Sztuczna inteligencja : jej korzenie i zakres

(I / XVI)

ĆWICZENIA

1. Utwórz i uzasadnij własną definicję sztucznej inteligencji.

2. Podaj kilka innych przykładów rozróżnienia Arystotelesa między materią i formą. Czy potrafisz pokazać, jak twoje przykłady mogą pasować do teorii abstrakcji?

3. Wiele tradycyjnych myśli zachodnich opierało się na relacjach umysł-ciało. Czy umysł i ciało to:

a. odrębne byty w jakiś sposób oddziałujące

b. umysł jest wyrazem "procesów fizycznych"

c. czy ciało jest jedynie iluzją racjonalnego umysłu?

Omów swoje przemyślenia na temat problemu umysł-ciało i jego znaczenia dla teorii sztucznej inteligencji.

4. Skrytykuj kryteria Turinga, aby oprogramowanie komputerowe było "inteligentne".

5. Opisz własne kryteria uznania oprogramowania komputerowego za "inteligentne".

6. Mimo że informatyka jest stosunkowo nową dyscypliną, filozofowie i matematycy zastanawiają się nad zagadnieniami związanymi z automatyzacją rozwiązywania problemów od tysięcy lat. Jakie jest Twoje zdanie na temat znaczenia tych zagadnień filozoficznych dla projektu urządzenia do inteligentnego rozwiązywania problemów? Uzasadnij swoją odpowiedź.

7. Biorąc pod uwagę różnice między architekturami współczesnych komputerów i architektury ludzkiego mózgu, jakie znaczenie mają badania nad strukturą fizjologiczną i funkcją systemów biologicznych w inżynierii programów AI? Uzasadnij swoją odpowiedź.

8. Wybierz jeden problem, który Twoim zdaniem uzasadniałby energię potrzebną do zaprojektowania specjalistycznego rozwiązania systemowego. Określ szczegółowo problem. Które aspekty tego rozwiązania byłyby najtrudniejsze do automatyzacji w oparciu o Twoją intuicję?

9. Dodaj jeszcze dwie korzyści dla systemów eksperckich do tych już wymienionych w tekście. Omów je w kategoriach wyników intelektualnych, społecznych lub finansowych.

10. Omów, dlaczego Twoim zdaniem problem "uczenia się" maszyn jest tak trudny.

11. Omów, czy uważasz, że komputer może zrozumieć i używać języka naturalnego (ludzkiego).

12. Wymień i omów dwa potencjalnie negatywne skutki rozwoju technologii sztucznej inteligencji dla społeczeństwa.

Materiały Szkoleniowe

Sztuczna inteligencja: próba definicji

Sztuczną inteligencję (AI) można zdefiniować jako dziedzinę informatyki zajmującą się automatyzacją inteligentnego zachowania. Ta definicja jest dla nas szczególnie odpowiednia, ponieważ podkreśla nasze przekonanie, że AI jest częścią informatyki i jako taka musi opierać się na solidnych teoretycznych i stosowanych zasadach tej dziedziny. Zasady te obejmują struktury danych stosowane w reprezentacji wiedzy, algorytmy potrzebne do zastosowania tej wiedzy oraz języki i techniki programowania wykorzystane w ich realizacji. Definicja ta cierpi jednak na fakt, że sama inteligencja nie jest zbyt dobrze zdefiniowana ani zrozumiana. Chociaż większość z nas jest pewna, że znamy inteligentne zachowanie, gdy je widzimy, wątpliwe jest, aby ktokolwiek mógł zbliżyć się do definiowania inteligencji w sposób, który byłby na tyle konkretny, aby pomóc w ocenie rzekomo inteligentnego programu komputerowego, jednocześnie przechwytując witalność i złożoność ludzkiego umysłu. W wyniku trudnego zadania polegającego na budowaniu ogólnej inteligencji badacze AI często przyjmują role inżynierów konstruujących określone inteligentne artefakty. Często mają one postać narzędzi diagnostycznych, prognostycznych lub wizualizacyjnych, które umożliwiają ich użytkownikom wykonywanie złożonych zadań. Przykłady tych narzędzi obejmują ukryte modele Markowa do rozumienia języka, zautomatyzowane systemy wnioskowania do dowodzenia nowych twierdzeń w matematyce, dynamiczne sieci bayesowskie do śledzenia sygnałów w sieciach korowych oraz wizualizacja wzorców danych ekspresji genów. Problem zdefiniowania pełnego pola sztucznej inteligencji polega na zdefiniowaniu samej inteligencji: czy inteligencja jest pojedynczym wydziałem, czy może jest tylko nazwą zbioru różnych i niezwiązanych ze sobą umiejętności? W jakim stopniu uczy się inteligencji w przeciwieństwie do istnienia a priori? Dokładnie to, co dzieje się, gdy następuje nauka? Czym jest kreatywność? Czym jest intuicja? Czy inteligencję można wywnioskować z obserwowalnego zachowania, czy też wymaga dowodów konkretnego mechanizmu wewnętrznego? W jaki sposób wiedza jest reprezentowana w tkance nerwowej żywej istoty i jakie są wnioski z projektowania inteligentnych maszyn? Czym jest samoświadomość; jaką rolę odgrywa w inteligencji? Ponadto, czy konieczne jest ukształtowanie inteligentnego programu komputerowego na podstawie tego, co wiadomo na temat ludzkiej inteligencji, czy też wystarczające jest ścisłe "inżynierskie" podejście do problemu? Czy w ogóle możliwe jest osiągnięcie inteligencji na komputerze, czy też inteligentna istota wymaga bogactwa wrażeń i doświadczenia, które można znaleźć tylko w istnieniu biologicznym? Są to pytania bez odpowiedzi, a wszystkie z nich pomogły ukształtować problemy i metodologie rozwiązań, które stanowią rdzeń współczesnej sztucznej inteligencji. W rzeczywistości część atrakcyjności sztucznej inteligencji polega na tym, że oferuje ona unikalne i potężne narzędzie do eksploracji dokładnie te pytania. AI oferuje medium i poligon doświadczalny dla teorii inteligencji: takie teorie można wypowiadać w języku programów komputerowych, a następnie testować i weryfikować poprzez wykonanie tych programów na rzeczywistym komputerze. Z tych powodów nasza początkowa definicja sztucznej inteligencji nie pozwala jednoznacznie zdefiniować pola. Co więcej, doprowadziło to tylko do dalszych pytań i paradoksalnego pojęcia dziedziny badań, której głównymi celami są własne definicje. Ale ta trudność w ustaleniu dokładnej definicji AI jest całkowicie odpowiednia. Sztuczna inteligencja jest wciąż młodą dyscypliną, a jej struktura, obawy i metody są mniej jasno określone niż w przypadku bardziej dojrzałej nauki, takiej jak fizyka. Sztuczna inteligencja zawsze bardziej

koncentrowała się na rozszerzaniu możliwości informatyki niż na określaniu jej granic. Utrzymanie tej eksploracji w oparciu o solidne zasady teoretyczne jest jednym z wyzwań, przed którymi stoją badacze AI . Ze względu na swój zakres i ambicje sztuczna inteligencja wymyka się prostej definicji. Na razie po prostu zdefiniujemy to jako zbiór problemów i metodologii badanych przez badaczy sztucznej inteligencji. Ta definicja może wydawać się głupia i pozbawiona znaczenia, ale stanowi ważny punkt: sztuczna inteligencja, jak każda nauka, jest przedsięwzięciem ludzkim i być może najlepiej jest ją rozumieć w tym kontekście. Istnieją powody, dla których jakakolwiek nauka, w tym sztuczna inteligencja, zajmuje się pewnym zestawem problemów i rozwija określony zestaw technik rozwiązywania tych problemów. W rozdziale 1 krótka historia sztucznej inteligencji oraz ludzi i założenia, które ją ukształtowały, wyjaśnią, dlaczego niektóre zestawy pytań zdominowały tę dziedzinę i dlaczego metody omówione w tej książce zostały zastosowane do ich rozwiązania.

AI: Wczesna historia i zastosowania

<u>Od Edenu do ENIAC: Postawy wobec inteligencji, wiedzy i sztuczności człowieka</u>

Prometeusz mówi o owocach swojego przestępstwa przeciwko bogom Olimpu: jego celem nie była jedynie kradzież ognia dla rodzaju ludzkiego, ale także oświecenie ludzkości poprzez dar inteligencji lub nous: rozumny umysł. Ta inteligencja stanowi podstawę całej ludzkiej technologii, a ostatecznie całej ludzkiej cywilizacji. Dzieło Ajschylosa, klasycznego dramaturga greckiego, ilustruje głęboką i starożytną świadomość niezwykłej siły wiedzy. Sztuczna inteligencja, w bezpośredniej trosce o dar Prometeusza, została zastosowana we wszystkich obszarach jego spuścizny - medycynie, psychologii, biologii, astronomii, geologii - i wielu obszarach naukowych, których Ajschylos nie mógł sobie wyobrazić. Chociaż działanie Prometeusza uwolniło ludzkość od choroby ignorancji, przyniosło mu również gniew Zeusa. Oburzony tą kradzieżą wiedzy, która wcześniej należał tylko do bogów Olimpu, Zeus nakazał, aby Prometeusz został przykuty do jałowej skały, aby cierpieć spustoszenie żywiołów na wieki. Pogląd, że ludzkie wysiłki mające na celu zdobycie wiedzy stanowią wykroczenia przeciwko prawom Bożym lub naturze, jest głęboko zakorzeniony w myśli zachodniej. Jest to podstawa historii Edenu i pojawia się w dziełach Dantego i Miltona. Zarówno Szekspir, jak i starożytni greccy tragicy przedstawiali ambicje intelektualne jako przyczynę katastrofy. Przekonanie, że pragnienie wiedzy musi ostatecznie doprowadzić do katastrofy, przetrwało przez całą historię, przeżywając Renesans, Wiek Oświecenia, a nawet postępy naukowe i filozoficzne dziewiętnastego i dwudziestego wieku. Dlatego nie powinniśmy się dziwić, że sztuczna inteligencja budzi tyle kontrowersji zarówno w środowisku akademickim, jak i popularnym. Rzeczywiście, zamiast rozwiać ten starożytny lęk przed konsekwencjami intelektualnej ambicji, nowoczesna technologia sprawiła, że konsekwencje te wydają się prawdopodobne, a nawet bezpośrednie. Legendy Prometeusza, Ewy i Fausta zostały powtórzone w tym języku społeczeństwa technologicznego. W swoim wstępie do Frankensteina, z podtytułem, co ciekawe, „The Modern Prometheus", Mary Shelley pisze:

"Wiele i długie były rozmowy między Lordem Byronem a Shelleyem, w których byłam pobożnym i cichym słuchaczem. Podczas jednej z nich dyskutowano o różnych doktrynach filozoficznych, między innymi o naturze zasady życia oraz o tym, czy istnieje jakiekolwiek prawdopodobieństwo jej odkrycia i przekazania. Mówili o eksperymentach doktora Darwina (nie mówię o tym, co doktor naprawdę zrobił lub powiedział, że to zrobił, ale, bardziej niż w moim celu, o czym wtedy mówiono jako o tym, co zrobił), który zachował kawałek wermiszelów w szklanej gablocie, aż w jakiś niezwykły sposób zaczął się poruszać dobrowolnym ruchem. W końcu nie da życia. Być może zwłoki zostaną ożywione; galwanizm dał znak do takich rzeczy: być może części składowe stworzenia mogą być wytwarzane, łączone i obdarzane żywotnym ciepłem."

Mary Shelley pokazuje nam, w jakim stopniu postępy naukowe, takie jak dzieło Darwina i odkrycie elektryczności, przekonały nawet nienaukowców, że działanie natury nie było boskimi tajemnicami, ale można je było rozbić i zrozumieć systematycznie. Potwór Frankensteina nie jest produktem szamańskich inkantacji lub niewymownych transakcji ze światem podziemnym: jest złożony z osobno "wyprodukowanych" komponentów i nasycony siłą witalną elektryczności. Chociaż dziewiętnastowieczna nauka nie była w stanie zrealizować celu, jakim jest pełne zrozumienie i stworzenie inteligentnego agent potwierdził pogląd, że tajemnice życia i intelektu mogą zostać ujawnione w świetle naukowej analizy.

Krótka historia podstaw sztucznej inteligencji

Zanim Mary Shelley w końcu i być może nieodwołalnie połączyła współczesną naukę z mitem prometiańskim, filozoficzne podstawy współczesnej pracy w sztucznej inteligencji rozwijały się przez kilka tysięcy lat. Chociaż kwestie moralne i kulturowe podniesione przez sztuczną inteligencję są zarówno interesujące, jak i ważne, nasze wprowadzenie bardziej odpowiednio dotyczy dziedzictwa intelektualnego AI. Logicznym punktem wyjścia dla takiej historii jest geniusz Arystotelesa, lub jak Dante w Boskiej komedii nazywa go "mistrzem tych, którzy wiedzą". Arystoteles splótł spostrzeżenia, cuda i obawy związane z wczesną grecką tradycją dzięki starannej analizie i zdyscyplinowanej myśli, które miały stać się standardem dla bardziej nowoczesnej nauki. Dla Arystotelesa najbardziej fascynującym aspektem natury była zmiana. W swojej fizyce zdefiniował swoją "filozofię przyrody" jako "badanie rzeczy, które się zmieniają". Rozróżniał materię i formę rzeczy: rzeźba wykonana jest z materialnego brązu i ma postać człowieka. Zmiana następuje, gdy brąz formuje się w nową formę. Rozróżnienie materia / forma stanowi filozoficzną podstawę dla współczesnych pojęć, takich jak obliczenia symboliczne i abstrakcja danych. W obliczeniach (nawet z liczbami) manipulujemy wzorcami, które są formami materiału elektromagnetycznego, a zmiany postaci tego materiału reprezentują aspekty procesu rozwiązania. Wyodrębnienie formy ze środka jej reprezentacji pozwala nie tylko na manipulowanie tymi formami obliczeniowo, ale także stanowi obietnicę teorii struktur danych, serca współczesnej informatyki. Wspiera również tworzenie "sztucznej" inteligencji. W swojej Metafizyce, poczynając od słów "Wszyscy ludzie z natury pragną wiedzieć", Arystoteles rozwinął naukę o rzeczach, które nigdy się nie zmieniają, w tym o swojej kosmologii i teologii. Bardziej istotna dla sztucznej inteligencji była jednak epistemologia Arystotelesa lub analiza tego, jak ludzie "znają" swój świat, omówiona w jego Logice. Arystoteles nazywał logikę "instrumentem" (organonem), ponieważ uważał, że samo badanie myśli było podstawą całej wiedzy. W swojej logice badał, czy niektóre twierdzenia można uznać za "prawdziwe", ponieważ są one związane z innymi rzeczami, o których wiadomo, że są "prawdziwe".

Zatem jeśli wiemy, że "wszyscy ludzie są śmiertelni" i że "Sokrates jest człowiekiem", możemy dojść do wniosku, że "Sokrates jest śmiertelny". Ten argument jest przykładem tego, co Arystoteles nazywał sylogizmem przy użyciu dedukcyjnej formy modus ponens. Chociaż formalne aksjomatyzowanie rozumowania wymagało kolejnych dwóch tysięcy lat do pełnego rozkwitu w dziełach Gottloba Frege, Bertranda Russella, Kurta Gödela, Alana Turinga, Alfreda Tarskiego i innych, jego korzenie można przypisać Arystotelesowi. Myśl renesansowa, budowana na tradycji greckiej, zapoczątkowała ewolucję odmiennego i silnego sposobu myślenia o ludzkości i jej związku ze światem przyrody. Nauka zaczęła zastępować mistycyzm jako sposób na zrozumienie natury. Zegary, a ostatecznie harmonogramy fabryczne zastąpiły rytmy natury dla tysięcy mieszkańców miast. Większość współczesnych nauk społecznych i fizycznych wywodzi się z poglądu, że procesy, naturalne lub sztuczne, mogą być analizowane matematycznie i rozumiane. W szczególności naukowcy i filozofowie zdali sobie sprawę, że sama myśl, sposób, w jaki wiedza jest reprezentowana i manipulowana w ludzkim umyśle, jest trudnym, ale niezbędnym przedmiotem badań naukowych. Być może głównym wydarzeniem w

rozwoju współczesnego poglądu na świat była rewolucja kopernikańska, zastąpienie starożytnego modelu wszechświata skoncentrowanego na Ziemi na ideę, że Ziemia i inne planety krążą wokół Słońca. Po wiekach "oczywistego" porządku, w którym naukowe wyjaśnienie natury kosmosu było zgodne z naukami religii i zdrowego rozsądku, zaproponowano drastycznie inny i wcale nieoczywisty model wyjaśniający ruchy ciał niebieskich. Być może po raz pierwszy nasze wyobrażenia o świecie były zasadniczo odmienne od wyglądu tego świata.

Ten podział między ludzkim umysłem a otaczającą go rzeczywistością, między wyobrażeniami o rzeczach i samych rzeczach, jest niezbędny do współczesnych badań nad umysłem i jego organizacją. Naruszenie to poszerzyły pisma Galileusza, którego obserwacje naukowe były dalej sprzeczne z "oczywistymi" prawdami o świecie naturalnym, a którego rozwój matematyki jako narzędzia do opisu tego świata podkreślał różnicę między światem a naszymi wyobrażeniami o nim. To właśnie z tego naruszenia ewoluowało współczesne pojęcie umysłu: introspekcja stała się powszechnym motywem w literaturze, filozofowie zaczęli badać epistemologię i matematykę, a systematyczne stosowanie metody naukowej rywalizowało ze zmysłami jako narzędziami do rozumienia świata. W 1620 r. Novum Organun Francisa Bacona zaoferował zestaw technik wyszukiwania dla tej nowej metodologii naukowej. Opierając się na arystotelesowskiej i platońskiej idei, że "forma" bytu była równoważna sumie jego niezbędnych i wystarczających "cech", Bacon sformułował algorytm określania istoty bytu. Najpierw stworzył zorganizowaną kolekcję wszystkich instancji encji, wyliczając cechy każdej z nich w tabeli. Następnie zebrał podobną listę negatywnych wystąpień bytu, koncentrując się szczególnie na bliskich wystąpieniach bytu, to znaczy tych, które odbiegały od "formy" bytu przez pojedyncze cechy. Następnie Bacon próbuje - ten krok nie jest do końca jasny - stworzyć systematyczną listę wszystkich cech istotnych dla bytu, czyli tych, które są wspólne dla wszystkich pozytywnych wystąpień bytu i których brakuje w negatywnych wystąpieniach. Interesujące jest, aby forma podejścia Francisa Bacona do uczenia się pojęć znalazła odzwierciedlenie we współczesnych algorytmach AI dla wyszukiwania w przestrzeni kosmicznej. Rozszerzenie algorytmów Bacona było również częścią programu sztucznej inteligencji do nauki odkrywania, odpowiednio zwanego Bacon. Program ten był w stanie zaindukować wiele praw fizycznych ze zbiorów danych związanych ze zjawiskami. Warto również zauważyć, że pytanie, czy możliwe jest zastosowanie algorytmu ogólnego do tworzenia dowodów naukowych, czekało na wyzwania matematyka z początku XX wieku Hilberta (jego entscheidungsproblem) i odpowiedź współczesnego geniuszu Alana Turinga (Testu Turinga)

Chociaż pierwsza maszyna licząca, liczydło, została stworzona przez Chińczyków w dwudziestym szóstym wieku p.n.e., dalsza mechanizacja procesów algebraicznych czekała na umiejętności siedemnastowiecznych Europejczyków. W 1614 r. Szkocki matematyk John Napier stworzył logarytmy, matematyczne transformacje, które pozwoliły na zwielokrotnienie i użycie wykładników zredukowane do dodawania i mnożenia. Napier stworzył również swoje kości, które posłużyły do przedstawienia wartości przepełnienia dla operacji arytmetycznych. Z tych kości później skorzystał Wilhelm Schickard (1592-1635), niemiecki matematyk i duchowny z Tybingi, który w 1623 r. wynalazł zegar obliczeniowy do dodawania i odejmowania. Ta maszyna zarejestrowała przepełnienie z obliczeń przez bicie zegara. Inną znaną maszyną liczącą była Pascaline, którą Blaise Pascal, francuski filozof i matematyk, stworzył w 1642 r. Chociaż mechanizmy Schickarda i Pascala ograniczały się do dodawania i odejmowania - w tym przenoszenia i pożyczania - pokazały, że procesy, które wcześniej uważano za wymagające ludzkiej myśli i umiejętności mogą być w pełni zautomatyzowane. Jak później stwierdził Pascal w Pensees (1670): "Maszyna arytmetyczna daje efekty bliższe myśleniu niż wszystkie działania zwierząt". Sukcesy Pascala w zakresie maszyn liczących zainspirowały Gottfrieda Wilhelma von Leibniza w 1694 r. do ukończenia działającej maszyny, która stała się znana jako Koło Leibniza. Zintegrował ruchomy wózek i ręczną korbę do napędzania kół i cylindrów, które wykonywały bardziej złożone operacje mnożenia i dzielenia. Leibniz był również zafascynowany możliwością zautomatyzowanej logiki dla dowodów

propozycji. Wracając do algorytmu specyfikacji encji Bacona, w którym pojęcia scharakteryzowano jako zbiór ich niezbędnych i wystarczających cech, Liebniz wyobraził sobie maszynę, która mogłaby obliczyć te cechy, aby uzyskać logicznie poprawne wnioski. Liebniz przewidział także maszynę odzwierciedlającą współczesne idee wnioskowania dedukcyjnego i dowodu, za pomocą których wytwarzanie wiedzy naukowej mogłoby zostać zautomatyzowane, rachunek rozumowania.

W siedemnastym i osiemnastym wieku wiele dyskusji poświęcono także zagadnieniom epistemologicznym; być może najbardziej wpływowa była praca René Descartesa, centralnej postaci w rozwoju nowoczesnych koncepcji myśli i teorii umysłu. W swoich Medytacjach Kartezjusz (1680) próbował znaleźć podstawę rzeczywistości wyłącznie poprzez introspekcję. Systematycznie odrzucając wkład swoich zmysłów jako niegodny zaufania, Kartezjusz był zmuszony wątpić nawet w istnienie świata fizycznego i pozostała mu jedynie rzeczywistość myśli; nawet jego własne istnienie musiało być uzasadnione w kategoriach myśli: "Cogito ergo sum" (tak ,myślę, dlatego jestem). Po tym, jak ustanowił swoje własne istnienie wyłącznie jako myślącą istotę, Kartezjusz wywnioskował istnienie Boga jako istotnego stwórcy i ostatecznie potwierdził rzeczywistość wszechświata fizycznego jako niezbędne stworzenie łagodnego Boga. Możemy tu dokonać dwóch spostrzeżeń: po pierwsze, schizma między umysłem a światem fizycznym stała się tak kompletna, że proces myślenia można omówić w oderwaniu od jakiegokolwiek określonego wkładu zmysłowego lub przedmiotu doczesnego; po drugie, związek między umysłem a światem fizycznym był tak słaby, że wymagał interwencji łagodnego Boga, aby poprzeć wiarygodną wiedzę o świecie fizycznym! Ten pogląd na dualność między umysłem a światem fizycznym leży u podstaw całej myśli Kartezjusza, w tym jego rozwoju geometrii analitycznej. Jak inaczej mógłby zjednoczyć tak pozornie ziemską gałąź matematyki, jak geometrię, z tak abstrakcyjną strukturą matematyczną, jak algebra? Dlaczego umieściliśmy tę dyskusję umysł / ciało w tekście na temat sztucznej inteligencji? Istnieją dwie konsekwencje tej analizy istotne dla przedsiębiorstwa AI:

1. Próbując oddzielić umysł od świata fizycznego, Kartezjusz i pokrewni myśliciele ustalili, że struktura idei świata niekoniecznie jest taka sama jak struktura ich przedmiotu. To leży u podstaw metodologii sztucznej inteligencji, a także dziedzin epistemologii, psychologii, znacznej części wyższej matematyki i większości współczesnej literatury: procesy mentalne istnieją samodzielnie, podlegają własnym prawom i mogą być badane same w sobie.

2. Po oddzieleniu umysłu i ciała filozofowie uznali za konieczne znalezienie sposobu na ponowne połączenie tych dwóch, ponieważ interakcja między Kartezjuszem umysłowym, res cogitans i fizycznym, res extensa, jest niezbędna dla ludzkiej egzystencji. Chociaż miliony słów zostały napisane na temat problemu umysł-ciało i zaproponowano liczne rozwiązania, nikt nie wyjaśnił oczywistych interakcji między stanami mentalnymi a działaniami fizycznymi, jednocześnie potwierdzając zasadniczą różnicę między nimi. Najszerzej akceptowana odpowiedź na ten problem oraz ta, która stanowi niezbędny fundament do badania AI, utrzymuje, że umysł i ciało wcale nie są zasadniczo różnymi istotami. Z tego punktu widzenia procesy umysłowe są rzeczywiście osiągane przez systemy fizyczne, takie jak mózgi (lub komputery). Procesy mentalne, podobnie jak procesy fizyczne, można ostatecznie scharakteryzować za pomocą formalnej matematyki. Lub, jak przyznał w swoim Lewiatanie XVII-wieczny angielski filozof Thomas Hobbes (1651): "Przez racjonalne myślenie rozumiem obliczenia".

AI a tradycje racjonalistyczne i empiryczne

Współczesne problemy badawcze w zakresie sztucznej inteligencji, podobnie jak w innych dyscyplinach naukowych, powstają i ewoluują poprzez połączenie nacisków historycznych, społecznych i kulturowych. Dwa z najbardziej znaczących nacisków na ewolucję AI to empiryczne i racjonalistyczne tradycje filozoficzne. Tradycja racjonalistyczna, jak widzieliśmy w poprzedniej części, miała wczesnego zwolennika w Platonie i była kontynuowana poprzez pisma Pascala, Kartezjusza i Liebniza. Dla

racjonalisty świat zewnętrzny jest rekonstruowany poprzez jasne i wyraźne idee matematyki. Krytyką tego dualistycznego podejścia jest przymusowe odłączenie systemów reprezentacyjnych od ich obszaru odniesienia. Problem polega na tym, czy znaczenie przypisane reprezentacji można zdefiniować niezależnie od warunków jej zastosowania. Jeśli świat różni się od naszych przekonań na temat świata, czy nasze stworzone koncepcje i symbole nadal mają znaczenie? Wiele programów AI ma bardzo dużo tego racjonalistycznego smaku. Na przykład wcześni planiści robotów opisywaliby swoją domenę aplikacji lub "świat" jako zestawy instrukcji rachunku predykatu, a następnie "plan" działania zostałby stworzony poprzez udowodnienie twierdzeń o tym "świecie". Hipoteza systemu fizycznego symboli Newella i Simona jest postrzegana przez wielu jako archetyp tego podejścia we współczesnej sztucznej inteligencji. Kilku krytyków skomentowało to racjonalistyczne uprzedzenie jako część niepowodzenia AI w rozwiązywaniu złożonych zadań, takich jak rozumienie ludzkich języków. Zamiast twierdzenia, że jest "prawdziwy" świat jasnych i odrębnych idei, empirycy nadal przypominają nam, że "nic nie wchodzi do umysłu inaczej niż poprzez zmysły". Ograniczenie to prowadzi do dalszych pytań, w jaki sposób człowiek może postrzegać ogólne koncepcje lub czyste formy jaskini Platona. Arystoteles był wczesnym empirystą, podkreślając w swojej De Anima ograniczenia ludzkiego systemu percepcyjnego. Nowocześni empirycy, zwłaszcza Hobbes, Locke i Hume, podkreślają tę wiedzę która musi być wyjaśniono przez introspektywną, ale empiryczną psychologię. Rozróżniają dwa typy percepcji zjawisk mentalnych z jednej strony, a myśl, pamięć i wyobraźnię z drugiej. Na przykład szkocki filozof David Hume rozróżnia wrażenia i pomysły. Wrażenia są żywe, odzwierciedlają obecność i istnienie zewnętrznego obiektu i nie podlegają dobrowolnej kontroli, jakości Dennetta. Z drugiej strony pomysły są mniej żywe i szczegółowe, a bardziej podlegają dobrowolnej kontroli podmiotu. Biorąc pod uwagę to rozróżnienie między wrażeniami a pomysłami, w jaki sposób może powstać wiedza? Dla Hobbesa, Locke'a i Hume'a podstawowym mechanizmem wyjaśniającym jest skojarzenie. Poszczególne właściwości percepcyjne są związane z powtarzanym doświadczeniem. To powtarzające się skojarzenie tworzy w umyśle skłonność do kojarzenia odpowiednich idei, co jest wstępem do behawiorystycznego podejścia XX wieku. Podstawową właściwością tego podejścia jest sceptycyzm Hume'a. Czysto opisowa relacja Hume'a o pochodzeniu pomysłów nie może, jak twierdzi, wspierać przekonania o przyczynowości. Nawet użycie logiki i indukcji nie może być racjonalnie poparte w tej radykalnej empistycznej epistemologii. W zapytaniu dotyczącym ludzkiego zrozumienia sceptycyzm Hume'a obejmował analizę cudów. Chociaż Hume nie odniósł się bezpośrednio do natury cudów, zakwestionował opartą na świadectwie wiarę w cuda. Ten sceptycyzm był oczywiście postrzegany jako bezpośrednie zagrożenie przez wierzących w Biblię, jak również przez wielu innych dostawców tradycji religijnych. Wielebny Thomas Bayes był zarówno matematykiem, jak i pastorem. Jeden z jego artykułów, zatytułowany Esej na temat rozwiązania problemu w doktrynie szans, dotyczył matematycznie pytań Hume'a. Twierdzenie Bayesa formalnie pokazuje, w jaki sposób ucząc się korelacji efektów działań, możemy określić prawdopodobieństwo ich przyczyn.

Skojarzone podejście wiedzy odgrywa znaczącą rolę w rozwoju struktur i programów reprezentacyjnych AI, na przykład w organizacji pamięci za pomocą sieci semantycznych i MOPS oraz pracy w zrozumieniu języka naturalnego. Konta stowarzyszeniowe mają istotny wpływ na uczenie maszynowe, szczególnie w sieciach łączności. Stowarzyszenie odgrywa również ważną rolę w psychologii poznawczej, w tym w schematach Bartletta i Piageta, a także w całym nurcie tradycji behawiorystycznej. Wreszcie, dzięki narzędziom AI do analizy stochastycznej, w tym bayesowskiej sieci wierzeń (BBN) i jego obecne rozszerzenia do kompletnych systemów Turinga pierwszego rzędu do modelowania stochastycznego, teorie asocjacyjne znalazły solidne podstawy matematyczne i dojrzałą moc ekspresyjną. Narzędzia bayesowskie są ważne dla badań, w tym diagnostyki, uczenia maszynowego i rozumienia języka naturalnego. Immanuel Kant, niemiecki filozof wyszkolony w tradycji racjonalistycznej, był pod silnym wpływem pisma Hume'a. W rezultacie rozpoczął nowoczesną syntezę

tych dwóch tradycji. Wiedza dla Kanta zawiera dwie współpracujące energie, element a priori pochodzący z powodu podmiotu oraz element a posteriori pochodzący z aktywnego doświadczenia. Doświadczenie ma sens tylko dzięki wkładowi przedmiotu. Bez zaproponowanej przez podmiot aktywnej formy organizacyjnej świat byłby niczym więcej niż przemijaniem wrażeń. Wreszcie, na poziomie osądu, twierdzenia Kanta, przekazywanie obrazów lub przedstawień są powiązane przez podmiot aktywny i przyjmowane jako różnorodne pozory tożsamości "obiektu". Realizm Kanta zapoczątkował współczesne przedsięwzięcie psychologów takich jak Bartlett, Brunner i Piaget. Praca Kanta ma wpływ na współczesne AI uczenia maszynowego, a także na dalszy rozwój konstruktywistycznej epistemologii

Rozwój logiki formalnej

Kiedy myślenie zaczęło być uważane za formę obliczeń, jego formalizacja i ewentualna mechanizacja były oczywistymi kolejnymi krokami. Jak wspomniano wcześniej, Gottfried Wilhelm von Leibniz wraz ze swoim Rachunkiem Filozoficznym wprowadził pierwszy system logiki formalnej oraz zaproponował maszynę do automatyzacji swoich zadań. Ponadto kroki i etapy tego mechanicznego rozwiązania można przedstawić jako ruch przez stany drzewa lub wykresu. Leonard Euler, w XVIII wieku, w swojej analizie "połączenia" mostów łączących brzegi rzek i wyspy w Królewcu, przedstawił studium reprezentacji, które mogą abstrakcyjnie uchwycić strukturę relacji na świecie, a także dyskretne kroki w obliczeniach dotyczących tych relacji. Formalizacja teorii grafów dała również możliwość przeszukiwania przestrzeni stanu, głównego narzędzia pojęciowego sztucznej inteligencji. Możemy użyć wykresów do modelowania głębszej struktury problemu. Węzły wykresu przestrzeni stanów reprezentują możliwe etapy rozwiązania problemu; łuki na wykresie przedstawiają wnioski, ruchy w grze lub inne etapy rozwiązania problemu. Rozwiązanie problemu polega na przeszukiwaniu wykresu przestrzeni stanów w poszukiwaniu ścieżki do rozwiązania. Opisując całą przestrzeń rozwiązań problemów, wykresy przestrzeni stanów stanowią potężne narzędzie do pomiaru struktury i złożoności problemów oraz analizy wydajności, poprawności i ogólności strategii rozwiązań. Jako jeden z pomysłodawców badań operacyjnych, a także projektant pierwszych programowalnych mechanicznych maszyn obliczeniowych, Charles Babbage, dziewiętnastowieczny matematyk, może również być uznany za wczesnego praktyka sztucznej inteligencji. Silnik różnicowy Babbage'a był maszyną specjalnego przeznaczenia do obliczania wartości niektórych funkcji wielomianowych i był tym prekursorem jego silnika analitycznego. Silnik analityczny, zaprojektowany, ale nie z powodzeniem skonstruowany za jego życia, był programowalną maszyną komputerową ogólnego przeznaczenia, która zapowiadała wiele założeń architektonicznych leżących u podstaw współczesnego komputera. Opisując silnik analityczny, Ada Lovelace, przyjaciółka, zwolennik i współpracownik Babbage'a powiedziała:

"Możemy najtrafniej powiedzieć, że silnik analityczny tka wzory algebraiczne, podobnie jak krosno żakardowe tka kwiaty i liście. Wydaje nam się, że w tym przypadku tkwi znacznie więcej oryginalności, niż silnik różnicowy może słusznie domagać się."

Inspiracją Babbage’a było jego pragnienie zastosowania technologii swoich czasów, aby uwolnić ludzi od znoju wykonywania obliczeń arytmetycznych. W tym sentymencie, podobnie jak w koncepcji komputerów jako urządzeń mechanicznych, Babbage myślał w kategoriach czysto dziewiętnastowiecznych. Jego silnik analityczny zawierał jednak również wiele nowoczesnych pojęć, takich jak oddzielenie pamięci i procesora, sklep i młyn w ujęciu Babbage'a, koncepcja maszyny cyfrowej zamiast analogowej oraz programowalność oparta na wykonaniu szeregu operacje zakodowane na perforowanych kartach kartonowych. Najbardziej uderzającą cechą opisu Ady Lovelace i ogólnie pracy Babbage'a jest traktowanie "wzorców" relacji algebraicznych jako bytów, które mogą być badane, charakteryzowane, a ostatecznie wdrażane i manipulowane mechanicznie bez

obawy o konkretne wartości, które są w końcu przeszedł przez młyn maszyny liczącej. Jest to przykładowa implementacja "abstrakcji i manipulacji formą" opisanej po raz pierwszy przez Arystotelesa i Liebniza. Cel stworzenia formalnego języka myśli pojawia się także w pracach George'a Boole'a, kolejnego dziewiętnastowiecznego matematyka, którego praca musi być uwzględniona w każdej dyskusji na temat korzeni sztucznej inteligencji. Chociaż przyczyniał się do wielu dziedzin matematyki, jego najbardziej znaną pracą była matematyczna formalizacja praw logiki, osiągnięcie, które stanowi sedno współczesnej informatyki. Chociaż rola algebry Boolean w projektowaniu układów logicznych jest dobrze znana, cele własne Boole w rozwoju jego systemu wydają się bliższe celom współczesnych badaczy AI. W pierwszym rozdziale Analizy prawa myśli, na której oparte są matematyczne teorie logiki i prawdopodobieństwa, Boole opisał swoje cele jako:

„zbadać podstawowe prawa tych operacji umysłu, za pomocą których odbywa się rozumowanie: dać im wyraz w symbolicznym języku rachunku różniczkowego i na tej podstawie ustanowić naukę logiki i poinstruować jej metodę; … I wreszcie, aby zebrać z różnych elementów prawdy przedstawionych w trakcie tych dociekań pewne prawdopodobne wskazówki dotyczące natury i budowy ludzkiego umysłu.”

Znaczenie osiągnięcia Boole’a polega na niezwykłej sile i prostocie opracowanego przez niego systemu: trzy operacje, "ORAZ" (oznaczone przez ? lub ?), "LUB" (oznaczone przez + lub ?) oraz "NIE" (oznaczone przez), uformował serce swojego rachunku logicznego. Operacje te pozostały podstawą wszystkich późniejszych zmian logiki formalnej, w tym projektowania nowoczesnych komputerów. Zachowując znaczenie tych symboli prawie identycznych z odpowiadającymi im operacjami algebraicznymi, Boole zauważył, że "Symbole logiki podlegają ponadto specjalnemu prawu, któremu nie podlegają symbole ilości jako takie". Prawo to stwierdza, że dla dowolnego X, elementu w algebrze, X ? X = X (lub że gdy coś okaże się prawdą, powtórzenie nie może zwiększyć tej wiedzy). Doprowadziło to do charakterystycznego ograniczenia wartości boolowskich tylko do dwóch liczb, które mogą spełnić to równanie: 1 i 0. Z tego wglądu wynikają standardowe definicje mnożenia boolowskiego (AND) i dodawania (OR). System Boole'a nie tylko stanowił podstawę arytmetyki binarnej, ale także pokazał, że niezwykle prosty system formalny był wystarczający do uchwycenia pełnej mocy logiki. To założenie i system Boole opracowany w celu wykazania, że stanowią podstawę wszystkich współczesnych wysiłków zmierzających do sformalizowania logiki, od Russell i Principia Mathematica Russella, poprzez pracę Turinga i Gödela, aż po nowoczesne zautomatyzowane systemy wnioskowania. Gottlob Frege w swoich Podstawach arytmetyki stworzył matematyczny język specyfikacji do opisywania podstaw arytmetyki w jasny i precyzyjny sposób. Za pomocą tego języka Frege sformalizował wiele problemów, które zostały po raz pierwszy rozwiązane przez Logikę Arystotelesa. Język Frege'a, zwany obecnie rachunkiem predykatów pierwszego rzędu, oferuje narzędzie do opisywania zdań i przypisywania wartości prawdy, które składają się na elementy rozumowania matematycznego, i opisuje aksjomatyczną podstawę "znaczenia" tych wyrażeń. System formalny rachunku predykatów, który obejmuje symbole predykatów, teorię funkcji i zmienne kwantyfikowane, miał być językiem opisu matematyki i jej filozoficznych podstaw. Odgrywa także fundamentalną rolę w tworzeniu teorii reprezentacji sztucznej inteligencji. Rachunek predykcyjny pierwszego rzędu oferuje narzędzia niezbędne do automatyzacji rozumowania: język wyrażeń, teorię założeń związanych ze znaczeniem wyrażeń i logicznie poprawny rachunek do wnioskowania o nowych prawdziwych wyrażeniach. Prace Whitehead i Russella są szczególnie ważne dla podstaw sztucznej inteligencji, ponieważ ich wyznaczonym celem było uzyskanie całej matematyki poprzez formalne operacje na zbiorze aksjomatów. Chociaż wiele systemów matematycznych zbudowano z podstawowych aksjomatów, interesujące jest zaangażowanie Russella i Whitehead w matematykę jako system czysto formalny. Oznaczało to, że aksjomaty i twierdzenia byłyby traktowane wyłącznie jako ciągi znaków: dowody przebiegałyby wyłącznie poprzez zastosowanie ściśle określonych reguł manipulowania tymi ciągami.

Nie byłoby polegania na intuicji ani znaczeniu twierdzeń jako podstawy dowodów. Każdy krok dowodu wynikał ze ścisłego zastosowania reguł formalnych (składniowych) do aksjomatów lub wcześniej udowodnionych twierdzeń, nawet jeśli tradycyjne dowody mogłyby uznać taki krok za "oczywisty". Jakie "znaczenie" twierdzeń i aksjomatów systemu w stosunku do świata byłoby niezależne od ich logicznych pochodnych. Takie podejście do rozumowania matematycznego w kategoriach czysto formalnych (a zatem mechanicznych) zapewniło istotną podstawę dla jego automatyzacji na fizycznych komputerach. Logiczna składnia i formalne reguły wnioskowania opracowane przez Russella i Whitehead są nadal podstawą automatycznych systemów dowodzenia twierdzeń, a także teoretycznych podstaw sztucznej inteligencji.

Alfred Tarski jest kolejnym matematykiem, którego praca jest niezbędna dla podstaw sztucznej inteligencji. Tarski stworzył teorię odniesienia, w której dobrze sformułowane formuły Frega lub Russella i Whiteheada można precyzyjnie odnieść do świata fizycznego. Ten wgląd leży u podstaw większości teorii semantyki formalnej. W swoim artykule Semantyczna koncepcja prawdy i podstawa semantyki Tarski opisuje swoją teorię relacji i relacji wartości z prawdą. Współcześni informatycy, zwłaszcza Scott, Strachey, Burstall i Plotkin, powiązali tę teorię z językami programowania i innymi specyfikacjami obliczeniowymi. Chociaż w XVIII, XIX i na początku XX wieku formalizacja nauki i matematyki stworzyła intelektualną przesłankę badań nad sztuczną inteligencją, dopiero w XX wieku i wprowadzeniu komputera cyfrowego AI stało się realną dyscypliną naukową. Do końca lat 40. XX wieku elektroniczne komputery cyfrowe wykazały potencjał do zapewnienia pamięci i mocy obliczeniowej wymaganej przez inteligentne programy. Możliwe było teraz wdrożenie formalnych systemów wnioskowania na komputerze i empiryczne przetestowanie ich wystarczalności do wykazania inteligencji. Istotnym elementem nauki o sztucznej inteligencji jest zaangażowanie w komputery cyfrowe jako narzędzie wyboru do tworzenia i testowania teorii inteligencji. Komputery cyfrowe to nie tylko narzędzie do testowania teorii inteligencji. Ich architektura sugeruje również specyficzny paradygmat takich teorii: inteligencja jest formą przetwarzania informacji. Na przykład pojęcie wyszukiwania jako metodologii rozwiązywania problemów wynika bardziej z sekwencyjnego charakteru działania komputera niż z dowolnego biologicznego modelu inteligencji. Większość programów AI reprezentuje wiedzę w jakimś formalnym języku, który jest następnie manipulowany przez algorytmy, honorując rozdzielenie danych i programu, fundamentalne dla stylu obliczeniowego von Neumanna. Logika formalna stała się ważnym narzędziem reprezentacyjnym dla badań nad AI, podobnie jak teoria grafów odgrywa nieodzowną rolę w analizie przestrzeni problemowych, a także stanowi podstawę dla sieci semantycznych i podobnych modeli znaczeń semantycznych. Te techniki i formalizacje zostały szczegółowo omówione w całym tekście; wspominamy o nich tutaj, aby podkreślić symbiotyczny związek między komputerem cyfrowym a teoretycznymi podstawami sztucznej inteligencji. Często zapominamy, że narzędzia, które tworzymy na własne potrzeby, kształtują naszą koncepcję świata poprzez ich strukturę i ograniczenia. Chociaż pozornie restrykcyjne, ta interakcja jest istotnym aspektem ewolucji ludzkiej wiedzy: narzędzie (i teorie naukowe są ostatecznie tylko narzędziami) opracowuje się w celu rozwiązania konkretnego problemu. Ponieważ jest używane i ulepszane, samo narzędzie wydaje się sugerować inne aplikacje, prowadząc do nowych pytań, a ostatecznie do rozwoju nowych narzędzi

Test Turinga

Jeden z pierwszych artykułów na temat inteligencji maszyn, szczególnie w odniesieniu do współczesnego komputera cyfrowego, został napisany w 1950 r. przez brytyjskiego matematyka Alana Turinga. Maszyny obliczeniowe i inteligencja pozostają na czasie zarówno w ocenie argumentów przeciwko możliwości stworzenia inteligentnej maszyny obliczeniowej, jak i w odpowiedziach na te argumenty. Turing, znany głównie ze swojego wkładu w teorię obliczalności, zastanawiał się, czy

rzeczywiście można zmusić maszynę do myślenia. Zauważając, że fundamentalne dwuznaczności samego pytania (co to jest myślenie? Co to jest maszyna?) wyklucza jakąkolwiek racjonalną odpowiedź, zaproponował zastąpienie pytania o inteligencję jaśniej zdefiniowanym testem empirycznym. Test Turinga mierzy wydajność rzekomo inteligentnej maszyny w porównaniu z ludzką istotą, prawdopodobnie najlepszym i jedynym standardem inteligentnego zachowania. Test, który Turing nazwał grą imitacji, umieszcza maszynę i człowieka w pokojach poza drugim człowiekiem, określanym jako osoba przesłuchująca. Osoba przesłuchująca nie jest w stanie widzieć ani mówić bezpośrednio do żadnego z nich, nie wie który podmiot jest faktycznie maszyną i może się z nimi komunikować wyłącznie za pomocą urządzenia tekstowego, takiego jak terminal. Przesłuchujący proszony jest o odróżnienie komputera od człowieka wyłącznie na podstawie ich odpowiedzi na pytania zadawane za pomocą tego urządzenia. Jeśli przesłuchujący nie potrafi odróżnić maszyny od człowieka, to, jak twierdzi Turing, można założyć, że maszyna jest inteligentna. Izolując przesłuchującego zarówno od maszyny, jak i od drugiego uczestnika, test zapewnia, że nie będzie stronniczy z powodu wyglądu maszyny lub jakichkolwiek właściwości mechanicznych jego głosu. Przesłuchujący może jednak zadawać dowolne pytania, bez względu na to, jak przebiegły lub pośredni, w celu odkrycia tożsamości komputera. Na przykład, śledczy może poprosić oba podmioty o wykonanie raczej zaangażowanego obliczenia arytmetycznego, zakładając, że komputer będzie bardziej prawdopodobny, aby uzyskać poprawną odpowiedź niż człowiek; aby przeciwdziałać tej strategii, komputer będzie musiał wiedzieć, kiedy nie uzyska prawidłowej odpowiedzi na takie problemy, aby wyglądać jak człowiek. Aby odkryć tożsamość człowieka na podstawie natury emocjonalnej, śledczy może poprosić oba podmioty o odpowiedź na wiersz lub dzieło sztuki; strategia ta będzie wymagała od komputera wiedzy o emocjonalnym składzie ludzi.

Ważnymi cechami testu Turinga są:

1. Próbuje podać obiektywne pojęcie inteligencji, tj. zachowanie znanej inteligentnej istoty w odpowiedzi na określony zestaw pytań. Zapewnia to standard określania inteligencji, która pozwala uniknąć nieuniknionych dyskusji na temat jej "prawdziwej" natury.

2. Zapobiega nam zejście z boku na takie mylące i obecnie niemożliwe do odpowiedzi pytania, jak to, czy komputer korzysta z odpowiednich procesów wewnętrznych, czy też maszyna jest faktycznie świadoma swoich działań.

3. Eliminuje wszelkie uprzedzenia na rzecz żywych organizmów, zmuszając przesłuchującego do skupienia się wyłącznie na treści odpowiedzi na pytania

Ze względu na te zalety test Turinga stanowi podstawę wielu schematów wykorzystywanych do oceny współczesnych programów AI. Program, który potencjalnie uzyskał inteligencję w niektórych obszarach wiedzy specjalistycznej, można ocenić, porównując jego skuteczność w odniesieniu do danego zestawu problemów z działaniem człowieka-eksperta. Ta technika oceny jest tylko odmianą testu Turinga: grupa ludzi jest proszona o ślepe porównanie wydajności komputera i człowieka w odniesieniu do określonego zestawu problemów. Jak zobaczymy, metodologia ta stała się niezbędnym narzędziem zarówno w rozwoju, jak i weryfikacji nowoczesnych systemów eksperckich. Test Turinga, pomimo intuicyjnego odwołania, jest podatny na szereg uzasadnionych zarzutów. Jednym z najważniejszych z nich jest nastawienie na czysto symboliczne zadania rozwiązywania problemów. Nie testuje zdolności wymagających umiejętności percepcyjnych lub zręczności manualnej, mimo że są to ważne elementy ludzkiej inteligencji. Odwrotnie, czasami sugeruje się, że test Turinga niepotrzebnie ogranicza inteligencję maszyny do dopasowania do ludzkiej formy. Być może inteligencja maszynowa po prostu różni się od inteligencji ludzkiej, a próba jej oceny w kategoriach ludzkich jest fundamentalnym błędem. Czy naprawdę chcielibyśmy, aby maszyna wykonywała matematykę tak wolno i niedokładnie

jak człowiek? Czy inteligentna maszyna nie powinna wykorzystywać własnych zasobów, takich jak duża, szybka i niezawodna pamięć, zamiast próbować naśladować ludzkie poznanie? W rzeczywistości wielu współczesnych praktyków AI uważa odpowiedź na pełne wyzwanie testu Turinga za pomyłkę i poważne odwrócenie uwagi od ważniejszej pracy: rozwijania ogólnych teorii wyjaśniających mechanizmy inteligencji u ludzi i maszyn oraz stosowania tych teorii do opracowania narzędzi do rozwiązywania konkretnych, praktycznych problemów. Chociaż zgadzamy się w dużej mierze z obawami Forda i Hayesa, nadal postrzegamy test Turinga jako ważny element weryfikacji i walidacji nowoczesnego oprogramowania AI. Turing zajął się również wykonalnością budowy inteligentnego programu na komputerze cyfrowym. Myśląc w kategoriach konkretnego modelu obliczeń (elektroniczna maszyna do obliczania stanów dyskretnych), sformułował dobrze uzasadnione przypuszczenia dotyczące pojemności pamięci, złożoności programu i podstawowej filozofii projektowania wymaganej dla takiego systemu. Na koniec skierował szereg moralnych, filozoficznych i naukowych zastrzeżeń do możliwości zbudowania takiego programu pod względem faktycznej technologii. Warto rozważyć dwa z zarzutów przytoczonych przez Turinga. Sprzeciw Lady Lovelace, po raz pierwszy stwierdzony przez Adę Lovelace, dowodzi, że komputery mogą robić tylko to, co im powiedziano, a zatem nie mogą wykonywać oryginalnych (a więc inteligentnych) działań. Sprzeciw ten stał się uspokajającą, choć nieco wątpliwą częścią współczesnego folkloru technologicznego. Systemy eksperckie, zwłaszcza w zakresie rozumowania diagnostycznego, doszły do nieprzewidzianych przez projektantów wniosków. Rzeczywiście, wielu badaczy uważa, że ludzką kreatywność można wyrazić w programie komputerowym. Drugi pokrewny zarzut, Argument z nieformalności zachowania, twierdzi, że niemożliwe jest stworzenie zestawu reguł, które dokładnie dadzą jednostce do zrozumienia, co robić w każdych możliwych okolicznościach. Z pewnością elastyczność, która umożliwia inteligencji biologicznej reagowanie na niemal nieskończony zakres sytuacji w rozsądny, choć niekoniecznie optymalny sposób, jest cechą inteligentnego zachowania. Chociaż prawdą jest, że struktura sterowania stosowana w większości tradycyjnych programów komputerowych nie wykazuje dużej elastyczności ani oryginalności, nie jest prawdą, że wszystkie programy muszą być napisane w ten sposób. Rzeczywiście, większość pracy w AI w ciągu ostatnich 25 lat polegała na opracowaniu języków programowania i modeli, takich jak systemy produkcyjne, systemy obiektowe, reprezentacje sieci neuronowej i inne, które próbują pokonać ten niedobór. Wiele współczesnych programów AI składa się z kolekcji elementów modułowych lub reguł zachowania, które nie są wykonywane w sztywnej kolejności, ale są wywoływane w razie potrzeby w odpowiedzi na strukturę konkretnego wystąpienia problemu. Dopasowanie wzorców pozwala na stosowanie ogólnych reguł w szeregu instancji. Systemy te mają wyjątkową elastyczność, która umożliwia stosunkowo małym programom prezentowanie szerokiej gamy możliwych zachowań w odpowiedzi na różne problemy i sytuacje.

To, czy systemy te mogą ostatecznie zostać stworzone w celu wykazania elastyczności wykazywanej przez żywy organizm, jest nadal przedmiotem wielu dyskusji. Laureat Nagrody Nobla Herbert Simon argumentował, że duża część oryginalności i różnorodności zachowań żywych stworzeń wynika raczej z bogactwa ich środowiska niż ze złożoności ich wewnętrznych programów. W The Sciences of the Artificial Simon opisuje mrówkę poruszającą się okrężnie po nierównym i zagraconym odcinku ziemi. Chociaż ścieżka mrówki wydaje się dość złożona, Simon twierdzi, że cel mrówki jest bardzo prosty: jak najszybszy powrót do kolonii. Zakręty na jej drodze są spowodowane przeszkodami, które napotka na swojej drodze. Simon konkluduje, że mrówka, postrzegana jako system zachowań, jest dość prosta. Pozorna złożoność jej zachowania w czasie jest w dużej mierze odzwierciedleniem złożoności środowiska, w którym się znajduje. Pomysł ten, jeśli ostatecznie okaże się, że ma zastosowanie do organizmów o wyższej inteligencji, a także do tak prostych stworzeń jak owady, stanowi silny argument, że takie systemy są stosunkowo proste, a zatem zrozumiałe. Warto zauważyć, że jeśli zastosujemy ten pomysł do ludzi, stanie się on silnym argumentem za znaczeniem kultury w

kształtowaniu inteligencji. Zamiast rosnąć w ciemności jak grzyby, inteligencja wydaje się zależeć od interakcji z odpowiednio bogatym środowiskiem. Kultura jest tak samo ważna w tworzeniu ludzi, jak ludzie w tworzeniu kultury. Zamiast oczerniać nasze intelekty, ta idea podkreśla cudowne bogactwo i spójność kultur, które powstały z życia oddzielnych ludzi. W rzeczywistości pomysł, że inteligencja wyłania się z interakcji poszczególnych elementów społeczeństwa, jest jednym ze spostrzeżeń wspierających podejście do technologii AI.

Biologiczne i społeczne modele inteligencji: teorie agentów

Do tej pory podchodziliśmy do problemu budowy inteligentnych maszyn z punktu widzenia matematyki, z domyślnym przekonaniem, że logiczne rozumowanie jest paradygmatem samej inteligencji, a także z zaangażowaniem w "obiektywne" podstawy logicznego rozumowania. Ten sposób patrzenia na wiedzę, język i myśl odzwierciedla racjonalistyczną tradycję zachodniej filozofii, która ewoluowała przez Platona, Galileusza, Kartezjusza, Leibniza i wielu innych filozofów omówionych wcześniej. Odzwierciedla również podstawowe założenia testu Turinga, w szczególności jego nacisk na rozumowanie symboliczne jako test inteligencji oraz przekonanie, że proste porównanie z ludzkim zachowaniem było wystarczające do potwierdzenia inteligencji maszynowej.

Poleganie na logice jako sposobie reprezentowania wiedzy i logicznym wnioskowaniu jako podstawowym mechanizmie inteligentnego rozumowania są tak dominujące w zachodniej filozofii, że ich "prawda" często wydaje się oczywista i niepodważalna. Nic więc dziwnego, że podejścia oparte na tych założeniach zdominowały naukę o sztucznej inteligencji od samego początku prawie do dnia dzisiejszego.

W drugiej połowie XX wieku pojawiły się jednak liczne wyzwania dla filozofii racjonalistycznej. Różne formy relatywizmu filozoficznego kwestionują obiektywne podstawy języka, nauki, społeczeństwa i samej myśli. Późniejsza filozofia Ludwiga Wittgensteina (zmusiła nas do ponownego rozważenia podstaw znaczenia w językach naturalnych i formalnych. Praca Godela i Turinga podważyła podstawy samej matematyki. Myśl postmodernistyczna zmieniła nasze rozumienie znaczenia i wartości w sztuce i społeczeństwie. Sztuczna inteligencja nie była odporna na te krytyki; w rzeczywistości trudności, jakie napotkała AI w osiąganiu swoich celów, są często traktowane jako dowód niepowodzenia racjonalistycznego punktu widzenia. Dwie tradycje filozoficzne, Wittgenstein, a także Husserl i Heidegger są kluczowe dla ponownej oceny zachodniej tradycji filozoficznej. W swojej późniejszej pracy Wittgenstein zakwestionował wiele założeń tradycji racjonalistycznej, w tym podstawy języka, nauki i wiedzy. Język ludzki był głównym przedmiotem analizy Wittgensteina: zakwestionował pogląd, że język wywodzi swoje znaczenie z jakiejkolwiek obiektywnej podstawy. Dla Wittgensteina, a także teorii aktu mowy, opracowanej przez Austina i jego zwolenników, znaczenie każdej wypowiedzi zależy od jej umiejscowienia w ludzkim kontekście kulturowym. Nasze rozumienie znaczenia słowa "krzesło" zależy na przykład od posiadania ciała fizycznego, które odpowiada pozycji siedzącej i kulturowych konwencji używania krzeseł. Kiedy na przykład duży, płaski kamień jest krzesłem? Dlaczego to dziwne nazywanie tronu Anglii krzesłem? Jaka jest różnica między rozumieniem człowieka przez człowieka a krzesłem lub psem, niezdolnym do siedzenia w ludzkim sensie? Opierając się na swoich atakach na podstawy znaczenia, Wittgenstein argumentował, że powinniśmy postrzegać użycie języka w kategoriach dokonywanych wyborów i działań podejmowanych w zmieniającym się kontekście kulturowym. Wittgenstein nawet rozszerzył swoją krytykę na naukę i matematykę, argumentując, że są one tak samo konstrukcjami społecznymi, jak używanie języka. Husserl, ojciec fenomenologii, był zaangażowany w abstrakcje zakorzenione w konkretnym Lebenswelt lub świecie życia: model racjonalistyczny był bardzo wtórny w stosunku do konkretnego świata, który go wspierał. Dla Husserla, jak również dla jego ucznia Heideggera i ich zwolennika Merleau-Ponty'ego, inteligencja nie wiedziała, co jest prawdą, ale raczej wiedziała, jak radzić sobie w świecie, który ciągle się zmieniał i ewoluował.

Gadamer również przyczynił się do tej tradycji. Dla egzystencjalisty / fenomenologa inteligencja jest postrzegana jako przetrwanie na świecie, a nie jako zbiór logicznych twierdzeń o świecie (w połączeniu z pewnym schematem wnioskowania). Wielu autorów, na przykład Dreyfus oraz Winograd i Flores, wykorzystali prace Wittgensteina i Husserla / Heideggera w swojej krytyce AI. Chociaż wielu praktyków AI nadal rozwija racjonalną / logiczną agendę, znaną również jako GOFAI lub Good Old Fashioned AI, coraz większa liczba badaczy w tej dziedzinie włączyła te krytyki do nowych i ekscytujących modeli inteligencji. Zgodnie z naciskiem Wittgensteina na antropologiczne i kulturowe korzenie wiedzy, dla ich inspiracji zwrócili się do społecznych, czasem określanych jako oparte na agentach lub umiejscowionych, modeli inteligentnego zachowania.

Jako przykład alternatywy dla podejścia opartego na logice, badania w ramach uczenia się przez konfrontację nie uwzględniają logiki i funkcjonowania racjonalnego umysłu w celu osiągnięcia inteligencji poprzez modelowanie architektury fizycznego mózgu. Neuronowe modele inteligencji podkreślają zdolność mózgu do przystosowania się do świata, w którym się znajduje, poprzez modyfikację relacji między poszczególnymi neuronami. Zamiast reprezentować wiedzę w wyraźnych logicznych zdaniach, ujmują ją w sposób dorozumiany, jako właściwość wzorców relacji. Inny biologiczny model inteligencji czerpie inspirację z procesów, w których cały gatunek dostosowuje się do swojego otoczenia. Praca w sztucznym życiu i algorytmy genetyczne stosują zasady ewolucji biologicznej do problemów ze znalezieniem rozwiązania trudnych problemów. Programy te nie rozwiązują problemów, logicznie je logując; raczej rozmnażają populacje konkurencyjnych rozwiązań kandydujących i napędzają je do ewolucji coraz lepszych rozwiązań poprzez proces wzorowany na ewolucji biologicznej: słabe rozwiązania kandydujące zazwyczaj umierają, a te, które pokazują obietnicę rozwiązania problemu, przetrwają i rozmnażają się przez tworzenie nowych rozwiązań z elementów ich udanych rodziców. Systemy społeczne stanowią kolejną metaforę inteligencji, ponieważ wykazują globalne zachowania, które pozwalają im rozwiązywać problemy, które wprawiłyby w zakłopotanie każdego z ich członków. Na przykład, chociaż nikt nie jest w stanie dokładnie przewidzieć liczby bochenków chleba, które zostaną spożyte w Nowym Jorku w danym dniu, cały system nowojorskich piekarni robi doskonałą robotę, utrzymując miasto zaopatrzone w chleb i robiąc to z minimalną ilością odpadów. Rynek papierów wartościowych doskonale sobie radzi z ustalaniem względnych wartości setek firm, chociaż każdy inwestor ma ograniczoną wiedzę o kilku firmach. Ostatni przykład pochodzi z nowoczesnej nauki. Osoby z uniwersytetów, przemysłu lub instytucji rządowych koncentrują się na typowych problemach. Ponieważ konferencje i czasopisma są głównymi środkami komunikacji, problemy ważne dla całego społeczeństwa są atakowane i rozwiązywane przez poszczególnych agentów pracujących częściowo niezależnie, chociaż postęp w wielu przypadkach jest również napędzany przez agencje finansujące. Przykłady te dotyczą dwóch tematów: po pierwsze, poglądu na inteligencję zakorzenioną w kulturze i społeczeństwie, a w konsekwencji pojawiającą się. Drugim tematem jest to, że inteligencja znajduje odzwierciedlenie w zbiorowych zachowaniach dużej liczby bardzo prostych, półautonomicznych jednostek lub agentów. Bez względu na to, czy czynniki te są komórkami nerwowymi, poszczególnymi członkami gatunku lub pojedynczą osobą w społeczeństwie, ich interakcje wytwarzają inteligencję. Jakie są główne tematy wspierające zorientowane na agenta i wschodzące spojrzenie na inteligencję? Zawierają:

1. Agenci są autonomiczni lub półautonomiczni. Oznacza to, że każdy agent ma pewne obowiązki w rozwiązywaniu problemów, przy niewielkiej lub żadnej wiedzy o tym, co robią inni agenci lub jak to robią. Każdy agent rozwiązuje problem samodzielnie i albo generuje wynik (robi coś), albo zgłasza wyniki innym członkom społeczności (agent komunikujący się).

2. Agenci są "usytuowani". Każdy agent jest wrażliwy na otaczające go środowisko i (zwykle) nie ma wiedzy o pełnej domenie wszystkich agentów. Zatem wiedza agenta ogranicza się do zadań: "plik-

przetwarzam" lub "ściana-obok mnie" bez znajomości całkowitego zakresu plików lub ograniczeń fizycznych w zadanie rozwiązywania problemów.

3. Agenci są interaktywni. Oznacza to, że tworzą zbiór osób, które współpracują przy określonym zadaniu. W tym sensie mogą być postrzegane jako "społeczeństwo" i, podobnie jak w przypadku społeczeństwa ludzkiego, wiedza, umiejętności i obowiązki, nawet postrzegane jako zbiorowe, są rozłożone na populację poszczególnych osób.

4. Społeczeństwo agentów jest zorganizowane. W większości poglądów na rozwiązywanie problemów zorientowanych na agentów każda osoba, choć ma swoje unikalne środowisko i zestaw umiejętności, będzie koordynować z innymi agentami w ogólnym rozwiązywaniu problemów. Tak więc ostateczne rozwiązanie będzie postrzegane nie tylko jako zbiorowe, ale także jako oparte na współpracy.

5. Wreszcie zjawisko inteligencji w tym środowisku ma charakter "wschodzący". Chociaż poszczególne podmioty są postrzegane jako posiadające zestawy umiejętności i obowiązków, ogólny wynik współpracy można postrzegać jako większy niż suma poszczególnych uczestników. Inteligencja jest postrzegana jako zjawisko zamieszkujące i wyłaniające się ze społeczeństwa, a nie tylko własność pojedynczego agenta.

W oparciu o te obserwacje definiujemy agenta jako element społeczeństwa, który może postrzegać (często ograniczone) aspekty swojego środowiska i oddziaływać na to środowisko bezpośrednio lub poprzez współpracę z innymi agentami. Większość inteligentnych rozwiązań wymaga różnych agentów. Należą do nich agenci zdalni, którzy po prostu przechwytują i przekazują informacje, agenci koordynacyjni, którzy mogą wspierać interakcje między innymi agentami, agenci wyszukiwania, którzy mogą badać wiele informacji i zwracają wybraną ich część, agenci uczący się, którzy mogą badać zbiory informacji oraz tworzyć koncepcje lub uogólnienia, a także podmioty decyzyjne, które mogą zarówno realizować zadania, jak i dochodzić do wniosków w świetle ograniczonych informacji i przetwarzania. Wracając do starszej definicji inteligencji, agentów można postrzegać jako mechanizmy wspierające podejmowanie decyzji w kontekście ograniczonych zasobów przetwarzania. Głównymi warunkami do zaprojektowania i zbudowania takiego społeczeństwa są:

1. struktury do przedstawiania informacji,

2. strategie wyszukiwania za pomocą alternatywnych rozwiązań oraz

3. tworzenie architektur, które mogą wspierać interakcję agentów.

Pozostałe części, zawierają zalecenia dotyczące budowy narzędzi wsparcia dla tego stowarzyszenia agentów, a także wiele przykładów rozwiązywania problemów opartych na agentach. Nasza wstępna dyskusja na temat teorii automatycznej inteligencji w żaden sposób nie ma na celu zawyżenia dotychczasowych postępów ani zminimalizowania przyszłych prac. Jak podkreślamy w tej książce, ważne jest, aby zdawać sobie sprawę z naszych ograniczeń i szczerze mówiąc o naszych sukcesach. Na przykład wyniki programów są ograniczone, a w jakimkolwiek interesującym sensie można powiedzieć, że "uczą się". Nasze osiągnięcia w modelowaniu semantycznych złożoności języka naturalnego, takiego jak angielski, były również bardzo skromne. Nawet podstawowe kwestie, takie jak organizacja wiedzy lub pełne zarządzanie złożonością i poprawnością bardzo dużych programów komputerowych (takich jak duże bazy wiedzy) wymagają dalszych badań. Systemy oparte na wiedzy, mimo że osiągnęły sukces rynkowy w dziedzinie inżynierii, wciąż mają wiele ograniczeń w jakości i ogólności ich rozumowania. Należą do nich ich niezdolność do prowadzenia zdrowego rozumowania lub wykazywania wiedzy o podstawowej rzeczywistości fizycznej, takiej jak zmiany rzeczy w czasie. Ale musimy zachować rozsądną perspektywę. Łatwo przeoczyć osiągnięcia sztucznej inteligencji, gdy uczciwie stawia się czoła

pozostałej pracy. W następnej części przedstawiamy tę perspektywę poprzez przegląd kilku ważnych obszarów badań i rozwoju sztucznej inteligencji.

Przegląd obszarów zastosowań AI

Wracamy teraz do celu, jakim jest zdefiniowanie sztucznej inteligencji poprzez badanie ambicji i osiągnięć pracowników w terenie. Dwie najbardziej fundamentalne obawy badaczy AI to reprezentacja wiedzy i wyszukiwanie. Pierwszy z nich dotyczy problemu przechwytywania w języku, tj. takim, który jest odpowiedni do manipulacji komputerowej, pełnym zakresem wiedzy wymaganym do inteligentnego zachowania. Wprowadzamy rachunek predykatów jako język opisujący właściwości i relacje między obiektami w domenach problemowych, które wymagają uzasadnienia jakościowego, a nie obliczeń arytmetycznych dla swoich rozwiązań. Późniejsza sekcja omawi narzędzia opracowane przez sztuczną inteligencję do reprezentowania dwuznaczności i złożoności obszarów, takich jak rozsądne rozumowanie i rozumienie języka naturalnego. Wyszukiwanie to technika rozwiązywania problemów, która systematycznie bada przestrzeń stanów problemowych, tj. kolejne i alternatywne etapy procesu rozwiązywania problemów. Przykłady stanów problemowych mogą obejmować różne konfiguracje planszy w grze lub pośrednie kroki w procesie wnioskowania. Ta przestrzeń alternatywnych rozwiązań jest następnie przeszukiwana w celu znalezienia odpowiedzi. Newell i Simon (1976) twierdzili, że jest to podstawowa podstawa rozwiązywania problemów człowieka. Rzeczywiście, gdy szachista bada skutki różnych ruchów lub lekarz rozważa szereg alternatywnych diagnoz, szuka alternatyw. Implikacje tego modelu i technik jego wdrażania omówiono później. Podobnie jak większość nauk, sztuczna inteligencja jest podzielona na szereg subdyscyplin, które, choć dzielą podstawowe podejście do rozwiązywania problemów, zajmują się różnymi aplikacjami. W tej sekcji przedstawiamy kilka z tych głównych obszarów zastosowań i ich wkład w sztuczną inteligencję jako całość.

Granie w gry

Wiele wczesnych badań w zakresie wyszukiwania w przestrzeni państwowej przeprowadzono przy użyciu popularnych gier planszowych, takich jak warcaby, szachy i 15-tki. Oprócz ich nieodłącznego intelektualnego uroku, gry planszowe mają pewne właściwości, które czyniły z nich idealne przedmioty do badań. Większość gier jest rozgrywana przy użyciu ściśle określonego zestawu reguł: ułatwia to generowanie przestrzeni wyszukiwania i uwalnia badacza od wielu dwuznaczności i złożoności związanych z mniej ustrukturyzowanymi problemami. Konfiguracje planszowe używane w grach są łatwo reprezentowane na komputerze, nie wymagając żadnego złożonego formalizmu potrzebnego do uchwycenia semantycznych subtelności bardziej złożonych domen problemowych. Ponieważ gry są łatwe do grania, testowanie programu do gier nie stanowi obciążenia finansowego ani etycznego. Gry mogą generować bardzo duże przestrzenie wyszukiwania. Są one wystarczająco duże i złożone, aby wymagać zaawansowanych technik określania alternatywnych rozwiązań w przestrzeni problemowej. Techniki te nazywane są heurystyką i stanowią główny obszar badań nad AI. Heurystyka to przydatna, ale potencjalnie omylna strategia rozwiązywania problemów, na przykład sprawdzanie, czy urządzenie nie reaguje, zanim zostanie założone, że jest zepsute lub zamyka się, aby uchronić króla przed schwytaniem w szachach. Wiele z tego, co zwykle nazywamy inteligencją, wydaje się opierać na heurystyce stosowanej przez ludzi do rozwiązywania problemów. Ponieważ większość z nas ma pewne doświadczenie z tymi prostymi grami, można opracować i przetestować skuteczność naszej własnej heurystyki. Nie musimy szukać eksperta w jakiejś ezoterycznej dziedzinie, takiej jak medycyna czy matematyka i konsultować się z nią (szachy są oczywistym wyjątkiem od tej reguły). Z tych powodów gry zapewniają bogatą domenę do badania wyszukiwania heurystycznego. Programy do gier, pomimo swojej prostoty, oferują własne wyzwania, w tym przeciwnika, którego ruchów nie można deterministycznie przewidzieć. Ostatnie sukcesy w grach komputerowych obejmują mistrzostwa

świata w backgammon i szachach. Warto również zauważyć, że w 2007 r. Została zmapowana pełna przestrzeń państwowa gry w warcaby, dzięki czemu może być od pierwszego ruchu deterministyczna!

Zautomatyzowane wnioskowanie i dowodzenie twierdzeń

Możemy argumentować, że automatyczne sprawdzanie twierdzeń jest najstarszą gałęzią sztucznej inteligencji, której korzenie sięgają wstecz, poprzez teoretyków logiki Newella i Simona oraz ogólnego rozwiązywania problemów, poprzez wysiłki Russella i Whiteheada, by traktować całą matematykę jako czysto formalne wyprowadzenie twierdzeń z podstawowych aksjomatów , do jego początków w pismach Babbage'a i Leibniza. W każdym razie była to z pewnością jedna z najbardziej owocnych gałęzi pola. Badania dowodzące twierdzeń były odpowiedzialne za większość wczesnych prac nad sformalizowaniem algorytmów wyszukiwania i opracowaniem formalnych języków reprezentacyjnych, takich jak rachunek predykatów i logiczny język programowania Prolog. Większość zautomatyzowanego dowodzenia twierdzeń leży w rygorze i ogólności logiki. Ponieważ jest to system formalny, logika nadaje się do automatyzacji. Szeroką gamę problemów można zaatakować, reprezentując opis problemu i istotne informacje podstawowe jako logiczne aksjomaty, a traktując przypadki problemu jako twierdzenia, które należy udowodnić. Ten wgląd jest podstawą pracy w automatycznych dowodzeniach twierdzeń i matematycznych systemach wnioskowania. Niestety wczesne próby napisania twierdzeń dowodowych nie doprowadziły do opracowania systemu, który mógłby konsekwentnie rozwiązywać skomplikowane problemy. Wynikało to ze zdolności dowolnego rozsądnie złożonego systemu logicznego do generowania nieskończonej liczby możliwych do udowodnienia twierdzeń: bez potężnych technik (heurystyki) do prowadzenia ich poszukiwań, automatyczne dowody twierdzeń udowodniły dużą liczbę nieistotnych twierdzeń przed natknięciem się na poprawne. W odpowiedzi na tę nieefektywność wielu twierdzi, że czysto formalne, składniowe metody kierowania wyszukiwaniem z natury nie są w stanie poradzić sobie z tak ogromną przestrzenią i że jedyną alternatywą jest poleganie na nieformalnych, doraźnych strategiach, które ludzie wydają się stosować przy rozwiązywaniu problemów. Takie podejście stanowi podstawę rozwoju systemów eksperckich i okazało się owocne. Jednak odwołanie do rozumowania opartego na formalnej logice matematycznej jest zbyt silne, aby je zignorować. Wiele ważnych problemów, takich jak projektowanie i weryfikacja układów logicznych, weryfikacja poprawności programów komputerowych i kontrola złożonych systemów, wydaje się odpowiadać na takie podejście. Ponadto społeczność dowodząca twierdzeń odniosła sukces w opracowywaniu potężnych heurystyk rozwiązań, które opierają się wyłącznie na ocenie składniowej formy logicznego wyrażenia, a w rezultacie zmniejszają złożoność przestrzeni wyszukiwania bez uciekania się do technik ad hoc używane przez większość ludzi rozwiązujących problemy. Innym powodem ciągłego zainteresowania automatycznymi dowodami twierdzeń jest świadomość, że taki system nie musi być zdolny do samodzielnego rozwiązywania niezwykle złożonych problemów bez pomocy człowieka. Wiele współczesnych twierdzeń dowodowych funkcjonuje jako inteligentni asystenci, pozwalając ludziom wykonywać bardziej wymagające zadania polegające na rozkładaniu dużego problemu na podproblemy i opracowywaniu heurystyki w celu przeszukiwania przestrzeni możliwych dowodów. Dowódca twierdzeń wykonuje następnie prostsze, ale wciąż wymagające zadanie udowodnienia lematów, weryfikacji mniejszych przypuszczeń i uzupełnienia formalnych aspektów dowodu przedstawionego przez jego współpracownika

Systemy eksperckie

Jednym z głównych wniosków zdobytych we wczesnej pracy nad rozwiązywaniem problemów było znaczenie wiedzy specyficznej dla danej dziedziny. Na przykład lekarz nie jest skuteczny w diagnozowaniu choroby tylko dlatego, że posiada pewne wrodzone ogólne umiejętności rozwiązywania problemów; jest skuteczna, ponieważ dużo wie o medycynie. Podobnie geolog skutecznie odkrywa złoża minerałów, ponieważ jest w stanie zastosować znaczną część wiedzy

teoretycznej i empirycznej na temat geologii w danym problemie. Wiedza ekspercka to połączenie teoretycznego zrozumienia problemu i zbioru heurystycznych zasad rozwiązywania problemów, które doświadczenie okazało się skuteczne w tej dziedzinie. Systemy eksperckie są konstruowane poprzez uzyskanie tej wiedzy od ludzkiego eksperta i zakodowanie jej w formie, którą komputer może zastosować do podobnych problemów. To poleganie na wiedzy eksperta w dziedzinie zagadnień ludzkich w strategiach rozwiązywania problemów w systemie jest główną cechą systemów eksperckich. Chociaż napisane są niektóre programy, w których projektant jest również źródłem wiedzy o domenie, o wiele bardziej typowe jest obserwowanie, jak takie programy powstają w wyniku współpracy między ekspertem w dziedzinie, takim jak lekarz, chemik, geolog lub inżynier, i osobnym specjalistą od sztucznej inteligencji. Ekspert w dziedzinie zapewnia niezbędną wiedzę na temat dziedziny problemowej poprzez ogólne omówienie jej metod rozwiązywania problemów i wykazanie tych umiejętności na starannie dobranym zestawie przykładowych problemów. Specjalista AI lub inżynier wiedzy, jak często znani są projektanci systemów ekspertowych, jest odpowiedzialny za wdrożenie tej wiedzy w programie, który jest zarówno skuteczny, jak i pozornie inteligentny w swoim zachowaniu. Po napisaniu takiego programu konieczne jest udoskonalenie jego wiedzy specjalistycznej poprzez proces dawania przykładowych problemów do rozwiązania, pozwalając ekspertowi domeny skrytykować jego zachowanie oraz wprowadzając wszelkie wymagane zmiany lub modyfikacje wiedzy o programie. Proces ten powtarza się, aż program osiągnie pożądany poziom wydajności. Jednym z najwcześniejszych systemów do wykorzystania wiedzy specyficznej dla domeny w rozwiązywaniu problemów był DENDRAL, opracowany w Stanford pod koniec lat 60. XX wieku. DENDRAL został zaprojektowany w celu wnioskowania o strukturze cząsteczek organicznych na podstawie ich wzorów chemicznych i informacji spektrografu masowego o wiązaniach chemicznych obecnych w cząsteczkach. Ponieważ cząsteczki organiczne są zwykle bardzo duże, liczba możliwych struktur dla tych cząsteczek jest na ogół ogromna. DENDRAL rozwiązuje ten problem dużej przestrzeni poszukiwań dzięki zastosowaniu heurystycznej wiedzy ekspertów chemików do problemu wyjaśnienia struktury. Metody DENDRAL-a okazały się niezwykle skuteczne, rutynowo znajdując prawidłową strukturę spośród milionów możliwości już po kilku próbach. Podejście to okazało się tak skuteczne, że rozszerzenia potomków DENDRAL są obecnie stosowane w laboratoriach chemicznych i farmaceutycznych na całym świecie. Podczas gdy DENDRAL był jednym z pierwszych programów efektywnie wykorzystujących wiedzę specyficzną dla danej dziedziny w celu osiągnięcia wydajności eksperckiej, MYCIN ustanowił metodologię współczesnych systemów eksperckich. MYCIN wykorzystuje specjalistyczną wiedzę medyczną do diagnozowania i przepisywania leczenia rdzeniowego zapalenia opon mózgowych i bakteryjnych zakażeń krwi. MYCIN, opracowany w Stanford w połowie lat siedemdziesiątych, był jednym z pierwszych programów, które rozwiązały problemy rozumowania za pomocą niepewnych lub niepełnych informacji. MYCIN dostarczył jasne i logiczne wyjaśnienia swojego uzasadnienia, zastosował strukturę kontroli odpowiednią do konkretnej dziedziny problemu i zidentyfikował kryteria w celu wiarygodnej oceny jego wydajności. Wiele obecnie stosowanych technik opracowywania systemów eksperckich zostało po raz pierwszy opracowanych w ramach projektu MYCIN. Inne wczesne systemy eksperckie obejmują program PROSPECTOR do określania prawdopodobnej lokalizacji i rodzaju złóż rudy na podstawie informacji geologicznych o miejscu, program INTERNIST do przeprowadzania diagnozy w dziedzinie chorób wewnętrznych, doradca Dipmeter do interpretacji wyników wiercenia odwiertu naftowego i XCON do konfiguracji komputerów VAX. XCON został opracowany w 1981 r. I za jednym razem każdy VAX sprzedawany przez Digital Equipment Corporation był konfigurowany przez to oprogramowanie. Wiele innych systemów eksperckich rozwiązuje obecnie problemy w takich dziedzinach, jak medycyna, edukacja, biznes, projektowanie i nauka. Zobacz także bieżące postępowanie w sprawie innowacyjnych wniosków sztucznej inteligencji (IAAI). Warto zauważyć, że większość systemów eksperckich została napisana dla stosunkowo wyspecjalizowanych domen na poziomie eksperckim. Domeny te są ogólnie dobrze zbadane i mają jasno określone strategie

rozwiązywania problemów. Problemy, które zależą od bardziej luźno zdefiniowanego pojęcia "zdrowego rozsądku", są znacznie trudniejsze do rozwiązania za pomocą tych środków. Mimo obietnicy systemów eksperckich błędem byłoby przecenianie możliwości tej technologii. Obecne braki obejmują:

1. Trudności w zdobyciu "głębokiej" wiedzy w dziedzinie problemowej. Na przykład MYCIN nie ma żadnej prawdziwej wiedzy na temat fizjologii człowieka. Nie wie, co robi krew ani funkcja rdzenia kręgowego. Folklor głosi, że raz, wybierając lek do leczenia zapalenia opon mózgowych, MYCIN zapytał, czy pacjentka jest w ciąży, nawet jeśli powiedziano jej, że jest mężczyzną. Niezależnie od tego, czy faktycznie miało to miejsce, czy nie, ilustruje to potencjalne zawężenie wiedzy w systemach eksperckich.

2. Brak solidności i elastyczności. Jeśli ludzie mają problem z wystąpieniem problemu, którego nie mogą natychmiast rozwiązać, zazwyczaj mogą wrócić do analizy pierwszych zasad i opracować strategię ataku na problem. Systemy eksperckie na ogół nie mają tej zdolności.

3. Niemożność udzielenia głębokich wyjaśnień. Ponieważ systemy eksperckie nie mają głębokiej wiedzy o swoich domenach problemowych, ich wyjaśnienia są na ogół ograniczone do opisu kroków, które podjęli w celu znalezienia rozwiązania. Na przykład często nie potrafią powiedzieć "dlaczego" przyjęto określone podejście.

4. Trudności w weryfikacji. Chociaż poprawność każdego dużego systemu komputerowego jest trudna do udowodnienia, systemy eksperckie są szczególnie trudne do zweryfikowania. Jest to poważny problem, ponieważ technologia systemów ekspertowych jest stosowana w krytycznych zastosowaniach, takich jak kontrola ruchu lotniczego, operacje reaktorów jądrowych i systemy uzbrojenia.

5. Niewiele uczenia się z doświadczenia. Obecne systemy eksperckie są wytwarzane ręcznie; po ukończeniu systemu jego wydajność nie poprawi się bez dalszej uwagi ze strony programistów, co prowadzi do wątpliwości co do inteligencji takich systemów.

Pomimo tych ograniczeń systemy eksperckie sprawdziły się w wielu ważnych zastosowaniach. Aktualne aplikacje często można znaleźć w materiałach z konferencji Inovative Applications of Artificial Intelligence (IAAI).

Zrozumienie języka naturalnego i semantyka

Jednym z długofalowych celów sztucznej inteligencji jest tworzenie programów zdolnych do rozumienia i generowania języka ludzkiego. Umiejętność posługiwania się i rozumienia języka naturalnego wydaje się nie tylko podstawowym aspektem ludzkiej inteligencji, ale także jego udana automatyzacja miałaby niesamowity wpływ na użyteczność i efektywność samych komputerów. Wiele wysiłku włożono w pisanie programów, które rozumieją język naturalny. Chociaż programy te odniosły sukces w ograniczonych kontekstach, systemy, które potrafią używać języka naturalnego z elastycznością i ogólnością, które charakteryzują ludzką mowę, wykraczają poza obecne metodologie. Zrozumienie języka naturalnego wymaga znacznie więcej niż analizowania zdań w poszczególnych częściach mowy i wyszukiwania tych słów w słowniku. Prawdziwe zrozumienie zależy od obszernej wiedzy podstawowej na temat dziedziny dyskursu i idiomów używanych w tej dziedzinie, a także umiejętności zastosowania ogólnej wiedzy kontekstualnej w celu wyeliminowania pominięć i dwuznaczności, które są normalną częścią ludzkiej mowy. Rozważmy na przykład trudności w prowadzeniu rozmowy o baseballu z osobą, która rozumie angielski, ale nie wie nic o zasadach, graczach ani historii gry. Czy ta osoba mogłaby zrozumieć znaczenie zdania: "Gdy nikt nie znalazł się u góry dziewiątej, a drugi wybiegł na drugie miejsce, menedżer zwrócił się z ulgą od byka"? Nawet jeśli

wszystkie słowa w zdaniu mogą być zrozumiane indywidualnie, zdanie to byłoby bełkotem nawet dla najbardziej inteligentnego fana niebędącego fanem baseballa. Zadanie gromadzenia i organizowania tej wiedzy podstawowej w taki sposób, aby można ją było zastosować do rozumienia języka, stanowi główny problem w automatyzacji rozumienia języka naturalnego. W odpowiedzi na tę potrzebę naukowcy opracowali wiele technik konstruowania znaczenia semantycznego stosowanego w sztucznej inteligencji. Ze względu na ogrom wiedzy wymaganej do zrozumienia języka naturalnego większość pracy wykonywana jest w dobrze zrozumiałych, specjalistycznych obszarach problemowych. Jednym z najwcześniejszych programów wykorzystujących metodologię "mikroświata" była SHRDLU Winograda, system języka naturalnego, który mógł "rozmawiać" o prostej konfiguracji bloków o różnych kształtach i kolorach. SHRDLU może odpowiedzieć na pytania takie jak "jaki blok koloru znajduje się na niebieskim sześcianie?", A także zaplanować działania, takie jak "przenieś czerwoną piramidę na zielony klocek". Problemy tego rodzaju, polegające na opisie i manipulowaniu prostymi układami bloków, pojawiły się z zaskakującą częstotliwością we wczesnych badaniach nad AI i są znane jako problemy "świata bloków". Pomimo sukcesu SHRDLU w rozmowach na temat ułożenia bloków, jego metody nie uogólniły się z tej domeny. Techniki reprezentacyjne zastosowane w programie były zbyt proste, aby w przydatny sposób uchwycić semantyczną organizację bogatszych i bardziej złożonych domen. Znaczna część bieżącej pracy nad zrozumieniem języka naturalnego poświęcona jest znalezieniu formalizmów reprezentacyjnych, które są na tyle ogólne, że można je stosować w szerokim zakresie aplikacji, ale dobrze dostosowują się do specyficznej struktury danej dziedziny. W tym celu badanych jest wiele różnych technik (z których wiele to rozszerzenia lub modyfikacje sieci semantycznych) i wykorzystywane w rozwoju programów, które potrafią zrozumieć język naturalny w ograniczonych, ale interesujących dziedzinach wiedzy. Wreszcie w obecnych badaniach modele stochastyczne opisujące, w jaki sposób słowa i struktury językowe "występują" w użyciu, są stosowane do charakteryzowania zarówno składni, jak i semantyki. Jednak pełne komputerowe rozumienie języka pozostaje poza aktualnym stanem wiedzy.

Modelowanie wydajności człowieka

Chociaż większość powyższej dyskusji wykorzystuje ludzką inteligencję jako punkt odniesienia przy rozważaniu sztucznej inteligencji, nie wynika z tego, że programy powinny wzorować się na organizacji ludzkiego umysłu. Rzeczywiście, wiele programów AI zaprojektowano w celu rozwiązania jakiegoś użytecznego problemu bez względu na ich podobieństwo do ludzkiej architektury mentalnej. Nawet systemy eksperckie, choć czerpią większość swojej wiedzy od ludzkich ekspertów, tak naprawdę nie próbują symulować ludzkich wewnętrznych procesów rozwiązywania problemów psychicznych. Jeśli wydajność jest jedynym kryterium oceny systemu, nie ma powodu, aby próbować symulować ludzkie metody rozwiązywania problemów; w rzeczywistości programy wykorzystujące nieludzkie podejście do rozwiązywania problemów (szachy) są często bardziej skuteczne niż ich ludzcy odpowiednicy. Mimo to projektowanie systemów, które wyraźnie modelują aspekty ludzkiej wydajności, jest płodnym obszarem badań zarówno w AI, jak i psychologii. Modelowanie wydajności człowieka, oprócz dostarczania sztucznej inteligencji dużej części swojej podstawowej metodologii, okazało się potężnym narzędziem do formułowania i testowania teorii ludzkiego poznania. Metodologie rozwiązywania problemów opracowane przez informatyków dały psychologom nową metaforę badania ludzkiego umysłu. Zamiast wyrzucać teorie poznania w niejasnym języku używanym we wczesnych badaniach lub porzucać problem całkowitego opisu wewnętrznego funkcjonowania ludzkiego umysłu (jak sugerują behawioryści), wielu psychologów przyjęło język i teorię informatyki w celu sformułowania modeli ludzkiej inteligencji. Techniki te nie tylko zapewniają nowe słownictwo opisujące ludzką inteligencję, ale także komputerowe implementacje tych teorii dają psychologom możliwość empirycznego przetestowania, krytyki i udoskonalenia swoich pomysłów

Planowanie i robotyka

Badania w zakresie planowania rozpoczęły się jako próba zaprojektowania robotów, które mogłyby wykonywać swoje zadania z pewnym stopniem elastyczności i szybkości reagowania na świat zewnętrzny. W skrócie, planowanie zakłada robota zdolnego do wykonywania pewnych działań atomowych. Próbuje znaleźć sekwencję czynności, które pozwolą wykonać jakieś zadanie na wyższym poziomie, takie jak przejście przez pomieszczenie wypełnione przeszkodami. Planowanie jest trudnym problemem z wielu powodów, między innymi z wielkości przestrzeni możliwych sekwencji ruchów. Nawet wyjątkowo prosty robot jest w stanie wygenerować ogromną liczbę potencjalnych sekwencji ruchów. Wyobraź sobie na przykład robota, który może poruszać się do przodu, do tyłu, w prawo lub w lewo, i zastanów się, ile różnych sposobów robot może poruszać się po pokoju. Załóżmy również, że w pomieszczeniu znajdują się przeszkody i że robot musi wybrać ścieżkę, która porusza się wokół nich w efektywny sposób. Napisanie programu, który potrafi znaleźć najlepszą ścieżkę w tych okolicznościach, bez przytłoczenia ogromną liczbą możliwości, wymaga wyrafinowanych technik reprezentowania wiedzy przestrzennej i kontrolowania wyszukiwania w możliwych środowiskach. Jedną z metod stosowanych przez ludzi w planowaniu jest hierarchiczny rozkład problemów. Jeśli planujesz podróż z Albuquerque do Londynu, zazwyczaj będziesz traktować problemy związane z zorganizowaniem lotu, dotarciem na lotnisko, nawiązaniem połączeń lotniczych i znalezieniem transportu naziemnego w Londynie osobno, mimo że wszystkie są częścią większego ogólnego planu . Każdy z nich można dalej rozłożyć na mniejsze podproblemy, takie jak znalezienie mapy miasta, negocjowanie systemu metra i znalezienie porządnego pubu. Nie tylko to podejście skutecznie ogranicza rozmiar przestrzeni, którą należy przeszukiwać, ale obsługuje także zapisywanie często używanych podplanów do przyszłego wykorzystania. Podczas gdy ludzie planują bez wysiłku, stworzenie programu komputerowego, który mógłby zrobić to samo, jest trudnym wyzwaniem. Pozornie proste zadanie, takie jak rozbicie problemu na niezależne podproblemy w rzeczywistości wymagają zaawansowanej heurystyki i rozległej wiedzy na temat dziedziny planowania. Równie trudnym problemem jest ustalenie, które podplany należy zapisać i jak można je uogólnić do wykorzystania w przyszłości. Robota, który ślepo wykonuje sekwencję działań bez reagowania na zmiany w swoim otoczeniu lub będąc w stanie wykryć i poprawić błędy we własnym planie, trudno uznać za inteligentnego. Robot może nie mieć odpowiednich czujników do zlokalizowania wszystkich przeszkód na drodze rzutowanej ścieżki. Taki robot musi zacząć poruszać się po pokoju w oparciu o to, co "zauważył" i korygować ścieżkę, gdy wykryte zostaną inne przeszkody. Poważnym problemem planowania jest uporządkowanie planów w sposób umożliwiający reagowanie na zmieniające się warunki środowiskowe. Wreszcie robotyka była jednym z obszarów badawczych w AI, który dostarczył wielu spostrzeżeń wspierających zorientowane na agenta rozwiązywanie problemów. Sfrustrowani zarówno złożonością utrzymania dużej przestrzeni reprezentacyjnej, jak i projektowaniem algorytmów wyszukiwania dla tradycyjnego planowania, badacze, w tym Agre i Chapman, Brooks, Thrun i inni ponownie sformułował problem pod względem interakcji wielu półautonomicznych czynników. Każdy agent był odpowiedzialny za swoją część zadania problemowego i dzięki ich koordynacji pojawiłoby się większe rozwiązanie. Badania w zakresie planowania wykraczają obecnie daleko poza dziedziny robotyki i obejmują koordynację dowolnego złożonego zestawu zadań i celów. Nowoczesne planery są stosowane do czynników, a także do sterowania akceleratorami wiązki cząstek

Języki i środowiska dla AI

Niektóre z najważniejszych produktów ubocznych badań nad sztuczną inteligencją to postępy w językach programowania i środowiskach programistycznych. Z wielu powodów, w tym wielkości wielu programów aplikacyjnych AI, znaczenia metodologii prototypowania, tendencji algorytmów wyszukiwania do generowania ogromnych przestrzeni oraz trudności w przewidywaniu zachowania

programów sterowanych heurystycznie, programiści AI zostali zmuszeni do opracowania potężnego zestawu metodologii programowania. Środowiska programowania obejmują techniki konstruowania wiedzy, takie jak programowanie obiektowe. Języki wysokiego poziomu, takie jak Lisp i Prolog, które obsługują programowanie modułowe, pomagają zarządzać wielkością i złożonością programu. Pakiety śledzenia pozwalają programiście zrekonstruować wykonanie złożonego algorytmu i umożliwić rozwikłanie złożoności wyszukiwania heurystycznego. Bez takich narzędzi i technik wątpliwe byłoby zbudowanie wielu znaczących systemów AI. Wiele z tych technik jest obecnie standardowymi narzędziami do inżynierii oprogramowania i ma niewielki związek z rdzeniem teorii sztucznej inteligencji. Inne, takie jak programowanie obiektowe, mają istotne znaczenie teoretyczne i praktyczne. Wreszcie, wiele algorytmów AI jest teraz wbudowanych w bardziej tradycyjne języki obliczeniowe, takie jak C ++ i Java. Języki opracowane do programowania sztucznej inteligencji są ściśle związane z teoretyczną strukturą dziedziny. Zbudowaliśmy wiele struktur reprezentacyjnych przedstawionych w tej książce w Prolog, Lisp i Java i udostępniamy je w Luger i Stubblefield oraz w Internecie.

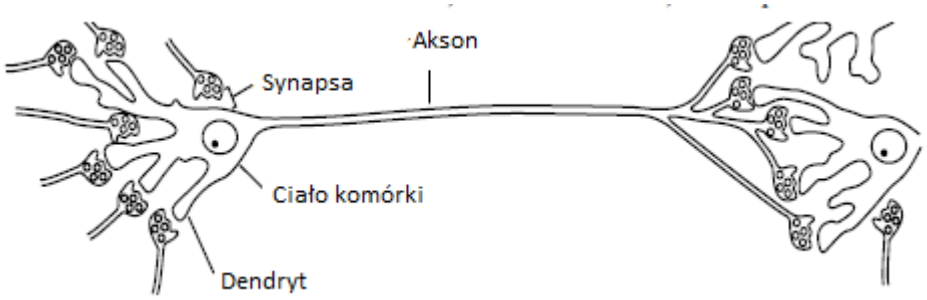
Nauczanie maszynowe

Uczenie się pozostaje wyzwaniem dla AI. Znaczenie uczenia się jest jednak niepodważalne, zwłaszcza że umiejętność ta jest jednym z najważniejszych elementów inteligentnego zachowania. System ekspercki może wykonywać rozległe i kosztowne obliczenia w celu rozwiązania problemu. Jednak w przeciwieństwie do człowieka, gdy drugi raz napotyka ten sam lub podobny problem, zwykle nie pamięta rozwiązania. Ponownie wykonuje tę samą sekwencję obliczeń. Dotyczy to drugiego, trzeciego, czwartego i za każdym razem, gdy rozwiązuje ten problem, nie jest to zachowanie inteligentnego rozwiązania problemu. Oczywistym rozwiązaniem tego problemu jest samodzielne uczenie się programów na podstawie doświadczenia, analogii, przykładów, "mówienie", co robić, nagradzanie lub karanie w zależności od wyników. Chociaż nauka jest trudnym obszarem, istnieje kilka programów, które sugerują, że nie jest to niemożliwe. Jednym z pierwszych programów jest AM, zautomatyzowany matematyk, zaprojektowany w celu odkrywania praw matematycznych. Początkowo biorąc pod uwagę pojęcia i aksjomaty teorii mnogości, AM był w stanie wywołać tak ważne pojęcia matematyczne, jak liczność, arytmetyka liczb całkowitych i wiele wyników teorii liczb. AM wymyślił nowe twierdzenia, modyfikując swoją bieżącą bazę wiedzy i wykorzystał heurystykę, aby realizować "najlepsze" z wielu możliwych alternatywnych twierdzeń. Cotton zaprojektował program, który automatycznie wymyśla "interesujące" sekwencje liczb całkowitych. Wczesne wpływowe prace obejmują również badania Winstona nad indukcją koncepcji konstrukcyjnych, takich jak "łuk" z zestawu przykładów w świecie bloków. Algorytm ID3 okazał się skuteczny w nauce ogólnych wzorców na podstawie przykładów. Meta-DENDRAL poznaje zasady interpretacji danych spektrograficznych mas w chemii organicznej na podstawie przykładów danych o związkach o znanej strukturze. Teiresias, inteligentny "front" dla systemów eksperckich, przekształca porady wysokiego szczebla w nowe zasady dotyczące swojej bazy wiedzy. Hacker opracowuje plany wykonywania manipulacji światem bloków poprzez iteracyjny proces opracowywania planu, testowania go i korygowania wszelkich wad wykrytych w planie kandydackim. Praca nad uczeniem się opartym na wyjaśnieniach wykazała skuteczność wcześniejszej wiedzy w uczeniu się. Istnieje również wiele ważnych biologicznych i socjologicznych modeli uczenia się; przeglądamy je w nauczaniu łączącym i kształceniu wschodzącym. Sukces programów uczenia maszynowego sugeruje istnienie zestawu ogólnych zasad uczenia się, które umożliwią budowę programów z możliwością uczenia się w realistycznych dziedzinach.

Alternatywne reprezentacje: sieci neuronowe i algorytmy genetyczne

Większość technik przedstawionych tutaj wykorzystuje jawnie reprezentowaną wiedzę i starannie zaprojektowane algorytmy wyszukiwania w celu wdrożenia inteligencji. Zupełnie inne podejście ma na

celu budowę inteligentnych programów z wykorzystaniem modeli, które równolegle budują neurony w ludzkim mózgu lub ewoluujące wzorce występujące w algorytmach genetycznych i sztucznym życiu. Prosty schemat neuronu



składa się z ciała komórkowego, które ma wiele rozgałęzionych występów, zwanych dendrytami, oraz pojedynczą gałąź zwaną aksonem. Dendryty odbierają sygnały z innych neuronów. Kiedy te połączone impulsy przekroczą pewien próg, neuron strzela i impuls lub kolec przepływa przez akson. Gałęzie na końcu aksonu tworzą synapsy z dendrytami innych neuronów. Synapsa jest punktem kontaktu między neuronami; synapsy mogą być pobudzające lub hamujące, albo zwiększając sumę sygnałów docierających do neuronu, albo odejmując od tej sumy. Ten opis neuronu jest zbyt prosty, ale oddaje te cechy, które są istotne w neuronowych modelach obliczeniowych. W szczególności każda jednostka obliczeniowa oblicza pewną funkcję swoich danych wejściowych i przekazuje wynik do połączonych jednostek w sieci: ostateczne wyniki są wytwarzane przez równoległe i rozproszone przetwarzanie tej sieci połączeń neuronowych i wag progowych. Architektury neuronowe są atrakcyjnymi mechanizmami wdrażania inteligencji z wielu powodów. Tradycyjne programy AI mogą być kruche i nadmiernie wrażliwe na hałas. Ludzka inteligencja jest znacznie bardziej elastyczna i dobrze interpretuje hałaśliwe informacje, takie jak twarz w zaciemnionym pokoju lub rozmowa na hałaśliwym przyjęciu. Architektury neuronowe, ponieważ przechwytują wiedzę w dużej liczbie drobnoziarnistych jednostek rozmieszczonych wokół sieci, wydają się mieć większy potencjał częściowego dopasowania hałaśliwych i niepełnych danych. Dzięki algorytmom genetycznym i sztucznemu życiu ewoluujemy nowe rozwiązania problemów z elementów poprzednich rozwiązań. Operatory genetyczne, takie jak krzyżowanie i mutacja, podobnie jak ich genetyczne odpowiedniki w świecie przyrody, pracują nad stworzeniem dla każdego nowego pokolenia coraz lepszych potencjalnych rozwiązań problemów. Sztuczne życie wytwarza swoje nowe pokolenie w funkcji "jakości" swoich sąsiadów w poprzednich pokoleniach. Zarówno architektury neuronowe, jak i algorytmy genetyczne zapewniają naturalny model równoległości, ponieważ każdy neuron lub segment rozwiązania jest niezależną jednostką. Hillis skomentował fakt, że ludzie stają się szybsi przy zadaniu, ponieważ zdobywają więcej wiedzy, podczas gdy komputery mają tendencję do spowalniania. To spowolnienie wynika z kosztu sekwencyjnego przeszukiwania bazy wiedzy; masowo równoległa architektura, taka jak ludzki mózg, nie cierpiałaby na ten problem. Wreszcie, coś jest wewnętrznie atrakcyjne w podejściu do problemów inteligencji z neuronowego lub genetycznego punktu widzenia. W końcu ewoluowany mózg osiąga inteligencję i robi to za pomocą architektury neuronowej.

Sztuczna inteligencja – Podsumowanie

Podjęliśmy próbę zdefiniowania sztucznej inteligencji poprzez omówienie jej głównych obszarów badań i zastosowań. Ta ankieta ujawnia młody i obiecujący kierunek badań, którego głównym celem jest znalezienie skutecznego sposobu zrozumienia i zastosowania inteligentnego rozwiązywania problemów, planowania i umiejętności komunikacyjnych w szerokim zakresie problemów praktycznych. Pomimo różnorodności problemów poruszanych w badaniach nad sztuczną inteligencją,

pojawia się szereg ważnych cech, które wydają się wspólne dla wszystkich działów w tej dziedzinie; obejmują one:

1. Wykorzystanie komputerów do rozumowania, rozpoznawania wzorców, uczenia się lub innych form wnioskowania.

2. Koncentracja na problemach, które nie reagują na rozwiązania algorytmiczne. To leży u podstaw polegania na wyszukiwaniu heurystycznym jako technice rozwiązywania problemów AI.

3. Problem związany z rozwiązywaniem problemów przy użyciu niedokładnych, brakujących lub źle zdefiniowanych informacji oraz stosowaniem formalnych form reprezentacji, które umożliwiają programiście zrekompensowanie tych problemów.

4. Rozumowanie znaczących cech jakościowych sytuacji.

5. Próba radzenia sobie z zagadnieniami o znaczeniu semantycznym oraz formą składniową.

6. Odpowiedzi, które nie są ani dokładne, ani optymalne, ale są w pewnym sensie "wystarczające". Wynika to z zasadniczego polegania na heurystycznych metodach rozwiązywania problemów w sytuacjach, w których optymalne lub dokładne wyniki są albo zbyt drogie, albo niemożliwe.

7. Wykorzystanie dużej ilości wiedzy specyficznej dla domeny w rozwiązywaniu problemów. To jest podstawa systemów ekspertowych.

8. Wykorzystanie wiedzy na poziomie meta do bardziej zaawansowanej kontroli strategii rozwiązywania problemów. Mimo że jest to bardzo trudny problem, rozwiązany w stosunkowo niewielu obecnych systemach, staje się on niezbędnym obszarem badań.

Mamy nadzieję, że wprowadzenie to w pewnym stopniu wyczuwa ogólną strukturę i znaczenie dziedziny sztucznej inteligencji. Mamy również nadzieję, że krótkie dyskusje na temat takich kwestii technicznych, jak wyszukiwanie i reprezentacja, nie były zbyt tajemnicze i niejasne; zostały one opracowane z odpowiednią szczegółowością w pozostałej części książki, ale zostały tu zawarte, aby wykazać ich znaczenie w ogólnej organizacji dziedziny. Jak wspomnieliśmy w dyskusji na temat rozwiązywania problemów zorientowanych na agentów, obiekty nabierają znaczenia poprzez swoje relacje z innymi obiektami. Dotyczy to w równym stopniu faktów, teorii i technik, które stanowią dziedzinę badań naukowych. Naszym celem było zrozumienie tych wzajemnych powiązań, aby po przedstawieniu osobnych technicznych zagadnień sztucznej inteligencji znalazły swoje miejsce w rozwijającym się zrozumieniu ogólnej istoty i kierunków tej dziedziny. W tym procesie kierujemy się spostrzeżeniem poczynionym przez Gregory’ego Batesona, psychologa i teoretyka systemów: Przełam wzór, który łączy elementy uczenia się, a ty koniecznie zniszcz całą jakość.

Sztuczna inteligencja : reprezentacja i wyszukiwanie

(II / XVI)

ĆWICZENIA

1. Korzystając z tabel prawdy, udowodnij tożsamość w sekcji "Semantyka rachunku zdań".

2. Nowy operator ⊕ lub exlusive-or lub może zostać zdefiniowany w poniższej tabeli prawdy:

P	Q	P ⊕ Q
T	T	F
T	F	T
F	T	T
F	F	F

Utwórz wyrażenie rachunku zdań, używając tylko ∧, ∨, i ¬, które jest równoważne P ⊕ Q. Udowodnij ich równoważność za pomocą tabel prawdy

3. Operator logiczny "↔" jest odczytywany "wtedy i tylko wtedy, gdy". P↔Q jest zdefiniowany jako równoważny (P→Q) ∧ (Q→P). Na podstawie tej definicji pokaż, że P↔Q jest logicznie równoważne (P ∨ Q)→ (P ∧ Q):

a. Za pomocą tabel prawdy.

4. Udowodnij, że implikacja jest przechodnia w rachunku zdań, to znaczy, że ((P → Q) ∧ (Q → R)) → (P → R).

5. a. Udowodnij, że modus ponens jest poprawny dla rachunku zdań. Wskazówka: użyj tabel prawdy, aby wyliczyć wszystkie możliwe interpretacje.

b. Uprowadzenie to zasada wnioskowania, która wnioskuje P z P → Q i Q. Pokaż, że uprowadzenie nie jest dźwiękiem

c. Pokaż modus tollens ((P→Q) ∧ ¬ Q)→¬ P to dźwięk.

6. Spróbuj ujednolicić następujące pary wyrażeń. Albo pokaż ich najbardziej ogólne unifikatory lub wyjaśnij, dlaczego nie będą jednoczyć.

a. p(X,Y) i p(a,Z)

b. p(X,X) i p(a,b)

c. przodek(X Y) i przodek(rachunek, ojciec (rachunek))

d. przodek(X,ojciec(X)) i przodek (David,George)

e. q(X) i ¬ q (a)

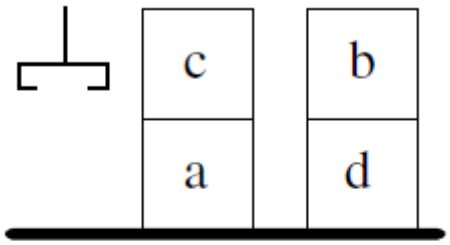
7. a. Skomponuj zestawy podstawień {a/X,Y/Z} i {X/W, b/Y}.

b. Udowodnij że skład zestawów podstawień jest asocjacyjny.

c. Skonstruuj przykład, aby pokazać, że kompozycja nie jest przemienna.

8. Zaimplementuj algorytm ujednolicenia opisany w części Unifikacja w wybranym języku komputerowym.

9. Podaj dwie alternatywne interpretacje opisu świata bloków na rysunku



on(c,a)
on(b,d)
ontable(a)
ontable(d)
clear(b)
clear(c)
hand_empty

10. Jane Doe ma cztery osoby na utrzymaniu, o stałym dochodzie w wysokości 30 000 USD i 15 000 USD na swoim koncie oszczędnościowym. Dodaj odpowiednie predykaty opisujące jej sytuację do ogólnego doradcy inwestycyjnego z przykładu "Aplikacja: Doradca finansowy oparty na logice" i wykonaj ujednolicenia i wnioski niezbędne do ustalenia jej sugerowanej inwestycji.

11. Napisz zestaw logicznych predykatów, które przeprowadzą prostą diagnostykę samochodową (np. Jeśli silnik się nie przekręci, a lampki nie zaświecą, oznacza to, że akumulator jest zły). Nie staraj się być zbyt skomplikowany, ale przykryj przypadki złej baterii, braku gazu, złych świec zapłonowych i złego silnika rozrusznika.

12. Poniższa historia pochodzi od N. Wirtha (1976) Algorytmy + struktury danych = programy. Poślubiłem wdowę (nazwijmy ją W), która ma dorosłą córkę (nazwij ją D). Mój ojciec (F), który często nas odwiedzał, zakochał się w mojej pasierbicy i poślubił ją. Dlatego mój ojciec został moim zięciem, a macocha stała się moją matką. Kilka miesięcy później moja żona urodziła syna (S1), który został szwagrem mojego ojca, a także wujkiem. Żona mojego ojca, czyli mojej przyrodniej córki, również miała syna (S2). Za pomocą rachunku predykatu utwórz zestaw wyrażeń, które reprezentują sytuację w powyższej historii. Dodaj wyrażenia określające podstawowe relacje rodzinne, takie jak definicja teścia i użyj modus ponens w tym systemie, aby udowodnić wniosek, że "jestem moim dziadkiem".

Sztuczna inteligencja : reprezentacja i wyszukiwanie(II)

Sztuczna inteligencja jako reprezentacja i poszukiwanie

Propozycja projektu badawczego Darthmouth Summer dotyczącego sztucznej inteligencji

„Proponujemy, aby latem 1956 r. W Dartmouth College w Hanover w stanie New Hampshire przeprowadzono dwumiesięczne, 10-osobowe badanie sztucznej inteligencji. Badanie ma przebiegać na podstawie przypuszczenia, że każdy aspekt uczenia się lub jakakolwiek inna cecha inteligencji może być w zasadzie tak precyzyjnie opisana, że można stworzyć maszynę do jej symulacji. Zostanie podjęta próba znalezienia sposobu zmuszenia maszyn do używania języka, tworzenia abstrakcji i pojęć, rozwiązywania rodzajów problemów zarezerwowanych obecnie dla ludzi i poprawy się. Uważamy, że można dokonać znaczącego postępu w zakresie jednego lub więcej z tych problemów, jeśli starannie wybrana grupa naukowców będzie pracować nad tym razem przez lato."

J. MCCARTHY, Dartmouth College , M. L. MINSKY, Harvard University , N. ROCHESTER, I.B.M. Corporation , C.E. SHANNON, Bell Telephone Laboratories , 31 sierpnia 1955 r

Wprowadzenie do reprezentacji i wyszukiwania

Z punktu widzenia inżynierii opis sztucznej inteligencji przedstawiony wcześniej można podsumować jako badanie reprezentacji i wyszukiwania, poprzez które inteligentna aktywność może być realizowana na mechanicznym urządzeniu. Ta perspektywa zdominowała początki i rozwój sztucznej inteligencji. Pierwsze nowoczesne warsztaty / konferencje dla praktyków AI odbyły się w Dartmouth College latem 1956 r. Propozycję tych warsztatów przedstawiono jako wstępny cytat z Części II. Warsztaty, na których wybrano samą nazwę sztuczna inteligencja, zgromadziły wielu ówczesnych badaczy skupionych na integracji obliczeń i inteligencji. W tym czasie napisano także kilka programów komputerowych odzwierciedlających te wczesne pomysły. Główne tematy do dyskusji na tej konferencji, skrócone tutaj od oryginalnej propozycji warsztatów, to:

1. Komputery automatyczne : Jeśli maszyna może wykonać zadanie, można zaprogramować automatyczny kalkulator do symulacji maszyny.

2. Jak można zaprogramować komputer do używania języka : Można spekulować, że duża część ludzkiej myśli polega na manipulowaniu słowami zgodnie z regułami rozumowania i przypuszczeniami.

3. Sieci neuronowe : Jak można zestaw (hipotetycznych) neuronów ułożyć koncepcje?

4. Teoria wielkości obliczenia : Jeśli otrzymamy dobrze zdefiniowany problem (taki, dla którego można mechanicznie sprawdzić, czy proponowana odpowiedź jest poprawną odpowiedzią), jednym ze sposobów rozwiązania tego problemu jest wypróbowanie wszystkich możliwych odpowiedzi w kolejności. Ta metoda jest nieefektywna i aby ją wykluczyć, trzeba mieć pewne kryterium wydajności obliczeń.

5. Samodoskonalenie (uczenie maszynowe) : Prawdopodobnie prawdziwie inteligentna maszyna będzie wykonywać czynności, które najlepiej można opisać jako samodoskonalenie.

6. Abstrakcje : Wiele rodzajów "abstrakcji" można wyraźnie zdefiniować, a kilka innych mniej wyraźnie. Bezpośrednia próba ich sklasyfikowania i opisania maszynowych metod formowania abstrakcji na podstawie danych sensorycznych i innych wydaje się opłacalna.

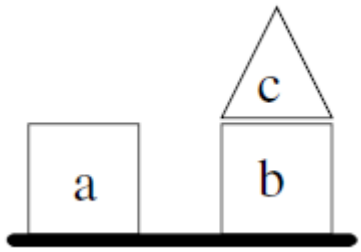
7. Losowość i kreatywność : Dość atrakcyjną, a jednak wyraźnie niekompletną hipotezą jest to, że różnica między kreatywnym myśleniem a niewyobrażalnym, kompetentnym myśleniem polega na zastrzyku pewnej przypadkowości.

Warto zauważyć, że tematy zaproponowane na pierwszą konferencję na temat sztucznej inteligencji obejmują wiele zagadnień, takich jak teoria złożoności, metodologia abstrakcji, projektowanie języków i uczenie maszynowe, które stanowią przedmiot współczesnej informatyki. W rzeczywistości wiele charakterystycznych cech informatyki, jaką znamy dzisiaj, ma swoje korzenie w sztucznej inteligencji. Sztuczna inteligencja również miała swoje historyczne i polityczne zmagania, a kilka z tych wczesnych tematów zaproponowanych do badań, takich jak "sieci neuronowe" i "losowość i kreatywność" zostały wprowadzone w tryb tła na dziesięciolecia. Mniej więcej w tym czasie pojawiło się nowe, potężne narzędzie obliczeniowe, język Lisp, zbudowane pod kierunkiem Johna McCarthy'ego, jednego z pierwszych twórców Warsztatu Dartmouth. Lisp zajął się kilkoma tematami Warsztatu, wspierając zdolność do tworzenia relacji, którymi mogłyby manipulować inne struktury języka. Lisp dał sztucznej inteligencji zarówno bardzo wyrazisty język, bogaty w abstrakcję, jak i medium do interpretacji tych wyrażeń. Dostępność języka programowania Lisp ukształtowała znaczną część wczesnego rozwoju sztucznej inteligencji, w szczególności wykorzystanie rachunku predykatów jako medium reprezentacyjnego, a także poszukiwanie w celu zbadania skuteczności różnych logicznych alternatyw, które obecnie nazywamy wyszukiwaniem grafowym. Prolog, stworzony pod koniec lat siedemdziesiątych, oferował AI podobne potężne narzędzie obliczeniowe.

Systemy reprezentacyjne

Funkcją każdego schematu reprezentacji jest uchwycenie - często nazywane streszczeniem - kluczowych cech domeny problemowej i udostępnienie tych informacji procedurze rozwiązywania problemów. Abstrakcja jest niezbędnym narzędziem do zarządzania złożonością, a także ważnym czynnikiem zapewniającym, że uzyskane programy są wydajne obliczeniowo. Ekspresyjność (wynik abstrakcyjnych cech) i wydajność (złożoność obliczeniowa algorytmów zastosowanych w abstrakcyjnych cechach) są głównymi wymiarami do oceny języków reprezentacji wiedzy. Czasami ekspresja musi zostać poświęcona, aby poprawić wydajność algorytmu. Należy tego dokonać bez ograniczania zdolności reprezentacji do przechwytywania niezbędnej wiedzy na temat rozwiązywania problemów. Optymalizacja kompromisu między wydajnością a ekspresją jest głównym zadaniem projektantów inteligentnych programów. Języki reprezentacji wiedzy mogą być również narzędziami pomagającymi ludziom w rozwiązywaniu problemów. Jako taka reprezentacja powinna zapewniać naturalne ramy dla wyrażania wiedzy na temat rozwiązywania problemów; powinna udostępnić tę wiedzę komputerowi i pomóc programiście w organizacji. Komputerowa reprezentacja liczb zmiennoprzecinkowych ilustruje te kompromisy. Mówiąc ściślej, liczby rzeczywiste wymagają pełnego opisu nieskończonego ciągu cyfr; nie można tego zrobić na urządzeniu skończonym, takim jak komputer. Jedną z odpowiedzi na ten dylemat jest przedstawienie liczby w dwóch częściach: cyfr znaczących i położenia w tych cyfrach miejsca dziesiętnego. Chociaż nie można faktycznie zapisać liczby rzeczywistej w komputerze, możliwe jest utworzenie reprezentacji, która będzie działać poprawnie w większości praktycznych zastosowań. Reprezentacja zmiennoprzecinkowa poświęca zatem pełną moc ekspresji, aby reprezentacja była wydajna, w tym przypadku, aby była możliwa. Ta reprezentacja obsługuje również algorytmy arytmetyki wielokrotnej precyzji, zapewniając efektywnie nieskończoną precyzję poprzez ograniczenie błędu zaokrąglenia do dowolnej z góry określonej tolerancji. Gwarantuje również dobrze zachowane błędy zaokrąglania. Podobnie jak wszystkie reprezentacje, jest to tylko abstrakcja, wzór symbolu, który określa pożądany byt, a nie sam byt. Tablica to kolejna reprezentacja powszechna w informatyce. W przypadku wielu problemów jest bardziej naturalny i wydajny niż architektura pamięci zaimplementowana w sprzęcie komputerowym. Ten wzrost naturalności i

wydajności pociąga za sobą kompromisy w ekspresji. Scena wizualna składa się z kilku punktów obrazu. Każdy punkt obrazu lub piksel ma zarówno lokalizację, jak i wartość liczbową reprezentującą jego intensywność lub poziom szarości. Naturalne jest zatem zebranie całej sceny w dwuwymiarową tablicę, w której adres wiersza i kolumny podaje lokalizację piksela (współrzędne X i Y), a zawartość elementu tablicy jest w tym miejscu poziomem szarości. Algorytmy są zaprojektowane do wykonywania takich operacji, jak szukanie izolowanych punktów w celu usunięcia szumów z obrazu, znajdowanie poziomów progowych dla rozpoznawania obiektów i krawędzi, sumowanie sąsiadujących elementów w celu określenia rozmiaru lub gęstości oraz na różne inne sposoby przekształcanie danych punktów obrazu. Wdrażane ich algorytmy są proste, biorąc pod uwagę na przykład reprezentację tablic i język FORTRAN. To zadanie byłoby dość kłopotliwe przy użyciu innych reprezentacji, takich jak rachunek predykatów, rekordy lub kod asemblacji, ponieważ nie mają one naturalnego dopasowania do reprezentowanego materiału. Reprezentując obraz jako układ pikseli, poświęcamy dokładność rozdzielczości (porównaj zdjęcie w gazecie z oryginalnym drukiem tego samego obrazu). Ponadto tablice pikseli nie mogą wyrazić głębszej semantycznej organizacji obrazu. Na przykład macierz pikseli nie może reprezentować organizacji chromosomów w jądrze pojedynczej komórki, ich funkcji genetycznej ani roli metafazy w podziale komórek. Wiedza ta jest łatwiejsza do zdobycia przy użyciu takich reprezentacji, jak rachunek predykatów lub sieci semantyczne. Podsumowując, schemat reprezentatywny powinien być odpowiedni do wyrażenia wszystkich niezbędnych informacji, wspierania wydajnego wykonania wynikowego kodu i zapewnienia naturalnego schematu wyrażania wymaganej wiedzy. Zasadniczo problemy, które AI próbuje rozwiązać, nie nadają się do reprezentacji oferowanych przez bardziej tradycyjne formalizmy, takie jak tablice. Sztuczna inteligencja dotyczy raczej jakościowego niż ilościowego rozwiązywania problemów, rozumowania zamiast obliczeń numerycznych oraz organizowania dużej i zróżnicowanej ilości wiedzy zamiast wdrażania jednego, dobrze zdefiniowanego algorytmu. Na przykład rozważmy rysunek, układ bloków na stole.



Załóżmy, że chcemy uchwycić właściwości i relacje wymagane do sterowania ramieniem robota. Musimy ustalić, które bloki są ułożone w stos na innych blokach, a które mają przezroczyste wierzchołki, aby można je było podnieść. Rachunek predykatów oferuje medium do przechwytywania tych opisowych informacji. Pierwsze słowo każdego wyrażenia (on, ontable itp.) Jest predykatem oznaczającym pewną właściwość lub związek między jego argumentami (występującymi w nawiasach). Argumenty to symbole oznaczające obiekty (bloki) w domenie. Zbiór klauzul logicznych opisuje ważne właściwości i relacje tego świata bloków:

clear(c)

clear(a)

ontable(a)

ontable(b)

on(c, b)

cube(b)

cube(a)

pyramid(c)

Rachunek predykatów zapewnia programistom sztucznej inteligencji dobrze zdefiniowany język do opisywania i wnioskowania o jakościowych aspektach systemu. Załóżmy, że w przykładzie świata bloków chcemy zdefiniować test w celu ustalenia, czy blok jest czysty, czyli nie ma na nim niczego ułożonego na sobie. Jest to ważne, jeśli ręka robota ma ją podnieść lub ułożyć na niej kolejny blok. Możemy zdefiniować ogólną zasadę:

$$\forall X \neg \exists Y \text{ on}(Y,X) \Rightarrow \text{clear}(X)$$

Jest to odczytywane "dla wszystkich X, X jest jasne, jeśli nie ma Y takiego, że Y jest na X". Tę ogólną zasadę można zastosować w różnych sytuacjach, zastępując różne nazwy bloków, a, b, c itp. , dla X i Y. Wspierając ogólne reguły wnioskowania, rachunek predykatów pozwala na oszczędność reprezentacji, a także na możliwość zaprojektowania systemów, które są wystarczająco elastyczne i ogólne, aby inteligentnie reagować na szereg sytuacji. Rachunek predykatów można również wykorzystać do przedstawienia właściwości jednostek i grup. Często nie wystarczy na przykład opisać samochodu, po prostu wymieniając jego części; możemy chcieć opisać sposoby łączenia tych części i interakcje między nimi. Ten widok struktury jest niezbędny w szeregu sytuacji, w tym w informacjach taksonomicznych, takich jak klasyfikacja roślin według rodzaju i gatunku lub opis złożonych obiektów, takich jak silnik Diesla lub ciało ludzkie, pod względem ich części składowych. Na przykład prosty opis niebieskiego ptaka może brzmieć: "niebieski ptak to mały niebieski ptak, a ptak to pierzasty latający kręgowiec", który można przedstawić jako zbiór logicznych predykatów:

hassize(bluebird,small)

hascovering(bird,feathers)

hascolor(bluebird,blue)

hasproperty(bird,flies)

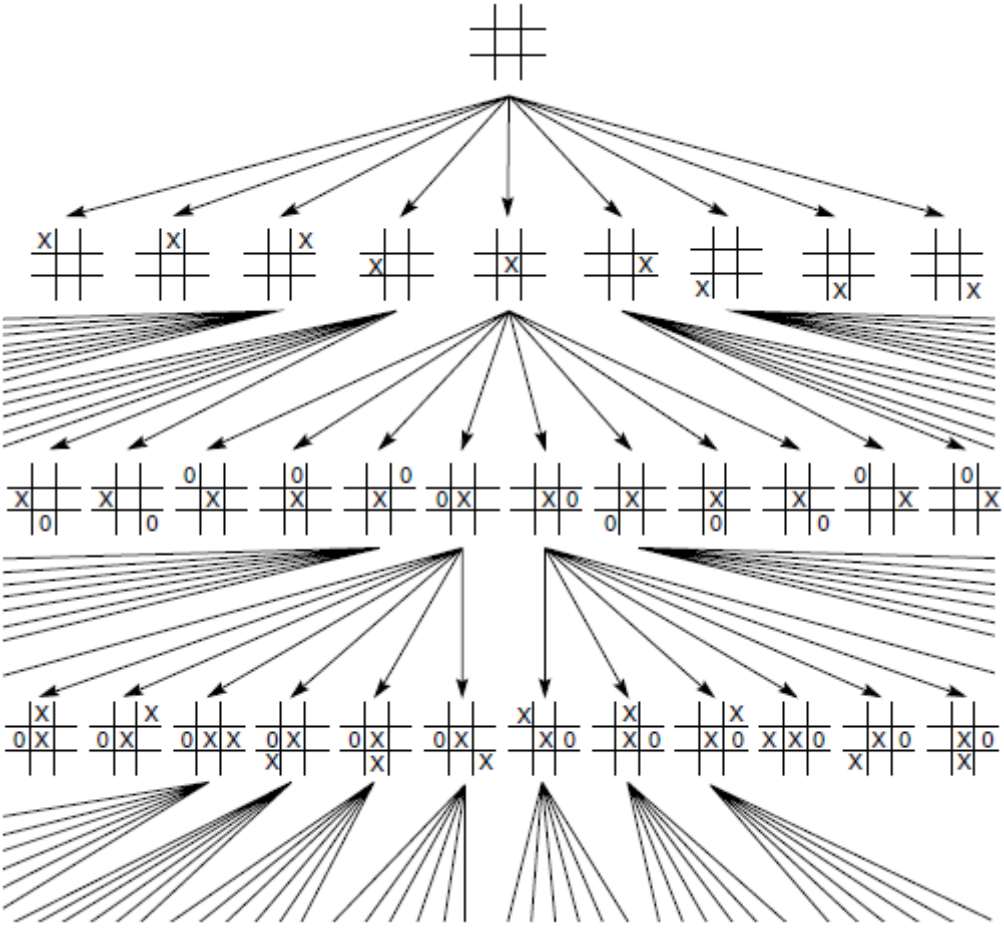
isa(bluebird,bird)

isa(bird,vertebrate)

Ten opis predykatu można przedstawić graficznie za pomocą łuków lub łączy na wykresie zamiast predykatów w celu wskazania relacji. Ta sieć semantyczna jest techniką reprezentowania znaczenia semantycznego. Ponieważ relacje są wyraźnie zaznaczone na wykresie, algorytm do wnioskowania o domenie problemowej może tworzyć odpowiednie powiązania, podążając za linkami. Na przykład na ilustracji bluebird, program musi podążać tylko za jednym linkiem, aby zobaczyć, że mucha leci, i dwoma linkami, aby ustalić, że bluebird jest kręgowcem. Być może najważniejszą aplikacją dla sieci semantycznych jest reprezentowanie znaczenia programów do rozumienia języka. Gdy konieczne jest zrozumienie historii dziecka, szczegółów artykułu w czasopiśmie lub zawartości strony internetowej, sieci semantyczne można wykorzystać do kodowania informacji i relacji odzwierciedlających wiedzę w tej aplikacji.

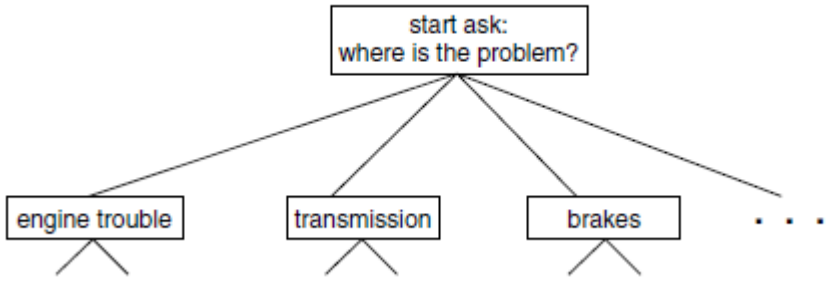
Wyszukiwanie

Biorąc pod uwagę reprezentację, drugim elementem inteligentnego rozwiązywania problemów jest wyszukiwanie. Ludzie na ogół rozważają szereg alternatywnych strategii na drodze do rozwiązania problemu. Gracz w szachy zazwyczaj sprawdza alternatywne ruchy, wybierając "najlepsze" zgodnie z kryteriami, takimi jak możliwe reakcje przeciwnika lub stopień, w jakim różne ruchy wspierają niektóre globalne strategie gry. Gracz bierze również pod uwagę krótkoterminowe korzyści (takie jak zabranie królowej przeciwnika), możliwości poświęcenia elementu dla uzyskania przewagi pozycyjnej lub domysły dotyczące psychologicznego wyglądu przeciwnika i poziomu umiejętności. Ten aspekt inteligentnego zachowania leży u podstaw techniki rozwiązywania problemów wyszukiwania w przestrzeni stanów. Rozważmy na przykład grę w kółko i krzyżyk. W każdej sytuacji na planszy gracz może wykonać tylko skończoną liczbę ruchów. Zaczynając od pustej planszy, pierwszy gracz może umieścić X w dowolnym z dziewięciu miejsc. Każdy z tych ruchów daje inną planszę, która pozwoli przeciwnikowi na osiem możliwych reakcji i tak dalej. Możemy przedstawić tę kolekcję możliwych ruchów i odpowiedzi, traktując każdą konfigurację płytki jako węzeł lub stan na wykresie. Łącza na wykresie przedstawiają legalne ruchy z jednej konfiguracji płyty na drugą. Powstała struktura jest wykresem w przestrzeni stanów. Reprezentacja przestrzeni stanu pozwala zatem traktować wszystkie możliwe gry w kółko i krzyżyk jako różne ścieżki na wykresie przestrzeni stanu. Biorąc pod uwagę tę reprezentację, skuteczna strategia gry przeszuka na wykresie ścieżki, które prowadzą do największej liczby wygranych i najmniejszych strat, i gra w sposób, który zawsze stara się wymusić grę wzdłuż jednej z tych optymalnych ścieżek, jak na rysunku

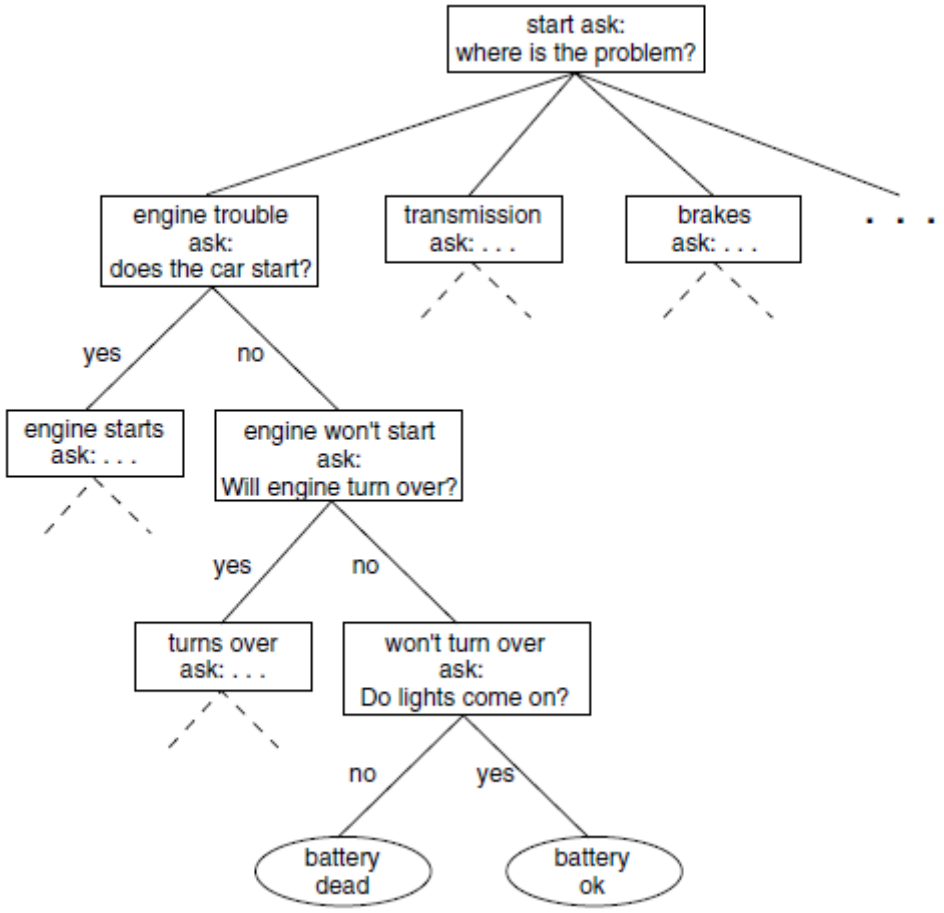


Jako przykład zastosowania wyszukiwania do rozwiązania bardziej skomplikowanego problemu, rozważ zadanie zdiagnozowania usterki mechanicznej w samochodzie. Chociaż początkowo problem ten nie wydaje się tak łatwy do przeszukiwania przestrzeni stanów jak kółko i krzyżyk lub szachy, w

rzeczywistości całkiem dobrze pasuje do tej strategii. Zamiast pozwolić, aby każdy węzeł wykresu reprezentował "stan tablicy", pozwalamy mu reprezentować stan częściowej wiedzy na temat problemów mechanicznych samochodu. Proces badania symptomów usterki i indukowania jej przyczyny można uznać za przeszukiwanie stanów o rosnącej wiedzy. Początkowy węzeł wykresu jest pusty, co wskazuje, że nic nie wiadomo na temat przyczyny problemu. Pierwszą rzeczą, jaką może zrobić mechanik, jest zapytanie klienta, który główny układ (silnik, skrzynia biegów, układ kierowniczy, hamulce itp.) Wydaje się powodować problemy. Jest to reprezentowane przez zbiór łuków od stanu początkowego do stanów, które wskazują na skupienie się na pojedynczym podsystemie samochodu, jak na rysunku



Na przykład węzeł usterki silnika ma łuki do węzłów oznaczone jako start silnika, a silnik nie chce się uruchomić. Z węzła "nie startuj" możemy przejść do węzłów oznaczonych jako zwroty i nie można go odwrócić. Węzeł, który nie przeszedł, ma łuki do węzłów oznaczone jako zużyta bateria i bateria w porządku



Narzędzie do rozwiązywania problemów może zdiagnozować awarię samochodu, szukając ścieżki na tym wykresie, która jest zgodna z objawami konkretnego uszkodzonego samochodu. Chociaż ten problem różni się bardzo od znalezienia optymalnego sposobu gry w kółko i krzyżyk lub w szachy, równie dobrze można go rozwiązać za pomocą wyszukiwania w przestrzeni stanów. Pomimo tej pozornej uniwersalności poszukiwanie przestrzeni stanów samo w sobie nie wystarcza do automatyzacji inteligentnego zachowania w rozwiązywaniu problemów; jest raczej ważnym narzędziem do projektowania inteligentnych programów. Gdyby przeszukiwanie przestrzeni stanu było wystarczające, napisanie programu, który gra w szachy, byłoby dość proste, przeszukując całą przestrzeń pod kątem sekwencji ruchów, które przyniosły zwycięstwo, metodą znaną jako wyczerpujące przeszukiwanie. Chociaż wyczerpujące wyszukiwanie można zastosować do dowolnej przestrzeni stanów, przytłaczająca wielkość przestrzeni dla interesujących problemów sprawia, że takie podejście jest praktycznie niemożliwe. Na przykład szachy mają około 10^{120} różnych stanów planszy. Jest to liczba większa niż liczba cząsteczek we wszechświecie lub liczba nanosekund, które minęły od Wielkiego Wybuchu. Poszukiwanie tej przestrzeni wykracza poza możliwości dowolnego urządzenia komputerowego, którego wymiary muszą być ograniczone do znanego wszechświata i którego wykonanie musi zostać zakończone, zanim wszechświat ulegnie spustoszeniu entropii. Ludzie używają inteligentnego wyszukiwania: szachista rozważa kilka możliwych ruchów, lekarz bada kilka możliwych diagnoz, informatyk bada różne projekty przed rozpoczęciem pisania kodu. Ludzie nie używają wyczerpującego wyszukiwania: szachista bada tylko ruchy, które doświadczalnie okazało się skuteczne, lekarz nie wymaga testów, które nie są w jakiś sposób wskazane przez objawy. Wydaje się, że ludzkie rozwiązywanie problemów opiera się na regułach oceny, które kierują wyszukiwaniem do tych części

przestrzeni państwowej, które wydają się najbardziej "obiecujące". Reguły te są znane jako heurystyka i stanowią jeden z głównych tematów badań nad AI. Heurystyka (nazwa pochodzi od greckiego słowa "odkryć") to strategia selektywnego przeszukiwania przestrzeni problemowej. Prowadzi wyszukiwanie wzdłuż linii, które mają duże prawdopodobieństwo sukcesu, unikając zmarnowanych lub pozornie głupich wysiłków. Ludzie używają dużej liczby heurystyk do rozwiązywania problemów. Jeśli zapytasz mechanika, dlaczego twój samochód się przegrzewa, może powiedzieć coś takiego: "Zwykle oznacza to, że termostat jest zły". Jeśli zapytasz lekarza, co może powodować nudności i bóle brzucha, może powiedzieć, że "prawdopodobnie jest to grypa żołądkowa lub zatrucie pokarmowe. "Wyszukiwanie w przestrzeni stanu pozwala nam sformalizować proces rozwiązywania problemów, a heurystyka pozwala nam nadać temu formalizmowi inteligencję. Techniki te pozostają w centrum najnowocześniejszych prac nad sztuczną inteligencją. Podsumowując, wyszukiwanie w przestrzeni stanu jest formalizmem, niezależnym od konkretnych strategii wyszukiwania i wykorzystywanym jako punkt wyjścia dla wielu podejść do rozwiązywania problemów. W całym tekście kontynuujemy eksplorację teoretycznych aspektów reprezentacji wiedzy i wyszukiwania oraz zastosowania tej teorii w budowaniu skutecznych programów.

Rachunek predykatów

Rachunek zdań

Symbole i zdania

Rachunek zdań i rachunek predykatów, są przede wszystkim językami. Używając ich słów, zwrotów i zdań, możemy reprezentować i uzasadniać właściwości i relacje na świecie. Pierwszym krokiem w opisie języka jest przedstawienie tworzących go elementów: zestawu symboli.

DEFINICJA

Symbole rachunku zdań

Symbole rachunku zdań są symbolami zdań:

P, Q, R, S, …

symbole prawdy: true i false

i łączniki: ∧, ∨, ¬, →, ≡

Symbole zdaniowe oznaczają zdania lub oświadczenia o świecie, które mogą być prawdziwe lub fałszywe, takie jak „samochód jest czerwony” lub „woda jest mokra”. Propozycje są oznaczone dużymi literami na końcu alfabetu angielskiego. Zdania w rachunku zdań są tworzone z tych symboli atomowych zgodnie z następującymi zasadami:

DEFINICJA

Zdania z rachunku zdań

Każdy symbol zdania i symbol prawdy to zdanie.

Na przykład: true, P, Q i R są zdaniami.

Negacja zdania jest zdaniem.

Na przykład: ¬ P i ¬ fałsz to zdania.

Koniunkcja lub and dwóch zdań jest zdaniem.

Na przykład: P ∧ ¬ P jest zdaniem.

Rozłączenie lub lub dwóch zdań jest zdaniem.

Na przykład: P ∨ ¬ P jest zdaniem.

Implikacja jednego zdania z drugiego jest zdaniem.

Na przykład: P → Q to zdanie.

Równoważność dwóch zdań jest zdaniem.

Na przykład: P ∨ Q ≡ R to zdanie.

Zdania legalne są również nazywane dobrze sformułowanymi formułami lub WFF.

W wyrażeniach postaci P ∧ Q, P i Q nazywane są połączonymi. W P ∨ Q, P i Q są nazywane rozłącznymi. W implikacji P → Q, P jest przesłanką lub poprzednikiem, a Q wnioskiem lub konsekwencją. W zdaniach zdań rachunku różniczkowego symbole () i [] są używane do grupowania symboli w podwyrażenia, a więc do kontrolowania ich kolejności oceny i znaczenia. Na przykład (P ∨ Q) ≡ R różni się znacznie od P ∨ (Q ≡ R), co można wykazać za pomocą tabel prawdy. Wyrażenie jest zdaniem lub dobrze sformułowaną formułą rachunku zdań, jeśli i tylko wtedy, gdy można je utworzyć z legalnych symboli poprzez pewną sekwencję tych reguł. Na przykład, ((P∧ Q) → R) ≡¬ P∨ ¬ Q∨ R jest dobrze uformowanym zdaniem w rachunku zdań, ponieważ:

P, Q i R to twierdzenia, a zatem zdania.

P ∧ Q, połączenie dwóch zdań, jest zdaniem.

(P ∧ Q) → R, implikacja zdania dla innego, jest zdaniem.

¬ P i ¬ Q, negacje zdań, są zdaniami.

¬ P ∨ ¬ Q, rozłączenie dwóch zdań, jest zdaniem.

¬ P ∨ ¬ Q ∨ R, rozłączenie dwóch zdań, jest zdaniem.

((P ∧ Q) → R) ≡ ¬ P ∨ ¬ Q ∨ R, równoważność dwóch zdań, jest zdaniem.

To jest nasze oryginalne zdanie, które zostało skonstruowane poprzez szereg zastosowań przepisów prawnych i dlatego jest „dobrze uformowane".

Semantyka rachunku zdań

We wcześniejszej sekcji przedstawiono składnię rachunku zdań, definiując zbiór zasad dotyczących wydawania wyroków sądowych. W tej sekcji formalnie definiujemy semantykę lub „znaczenie" tych zdań. Ponieważ programy AI muszą rozumować za pomocą swoich struktur reprezentacyjnych, ważne jest wykazanie, że prawdziwość ich wniosków zależy tylko od prawdy ich początkowej wiedzy lub przesłanek, tj. że błędy logiczne nie są wprowadzane przez procedury wnioskowania. Precyzyjne traktowanie semantyki jest niezbędne do osiągnięcia tego celu. Symbol propozycji odpowiada stwierdzeniu o świecie. Na przykład P może oznaczać stwierdzenie „pada deszcz" lub Q, stwierdzenie „Mieszkam w brązowym domu". Twierdzenie musi być prawdziwe lub fałszywe, biorąc pod uwagę pewien stan świata. Przypisanie wartości prawdy do zdań zdań jest nazywane interpretacją, twierdzeniem o ich prawdzie w jakimś możliwym świecie. Formalnie interpretacja jest odwzorowaniem symboli zdań na zbiór {T, F}. Jak wspomniano w poprzedniej sekcji, symbole prawda i fałsz są częścią zestawu dobrze uformowanych zdań rachunku zdań; tzn. różnią się od wartości prawdy przypisanej do

zdania. Aby wprowadzić to rozróżnienie, symbole T i F służą do przypisania wartości prawdy. Każde możliwe mapowanie wartości prawdy na zdania odpowiada możliwemu światowi interpretacji. Na przykład, jeśli P oznacza zdanie „pada deszcz", a Q oznacza „Jestem w pracy", to zestaw zdań {P, Q} ma cztery różne mapowania funkcjonalne na wartości prawdy {T, F}. Te odwzorowania odpowiadają czterem różnym interpretacjom. Semantyka rachunku zdań, podobnie jak jego składnia, jest definiowana indukcyjnie:

DEFINICJA

Semantyka rachunku zdań

Interpretacja zbioru zdań jest przypisaniem wartości prawdy, T lub F, do każdego symbolu zdania.

Symbolowi true przypisuje się zawsze T, a symbolowi false - F.

Interpretacja lub wartość prawdy dla zdań jest określona przez:

Prawidłowe przypisanie negacji, ¬P, gdzie P jest dowolnym symbolem zdaniowym, to F, jeśli przypisanie do P jest T, a T, jeśli przypisanie do P jest F.

Prawidłowe przypisanie koniunkcji ∧ jest T tylko wtedy, gdy obie koniunkcje mają wartość prawdy T; w przeciwnym razie jest to F.

Prawidłowe przypisanie rozłączenia ∨ jest F tylko wtedy, gdy oba rozłączenia mają wartość prawdy F; w przeciwnym razie jest to T.

Prawidłowe przypisanie implikacji, →, ma wartość F tylko wtedy, gdy przesłanką lub symbolem przed implikacją jest T, a wartość prawdy wyniku lub symbolu po implikacji wynosi F; w przeciwnym razie jest to T.

Przypisanie równoważności przez prawdę ≡ jest T tylko wtedy, gdy oba wyrażenia mają to samo przypisanie prawdy dla wszystkich możliwych interpretacji; w przeciwnym razie jest to F.

Przypisania prawdy zdań złożonych są często opisywane w tabelach prawdy. Tabela prawdy zawiera listę wszystkich możliwych przypisań wartości prawdy do zdań atomowych wyrażenia i podaje wartość prawdy wyrażenia dla każdego zadania. Zatem tabela prawdy wylicza wszystkie możliwe światy interpretacji, które można nadać wyrażeniu. Na przykład tabela prawdy dla P ∧ Q, zawiera wartości prawdy dla każdego możliwego przypisania prawdy operandów. P ∧ Q jest prawdziwe tylko wtedy, gdy P i Q są jednocześnie T. Lub (∨), nie (¬), implikuje (→) i równoważność (≡) są zdefiniowane w podobny sposób. Konstrukcja tabel prawdy pozostawia się jako ćwiczenie. Dwa wyrażenia w rachunku zdań są równoważne, jeśli mają tę samą wartość w ramach wszystkich przypisań wartości prawdy. Tę równoważność można wykazać za pomocą tabel prawdy. Na przykład dowód równoważności P → Q i ¬ P ∨ Q podano w tabeli prawdy na rycinie 2.2. Wykazując, że dwa różne zdania w rachunku zdań mają identyczne tabele prawdy, możemy udowodnić następujące równoważności. W przypadku wyrażeń zdań P, Q i R:

¬ (¬ P) ≡ P

(P ∨ Q) ≡ (¬ P → Q)

prawo przeciwstawności : (P → Q) ≡ (¬ Q → ¬P)

prawo de Morgana: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) i ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)

prawa przemienne: (P ∧ Q) ≡ (Q ∧ P) i (P ∨ Q) ≡ (Q ∨ P)

prawo łączności: ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))

prawo łączności: ((P ∨ Q) ∨ R) ≡ (P ∨ (Q ∨ R))

prawo rozdzielności: P ∨ (Q ∧ R) ≡ (P ∨ Q) ∧ (P ∨ R)

prawo rozdzielności: P ∧ (Q ∨ R) ≡ (P ∧ Q) ∨ (P ∧ R)

Tożsamości takie jak te mogą być użyte do zmiany wyrażeń rachunku zdań w odmienną składniowo, ale logicznie równoważną formę. Tych tożsamości można użyć zamiast tabel prawdy, aby udowodnić, że dwa wyrażenia są równoważne: znaleźć serię tożsamości, które przekształcają jedno wyrażenie w drugie. Wczesny program sztucznej inteligencji, Teoretyk Logiki, zaprojektowany przez Newella, Simona i Shawa, wykorzystali transformacje między równoważnymi formami wyrażeń, aby udowodnić wiele twierdzeń Whiteheada i Russella, Principia Mathematica (1950). Zdolność do zmiany logicznego wyrażenia na inną formę o równoważnych wartościach prawdy jest również ważna, gdy stosuje się reguły wnioskowania , które wymagają, aby wyrażenia były w formie specyficznej.

Rachunek predykatów

W rachunku zdań każdy symbol atomowy (P, Q itd.) oznacza pojedyncze zdanie. Nie ma możliwości uzyskania dostępu do składników pojedynczego stwierdzenia. Rachunek predykatów zapewnia tę zdolność. Na przykład, zamiast pozwolić jednemu symbolowi zdaniowemu, P, oznaczać całe zdanie "padało we wtorek", możemy stworzyć predykatową pogodę, która opisuje związek między datą a pogodą: pogoda (wtorek, deszcz). Dzięki regułom wnioskowania możemy manipulować wyrażeniami rachunku predykatów, uzyskując dostęp do ich poszczególnych składników i wprowadzając nowe zdania. Rachunek predykatów pozwala również, aby wyrażenia zawierały zmienne. Zmienne pozwalają nam tworzyć ogólne twierdzenia o klasach bytów. Na przykład możemy stwierdzić, że dla wszystkich wartości X, gdzie X jest dniem tygodnia, wyrażenie pogoda (X, deszcz) jest prawdziwe; czyli pada codziennie. Podobnie jak w przypadku rachunku zdań, najpierw zdefiniujemy składnię języka, a następnie omówimy jego semantykę.

Składnia predykatów i zdań

Przed zdefiniowaniem składni poprawnych wyrażeń w rachunku predykatów definiujemy alfabet i gramatykę do tworzenia symboli języka. Odpowiada to leksykalnemu aspektowi definicji języka programowania. Symbole rachunku predykatu, podobnie jak tokeny w języku programowania, są nieredukowalnymi elementami składniowymi: nie można ich rozbić na części składowe za pomocą operacji języka. W naszej prezentacji reprezentujemy symbole rachunku predykatu jako ciągi liter i cyfr rozpoczynające się na literę. Nie mogą pojawiać się spacje i znaki niealfanumeryczne wewnątrz ciągu znaków, chociaż podkreślenie _ może być użyte do poprawy czytelności.

DEFINICJA

SYMBOLE RACHUNKU PREDYKATÓW

Alfabet, który składa się z symboli rachunku predykatu, składa się z:

1. Zestaw liter alfabetu angielskiego, zarówno wielkich, jak i małych.

2. Zbiór cyfr, 0, 1, …, 9.

3. Podkreślenie, _.

Symbole w rachunku predykatów zaczynają się od litery, po której następuje dowolna sekwencja tych legalnych znaków. Do prawidłowych znaków alfabetu symboli rachunku predykatu należą

a R 6 9 p _ z

Przykłady znaków spoza alfabetu obejmują

#% @ / &

Uzasadnione symbole rachunku predykatu obejmują

George fire3 tom_and_jerry bill XXXX friends_of

Przykłady ciągów znaków, które nie są legalnymi symbolami

3jack bez pustych miejsc ab% cd *** 71 kaczka !!!

Symbole, jak widzimy , są używane do oznaczania obiektów, właściwości lub relacji w świecie dyskursu. Podobnie jak w przypadku większości języków programowania, użycie "słów" sugerujących zamierzone znaczenie symbolu pomaga nam zrozumieć kod programu. Tak więc, chociaż l (g, k) i polubienia (George, Kate) są formalnie równoważne (tj. mają tę samą strukturę), drugi może być bardzo pomocny (dla ludzkich czytelników) we wskazywaniu, jaką relację reprezentuje to wyrażenie. Należy podkreślić, że te opisowe nazwy służą wyłącznie poprawie czytelności wyrażeń. Jedyne "znaczenie", które mają wyrażenia rachunku predykatu, podane jest poprzez ich formalną semantykę. Nawiasy "()", przecinki "," i kropki "." Są używane wyłącznie do konstruowania dobrze sformułowanych wyrażeń i nie oznaczają obiektów ani relacji na świecie. Są to tak zwane niewłaściwe symbole. Symbole rachunku predykatu mogą reprezentować zmienne, stałe, funkcje lub predykaty. Stałe nazywają określone obiekty lub właściwości na świecie. Symbole stałe muszą zaczynać się małą literą. Tak więc George, drzewo, wysoki i niebieski są przykładami dobrze uformowanych stałych symboli. Stałe prawda i fałsz są zastrzeżone jako symbole prawdy. Symbole zmienne są używane do oznaczania ogólnych klas obiektów lub właściwości na świecie. Zmienne są reprezentowane przez symbole rozpoczynające się od dużej litery. A zatem George, BILL i KAte są zmiennymi prawnymi, podczas gdy George i Bill nie. Rachunek predykatów umożliwia także funkcje na obiektach w świecie dyskursu. Symbole funkcji (jak stałe) zaczynają się od małej litery. Funkcje oznaczają odwzorowanie jednego lub więcej elementów w zestawie (zwanym domeną funkcji) na unikalny element drugiego zestawu (zakres funkcji). Elementami dziedziny i zasięgu są obiekty w świecie dyskursu. Oprócz typowych funkcji arytmetycznych, takich jak dodawanie i mnożenie, funkcje mogą definiować odwzorowania między domenami nieliczbowymi. Zauważ, że nasza definicja symboli rachunku predykatu nie obejmuje liczb ani operatorów arytmetycznych. System liczbowy nie jest uwzględniony w operacjach pierwotnych rachunku predykatu; zamiast tego jest definiowany aksjomatycznie przy użyciu "czystego" rachunku predykatów jako podstawy. Chociaż szczegóły tej pochodnej mają znaczenie teoretyczne, są one mniej ważne dla zastosowania rachunku predykatów jako języka reprezentacji AI. Dla wygody przyjmujemy to wyprowadzenie i uwzględniamy arytmetykę w języku. Każdy symbol funkcji ma powiązaną aranżację, wskazując liczbę elementów w domenie odwzorowanych na każdy element zakresu. W ten sposób ojciec mógłby określić funkcję arity 1, która mapuje ludzi na ich (unikalnego) męskiego rodzica. plus może być funkcją arity 2, która odwzorowuje dwie liczby na ich sumę arytmetyczną. Wyrażenie funkcyjne to symbol funkcji, po której następują jej argumenty. Argumenty są elementami z dziedziny funkcji; liczba argumentów jest równa arity funkcji. Argumenty są ujęte w nawiasy i oddzielone przecinkami. Na przykład,

f(X, Y)

ojciec(dawid)

cena(banany)

są dobrze sformułowanymi wyrażeniami funkcyjnymi.

Każde wyrażenie funkcji oznacza odwzorowanie argumentów na pojedynczy obiekt w zakresie, zwany wartością funkcji. Na przykład, jeśli ojciec jest funkcją jednoargumentową, to ojciec (David) jest wyrażeniem funkcji, którego wartością (w świecie dyskursu autora) jest George. Jeśli plus jest funkcją arity 2, z domeną liczb całkowitych, to plus (2,3) jest wyrażeniem funkcyjnym, którego wartością jest liczba całkowita 5. Czynność zastąpienia funkcji jej wartością nazywana jest oceną. Pojęcie symbolu lub terminu rachunku predykatu sformalizowano w następującej definicji:

DEFINICJA

SYMBOLE I WARUNKI

Symbole rachunku predykatu obejmują:

1. Symbole prawdy prawda i fałsz (są to symbole zastrzeżone).

2. Symbole stałe to wyrażenia symboli, których pierwsza litera jest mała.

3. Symbole zmienne są wyrażeniami symboli rozpoczynającymi się od wielkich liter.

4. Symbole funkcyjne to wyrażenia symboliczne zawierające pierwszą literę małej litery. Funkcje mają dołączony arian wskazujący liczbę elementów domeny odwzorowanych na każdy element zakresu.

Wyrażenie funkcyjne składa się ze stałej funkcji arity n, po której następuje n terminów, t_1, t_2 ... t_n, otoczonych nawiasami i oddzielonych przecinkami. Termin rachunku predykatu jest wyrażeniem stałym, zmiennym lub funkcyjnym. Zatem predykcyjny rachunek różniczkowy może być użyty do oznaczenia obiektów i właściwości w dziedzinie problemowej. Przykłady terminów to:

kot

razy(2,3)

X

niebieski

matka(Sarah)

Kate

Symbole w rachunku predykatów mogą również reprezentować predykaty. Symbole predykatów, takie jak stałe i nazwy funkcji, zaczynają się od małej litery. Predykat określa związek między zero lub większą liczbą obiektów na świecie. Tak powiązana liczba obiektów jest rodzajem predykatu. Przykładami predykatów są polubienia równe w pobliżu części_ Zdanie atomowe, najbardziej prymitywna jednostka języka rachunku predykatów, jest predykatem arity n, po którym następuje n terminów zamkniętych w nawiasach i oddzielonych przecinkami. Przykładami zdań atomowych są

likes(george,kate) likes(X,george)

likes(george,susie) likes(X,X)

likes(george,sarah,tuesday) friends(bill,richard)

friends(bill,george) friends(father_of(david),father_of(andrew))

helps(bill,george) helps(richard,bill)

Symbole predykatów w tych wyrażeniach to polubienia, przyjaciele i pomoc. Symbol predykatu może być używany z różną liczbą argumentów. W tym przykładzie są dwa różne polubienia, jeden z dwoma, a drugi z trzema argumentami. Gdy w zdaniach o różnych aracjach używany jest symbol predykatu, uważa się, że reprezentuje on dwie różne relacje. Tak więc relacja predykatu jest definiowana przez jej nazwę i aranżację. Nie ma powodu, dla którego te dwa różne upodobania nie mogą stanowić części tego samego opisu świata; jednak zwykle tego się unika, ponieważ często może powodować zamieszanie. W powyższych predykatach rachunek, George, Kate itp. Są stałymi symbolami i reprezentują obiekty w dziedzinie problemowej. Argumentami predykatu są terminy i mogą również zawierać zmienne lub wyrażenia funkcyjne. Na przykład przyjaciele (father_of (David), father_of (andrew)) jest predykatem opisującym związek między dwoma obiektami w dziedzinie dyskursu. Argumenty te są reprezentowane jako wyrażenia funkcyjne, których odwzorowania (biorąc pod uwagę, że ojciec_david to George, a ojciec_ andrew to allen) tworzą parametry. Jeśli wyrażenia funkcyjne są oceniane, wyrażenie staje się przyjaciółmi (George, Allen). Te pomysły zostały sformalizowane w następującej definicji.

DEFINICJA

PREDYKAT iI ZDARZENIA ATOMOWE

Symbole predykatów to symbole rozpoczynające się od małej litery.

Predykaty mają powiązaną dodatnią liczbę całkowitą określaną jako arity lub "liczba argumentów" dla predykatu. Predykaty o tej samej nazwie, ale różnych ariach są uważane za odrębne. Zdanie atomowe jest predykatem stałej arity n, po której następuje n terminów, ujętych w nawiasy i oddzielonych przecinkami. Wartości prawdy, prawda i fałsz, są również zdaniami atomowymi.

Zdania atomowe nazywane są również wyrażeniami atomowymi, atomami lub zdaniami. Możemy łączyć zdania atomowe za pomocą operatorów logicznych, aby tworzyć zdania w rachunku predykatów. Są to te same logiczne łączniki, które są używane w rachunku zdań:

∧, ∨, ¬, →, i ≡

Kiedy zmienna pojawia się jako argument w zdaniu, odnosi się do nieokreślonych obiektów w domenie. Rachunek predykatu pierwszego rzędu zawiera dwa symbole, kwantyfikatory zmiennych ∀ i ∃, które ograniczają znaczenie zdania zawierającego zmienną. Po kwantyfikatorze następuje zmienna i zdanie, takie jak

∃Y friends(Y, peter)

∀X likes(X, ice_cream)

Uniwersalny kwantyfikator ∀ wskazuje, że zdanie jest prawdziwe dla wszystkich wartości zmiennej. W przykładzie ∀X polubień (X, lody_kremowe) jest prawdziwe dla wszystkich wartości w dziedzinie definicji X. Egzystencjalny kwantyfikator ∃ wskazuje, że zdanie jest prawdziwe dla co najmniej jednej wartości w domenie. Friends Przyjaciele Y (Y, Peter) są prawdziwe, jeśli istnieje co najmniej jeden obiekt wskazany przez Y, który jest przyjacielem Petera. Zdania w rachunku predykatów są definiowane indukcyjnie.

DEFINICJA

PREYKAT RACHUNKU ZDAŃ

Każde zdanie atomowe jest zdaniem.

1. Jeśli s jest zdaniem, to i jego negacja, ¬s.

2. Jeśli s_1 i s_2 są zdaniami, to także ich koniunkcja, $s_1 \wedge s_2$.

3. Jeśli s_1 i s_2 są zdaniami, to i ich rozłączenie, $s_1 \vee s_2$.

4. Jeśli s_1 i s_2 są zdaniami, to ich implikacja, $s_1 \rightarrow s_2$.

5. Jeśli s_1 i s_2 są zdaniami, to ich równoważność, $s_1 \equiv s_2$.

6. Jeśli X jest zmienną a s zdaniem, to ∀ X s jest zdaniem.

7. Jeśli X jest zmienną a s zdaniem, to ∃ X s jest zdaniem.

Poniżej podano przykłady poprawnie sformułowanych zdań. Niech czasy i plus będą symbolami funkcyjnymi arity 2, a równe i foo będą predykatami symboli odpowiednio arity 2 i 3.

plus(dwa, trzy) jest funkcją, a zatem nie jest zdaniem atomowym.

równe(plus(dwa, trzy), pięć) to zdanie atomowe.

równe(plus(2, 3), siedem) to zdanie atomowe. Zauważ, że to zdanie, biorąc pod uwagę standardową interpretację plus i równość, jest fałszywe. Dobra forma i wartość prawdy są niezależnymi zagadnieniami.

∃Xfoo (X, dwa, plus (dwa, trzy)) ∧ równe (plus (dwa, trzy), pięć) to zdanie, ponieważ obie koniunkcje są zdaniami.

(foo(dwa, dwa, plus (dwa, trzy))) → (równe (plus (trzy, dwa), pięć) ≡ prawda) to zdanie, ponieważ wszystkie jego elementy są zdaniami, odpowiednio połączonymi operatorami logicznymi.

Definicja zdań rachunku predykatu i właśnie przedstawione przykłady sugerują metodę weryfikacji, czy wyrażenie jest zdaniem. Jest to zapisywane jako algorytm rekurencyjny, Verse_sentence. Verit_sentence przyjmuje jako argument wyrażenie kandydujące i zwraca sukces, jeśli wyrażenie jest zdaniem.

function verify_sentence(expression);

begin

case

expression is an atomic sentence: return SUCCESS;

expression is of the form Q X s, where Q is either ∀ or ∃, X is a variable,

if verify_sentence(s) returns SUCCESS

then return SUCCESS

else return FAIL;

expression is of the form ¬ s:

if verify_sentence(s) returns SUCCESS

then return SUCCESS

else return FAIL;

expression is of the form s_1 op s_2, where op is a binary logical operator:

if verify_sentence(s_1) returns SUCCESS and

verify_sentence(s_2) returns SUCCESS

then return SUCCESS

else return FAIL;

otherwise: return FAIL

end

end.

Kończymy tę sekcję przykładem użycia rachunku predykatu do opisu prostego świata. Dziedziną dyskursu jest zbiór relacji rodzinnych w biblijnej genealogii:

mother(eve,abel)

mother(eve,cain)

father(adam,abel)

father(adam,cain)

$\forall X \forall Y$ father(X, Y) $\vee$ mother(X, Y) $\rightarrow$ parent(X, Y)

$\forall X \forall Y \forall Z$ parent(X, Y) $\wedge$ parent(X, Z) $\rightarrow$ sibling(Y, Z)

W tym przykładzie używamy predykatów matka i ojciec, aby zdefiniować zestaw relacji rodzic-dziecko. Implikacje podają ogólne definicje innych relacji, takich jak rodzic i rodzeństwo, w odniesieniu do tych predykatów. Intuicyjnie jasne jest, że implikacje te można wykorzystać do wnioskowania o faktach, takich jak rodzeństwo (cain, abel). Aby sformalizować ten proces, aby można go było przeprowadzić na komputerze, należy dołożyć starań, aby zdefiniować algorytmy wnioskowania i upewnić się, że takie algorytmy rzeczywiście wyciągają prawidłowe wnioski z zestawu twierdzeń rachunku predykatu. W tym celu definiujemy semantykę rachunku predykatów, a następnie rozwiązujemy problem reguł wnioskowania.

Semantyka dla rachunku predykatów

Po zdefiniowaniu dobrze sformułowanych wyrażeń w rachunku predykatów ważne jest, aby określić ich znaczenie w kategoriach obiektów, właściwości i relacji na świecie. Semantyka rachunku predykatów zapewnia formalną podstawę do określenia wartości prawdy poprawnie sformułowanych wyrażeń. Prawda wyrażeń zależy od mapowania stałych, zmiennych, predykatów i funkcji na obiekty i relacje w dziedzinie dyskursu. Prawda relacji w dziedzinie określa prawdziwość odpowiednich wyrażeń. Na przykład informacje na temat osoby, George'a i jego przyjaciół Kate i Susie mogą być wyrażone przez

przyjaciele (George, Susie)

przyjaciele (George, Kate)

Jeśli to prawda, że George jest przyjacielem Susie, a George jest przyjacielem Kate, wówczas każde z tych wyrażeń miałoby wartość prawdy (przypisanie) T. Jeśli George jest przyjacielem Susie, ale nie Kate, wówczas pierwsze wyrażenie byłoby mają wartość prawdy T, a druga miałaby wartość prawdy F. Aby wykorzystać rachunek predykatów jako reprezentację do rozwiązywania problemów, opisujemy obiekty i relacje w dziedzinie interpretacji za pomocą zestawu dobrze sformułowanych wyrażeń. Terminy i predykcje tych wyrażeń oznaczają obiekty i relacje w domenie. Ta baza wyrażeń rachunku predykatu, z których każde ma wartość prawdy T, opisuje "stan świata". Opis George'a i jego przyjaciół jest prostym przykładem takiej bazy danych. Innym przykładem jest świat bloków we wstępie do części II. W oparciu o te intuicje formalnie definiujemy semantykę rachunku predykatów. Najpierw definiujemy interpretację w domenie D. Następnie używamy tej interpretacji do określenia przypisania zdań prawdy do wartości w języku.

DEFINICJA

INTERPRETACJA

Niech domena D będzie niepustym zestawem.

Interpretacja nad D jest przypisaniem bytów D do każdego z symboli stałych, zmiennych, predykatów i funkcji wyrażenia rachunku predykatów, tak że:

1. Każda stała ma przypisany element D.

2. Każda zmienna jest przypisana do niepustego podzbioru D; są to dopuszczalne podstawienia dla tej zmiennej.

3. Każda funkcja f arity m jest zdefiniowana na m argumentach D i definiuje odwzorowanie z D^m na D.

4. Każdy predykat ar ar n jest zdefiniowany na n argumentach z D i definiuje odwzorowanie z D^n na {T, F}.

Biorąc pod uwagę interpretację, znaczenie wyrażenia jest przypisaniem wartości prawdy nad interpretacją.

DEFINICJA

WARTOŚĆPRAWDY WYRAŻEŃ RAHUNKU ZDAŃ

Załóżmy wyrażenie E i interpretację I dla E w niepustej domenie D. Wartość prawdy dla E jest określona przez:

1. Wartość stałej jest elementem D, do którego przypisuje ją I.

2. Wartością zmiennej jest zbiór elementów D, do którego przypisuje ją I.

3. Wartością wyrażenia funkcyjnego jest ten element D uzyskany przez ocenę funkcji dla wartości parametrów przypisanych przez interpretację.

4. Wartość symbolu prawdy "prawda" to T, a "fałsz" to F.

5. Wartość zdania atomowego wynosi T lub F, zgodnie z interpretacją I.

6. Wartość negacji zdania wynosi T, jeśli wartością zdania jest F, a F, jeśli wartością zdania jest T.

7. Wartość koniunkcji dwóch zdań wynosi T, jeśli wartość obu zdań to T, a w przeciwnym razie F.

8. - 10. Wartość prawdziwości wyrażeń używających ∨, →, i ≡ określa się na podstawie wartości ich argumentów określonych w sekcji wcześniejszej. Wreszcie dla zmiennej X i zdania S zawierającego X:

11. Wartość ∀ X S wynosi T, jeśli S jest T dla wszystkich przypisań do X w ramach I, a F w przeciwnym razie.

12. Wartość ∃ X S wynosi T, jeżeli w interpretacji istnieje przyporządkowanie do X; w przeciwnym razie jest to F.

Kwantyfikacja zmiennych jest ważną częścią semantyki rachunku predykatu. Kiedy zmienna pojawia się w zdaniu, na przykład X w polubieniach (George, X), zmienna działa jako symbol zastępczy. Każda stała dozwolona w ramach interpretacji może zostać zastąpiona w wyrażeniu. Podstawienie kate lub susie za X w polubieniach (George, X) tworzy stwierdzenia polubienia (George, Kate) i polubienia (George, Susie), jak widzieliśmy wcześniej. Zmienna X oznacza wszystkie stałe, które mogą pojawić się jako drugi parametr zdania. Ta nazwa zmiennej może zostać zastąpiona dowolną inną nazwą zmiennej, taką jak Y lub LUDZIE, bez zmiany znaczenia wyrażenia. Zatem zmienna jest uważana za manekina. W rachunku predykatów zmienne muszą być kwantyfikowane na jeden z dwóch sposobów: uniwersalnie lub egzystencjalnie. Zmienna jest uważana za swobodną, jeżeli nie wchodzi w zakres kwantyfikatorów uniwersalnych lub egzystencjalnych. Wyrażenie jest zamykane, jeśli wszystkie jego zmienne są kwantyfikowane. Wyrażenie podstawowe nie ma żadnych zmiennych. W rachunku predykatów wszystkie zmienne muszą być kwantyfikowane. Nawiasy są często używane w celu wskazania zakresu kwantyfikacji, to znaczy wystąpień nazwy zmiennej, które dotyczą kwantyfikacji. Zatem dla symbolu wskazującego uniwersalną kwantyfikację ∀:

$$\forall X\,(p(X) \vee q(Y) \rightarrow r(X))$$

wskazuje, że X jest powszechnie określany ilościowo zarówno w p(X), jak i r(X). Uniwersalna kwantyfikacja wprowadza problemy w obliczaniu prawdziwej wartości zdania, ponieważ wszystkie możliwe wartości symbolu zmiennej muszą zostać przetestowane, aby sprawdzić, czy wyrażenie pozostaje prawdziwe. Na przykład, aby przetestować wartość prawdy likes X polubień (George, X), gdzie X rozciąga się na zbiór wszystkich ludzi, należy przetestować wszystkie możliwe wartości X. Jeśli dziedzina interpretacji jest nieskończona, wyczerpujące testowanie wszystkich podstawień zmiennej uniwersalnej jest niemożliwe obliczeniowo: algorytm nigdy nie może zostać zatrzymany. Z tego powodu mówi się, że rachunek predykatów jest nierozstrzygalny. Ponieważ rachunek zdań nie obsługuje zmiennych, zdania mogą mieć tylko skończoną liczbę przypisań prawdy i możemy wyczerpująco przetestować wszystkie te możliwe przypisania. Dokonuje się tego za pomocą tabeli prawdy. Zmienne mogą być również egzystencjalnie określone ilościowo, oznaczone symbolem ?. W przypadku egzystencjalnym mówi się, że wyrażenie zawierające zmienną jest prawdziwe dla co najmniej jednego podstawienia z jego dziedziny definicji. Zakres egzystencjalnie skwantyfikowanej zmiennej jest również wskazany poprzez umieszczenie skwantyfikowanych wystąpień zmiennej w nawiasach. Ocena prawdziwości wyrażenia zawierającego zmienną skwantyfikowaną egzystencjalnie może nie być łatwiejsze niż ocena prawdziwości wyrażeń zawierających zmienne skwantyfikowane. Załóżmy, że próbujemy ustalić prawdziwość wyrażenia, próbując podstawienia, dopóki nie zostanie znalezione takie, które sprawi, że wyrażenie będzie prawdziwe. Jeśli domena zmiennej jest nieskończona a wyrażenie jest fałszywe przy wszystkich podstawieniach, algorytm nigdy się nie zatrzyma. Kilka zależności między negacją a uniwersalnymi i egzystencjalnymi kwantyfikatorami podano poniżej. Zależności te są wykorzystywane w opisanych powyżej systemach odrzucania rozdzielczości. Zwraca się również uwagę na nazwę zmiennej jako symbol zastępczy, który oznacza zbiór stałych. Dla predykatów p i q oraz zmiennych X i Y:

$$\neg \exists X\, p(X) \equiv \forall X \neg p(X)$$

¬ ∀ X p(X) ≡ ∃ X ¬ p(X)

∃ X p(X) ≡ ∃ Y p(Y)

∀ X q(X) ≡ ∀ Y q(Y)

∀ X (p(X) ∧ q(X)) ≡ ∀ X p(X) ∧ ∀ Y q(Y)

∃ X (p(X) ∨ q(X)) ≡ ∃ X p(X) ∨ ∃ Y q(Y)

W zdefiniowanym przez nas języku uniwersalnie i egzystencjalnie zmienne kwantyfikowane mogą odnosić się tylko do obiektów (stałych) w dziedzinie dyskursu. Predykatów i nazw funkcji nie można zastępować zmiennymi kwantyfikowanymi. Ten język nosi nazwę rachunek predykatów pierwszego rzędu

DEFINICJA

RACHUNEK PREDYKATÓW PIERWSZEGO RZĘDU

Rachunek predykatów pierwszego rzędu pozwala zmiennym kwantyfikowanym odwoływać się do obiektów w dziedzinie dyskursu, a nie do predykatów lub funkcji.

Na przykład ∀ (Likes) Likes (George, Kate) nie jest dobrze sformułowanym wyrażeniem w rachunku predykatów pierwszego rzędu. Istnieją wyliczenia predykatów wyższego rzędu, w których takie wyrażenia są znaczące. Niektórzy badacze używali języków wyższego rzędu do reprezentowania wiedzy w programach do rozumienia języka naturalnego. Wiele poprawnych gramatycznie zdań w języku angielskim może być reprezentowanych w rachunku predykatów pierwszego rzędu za pomocą symboli, łączników i symboli zmiennych zdefiniowanych w tej sekcji. Należy zauważyć, że nie ma unikalnego mapowania zdań na wyrażenia rachunku predykatu; w rzeczywistości zdanie angielskie może mieć dowolną liczbę różnych reprezentacji rachunku predykatu. Głównym wyzwaniem dla programistów AI jest znalezienie schematu wykorzystania tych predykatów, który optymalizuje ekspresję i wydajność wynikowej reprezentacji. Przykłady zdań w języku angielskim reprezentowanych w rachunku predykatów są:

Jeśli w poniedziałek nie pada, Tom pojedzie w góry.

¬ pogoda (deszcz, poniedziałek) → idź (tom, góry)

Emma to Doberman Pinczer i dobry pies.

gooddog (emma) ∧ isa (emma, doberman)

Wszyscy koszykarze są wysocy.

∀ X (gracz koszykówki (X) → wysoki (X))

Niektórzy ludzie lubią sardele.

∃ X (osoba (X) ∧ lubi (X, anchois))

Gdyby życzenia były koniami, żebracy jeździliby

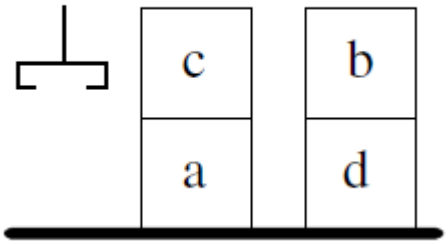
równe (życzenia, konie) → jazda (żebracy)

Nikt nie lubi podatków.

¬ ∃ Polubienia (X, podatki)

Przykład "świata bloków" o znaczeniu semantycznym

Kończymy tę sekcję, podając rozszerzony przykład przypisania wartości prawdy do zestawu wyrażeń rachunku predykatu. Załóżmy, że chcemy modelować świat bloków z rysunku w celu zaprojektowania, na przykład, algorytmu sterowania dla ramienia robota.



Możemy użyć zdań rachunku predykatu do przedstawienia relacji jakościowych na świecie: czy dany blok ma wyraźną górną powierzchnię? czy możemy odebrać blok a? itp. Załóżmy, że komputer ma wiedzę o położeniu każdego bloku i ramienia i jest w stanie śledzić te położenia (przy użyciu trójwymiarowych współrzędnych), gdy ręka przesuwa bloki wokół stołu. Musimy być bardzo precyzyjni w kwestii tego, co proponujemy na tym przykładzie "świata bloków". Po pierwsze, tworzymy zestaw wyrażeń rachunku predykatu, który ma reprezentować statyczną migawkę domeny problemu świata bloków. Jak zobaczymy później, ten zestaw bloków oferuje interpretację i możliwy model zbioru wyrażeń rachunku predykatu. Po drugie, rachunek predykatów jest deklaratywny, to znaczy nie ma założonego terminu ani kolejności uwzględnienia każdego wyrażenia. Niemniej jednak w sekcji planowania , dodamy "semantykę proceduralną" lub jasno określoną metodologię oceny tych wyrażeń w czasie. Konkretnym przykładem semantyki proceduralnej dla wyrażeń rachunku predykatu jest Prolog. Ten rachunek sytuacji, który tworzymy, wprowadzi szereg zagadnień, w tym problem ramowy i zagadnienie niemonotoniczności interpretacji logicznych, które zostaną omówione w dalszej części tej książki. W tym przykładzie wystarczy jednak powiedzieć, że nasze wyrażenia rachunku predykatu będą oceniane od góry do dołu i od lewej do prawej. Aby podnieść blok i ułożyć go w innym bloku, oba bloki muszą być czyste. Na rysunku , blok a nie jest jasny. Ponieważ ramię może przesuwać bloki, może zmieniać stan świata i usuwać blok. Załóżmy, że usuwa blok c z bloku a i aktualizuje bazę wiedzy, aby to odzwierciedlić, usuwając twierdzenie z (c, a). Program musi być w stanie wywnioskować, że blok a stał się jasny. Poniższa reguła opisuje, kiedy blok jest wyczyszczony:

$\forall$ X ($\neg$ $\exists$ Y on (Y, X) $\rightarrow$ wyczyść (X))

Oznacza to, że dla wszystkich X, X jest jasne, jeśli nie istnieje Y takie, że Y jest na X. Ta reguła nie tylko określa, co to znaczy, że blok jest czysty, ale także stanowi podstawę do ustalenia, jak usunąć bloki, które nie są. Na przykład blok d nie jest jasny, ponieważ jeśli zmienna X otrzyma wartość d, podstawienie b na Y spowoduje, że instrukcja będzie fałszywa. Dlatego, aby ta definicja była prawdziwa, blok b należy usunąć z bloku d. Można to łatwo zrobić, ponieważ komputer ma zapis wszystkich bloków i ich lokalizacji. Oprócz użycia implikacji do zdefiniowania, kiedy blok jest czysty, można dodać inne reguły opisujące operacje, takie jak układanie jednego bloku na drugim. Na przykład: aby ułożyć X na Y, najpierw opróżnij rękę, następnie wyczyść X, następnie wyczyść Y, a następnie pick_up X i umieść X na Y.

$\forall$ X $\forall$ Y ((hand_empty $\wedge$ clear(X) $\wedge$ clear(Y) $\wedge$ pick_up(X) $\wedge$ put_down(X,Y)) $\rightarrow$stack(X,Y))

Zauważ, że przy implementacji powyższego opisu konieczne jest "dołączenie" działania ramienia robota do każdego predykatu, takiego jak pick_up (X). Jak wspomniano wcześniej, dla takiej implementacji konieczne było zwiększenie semantyki rachunku predykatów poprzez wymaganie, aby działania były wykonywane w kolejności, w jakiej występują w przesłance reguły. Jednak wiele zyskuje się poprzez oddzielenie tych zagadnień od użycia rachunku predykatu do zdefiniowania relacji i operacji w domenie. Rysunek powyższy przedstawia interpretację semantyczną tych wyrażeń rachunku predykatu. Ta interpretacja odwzorowuje stałe i predykaty w zestawie wyrażeń na domenę D, tutaj bloki i relacje między nimi. Interpretacja nadaje wartości T wartość każdemu wyrażeniu w opisie. Inną interpretację może zaoferować inny zestaw klocków w innym miejscu lub zespół czterech akrobatów. Ważnym pytaniem nie jest wyjątkowość interpretacji, ale to, czy interpretacja zapewnia wartość prawdy dla wszystkich wyrażeń w zbiorze i czy wyrażenia opisują świat wystarczająco szczegółowo, aby można było przeprowadzić wszystkie niezbędne wnioskowania poprzez manipulację wyrażeniami symbolicznymi. W następnej sekcji wykorzystano te pomysły, aby zapewnić formalną podstawę reguł wnioskowania rachunku predykatu.

Używanie reguł wnioskowania do tworzenia wyrażeń rachunku predykatu

Zasady wnioskowania

Semantyka rachunku predykatów stanowi podstawę formalnej teorii wnioskowania logicznego. Zdolność wnioskowania nowych poprawnych wyrażeń na podstawie zestawu prawdziwych twierdzeń jest ważną cechą rachunku predykatów. Te nowe wyrażenia są w tym poprawne są one zgodne ze wszystkimi poprzednimi interpretacjami oryginalnego zestawu wyrażeń. Najpierw omawiamy te pomysły nieoficjalnie, a następnie tworzymy zestaw definicji, aby były precyzyjne. Mówi się, że interpretacja, która sprawia, że zdanie jest zgodne z prawdą. Mówi się, że interpretacja, która spełnia każdy element zestawu wyrażeń, spełnia zestaw. Wyrażenie X logicznie wynika z zestawu wyrażeń rachunku predykatów S, jeśli każda interpretacja, która spełnia S, również spełnia X. To pojęcie daje nam podstawę do weryfikacji poprawności reguł wnioskowania: funkcją wnioskowania logicznego jest tworzenie nowych zdań, które logicznie postępuj zgodnie z danym zestawem zdań rachunku różniczkowego. Ważne jest, aby zrozumieć dokładne znaczenie logicznie następującego: aby wyrażenie X logicznie następowało po S, musi być prawdą w przypadku każdej interpretacji, która spełnia oryginalny zestaw wyrażeń S. Oznaczałoby to na przykład, że każde nowe wyrażenie rachunku predykatu dodane do świata bloków z rysunku 2.3 musi być prawdziwe w tym świecie, a także w każdej innej interpretacji, jaką może mieć ten zestaw wyrażeń. Sam termin "logicznie podąża" może być nieco mylący. Nie oznacza to, że X można wywnioskować z, ani nawet, że można go wydedukować ze S. To po prostu oznacza, że X jest prawdą dla każdej interpretacji spełniającej S. Jednak, ponieważ systemy predykatów mogą mieć potencjalnie nieskończoną liczbę możliwych interpretacji, jest to rzadko praktyczne jest wypróbowanie wszystkich interpretacji. Zamiast tego reguły wnioskowania zapewniają obliczeniowo wykonalny sposób ustalenia, kiedy wyrażenie, składnik interpretacji, logicznie podąża za tą interpretacją. Pojęcie "logicznie podąża" stanowi formalną podstawę dla dowodów prawidłowości i poprawności reguł wnioskowania. Reguła wnioskowania jest w istocie mechanicznym środkiem do tworzenia nowych zdań predykatowych z innych zdań. Oznacza to, że reguły wnioskowania tworzą nowe zdania na podstawie składniowej formy podanych twierdzeń logicznych. Gdy każde zdanie X wytworzone przez regułę wnioskowania działającą na zbiorze S wyrażeń logicznych logicznie wynika z S, o reguły wnioskowania mówi się, że jest solidna. Jeśli reguła wnioskowania jest w stanie wygenerować każde wyrażenie logicznie wynikające z S, to mówi się, że jest kompletne. Modus ponens, który zostanie wprowadzony poniżej, oraz rozdzielczość, wprowadzona w rozdziale 11, są przykładami zasad wnioskowania, które są rozsądne i, jeśli są stosowane z odpowiednimi strategiami aplikacji, kompletne. Logiczne systemy wnioskowania na ogół używają rozsądnych reguł wnioskowania, chociaż

późniejsze części badają rozumowanie heurystyczne i rozumowanie zdrowego rozsądku, które rozluźniają ten wymóg. Formalizujemy te pomysły za pomocą następujących definicji.

DEFINICJA

ZADOWOLONY, MODELOWY, WAŻNY, NIEZGODNY

W przypadku rachunku predykatu wyrażenie X i interpretacja I:

Jeśli X ma wartość T w ramach I i szczególne przypisanie zmiennej, to mówi się, że spełniam X.

Jeśli spełniam X dla wszystkich przypisań zmiennych, to jestem modelem X.

X jest zadowalające tylko wtedy, gdy istnieje interpretacja i przypisanie zmiennych, które go spełniają; w przeciwnym razie jest niezadowalający.

Zestaw wyrażeń jest zadowalający tylko wtedy, gdy istnieje interpretacja i przypisanie zmiennych, które spełniają każdy element.

Jeśli zestaw wyrażeń nie jest zadowalający, mówi się, że jest niespójny.

Jeśli X ma wartość T dla wszystkich możliwych interpretacji, mówi się, że X jest poprawny.

W przykładzie świata bloków z rysunku powyżej świat bloków był modelem jego logicznego opisu. Wszystkie zdania w tym przykładzie były zgodne z tą interpretacją. Gdy baza wiedzy jest implementowana jako zbiór prawdziwych stwierdzeń na temat problematycznej domeny, domena ta jest modelem dla bazy wiedzy. Wyrażenie $\exists X (p(X) \wedge \neg p(X))$ jest niespójne, ponieważ nie może być spełnione przy żadnej interpretacji lub przypisaniu zmiennej. Z drugiej strony prawidłowe jest wyrażenie $\forall X (p(X) \vee \neg p(X))$.

Metodę tabeli prawdy można wykorzystać do przetestowania poprawności dowolnego wyrażenia niezawierającego zmiennych. Ponieważ nie zawsze jest możliwe ustalenie ważności wyrażeń zawierających zmienne (jak wspomniano powyżej, proces może się nie zakończyć), pełny rachunek predykatów jest "nierozstrzygalny". Istnieją jednak procedury dowodowe, które mogą wygenerować dowolne wyrażenie, które logicznie następuje z zestawu wyrażeń. Są to tak zwane procedury pełnego dowodu.

DEFINICJA

PROCEDURA DOWODOWA

Procedura sprawdzająca jest kombinacją reguły wnioskowania i algorytmu do zastosowania tej reguły do zestawu wyrażeń logicznych w celu wygenerowania nowych zdań. Później przedstawiamy procedury sprawdzające regułę wnioskowania o rozstrzygnięciu. Korzystając z tych definicji, możemy formalnie zdefiniować "logicznie podąża".

DEFINICJA

LOGICZNE PODĄŻANIE, BRZMIENIE I KOMPLET

Wyrażenie rachunku predykatów X logicznie wynika ze zbioru S wyrażeń rachunku predykatów, jeżeli każda interpretacja i przypisanie zmiennych, które spełnia S, również spełnia X. Reguła wnioskowania jest skuteczna, jeśli każde wyrażenie rachunku predykatu wygenerowane przez regułę ze zbioru S wyrażeń rachunku predykatu również logicznie wynika ze S. Reguła wnioskowania jest kompletna, jeśli biorąc pod uwagę zbiór S wyrażeń rachunku predykatu, reguła może wywnioskować każde wyrażenie, które logicznie wynika z S. Modus ponens jest regułą wnioskowania dźwiękowego. Jeśli otrzymamy

wyrażenie formy P → Q i inne wyrażenie formy P, takie, że oba są prawdziwe zgodnie z interpretacją I, wówczas modus ponens pozwala nam wnioskować, że Q jest również prawdziwe dla tej interpretacji. Rzeczywiście, ponieważ modus ponens jest dźwiękiem, Q jest prawdziwe dla wszystkich interpretacji, dla których P i P → Q są prawdziwe. Modus ponens i szereg innych przydatnych reguł wnioskowania określono poniżej.

DEFINICJA

MODUS PONENS, MODUS TOLLENS I ELIMINACJA ORAZ WPROWADZENIE I UNIWERSALNA INSTALACJA

Jeśli wiadomo, że zdania P i P →Q są prawdziwe, to modus ponens pozwala wywnioskować Q. Zgodnie z regułą wnioskowania modus tollens, jeśli P → Q jest znane jako prawda, a Q wiadomo, że jest fałszem, możemy wywnioskować, że P jest fałszem: ¬P. A eliminacja pozwala nam wywnioskować prawdę jednego z koniunkcji z prawdy zdania łączącego. Na przykład P ∧ Q pozwala stwierdzić, że P i Q są prawdziwe. A wprowadzenie pozwala nam wywnioskować prawdę koniunkcji z prawdy jej koniunkcji. Na przykład, jeśli P i Q są prawdziwe, to P ∧ Q jest prawdziwe. Instancja uniwersalna stwierdza, że jeśli dowolna uniwersalnie skwantyfikowana zmienna w prawdziwym zdaniu, powiedzmy p(X), zostanie zastąpiona odpowiednim terminem z dziedziny, wynikiem jest prawdziwe zdanie. Zatem, jeśli a pochodzi z domeny X, ∀ X p(X) pozwala wywnioskować p(a). Jako prosty przykład zastosowania modus ponens w rachunku zdań, załóżmy następujące spostrzeżenia: "jeśli pada deszcz, wówczas ziemia jest mokra" i "pada deszcz". Jeśli P oznacza "pada deszcz", a Q oznacza " ziemia jest mokra ", wówczas pierwsze wyrażenie zmienia się w P → Q. Ponieważ rzeczywiście teraz pada (P to prawda), nasz zestaw aksjomatów staje się P → QP Dzięki zastosowaniu modus ponens fakt, że ziemia jest mokra (Q) można dodać do zestawu prawdziwych wyrażeń. Modus ponens można również stosować do wyrażeń zawierających zmienne. Rozważmy jako przykład powszechny sylogizm: "wszyscy ludzie są śmiertelni, a Sokrates jest mężczyzną; dlatego Sokrates jest śmiertelny. "" Wszyscy ludzie są śmiertelni "mogą być reprezentowani w rachunku predykatu przez: ∀X (człowiek (X) → śmiertelnik (X)).

"Sokrates jest mężczyzną" to człowiek (Sokrates). Ponieważ X w implikacji jest powszechnie kwantyfikowane, możemy podstawić dowolną wartość w domenę dla X i nadal mają prawdziwe stwierdzenie zgodnie z regułą wnioskowania o uniwersalnej instancji. Zastępując X w implikacji Sokratesa, wnioskujemy o wyrażeniu człowiek (Sokrates) → śmiertelny (Sokrates).

Możemy teraz zastosować modus ponens i wnioskować, że śmiertelnik (Sokrates). Jest to dodawane do zestawu wyrażeń, które logicznie wynikają z pierwotnych asercji. Algorytm zwany unifikacją może zostać wykorzystany przez zautomatyzowane narzędzie do rozwiązywania problemów w celu ustalenia, że Sokrates może zastąpić X w celu zastosowania modus ponens.

Unifikacja

Aby zastosować reguły wnioskowania, takie jak modus ponens, system wnioskowania musi być w stanie określić, kiedy dwa wyrażenia są takie same lub pasują. W rachunku zdań jest to trywialne: dwa wyrażenia pasują tylko wtedy, gdy są identyczne pod względem składniowym. W rachunku predykatów proces dopasowywania dwóch zdań jest skomplikowany przez istnienie zmiennych w wyrażeniach. Uniwersalne tworzenie instancji pozwala na zastąpienie uniwersalnych zmiennych ilościowych terminami z dziedziny. Wymaga to procesu decyzyjnego w celu ustalenia podstawień zmiennych, w ramach których dwa lub więcej wyrażeń może być identycznych (zwykle w celu zastosowania reguł wnioskowania).

Ujednolicenie jest algorytmem służącym do określania podstawień potrzebnych do dopasowania dwóch wyrażeń rachunku predykatu. Widzieliśmy to już w poprzednim podrozdziale, w którym

Sokrates u człowieka (Sokrates) zastąpiono X w ∀ X(człowiek(X) ⇒ śmiertelny(X)). To pozwoliło na zastosowanie modus ponens i wniosek śmiertelnika (Sokrates). Inny przykład unifikacji zaobserwowano wcześniej, gdy omawiano zmienne fikcyjne. Ponieważ p(X) i p(Y) są równoważne, Y może zastąpić X, aby dopasować zdania. Reguły unifikacji i wnioskowania, takie jak modus ponens, pozwalają nam wnioskować na podstawie zestawu logicznych twierdzeń. Aby to zrobić, logiczna baza danych musi być wyrażona w odpowiedniej formie. Zasadniczym aspektem tej formy jest wymaganie, aby wszystkie zmienne były uniwersalne i skwantyfikowane. Pozwala to na pełną swobodę w zastępowaniu obliczeń. Zmienne ilościowe można wyeliminować ze zdań w bazie danych, zastępując je stałymi, dzięki którym zdanie jest prawdziwe. Na przykład parent X rodzic (X, tom) może zostać zastąpiony przez wyrażenie rodzic (Bob, Tom) lub rodzic (Mary, Tom), zakładając, że Bob i Mary są rodzicami Toma w interpretacji. Proces eliminowania egzystencjalnie skwantyfikowanych zmiennych jest skomplikowany przez fakt, że wartość tych podstawień może zależeć od wartości innych zmiennych w wyrażeniu. Na przykład w wyrażeniu ∀ X ∃ Matka Y (X, Y) wartość egzystencjalnie skwantyfikowanej zmiennej Y zależy od wartości X. Skolemizacja zastępuje każdą egzystencjalnie skwantyfikowaną zmienną funkcją, która zwraca odpowiednią stałą w funkcji niektóre lub wszystkie inne zmienne w zdaniu. W powyższym przykładzie, ponieważ wartość Y zależy od X, Y można zastąpić funkcją skolem, f, z X. Daje to predykat mother X matkę (X, f (X)). Skolemizacja, proces, który może również wiązać zmienne o wartościach uniwersalnych ze stałymi. Po usunięciu istniejących ilościowo zmiennych z logicznej bazy danych można zastosować unifikację do dopasowania zdań w celu zastosowania reguł wnioskowania, takich jak modus ponens. Ujednolicenie komplikuje fakt, że zmienną można zastąpić dowolnym terminem, w tym innymi zmiennymi i wyrażeniami funkcji o dowolnej złożoności. Te wyrażenia mogą same zawierać zmienne. Na przykład ojciec (walet) może zastąpić X u mężczyzny (X), aby wywnioskować, że ojciec waletera jest śmiertelny. Niektóre wystąpienia wyrażenia foo (X, a, goo (Y)). generowane przez prawne zastępstwa podano poniżej:
1) foo (fred, a, goo (Z))

2) foo (W, a, goo (jack))

3) foo (Z, a, goo (moo (Z)))

W tym przykładzie wystąpienia lub ujednolicenia podstawienia, które spowodowałyby, że początkowe wyrażenie byłoby identyczne z każdym z pozostałych trzech, zapisano jako zestawy:

1) {fred / X, Z / Y}

2) {W / X, jack / Y}

3) {Z / X, moo (Z) / Y}

Notacja X / Y,... wskazuje, że X zastępuje zmienną Y w pierwotnym wyrażeniu. Podstawienia są również nazywane wiązaniami. Mówi się, że zmienna jest związana z podstawioną wartością. Przy definiowaniu algorytmu unifikacji, który oblicza podstawienia wymagane do dopasowania dwóch wyrażeń, należy wziąć pod uwagę szereg kwestii. Po pierwsze, chociaż stała może być systematycznie zastępowana zmienną, każda stała jest uważana za "instancję podstawową" i nie może być zastąpiona. Ani jednej zmiennej nie można zastąpić dwoma różnymi instancjami naziemnymi. Po drugie, zmiennej nie można ujednolicić z terminem zawierającym tę zmienną. X nie może być zastąpione przez p (X), ponieważ tworzy to nieskończone wyrażenie: p (p (p (p (p (… X) …))). Test na tę sytuację nazywa się sprawdzaniem zachodzącym. Ponadto proces rozwiązywania problemów często wymaga wielu wnioskowania, a w konsekwencji wielu kolejnych unifikacji. Rozwiązania problemów logicznych muszą zachować spójność podstawień zmiennych. Ważne jest, aby wszelkie podstawienia ujednolicające były konsekwentnie wykonywane we wszystkich wystąpieniach w zakresie zmiennej w dopasowywanych obu wyrażeniach. Było to widoczne wcześniej, gdy Sokrates zastąpiono nie tylko zmienną X u człowieka (X), ale także zmienną X u śmiertelnika (X). Po związaniu zmiennej przyszłe ujednolicenia i wnioski muszą

uwzględniać wartość tego wiązania. Jeśli zmienna jest powiązana ze stałą, zmienna ta może nie otrzymać nowego wiązania w przyszłej unifikacji. Jeśli zmienna X1 zostanie zastąpiona inną zmienną X2, a później X1 zostanie zastąpiona stałą, wówczas X2 musi również odzwierciedlać to wiązanie. Zestaw podstawień zastosowany w sekwencji wnioskowania jest ważny, ponieważ może zawierać odpowiedź na pierwotne zapytanie. Na przykład, jeśli p (a, X) łączy się z założeniem p (Y, Z) ⇒ q (Y, Z) z podstawieniem {a / Y, X / Z}, modus ponens pozwala nam wnioskować q (a, X) w ramach tego samego zastąpienia. Jeśli dopasujemy ten wynik do założenia q (W, b) ⇒ r (W, b), wnioskujemy r (a, b) w ramach zestawu podstawień {a / W, b / X}. Inną ważną koncepcją jest skład podstawień unifikacyjnych. Jeśli S i S 'są dwoma zestawami podstawień, wówczas skład S i S' (zapisane SS ') uzyskuje się przez zastosowanie S' do elementów S i dodanie wyniku do S. Rozważ przykład komponowania następujących trzech zestawów podstawień:

{X / Y, W / Z}, {V / X}, {a / V, f (b) / W}.

Komponowanie trzeciego zestawu {a / V, f (b) / W} z drugim {V / X} daje:

{a / X, a / V, f (b) / W}.

Skomponowanie tego wyniku z pierwszym zestawem {X / Y, W / Z} daje zestaw podstawień:

{a / Y, a / X, a / V, f (b) / Z, f (b) / W}.

Kompozycja to metoda łączenia podstawień unifikacyjnych i zwracania ich w funkcji rekurencyjnej unify, przedstawiona poniżej. Kompozycja jest asocjacyjna, ale nie przemienna. Ćwiczenia przedstawiają te kwestie bardziej szczegółowo. Kolejnym wymogiem algorytmu unifikacji jest to, aby unifikator był jak najbardziej ogólny: aby znaleźć najbardziej ogólny unifikator. Jest to ważne, jak zobaczymy w następnym przykładzie, ponieważ jeśli ogólność zostanie utracona w procesie rozwiązania, może zmniejszyć zakres ostatecznego rozwiązania lub nawet całkowicie wyeliminować możliwość rozwiązania. Na przykład w ujednolicaniu p(X) i p(Y) będzie działać dowolne wyrażenie stałe, takie jak {fred / X, fred / Y}. Jednak fred nie jest najbardziej ogólnym unifikatorem; każda zmienna dałaby bardziej ogólne wyrażenie: {Z / X, Z / Y}. Rozwiązania uzyskane z pierwszej instancji podstawienia byłyby zawsze ograniczone przez stałe ograniczenie prędkości wynikowych wniosków; tj. fred byłby unifikatorem, ale zmniejszyłby ogólność wyniku

DEFINICJA

NAJBARDZIEJ OGÓLNY UNIFIKATOR (mgu)

Jeśli s jest dowolnym unifikatorem wyrażeń E, a g jest najbardziej ogólnym unifikatorem tego zestawu wyrażeń, to dla s zastosowanego do E istnieje inny unifikator s ? taki, że Es = Egs', gdzie Es i Egs' są kompozycją unifikatory zastosowane do wyrażenia E. Najbardziej ogólny unifikator dla zestawu wyrażeń jest unikalny, z wyjątkiem odmian alfabetycznych; tj. to, czy zmienna zostanie ostatecznie nazwana X, czy Y, tak naprawdę nie ma żadnego wpływu na ogólność wynikowych unifikacji. Ujednolicenie jest ważne dla każdego rozwiązania problemu sztucznej inteligencji, który używa rachunku predykatu do reprezentacji. Ujednolicenie określa warunki, w których można powiedzieć, że dwa (lub więcej) wyrażenia rachunku predykatu są równoważne. Pozwala to na stosowanie reguł wnioskowania, takich jak rozwiązywanie, z reprezentacjami logicznymi, co często wymaga cofnięcia w celu znalezienia wszystkich możliwych interpretacji. Następnie przedstawiamy pseudo-kod dla funkcji unify, która może obliczać podstawienia unifikujące (jeśli jest to możliwe) między dwoma wyrażeniami rachunku predykatu. Unify przyjmuje jako argumenty dwa wyrażenia w rachunku predykatów i zwraca albo najbardziej ogólny zestaw podstawień unifikujących, albo stały FAIL, jeśli żadna unifikacja nie jest możliwa. Jest zdefiniowany jako funkcja rekurencyjna: po pierwsze, rekurencyjnie próbuje ujednolicić

początkowe komponenty wyrażeń. Jeśli to się powiedzie, wszelkie podstawienia zwrócone przez tę unifikację zostaną zastosowane do reszty obu wyrażeń. Są one następnie przekazywane w drugim rekurencyjnym wezwaniu do unifikacji, które próbuje zakończyć unifikację. Rekurencja kończy się, gdy dowolny argument jest symbolem (predykatem, nazwą funkcji, stałą lub zmienną) lub gdy wszystkie elementy wyrażenia zostały dopasowane. Aby uprościć manipulację wyrażeniami, algorytm zakłada nieco zmodyfikowaną składnię. Ponieważ unify po prostu wykonuje składniowe dopasowanie wzorca, może skutecznie ignorować rozróżnienie rachunku predykatów między predykatami, funkcjami i argumentami. Reprezentując wyrażenie jako listę (uporządkowaną sekwencję elementów) z nazwą predykatu lub funkcji jako pierwszym elementem, a następnie jego argumentami, upraszczamy manipulację wyrażeniami. Wyrażenia, w których sam argument jest predykatem lub wyrażeniem funkcyjnym, są reprezentowane jako listy na liście, zachowując w ten sposób strukturę wyrażenia. Listy są rozdzielane nawiasami (), a elementy listy są oddzielane spacjami. Przykłady wyrażeń zarówno w rachunku predykatów, na PC, jak i na liście:

PC SYNTAX	LIST SYNTAX
p(a,b)	(p a b)
p(f(a),g(X,Y))	(p (f a) (g X Y))
equal(eve,mother(cain))	(equal eve (mother cain))

We next present the function unify:

```
function unify(E1, E2);
  begin
    case
      both E1 and E2 are constants or the empty list:          %recursion stops
        if E1 = E2 then return {}
          else return FAIL;
      E1 is a variable:
        if E1 occurs in E2 then return FAIL
          else return (E2/E1);
      E2 is a variable:
        if E2 occurs in E1 then return FAIL
          else return (E1/E2)
      either E1 or E2 are empty then return FAIL     %the lists are of different sizes
      otherwise:                                          %both E1 and E2 are lists
        begin
            HE1 := first element of E1;
            HE2 := first element of E2;
            SUBS1 := unify(HE1,HE2);
            if SUBS1 : = FAIL then return FAIL;
            TE1 := apply(SUBS1, rest of E1);
            TE2 : = apply (SUBS1, rest of E2);
            SUBS2 : = unify(TE1, TE2);
            if SUBS2 = FAIL then return FAIL;
                else return composition(SUBS1,SUBS2)
        end
    end                                                          %end case
  end
```

Przykład unifikacji

Zachowanie powyższego algorytmu można wyjaśnić, śledząc wywołanie unify ((rodzice X (ojciec X) (rachunek matki)), (rachunek rodziców (rachunek ojca) Y)). Kiedy unify jest wywoływane po raz pierwszy, ponieważ żaden argument nie jest symbolem atomowym, funkcja spróbuje rekurencyjnie ujednolicić pierwsze elementy każdego wyrażenia, wywołując

unify(rodzice, rodzice).

To zjednoczenie się powiodło, zwracając puste podstawienie {}. Zastosowanie tego do pozostałych wyrażeń nie powoduje żadnych zmian; algorytm następnie wywołuje

unify ((X (ojciec X) (rachunek matki)), (rachunek (rachunek ojca) Y)).

W drugim wywołaniu unifikacji żadne wyrażenie nie jest atomowe, więc algorytm dzieli każde wyrażenie na pierwszy składnik i pozostałą część wyrażenia. To prowadzi do połączenia

unify (X, rachunek).

Wywołanie to się powiedzie, ponieważ oba wyrażenia są atomowe, a jedno z nich jest zmienną. Wywołanie zwraca podstawienie {bill / X}. Podstawienie to stosuje się do reszty każdego wyrażenia, a wyniki unify są wywoływane w wynikach:

unify (((rachunek ojca) (rachunek matki)), ((rachunek ojca) Y)).

Wynikiem tego połączenia jest ujednolicenie (rachunek ojca) z (rachunek ojca). To prowadzi do połączeń

unify (ojciec, ojciec)

unify (rachunek, rachunek)

unify ((), ())

Wszystko to się udaje, zwracając pusty zestaw podstawień. Następnie wywoływana jest funkcja Unify dla pozostałych wyrażeń:

unify (((rachunek główny)), (Y)).

To z kolei prowadzi do połączeń

unify ((rachunek główny), Y)

unify ((), ()).

W pierwszym z nich (rachunek macierzysty) łączy się z Y. Zwróć uwagę, że unifikacja zastępuje całą strukturę (rachunek macierzysty) zmienną Y. Tak więc unifikacja kończy się powodzeniem i zwraca podstawienie {(rachunek główny) / Y}. Wywołanie

unify ((), ())

zwraca { }. Wszystkie podstawienia są tworzone po zakończeniu każdego połączenia rekurencyjnego, aby zwrócić odpowiedź {bill / X (mother bill) / Y}. Każde połączenie jest ponumerowane, aby wskazać kolejność, w jakiej zostało wykonane; podstawienia zwracane przez każde wywołanie są odnotowywane na łukach drzewa.

Aplikacja: Doradca finansowy oparty na logice

Jako ostatni przykład wykorzystania rachunku predykatów do reprezentowania i uzasadniania domen problemowych, projektujemy doradcę finansowego przy użyciu rachunku predykatu. Chociaż jest to prosty przykład, ilustruje on wiele problemów związanych z realistycznymi aplikacjami. Zadaniem doradcy jest pomoc użytkownikowi w podjęciu decyzji, czy zainwestować w rachunek oszczędnościowy czy giełdę. Niektórzy inwestorzy mogą chcieć podzielić swoje pieniądze między nimi. Inwestycja, która będzie zalecana inwestorom indywidualnym, zależy od ich dochodów i bieżącej kwoty, którą zaoszczędzili, zgodnie z następującymi kryteriami:

1. Osoby z nieodpowiednim kontem oszczędnościowym powinny zawsze zwiększać zaoszczędzoną kwotę na pierwszym miejscu, niezależnie od swoich dochodów.

2. Osoby posiadające odpowiedni rachunek oszczędnościowy i odpowiedni dochód powinny rozważyć bardziej ryzykowną, ale potencjalnie bardziej opłacalną inwestycję na giełdzie.

3. Osoby o niższych dochodach, które mają już odpowiedni rachunek oszczędnościowy, mogą rozważyć podzielenie nadwyżki dochodu na oszczędności i zapasy, aby zwiększyć poduszkę oszczędności, próbując jednocześnie zwiększyć swoje dochody poprzez zapasy.

Adekwatność zarówno oszczędności, jak i dochodu zależy od liczby osób na utrzymaniu, które jednostka musi wspierać. Naszą zasadą jest co najmniej 5000 USD oszczędności na każdego członka rodziny. Odpowiedni dochód musi być stałym dochodem i zapewniać co najmniej 15 000 USD rocznie plus dodatkowe 4000 USD na każdego członka rodziny. Aby zautomatyzować tę poradę, tłumaczymy te wytyczne na zdania w rachunku predykatu. Pierwszym zadaniem jest określenie głównych cech, które należy wziąć pod uwagę. Tutaj są adekwatnością oszczędności i dochodu. Są one reprezentowane odpowiednio przez rachunek oszczędnościowy i dochód. Oba są jednoznacznymi predykatami, a ich argument może być odpowiedni lub nieodpowiedni. A zatem,

savings_account(adequate).

savings_account(inadequate).

income(adequate).

income(inadequate).

są ich możliwymi wartościami.

Wnioski są reprezentowane przez jednostkową inwestycję predykatową, przy czym możliwymi wartościami tego argumentu są zapasy, oszczędności lub połączenie (co oznacza, że inwestycja powinna zostać podzielona). Stosując te predykaty, różne strategie inwestycyjne są reprezentowane przez implikacje. Pierwsza zasada, że osoby z niewystarczającymi oszczędnościami powinny poczynić do wzrostu oszczędności jest ich głównym priorytetem, jest reprezentowany przez

savings_account(inadequate) → investment(savings)

Podobnie pozostałe dwie możliwe alternatywy inwestycyjne są reprezentowane przez

savings_account(adequate) ∧ income(adequate) → investment(stocks).

savings_account(adequate) ∧ income(inadequate) → investment(combination).

Następnie doradca musi ustalić, kiedy oszczędności i dochód są odpowiednie lub niewystarczające. Zostanie to również wykonane przy użyciu implikacji. Konieczność wykonywania obliczeń arytmetycznych wymaga użycia funkcji. Aby określić minimalne odpowiednie oszczędności, zdefiniowano funkcję oszczędzania min. minsavings pobiera jeden argument, liczbę osób zależnych i zwraca 5000 razy ten argument. Stosując oszczędności minimalne, adekwatność oszczędności zależy od zasad

∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧ greater(X, minsavings(Y)))

→ savings_account(adequate).

∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧ ¬ greater(X, minsavings(Y)))

→ savings_account(inadequate).

gdzie minsavings (X) ≡ 5000 ? X.

W tych definicjach amount_saved (X) i osoby na dependents(Y) potwierdzają bieżącą kwotę oszczędności i liczbę osób na utrzymaniu inwestora; greater(X, Y) jest standardowym testem

arytmetycznym dla jednej liczby większej od drugiej i nie jest formalnie zdefiniowana w tym przykładzie. Podobnie, funkcja minincome jest zdefiniowana jako

minincome (X) ≡ 15000 + (4000 ? X).

minincome służy do obliczenia minimalnego odpowiedniego dochodu, biorąc pod uwagę liczbę osób na utrzymaniu. Bieżące dochody inwestora reprezentowane są przez predykat, zyski. Ponieważ odpowiedni dochód musi być zarówno stały, jak i powyżej minimum, zarobki wymagają dwóch argumentów: pierwszy to kwota zarobiona, a drugi musi być równy albo stały, albo niestały. Pozostałe zasady potrzebne doradcy to:

∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧ greater(X, minincome(Y))) → income(adequate).

∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧ ¬ greater(X, minincome(Y))) → income(inadequate).

∀ X earnings(X, unsteady) → income(inadequate).

W celu przeprowadzenia konsultacji do tego zestawu zdań rachunku predykatu dodawany jest opis konkretnego inwestora z wykorzystaniem predykatów kwota_zapisana, zarobki i osoby na utrzymaniu. Tak więc osoba z trzema osobami na utrzymaniu, o oszczędności 22 000 USD i stałym dochodzie w wysokości 25 000 USD, opisałaby

amount_saved(22000)

earnings(25000, steady).

dependents(3)

To daje logiczny system składający się z następujących zdań:

1. savings_account(inadequate) → investment(savings).

2. savings_account(adequate) ∧ income(adequate) → investment(stocks).

3. savings_account(adequate) ∧ income(inadequate)

→ investment(combination).

4. ∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧

greater(X, minsavings(Y))) → savings_account(adequate).

5. ∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧

¬ greater(X, minsavings(Y))) → savings_account(inadequate).

6. ∀ X earnings(X, steady) ∧ ∃ Y (dependents (Y) ∧

greater(X, minincome(Y))) → income(adequate).

7. ∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧

¬ greater(X, minincome(Y))) → income(inadequate).

8. ∀ X earnings(X, unsteady) → income(inadequate).

9. amount_saved(22000).

10. earnings(25000, steady).

11. dependents(3).

gdzie minsavings (X) ≡ 5000 * X i minincome (X) ≡ 15000 + (4000 * X). Ten zestaw zdań logicznych opisuje domenę problemu. Asercje są ponumerowane, dzięki czemu można się do nich odwoływać w następującym śladzie. Używając unifikacji i modus ponens, właściwą strategię inwestycyjną dla tej osoby można wywnioskować jako logiczną konsekwencję tych opisów. Pierwszym krokiem byłoby ujednolicenie połączenia 10 i 11 z pierwszymi dwoma składnikami założenia 7; to znaczy.,

earnings(25000,steady) ∧ dependents(3)

unifies with

earnings(X,steady) ∧ dependents(Y)

pod podstawieniem {25000 / X, 3 / Y}. Ta zamiana daje nową implikację:

earnings(25000, steady) ∧ dependents(3) ∧ ¬ greater(25000, minincome(3))

→ income(inadequate).

Ocena funkcji minincome daje wyrażenie

earnings(25000, steady) ∧ dependents(3) ∧ ¬ greater(25000, 27000)

→ income(inadequate).

Ponieważ wszystkie trzy składniki przesłanki są indywidualnie prawdziwe, o 10, 3, a matematyczna definicja większej, ich połączenie jest prawdziwe, a cała przesłanka jest prawdziwa. Można zatem zastosować modus ponens, co da dochód z wniosku (nieodpowiedni). Jest to dodane jako twierdzenie 12.

12. income(inadequate).

Podobnie,

amount_saved(22000) ∧ dependents(3)

łączy się z pierwszymi dwoma elementami założenia asercji 4 pod podstawieniem {22000 / X, 3 / Y}, dając implikację

amount_saved(22000) ∧ dependents(3) ∧ greater(22000, minsavings(3))

→ savings_account(adequate).

Tutaj ocena funkcji minsavings (3) daje wyrażenie

amount_saved(22000) ∧ dependents(3) ∧ greater(22000, 15000)

→ savings_account(adequate)

Ponownie, ponieważ wszystkie składniki przesłanki tej implikacji są prawdziwe, cała przesłanka ocenia się na prawdę, a modus ponens można ponownie zastosować, uzyskując wniosek o oszczędności_konta (odpowiedni), który dodaje się jako wyrażenie 13.

13. savings_account(adequate).

Jak wskazuje analiza wyrażeń 3, 12 i 13, przesłanka implikacji 3 jest również prawdziwa. Kiedy ponownie zastosujemy modus ponens, wnioskiem jest inwestycja (połączenie), sugerowana inwestycja

dla naszej osoby. Ten przykład ilustruje, w jaki sposób można zastosować rachunek predykatu do uzasadnienia realistycznego problemu, wyciągając prawidłowe wnioski, stosując reguły wnioskowania do wstępnego opisu problemu. Nie dyskutowaliśmy dokładnie, w jaki sposób algorytm może określić prawidłowe wnioski, aby rozwiązać dany problem, ani sposób, w jaki można to zaimplementować na komputerze.

Epilog

Wprowadziliśmy rachunek predykatów jako język reprezentacyjny do rozwiązywania problemów AI. Symbole, terminy, wyrażenia i semantyka języka zostały opisane i zdefiniowane. W oparciu o semantykę rachunku predykatów zdefiniowaliśmy reguły wnioskowania, które pozwalają nam wyprowadzać zdania, które logicznie wynikają z danego zestawu wyrażeń. Zdefiniowaliśmy algorytm unifikacji, który określa podstawienia zmiennych, które dopasowują dwa wyrażenia, co jest niezbędne do zastosowania reguł wnioskowania. Zakończyliśmy ten rozdział przykładem doradcy finansowego, który reprezentuje wiedzę finansową za pomocą rachunku predykatu i wykazuje logiczne wnioskowanie jako technikę rozwiązywania problemów.

Sztuczna inteligencja : struktury i strategie wyszukiwania w przestrzeni stanów

(III / XVI)

ĆWICZENIA

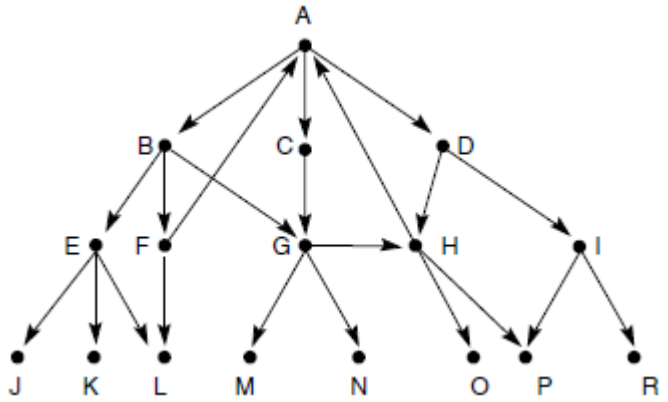
1. Ścieżka hamiltonowska to ścieżka, która wykorzystuje każdy węzeł wykresu dokładnie raz. Jakie warunki są konieczne do istnienia takiej ścieżki? Czy istnieje taka ścieżka na mapie Królewca?

2. Podaj graficzną reprezentację problemu dla farmera, wilka, kozy i kapusty: farmer ze swoim wilkiem, kozą i kapustą zbliża się do brzegu rzeki, którą chce przepłynąć. Na brzegu rzeki znajduje się łódź, ale oczywiście tylko rolnik może wiosłować. Łódź może jednocześnie przenosić tylko dwie rzeczy (w tym wioślarza). Jeśli wilk zostanie kiedykolwiek sam z kozą, wilk zje kozę; podobnie, jeśli koza zostanie sama z kapustą, koza zje kapustę. Opracuj sekwencję przepraw przez rzekę, aby wszystkie cztery postacie bezpiecznie dotarły na drugą stronę rzeki. Niech węzły reprezentują stany świata; np. farmer i koza są na zachodnim brzegu, a wilk i kapusta na wschodzie. Omów zalety pierwszego i pełnego wyszukiwania tego problemu.

3. Zbuduj akceptor stanu skończonego, który rozpoznaje wszystkie ciągi cyfr binarnych a), które zawierają "111", b), które kończą się na "111", c), które zawierają "111", ale nie więcej niż trzy kolejne "1".

4. Podaj przykład problemu sprzedawcy podróży, dla którego strategia najbliższego sąsiada nie znajduje optymalnej ścieżki. Zaproponuj inną heurystykę dla tego problemu.

5. "Uruchom ręcznie" algorytm cofania na wykresie na rysunku



Rozpocznij od stanu A. Śledź kolejne wartości NSL, SL, CS itp.

6. Wdróż algorytm cofania w wybranym języku programowania.

7. Ustal, czy wyszukiwanie ukierunkowane na cel czy na podstawie danych byłoby lepsze do rozwiązania każdego z poniższych problemów. Uzasadnij swoją odpowiedź.

a. Diagnozowanie problemów mechanicznych w samochodzie.

b. Poznałeś osobę, która twierdzi, że jest twoim dalekim kuzynem, ze wspólnym przodkiem o imieniu John Doe. Chcesz zweryfikować jej roszczenie.

c. Inna osoba twierdzi, że jest twoim dalekim kuzynem. Nie zna imienia wspólnego przodka, ale wie, że było to nie więcej niż osiem pokoleń wstecz. Chcesz znaleźć tego przodka lub ustalić, że on nie istniał.

d. Dowiedzenie twierdzenia o geometrii płaskiej.

e. Program do badania odczytów sonaru i ich interpretacji, na przykład mówienie dużej łodzi podwodnej od małej łodzi podwodnej od wieloryba ze szkoły ryb.

f. System ekspercki, który pomoże człowiekowi sklasyfikować rośliny według gatunku, rodzaju itp.

8. Wybierz i uzasadnij wybór pierwszego lub drugiego zakresu dla przykładów ćwiczenia 6.

9. Napisz algorytm cofania dla i / lub wykresów.

10. Prześledzić ukierunkowany na cel problem z psem z przykładu 3.3.4 w sposób oparty na danych.

11. Podaj kolejny przykład problemu z grafem i / lub wykresu i opracuj część przestrzeni wyszukiwania.

12. Śledź oparte na danych wykonanie doradcy finansowego z przykładu 3.3.5 w przypadku osoby posiadającej cztery osoby na utrzymaniu, 18 000 USD w banku i stały dochód w wysokości 25 000 USD rocznie. Na podstawie porównania tego problemu i przykładu w tekście zasugeruj: ogólnie "najlepsza" strategia rozwiązania problemu.

13. Dodaj reguły dla przymiotników i przysłówków do gramatyki angielskiej w przykładzie 3.3.6.

14. Dodaj reguły dla (wielu) wyrażeń przyimkowych do gramatyki angielskiej w Przykładzie 3.3.6.

15. Dodaj reguły gramatyczne do Przykładu 3.3.6, które dopuszczają złożone zdania, takie jak zdanie ? zdanie ORAZ zdanie.

Wprowadzenie

Wcześniej wprowadziliśmy rachunek predykatów jako przykład języka reprezentacji sztucznej inteligencji. Dobrze sformułowane wyrażenia rachunku predykatu umożliwiają opisywanie obiektów i relacji w dziedzinie problemów, a reguły wnioskowania, takie jak modus ponens, pozwalają wnioskować o nowej wiedzy z tych opisów. Wnioski te definiują przestrzeń, która jest przeszukiwana, aby znaleźć rozwiązanie problemu. Tutaj ,wprowadzimy teorię wyszukiwania w przestrzeni stanów. Aby pomyślnie zaprojektować i wdrożyć algorytmy wyszukiwania, programista musi być w stanie przeanalizować i przewidzieć swoje zachowanie. Pytania, na które należy odpowiedzieć, obejmują:

* Czy rozwiązanie problemu gwarantuje znalezienie rozwiązania?

* Czy rozwiązywanie problemów zawsze kończy się? Czy może zostać złapany w nieskończoną pętlę?

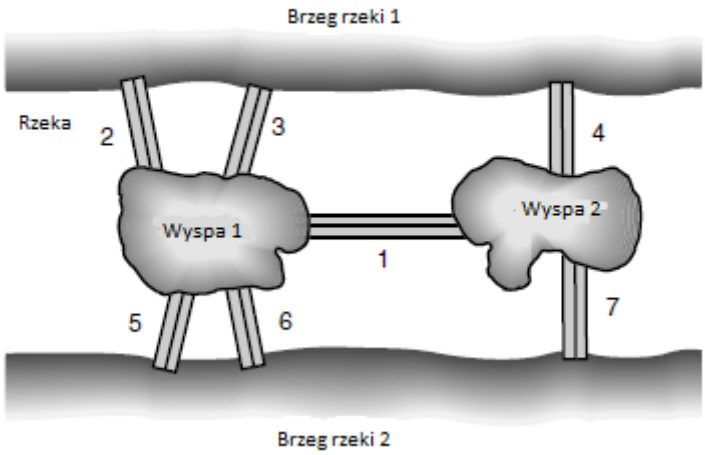
* Czy po znalezieniu rozwiązania jest ono optymalne?

* Jaka jest złożoność procesu wyszukiwania pod względem wykorzystania czasu? Zużycie pamięci?

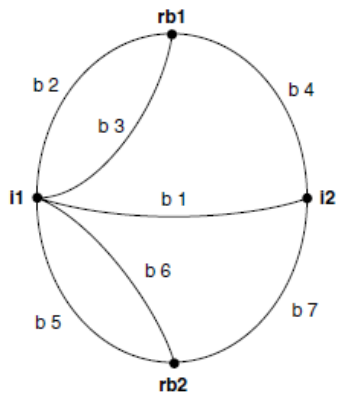
* W jaki sposób tłumacz może najskuteczniej zmniejszyć złożoność wyszukiwania?

* Jak można zaprojektować tłumacza, aby najskuteczniej wykorzystywać język reprezentacji?

Teoria przeszukiwania przestrzeni stanów jest naszym podstawowym narzędziem do odpowiedzi na te pytania. Reprezentując problem jako wykres przestrzeni stanu, możemy użyć teorii grafów do analizy struktury i złożoności problemu oraz procedur wyszukiwania, które stosujemy do jego rozwiązania. Wykres składa się z zestawu węzłów i zestawu łuków lub łączy łączących pary węzłów. W modelu rozwiązywania problemów w przestrzeni stanów węzły grafu są reprezentowane przez dyskretne stany w procesie rozwiązywania problemów, takie jak wyniki wnioskowania logicznego lub różne konfiguracje planszy. Łuki na wykresie reprezentują przejścia między stanami. Przejścia te odpowiadają logicznym wnioskom lub legalnym ruchom gry. Na przykład w systemach eksperckich stany opisują naszą wiedzę o wystąpieniu problemu na pewnym etapie procesu wnioskowania. Wiedza ekspercka w postaci reguły if … else, pozwala nam generować nowe informacje; akt zastosowania reguły jest przedstawiany jako łuk między stanami. Teoria grafów jest naszym najlepszym narzędziem do wnioskowania o strukturze obiektów i relacji; w rzeczy samej właśnie ta potrzeba doprowadziła do jej powstania na początku XVIII wieku. Szwajcarski matematyk Leonhard Euler wynalazł teorię grafów, aby rozwiązać „problem mostów Królewca”. Miasto Królewskie zajmowało zarówno brzegi, jak i dwie wyspy rzeki. Wyspy i brzegi rzeki były połączone siedmioma mostami, jak pokazano na rysunku



Problem mostów w Królewcu pyta, czy istnieje spacer wokół miasta, który przecina każdy most dokładnie raz. Chociaż mieszkańcy nie znaleźli takiego rozwiązania i wątpili, czy to możliwe, nikt nie udowodnił jego niemożliwości. Opracowując formę teorii grafów, Euler stworzył alternatywną reprezentację mapy, przedstawioną na rysunku



Brzegi rzek (rb1 i rb2) i wyspy (i1 i i2) są opisane przez węzły wykresu; mosty są reprezentowane przez oznaczone łuki między węzłami (b1, b2,, b7). Reprezentacja wykresu zachowuje zasadniczą strukturę systemu mostów, ignorując przy tym cechy zewnętrzne, takie jak długości mostów, odległości i kolejność mostów podczas marszu. Alternatywnie możemy reprezentować system mostów Królewca za pomocą rachunku predykatu. Predykat połączenia odpowiada łukowi wykresu, zapewniając, że dwie masy lądowe są połączone określonym mostem. Każdy most wymaga dwóch predykatów łączenia, po jednym dla każdego kierunku, w którym most może zostać przekroczony. Wyrażenie predykatu, connect (X, Y, Z) = connect (Y, X, Z), wskazujące, że dowolny most może zostać przekroczony w dowolnym kierunku, pozwoliłoby na usunięcie połowy następujących faktów połączenia:

connect (i1, i2, b1) connect (i2, i1, b1)

connect (rb1, i1, b2) connect (i1, rb1, b2)

connect (rb1, i1, b3) connect (i1, rb1, b3)

connect (rb1, i2, b4) connect (i2, rb1, b4)

connect (rb2, i1, b5) connect (i1, rb2, b5)

connect (rb2, i1, b6) connect (i1, rb2, b6)

connect (rb2, i2, b7) connect (i2, rb2, b7)

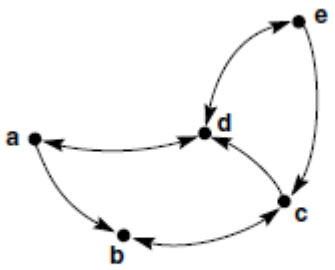
Reprezentacja rachunku predykatu jest równoważna reprezentacji wykresu, ponieważ zachowane jest połączenie. Rzeczywiście, algorytm mógłby tłumaczyć między tymi dwoma reprezentacjami bez utraty informacji. Jednak strukturę problemu można zwizualizować bardziej bezpośrednio w reprezentacji grafu, podczas gdy pozostaje on domyślny w wersji rachunku predykatu. Dowód Eulera ilustruje to rozróżnienie. Udowadniając, że spacer był niemożliwy, Euler skupił się na stopniu węzłów wykresu, obserwując, że węzeł może być równy lub nieparzysty. Węzeł parzystego stopnia ma parzystą liczbę łuków łączących go z sąsiednimi węzłami. Węzeł nieparzystego stopnia ma nieparzystą liczbę łuków. Z wyjątkiem początkowych i końcowych węzłów pożądany spacer musiałby opuszczać każdy węzeł dokładnie tak często, jak do niego wchodził. Węzły nieparzystego stopnia mogą być używane tylko jako początek lub koniec marszu, ponieważ takie węzły można było przekroczyć tylko określoną liczbę razy, zanim okażą się ślepym zaułkiem. Podróżny nie mógł opuścić węzła bez użycia poprzednio przebytego łuku. Euler zauważył, że jeśli wykres nie zawiera dokładnie zero lub dwóch węzłów o nieparzystym stopniu, przejście jest niemożliwe. Gdyby istniały dwa węzły nieparzystego stopnia, spacer mógłby rozpocząć się od pierwszego, a zakończyć na drugim; gdyby nie było węzłów o nieparzystym stopniu, spacer mógłby się rozpocząć i zakończyć w tym samym węźle. Spacer nie jest możliwy w przypadku wykresów zawierających inną liczbę węzłów o nieparzystym stopniu, jak ma to miejsce w przypadku miasta Królewca. Ten problem nazywa się teraz znajdowaniem ścieżki Eulera przez wykres. Zauważ, że reprezentacja rachunku predykatu, chociaż uwzględnia związki między mostami a ziemią w mieście, nie sugeruje koncepcji stopnia węzła. W reprezentacji graficznej jest jedno wystąpienie każdego węzła ze łukami między węzłami, a nie wielokrotne występowanie stałych jako argumentów w zestawie predykatów. Z tego powodu reprezentacja wykresu sugeruje koncepcję stopnia węzła i skupienie dowodu Eulera. To ilustruje moc teorii grafów do analizy struktury obiektów, właściwości i relacji. Dokonamy przeglądu podstawowej teorii grafów, a następnie prezentujemy skończone maszyny stanów i opis problemów w przestrzeni stanów. W sekcji kolejnej przedstawiamy wyszukiwanie grafów jako metodologię rozwiązywania problemów. Wyszukiwanie w pierwszej kolejności i w pierwszej kolejności to dwie strategie przeszukiwania przestrzeni stanów. Porównujemy je i wprowadzamy dodatkowe rozróżnienie między wyszukiwaniem goaldriven a wyszukiwaniem opartym na danych.

Kolejna sekcja pokazuje, w jaki sposób można wykorzystać wyszukiwanie w przestrzeni stanów do scharakteryzowania rozumowania za pomocą logiki. Wykorzystamy teorię grafów do analizy struktury i złożoności różnych problemów.

Struktury wyszukiwania w przestrzeni stanu

Teoria grafów (opcjonalnie)

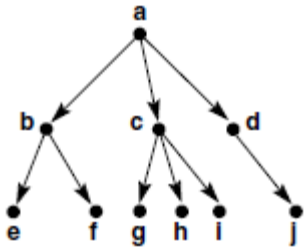
Wykres to zestaw węzłów lub stanów oraz zestaw łuków łączących węzły. Wykres z etykietami ma jeden lub więcej deskryptorów (etykiet) dołączonych do każdego węzła, które odróżniają ten węzeł od dowolnego innego węzła na wykresie. Na wykresie przestrzeni stanów deskryptory te identyfikują stany w procesie rozwiązywania problemów. Jeśli nie ma różnic opisowych między dwoma węzłami, są one uważane za takie same. Łuk między dwoma węzłami jest oznaczony etykietami połączonych węzłów. Łuki na wykresie mogą być również oznaczone. Etykiety łuku służą do wskazania, że łuk reprezentuje nazwaną relację (jak w sieci semantycznej) lub do przypisania wag do łuków (jak w przypadku problemu sprzedawcy podróży). Jeśli między tymi samymi węzłami występują różne łuki, można je również rozróżnić poprzez etykietowanie. Wykres jest kierowany, jeśli łuki mają powiązaną kierunkowość. Łuki na ukierunkowanym wykresie są zwykle rysowane jako strzałki lub mają dołączoną strzałkę wskazującą kierunek. Łuki które mogą być skrzyżowane w dowolnym kierunku, mogą mieć dołączone dwie strzałki, ale częściej nie mają żadnych kierunkowskazów. Poniższy rysunek to nakreślony, ukierunkowany wykres: łuk (a, b) można krzyżować tylko z węzła a do węzła b, ale łuk (b, c) można krzyżować w dowolnym kierunku.



Węzły = **{a,b,c,d,e}**

Łuki = **{(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}**

Ścieżka przez wykres łączy sekwencję węzłów poprzez kolejne łuki. Ścieżka jest reprezentowana przez uporządkowaną listę, która rejestruje węzły w kolejności, w jakiej występują na ścieżce. Na rysunku [a, b, c, d] reprezentuje ścieżkę przez węzły a, b, cid w tej kolejności. Zrootowany wykres ma unikalny węzeł, zwany korzeniem, tak że istnieje ścieżka od korzenia do wszystkich węzłów na wykresie. Podczas rysowania zrootowanego wykresu korzeń jest zwykle rysowany u góry strony, nad innymi węzłami. Wykresy przestrzeni stanów dla gier są zwykle wykresami zrootowanymi z początkiem gry jako rootem. Początkowe ruchy wykresu gry w kółko i krzyżyk są reprezentowane przez zrootowany wykres z rysunku.



Jest to ukierunkowany wykres, w którym wszystkie łuki mają jeden kierunek. Zauważ, że ten wykres nie zawiera cykli; gracze nie mogą (tak często, jak by sobie tego życzyli!) cofnąć ruch. Drzewo to wykres, na którym dwa węzły mają co najwyżej jedną ścieżkę między nimi. Drzewa często mają korzenie, w którym to przypadku są zwykle rysowane korzeniem u góry, jak zrootowany wykres. Ponieważ każdy węzeł w drzewie ma tylko jedną ścieżkę dostępu z dowolnego innego węzła, ścieżka nie może zapętlać ani przełączać się przez sekwencję węzłów. W przypadku ukorzenionych drzew lub wykresów relacje między węzłami obejmują nadrzędny, podrzędny i rodzeństwo. Są one używane w zwykły rodzinny sposób, a rodzic poprzedzający dziecko wzdłuż ukierunkowanego łuku. Dzieci węzła nazywane są rodzeństwem. Podobnie przodek pojawia się przed potomkiem na jakiejś ścieżce ukierunkowanego wykresu. Na rycinie 3.4, b jest rodzicem węzłów e i f (które są zatem potomkami b i swoim rodzeństwem). Węzły a i c są przodkami stanów g, h i i, a g, h i i są potomkami a i c. Przed wprowadzeniem reprezentacji problemów w przestrzeni stanów formalnie definiujemy te pojęcia.

DEFINICJA

WYKRES

Wykres składa się z:

Zestaw węzłów N_1, N_2, N_3, … , N_n, ..., które nie muszą być skończone.

Zestaw łuków łączących pary węzłów.

Łuki są uporządkowane parami węzłów; tzn. łuk (N_3, N_4) łączy węzeł N_3 z węzłem N_4. Wskazuje to bezpośrednie połączenie z węzła N_3 do N_4, ale nie z N_4 do N_3, chyba że (N_4, N_3) jest również łukiem, a następnie łuk łączący N_3 i N_4 nie jest kierowany.

Jeśli skierowany łuk łączy N_j i N_k, wówczas N_j jest nazywany rodzicem N_k i N_k, dzieckiem N_j. Jeśli wykres zawiera również łuk (N_j, N_l), wówczas N_k i N_l są nazywane rodzeństwem.

Zrootowany wykres ma unikalny węzeł N_S, z którego pochodzą wszystkie ścieżki na wykresie. Oznacza to, że root nie ma rodzica na wykresie.

Węzeł wierzchołka lub liścia jest węzłem, który nie ma potomków.

Uporządkowana sekwencja węzłów [N_1, N_2, N_3, ..., N_n], gdzie każda para N_i, N_{i+1} w sekwencji reprezentuje łuk, tj. (N_i, N_{i+1}), nazywana jest ścieżką długości n - 1.

Na ścieżce w zrootowanym grafie węzeł jest uważany za przodka wszystkich węzłów umieszczonych po nim (po jego prawej stronie), a także jako potomek wszystkich węzłów przed nim.

Ścieżka zawierająca dowolny węzeł więcej niż jeden raz (część N_j w powyższej definicji ścieżki jest powtarzana) zawiera cykl lub pętlę. Drzewo to wykres, na którym istnieje unikalna ścieżka między każdą parą węzłów. (Ścieżki w drzewie nie zawierają zatem cykli).

Krawędzie ukorzenionego drzewa są skierowane z dala od korzenia. Każdy węzeł w zrootowanym drzewie ma unikalny element nadrzędny.

Mówi się, że dwa węzły są połączone, jeśli istnieje ścieżka, która obejmuje je oba.

Następnie wprowadzamy maszynę skończoną, abstrakcyjną reprezentację urządzeń obliczeniowych, którą można postrzegać jako automat do przechodzenia ścieżek na wykresie.

Maszyna stanów skończonych (opcjonalnie)

Możemy myśleć o maszynie jako o systemie, który akceptuje wartości wejściowe, ewentualnie produkuje wartości wyjściowe i ma pewien wewnętrzny mechanizm (stany) do śledzenia informacji o poprzednich wartościach wejściowych. Skończona maszyna stanu (FSM) jest skończonym, ukierunkowanym, połączonym wykresem, mającym zestaw stanów, zestaw wartości wejściowych i stan funkcji przejścia opisująca wpływ elementów strumienia wejściowego na stany wykresu. Strumień wartości wejściowych tworzy ścieżkę na wykresie stanów tego skończonego automatu. Zatem FSM można postrzegać jako abstrakcyjny model obliczeń. Podstawowym zastosowaniem takiej maszyny jest rozpoznawanie elementów języka formalnego. Składniki te są często ciągami znaków („słów" złożonych ze znaków „alfabetu"). Te maszyny stanów odgrywają ważną rolę w analizie wyrażeń w językach, zarówno obliczeniowych, jak i ludzkich

DEFINICJA

AUTOMAT SKOŃCZONY (FSM)

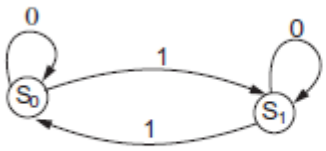
Automat skończony jest uporządkowanym potrójnym (S, I, F), gdzie:

S jest skończonym zestawem stanów na połączonym wykresie s_1 , s_2 , s_3 , …, s_n .

I jest skończonym zestawem wartości wejściowych i_1 , i_2 , i_3 ,… i_m .

F jest funkcją przejścia stanu, która dla dowolnego i ∈ I opisuje jej wpływ na stany S maszyny, a zatem ∀ i ∈ I, Fi: (S → S). Jeśli maszyna jest w stanie s_j i wystąpi wejście i, następnym stanem maszyny będzie $F_i(s_j)$.

Dla prostego przykładu maszyny stanów skończonych, niech S = {s_0 , s_1 }, I = {0,1}, $f_0(s_0) = s_0$, $f_0(s_1) = (s_1)$, $f_1(s_0) = s_1$, i $f_1(s_1) = s_0$. W tym urządzeniu, zwanym czasem flip-flop, wartość wejściowa równa zero pozostawia stan bez zmian, a wejście 1 zmienia stan maszyny. Możemy wizualizować tę maszynę z dwóch równoważnych perspektyw, jako skończonego wykresu ze znakowanymi, skierowanymi łukami, jak na rysunku a lub jako macierzy przejścia, rysunek b. W macierzy przejścia wartości wejściowe są wymienione wzdłuż górnego wiersza, stany znajdują się w lewej kolumnie, a dane wyjściowe dla danych wejściowych zastosowanych do stanu znajdują się w punkcie przecięcia.

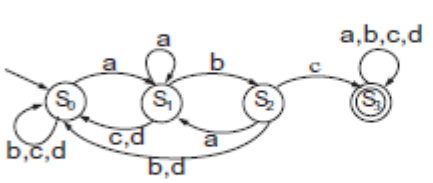


(a)

	0	1
S_0	S_0	S_1
S_1	S_1	S_0

(b)

Drugi przykład maszyny stanów skończonych jest reprezentowany przez ukierunkowany wykres z rysunku a i (równoważną) macierz przejściową z rysunku b. Ktoś może zapytać, co może reprezentować skończona maszyna stanów z rysunku.



(a)

	a	b	c	d
→ S_0	S_1	S_0	S_0	S_0
S_1	S_1	S_2	S_0	S_0
S_2	S_1	S_0	S_3	S_0
(S_3)	S_3	S_3	S_3	S_3

(b)

Przy dwóch założeniach maszynę tę można uznać za rozpoznawanie wszystkich ciągów znaków z alfabetu {a, b, c, d}, które zawierają dokładną sekwencję „abc”. Dwa założenia to, po pierwsze, że stan s0 ma szczególną rolę jako stan początkowy, a po drugie, że s_3 jest stanem akceptującym. Zatem strumień wejściowy przedstawi swój pierwszy element do stanu s0. Jeśli strumień zakończy się później z maszyną w stanie s3, rozpozna sekwencję „abc” w tym strumień wejściowy. To, co właśnie opisaliśmy, to maszyna akceptująca stan skończony, czasami nazywana maszyną Moore'a. Stosujemy konwencję umieszczania strzałki z żadnego stanu, która kończy się w stanie początkowym maszyny Moore, i reprezentujemy stan (lub stany) akceptujące jako szczególny, często za pomocą podwójnego koła, jak na rysunku powyżej. Przedstawiamy teraz formalną definicję maszyny Moore:

DEFINICJA

AKCEPTOR STANU SKOŃCZONEGI (MOORE MACHINE)

Akceptant stanu skończonego jest maszyną stanu skończonego (S, I, F), gdzie:

$\exists\ s_0 \in S$ tak, że strumień wejściowy zaczyna się od s_0,

a $\exists\ s_n \in S$, stan akceptacji. Strumień wejściowy jest akceptowany, jeśli kończy się w tym stanie. W rzeczywistości może istnieć zestaw stanów akceptacji.

Akceptor stanu skończonego jest reprezentowany jako $(S, s_0, \{s_n\}, I, F)$

Przedstawiliśmy dwa dość proste przykłady potężnej koncepcji. Jak zobaczymy w zrozumieniu języka naturalnego, rozpoznawanie stanów skończonych jest ważnym narzędziem do określania, czy wzory znaków, słów lub zdań mają pożądane właściwości. Zobaczymy, że akceptant stanu skończonego domyślnie definiuje język formalny na podstawie zestawów liter (znaków) i słów (ciągów znaków), które akceptuje. Pokazaliśmy także tylko deterministyczne maszyny stanów skończonych, w których funkcja przejścia dla dowolnej wartości wejściowej do stanu daje unikalny następny stan. Ważną techniką modelowania są również probabilistyczne maszyny stanów skończonych, w których funkcja przejścia definiuje rozkład stanów wyjściowych dla każdego wejścia do stanu.

Reprezentacja problemów w przestrzeni stanu

W reprezentacji problemu w przestrzeni stanów ,węzły wykresu odpowiadają stanom częściowego rozwiązania problemu, a łuki odpowiadają etapom w procesie rozwiązywania problemu. Jeden lub więcej stanów początkowych, odpowiadających danym informacyjnym w wystąpieniu problemu, tworzy rdzeń wykresu. Wykres definiuje również jeden lub więcej warunków celu, które są

rozwiązaniami problemu. Wyszukiwanie w przestrzeni stanów charakteryzuje rozwiązywanie problemów jako proces znajdowania ścieżki rozwiązania od stanu początkowego do celu. Cel może opisywać stan, na przykład zwycięską planszę w kółko i krzyżyk lub konfigurację bramki w 8-puzzlach. Alternatywnie cel może opisać niektóre właściwości samej ścieżki rozwiązania. W przypadku problemu sprzedawcy podróży wyszukiwanie kończy się, gdy „najkrótsza" ścieżka zostanie znaleziona we wszystkich węzłach wykresu. W przypadku problemu analizowania ścieżka rozwiązania stanowi udaną analizę struktury zdania. Łuki w przestrzeni stanów odpowiadają krokom w procesie rozwiązania, a ścieżki w przestrzeni reprezentują rozwiązania na różnych etapach realizacji. Ścieżki są przeszukiwane, zaczynając od stanu początkowego i kontynuując wykres, aż opis celu zostanie spełniony lub zostaną porzucone. Rzeczywiste generowanie nowych stanów na ścieżce odbywa się poprzez zastosowanie operatorów, takich jak „legalne ruchy" w grze lub reguły wnioskowania w problemach logicznych lub systemie eksperckim, do istniejących stanów na ścieżce. Algorytm wyszukiwania ma za zadanie znaleźć ścieżkę rozwiązania w takiej przestrzeni problemów. Algorytmy wyszukiwania muszą śledzić ścieżki od początku do węzła celu, ponieważ ścieżki te zawierają serię operacji prowadzących do rozwiązania problemu. Teraz formalnie definiujemy reprezentację problemów w przestrzeni stanu:

DEFINICJA

WYSZUKIWANIE PRZESTRZENI STANU

Przestrzeń stanu jest reprezentowana przez czterokrotną [N, A, S, GD], gdzie:

N jest zbiorem węzłów lub stanów wykresu. Odpowiadają one stanom w procesie rozwiązywania problemów. A to zestaw łuków (lub łączy) między węzłami. Odpowiadają one etapom procesu rozwiązywania problemów. S, niepusty podzbiór N, zawiera stan początkowy problemu. GD, niepusty podzbiór N, zawiera stan (cele) problemu. Stany w GD są opisane za pomocą:

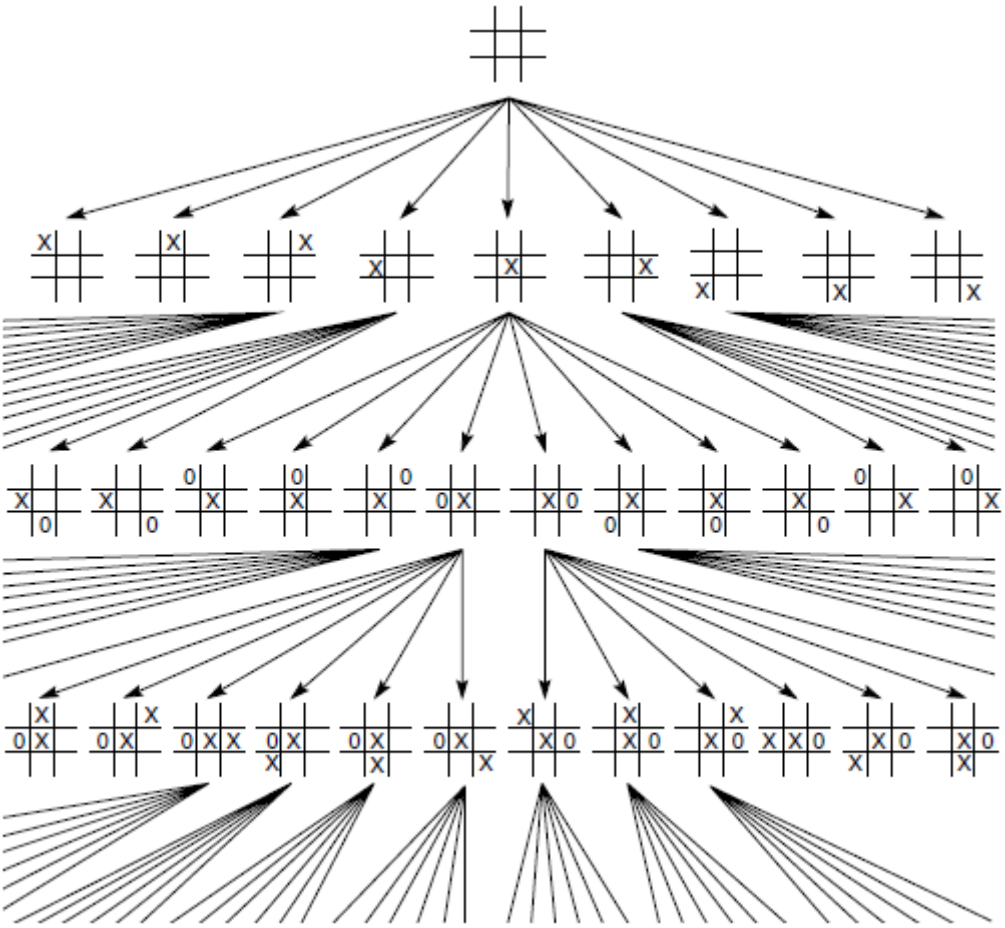
1. Mierzalna właściwość stanów napotkanych podczas wyszukiwania.

2. Mierzalna właściwość ścieżki opracowana podczas wyszukiwania, na przykład suma kosztów przejścia dla łuków ścieżki.

Ścieżka rozwiązania to ścieżka przez ten wykres od węzła w S do węzła w GD. Jedną z ogólnych cech grafu i jednym z problemów, które pojawiają się przy projektowaniu algorytmu wyszukiwania grafów, jest to, że do stanów można czasem dotrzeć różnymi ścieżkami. Na przykład ścieżka może być utworzona ze stanu od a do stanu d albo przez b i c, albo bezpośrednio od a do d. Dlatego ważne jest, aby wybrać najlepszą ścieżkę zgodnie z potrzebami problemu. Ponadto wiele ścieżek do stanu może prowadzić do powstawania pętli lub cykli w ścieżce rozwiązania, które uniemożliwiają algorytmowi osiągnięcie celu. Jeśli miejscem do przeszukania jest drzewo , problem cykli nie występuje. Dlatego ważne jest rozróżnienie między problemami, których przestrzenią stanu jest drzewo, a tymi, które mogą zawierać pętle. Ogólne algorytmy wyszukiwania grafów muszą wykrywać i eliminować pętle z potencjalnych ścieżek rozwiązań, podczas gdy wyszukiwanie drzew może zwiększyć wydajność poprzez wyeliminowanie tego testu i jego narzutu. Kółko i krzyżyk oraz 8-puzzle stanowią przestrzeń stanów prostych gier. Oba te przykłady pokazują warunki zakończenia typu 1 w naszej definicji wyszukiwania w przestrzeni stanów. Przykład problemu sprzedawcy podróży, zawiera opis celu typu 2, czyli całkowity koszt samej ścieżki.

PRZYKŁAD 3.1.1: Kółko i krzyżyk

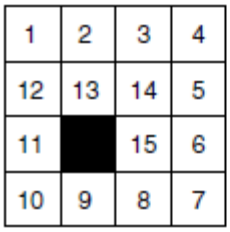
Reprezentacja przestrzeni stanowej kółko i krzyżyk jest pokazana na rysunku

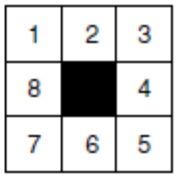


Stan początkowy to pusta tablica, a zakończenie lub opis celu to stan planszy mający trzy X-y w rzędzie, kolumnie lub przekątnej (przy założeniu, że celem jest wygrana dla X). Ścieżka od stanu początkowego do stanu bramkowego daje serię ruchów w zwycięskiej grze. Stany w przestrzeni to różne konfiguracje X i O-er, które może mieć gra. Oczywiście, chociaż istnieje 39 sposobów na ułożenie {puste, X, O} w dziewięciu polach, większość z nich nigdy nie wystąpiłaby w prawdziwej grze. Łuki są generowane przez legalne ruchy w grze, na przemian między umieszczeniem X i O w nieużywanym miejscu. Przestrzeń stanu jest raczej grafem niż drzewem, ponieważ do niektórych stanów na trzecim i głębszym poziomie można dotrzeć różnymi ścieżkami. Jednak w przestrzeni stanu nie ma cykli, ponieważ skierowane łuki wykresu nie pozwalają na cofnięcie ruchu. Po osiągnięciu stanu niemożliwe jest „cofnięcie się" w górę struktury. Nie jest konieczne sprawdzanie cykli w generowaniu ścieżki. Struktura wykresu z tą właściwością nazywana jest ukierunkowanym wykresem acyklicznym lub DAG i jest powszechna w wyszukiwaniu przestrzeni stanów i modelach graficznych. Reprezentacja przestrzeni stanów umożliwia określenie złożoności problemu. W kółko i krzyżyk występuje dziewięć pierwszych ruchów z ośmioma możliwymi reakcjami na każde z nich, a następnie siedem możliwych odpowiedzi na każde z nich i tak dalej. Wynika z tego, że 9 × 8 × 7 × ... lub 9! można wygenerować różne ścieżki. Chociaż komputer nie jest w stanie wyczerpująco przeszukać tej liczby ścieżek (362 880), wszelkie ważne problemy wykazują również złożoność czynnikową lub wykładniczą, chociaż na znacznie większą skalę. Szachy mają 10^{120} możliwych ścieżek gry; warcaby ma 10^{40}, z których niektóre mogą nigdy nie wystąpić w rzeczywistej grze. Przestrzenie te są trudne lub niemożliwe do wyczerpującego przeszukania. Strategie przeszukiwania tak dużych przestrzeni często opierają się na heurystyce w celu zmniejszenia złożoności wyszukiwania

PRZYKŁAD 3.1.2: 8-PUZZLEL

W Piętnastce z rysunku





15 różnie ponumerowanych płytek umieszczonych jest na 16 polach na siatce. Jedno miejsce pozostawia się puste, aby można było przesuwać płytki, tworząc różne wzory. Celem jest znalezienie serii ruchów płytek w puste miejsce, które ustawiają planszę w konfiguracji celu. Jest to popularna gra, w którą większość z nas grała jako dzieci. Wiele interesujących aspektów tej gry sprawiło, że jest ona przydatna dla badaczy w rozwiązywaniu problemów. Przestrzeń stanu jest wystarczająco duża, aby być interesująca, ale nie jest całkowicie trudna do rozwiązania (16! Jeśli stany symetryczne są traktowane jako odrębne). Stany gry są łatwe do przedstawienia. Gra jest wystarczająco bogata, aby zapewnić szereg interesujących heurystyk. 8-puzzle to wersja Piętnastki, 3 × 3, w której można przesuwać osiem płytek na dziewięciu polach. Ponieważ 8-puzzle generuje mniejszą przestrzeń stanu niż pełne 15-łamigłówek, a jej wykres łatwo mieści się na stronie, jest wykorzystywany w wielu przykładach w tej książce. Chociaż w fizycznych łamigłówkach ruchy są wykonywane przez przesuwanie płytek („przesuń 7 płytek w prawo, pod warunkiem, że puste miejsce znajduje się po prawej stronie płytki" lub „przesuń 3 płytki w dół"), o wiele łatwiej jest myśleć w kategoriach „ przesunięcie pustej przestrzeni ". Upraszcza to definicję reguł ruchu, ponieważ istnieje osiem kafelków, ale tylko jeden pusty. Aby zastosować ruch, musimy upewnić się, że nie spowoduje to przesunięcia pustej planszy. Dlatego wszystkie cztery ruchy nie mają zastosowania przez cały czas; na przykład, gdy półfabrykat znajduje się w jednym z rogów, możliwe są tylko dwa ruchy.

Legalne ruchy to:

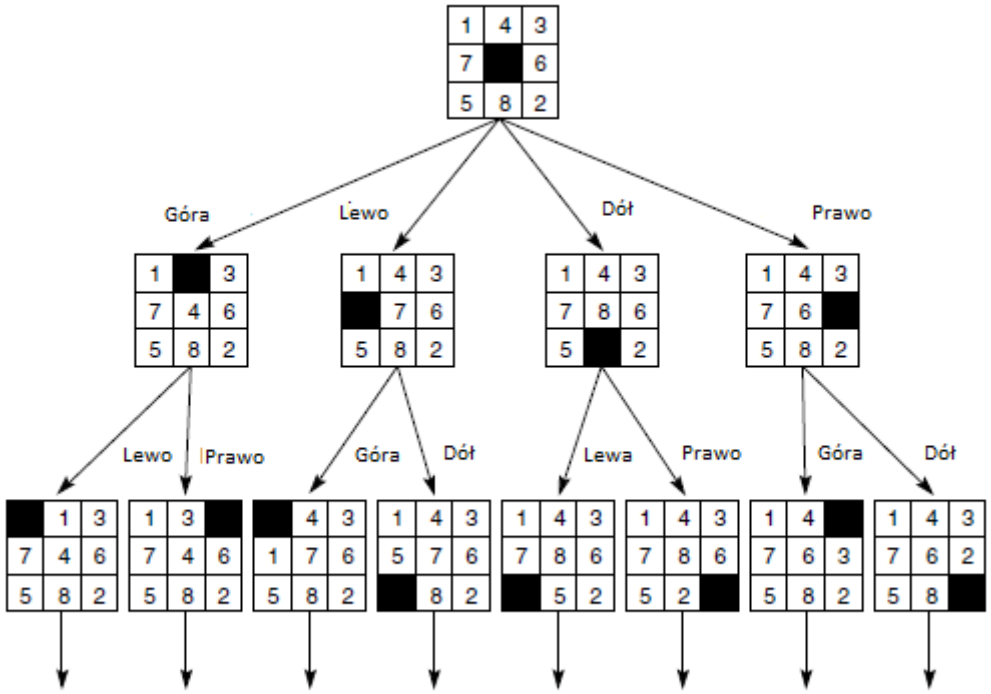
przesuń puste w górę ↑

przesuń puste prawo →

przesuń puste miejsce ↓

przesuń puste w lewo ←

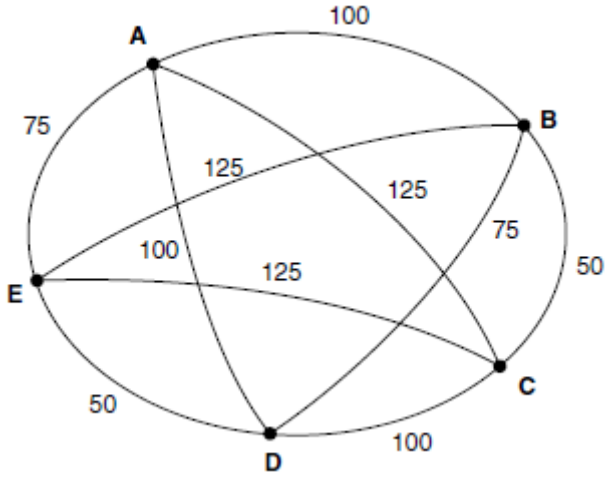
Jeśli określimy stan początkowy i stan celu dla 8-puzzli, możliwe jest podanie rozliczenia w przestrzeni stanu procesu rozwiązywania problemów



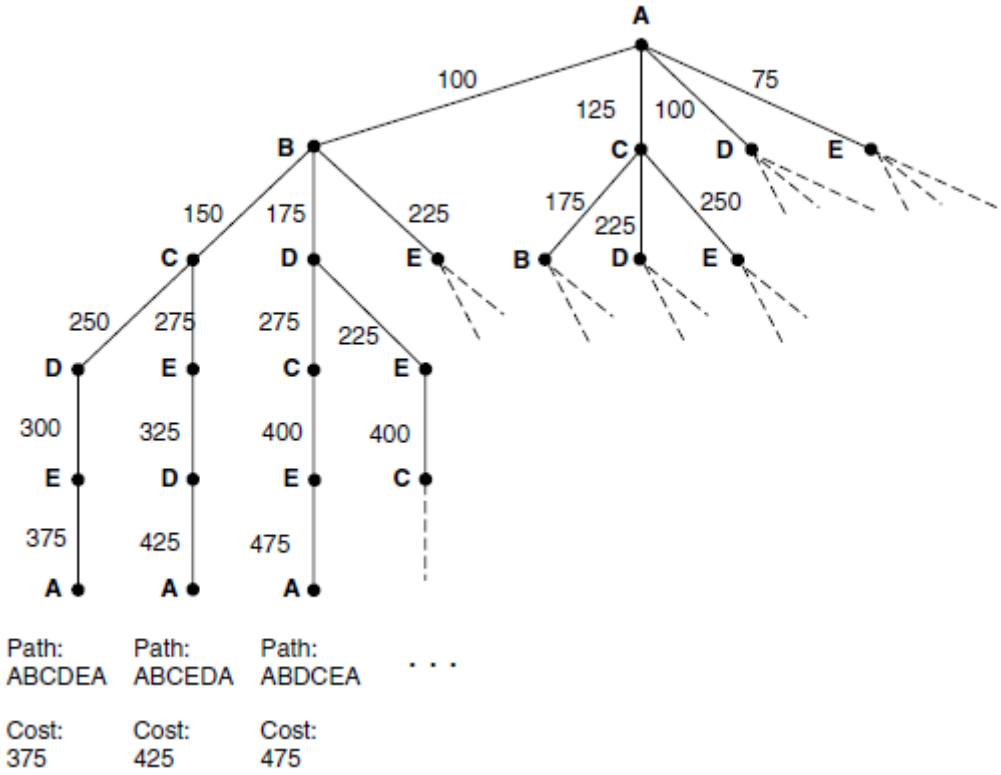
Stany mogą być reprezentowane przy użyciu prostej tablicy 3 × 3. Reprezentacja rachunku predykatu może wykorzystywać predykat „stanowy” z dziewięcioma parametrami (dla lokalizacji liczb w siatce). Cztery procedury opisujące każdy z możliwych ruchów pustki definiują łuki w przestrzeni stanu. Podobnie jak w przypadku kółko i krzyżyk, przestrzeń stanu dla 8-puzzli jest wykresem (większość stanów ma wielu rodziców), ale w przeciwieństwie do kółko i krzyżyk, możliwe są cykle. GD lub opis celu przestrzeni stanów to konkretna konfiguracja stanu lub płytki. Po znalezieniu tego stanu na ścieżce wyszukiwanie kończy się. Ścieżka od początku do celu jest pożądaną serią ruchów. Warto zauważyć, że pełna przestrzeń stanów 8 i 15 łamigłówek składa się z dwóch rozłączonych (iw tym przypadku jednakowych rozmiarów) podgrafów. To sprawia, że połowa możliwych stanów w przestrzeni wyszukiwania jest niemożliwa do osiągnięcia z dowolnego stanu początkowego. Jeśli wymienimy (poprzez podważanie!) Dwie bezpośrednio sąsiadujące ze sobą płytki, stany w drugim elemencie przestrzeni będą osiągalne.

PODRÓŻ SPRZEDAWCY

Załóżmy, że sprzedawca ma pięć miast do odwiedzenia, a następnie musi wrócić do domu. Problem polega na znalezieniu najkrótszej ścieżki, po której sprzedawca podróżuje, odwiedza każde miasto, a następnie wraca do miasta początkowego. Rysunek pokazuje przykład tego problemu.

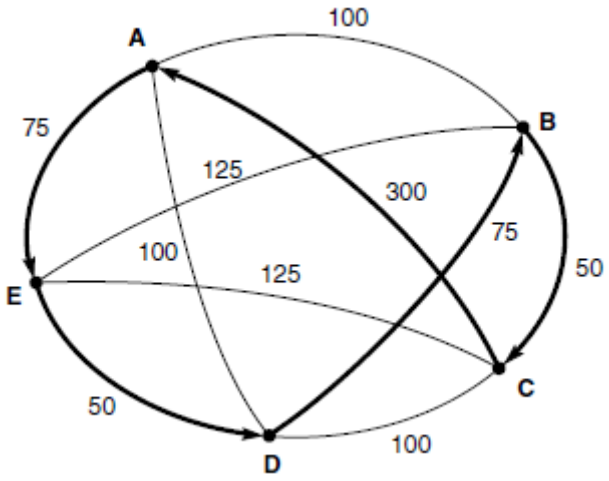


Węzły wykresu przedstawiają miasta, a każdy łuk jest oznaczony wagą wskazującą koszt podróży tego łuku. Koszt ten może stanowić reprezentację mil niezbędnych w podróży samochodem lub koszt lotu lotniczego między dwoma miastami. Dla wygody zakładamy, że sprzedawca mieszka w mieście A i tam wróci, chociaż założenie to po prostu ogranicza problem N miast do problemu (N - 1) miast. Ścieżka [A, D, C, B, E, A], z powiązanym kosztem 450 mil, jest przykładem możliwego obwodu. Opis celu wymaga kompletnego obwodu przy minimalnych kosztach. Pamiętaj, że opis celu jest właściwością całej ścieżki, a nie pojedynczego stanu. Jest to opis celu typu 2 z definicji wyszukiwania w przestrzeni stanów. Rysunek poniższy pokazuje jeden ze sposobów generowania i porównywania możliwych ścieżek rozwiązań.



Począwszy od węzła A, możliwe są kolejne stany, dopóki wszystkie miasta nie zostaną uwzględnione i ścieżka nie wróci do domu. Celem jest ścieżka o najniższych kosztach. Jak sugeruje rysunek , złożoność wyczerpujących poszukiwań w problemie sprzedawcy podróży to (N - 1) !, gdzie N to liczba miast na

wykresie. W przypadku 9 miast możemy wyczerpująco wypróbować wszystkie ścieżki, ale w przypadku dowolnego problemu o interesującej wielkości, na przykład w przypadku 50 miast, nie można przeprowadzić prostego, wyczerpującego wyszukiwania w praktycznym czasie. W rzeczywistości koszty złożoności dla N! Wyszukiwanie rośnie tak szybko, że już wkrótce kombinacje wyszukiwania stają się trudne. Kilka technik może zmniejszyć złożoność wyszukiwania. Jeden nazywa się odgałęzieniem i związaniem). Rozgałęzienie i powiązanie generuje ścieżki pojedynczo, śledząc najlepszy znaleziony obwód. Ta wartość jest używana jako ograniczenie dla przyszłych kandydatów. Ponieważ ścieżki są budowane po jednym mieście na raz, algorytm sprawdza każdą częściowo ukończoną ścieżkę. Jeśli algorytm ustali, że najlepsze możliwe rozszerzenie ścieżki, gałąź, będzie miało większy koszt niż granica, eliminuje tę ścieżkę częściową i wszystkie możliwe rozszerzenia. Zmniejsza to znacznie wyszukiwanie, ale wciąż pozostawia wykładniczą liczbę ścieżek ($1{,}26^N$ zamiast N!). Inna strategia kontrolowania wyszukiwania konstruuje ścieżkę zgodnie z zasadą „idź do najbliższego nieodwiedzonego miasta". Ścieżka najbliższego sąsiada przez wykres z poniższego rysunku to [A, E, D, B, C, A], kosztem 375 mil. Ta metoda jest bardzo wydajna, ponieważ można wypróbować tylko jedną ścieżkę! Najbliższy sąsiad, czasem nazywany chciwym, heurystycznym jest omylny, ponieważ istnieją wykresy, dla których nie znajduje on najkrótszej ścieżki, ale jest to możliwy kompromis, gdy wymagany czas sprawia, że wyczerpujące poszukiwania są niepraktyczne



Strategie wyszukiwania w przestrzeni stanu

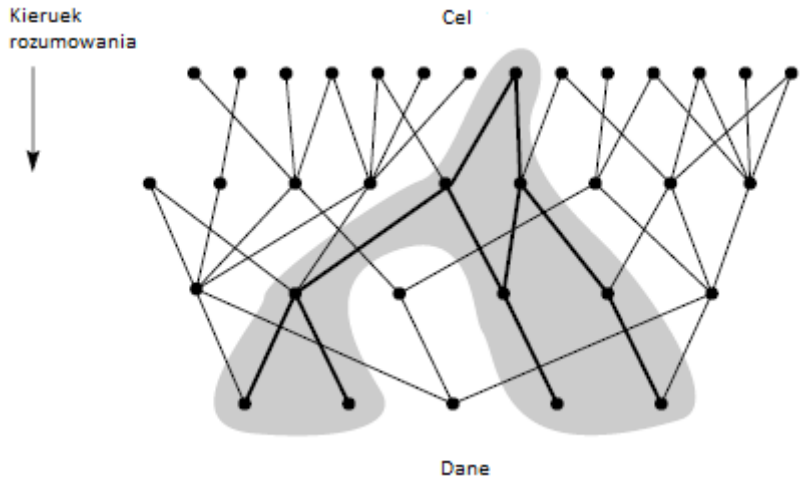
Wyszukiwanie oparte na danych i celu

Przestrzeń stanu może być przeszukiwana w dwóch kierunkach: od podanych danych wystąpienia problemu w kierunku celu lub z celu z powrotem do danych. W wyszukiwaniu opartym na danych, czasami nazywanym łańcuchem naprzód, rozwiązanie problemu rozpoczyna się od podanych faktów problemu i zestawu legalnych ruchów lub reguł dotyczących zmiany stanu. Wyszukiwanie przebiega przez zastosowanie reguł do faktów w celu wygenerowania nowych faktów, które z kolei są wykorzystywane przez reguły do generowania kolejnych nowych faktów. Proces ten trwa, dopóki (mamy nadzieję!) Nie wygeneruje ścieżki, która spełnia warunek celu. Możliwe jest alternatywne podejście: postaw cel, który chcemy rozwiązać. Zobacz, jakie reguły lub legalne ruchy można zastosować do wygenerowania tego celu, i określ, jakie warunki muszą być spełnione, aby z nich skorzystać. Warunki te stają się nowymi celami lub podzadaniami poszukiwania. Wyszukiwanie trwa, pracując wstecz przez kolejne podzadania, dopóki (mamy nadzieję!) Nie wróci do faktów dotyczących problemu. Znajduje łańcuch ruchów lub reguł prowadzących od danych do celu, chociaż robi to w odwrotnej kolejności. Takie podejście nazywa się rozumowaniem ukierunkowanym na cel lub

łańcuchem wstecznym i przypomina prostą sztuczkę z dzieciństwa polegającą na próbie rozwiązania labiryntu poprzez powrót do mety od początku do końca. Podsumowując: rozumowanie oparte na danych uwzględnia fakty dotyczące problemu i stosuje reguły lub legalne działania w celu uzyskania nowych faktów prowadzących do celu; Rozumowanie zorientowane na cel koncentruje się na celu, znajduje reguły, które mogłyby doprowadzić do celu, i łączy łańcuchy wstecz poprzez kolejne reguły i podzadania do danych faktów dotyczących problemu. W końcowej analizie, zarówno rozwiązujący problemy oparte na danych, jak i na celu przeszukają ten sam wykres przestrzeni stanów; jednak kolejność i rzeczywista liczba przeszukiwanych stanów mogą się różnić. Preferowaną strategię określają właściwości samego problemu. Należą do nich złożoność reguł, „kształt" przestrzeni stanu oraz charakter i dostępność danych o problemach. Wszystkie różnią się w zależności od problemów. Jako przykład wpływu strategii wyszukiwania na złożoność wyszukiwania, rozważ problem potwierdzenia lub odmowy stwierdzenia „Jestem potomkiem Thomasa Jeffersona". Rozwiązanie to ścieżka bezpośredniej linii między „ja" a Thomas Jefferson. Przestrzeń tę można przeszukiwać w dwóch kierunkach, zaczynając od „ja" i pracując wzdłuż linii przodków do Thomasa Jeffersona lub zaczynając od Thomasa Jeffersona i pracując przez jego potomków. Kilka prostych założeń pozwala oszacować rozmiar przeszukiwanej przestrzeni w każdym kierunku. Thomas Jefferson urodził się około 250 lat temu; jeśli założymy, że 25 lat na pokolenie, wymagana ścieżka będzie miała długość około 10. Ponieważ każda osoba ma dokładnie dwóch rodziców, wyszukiwanie z „ja" badałoby na 210 przodków. Poszukiwania przeprowadzone przez Thomasa Jeffersona zbadałyby więcej stanów, ponieważ ludzie mają zwykle więcej niż dwoje dzieci (szczególnie w XVIII i XIX wieku). Gdy zakładamy, że średnio tylko troje dzieci na rodzinę, wyszukiwanie badałoby w kolejności 310 węzłów drzewa genealogicznego. Zatem wyszukiwanie wstecz od „ja" zbadałoby mniej węzłów. Należy jednak pamiętać, że oba kierunki powodują wykładniczą złożoność. Decyzja o wyborze między wyszukiwaniem opartym na danych i celu zależy od struktury problemu do rozwiązania. Wyszukiwanie zorientowane na cel jest zalecane, jeśli:

1. Cel lub hipoteza jest podana w opisie problemu lub może być łatwo sformułowana. Na przykład w przysłowie twierdzeń matematycznych celem jest twierdzenie, które należy udowodnić. Wiele systemów diagnostycznych rozważa potencjalne diagnozy w sposób systematyczny, potwierdzając lub eliminując je przy użyciu rozumowania ukierunkowanego na cel.

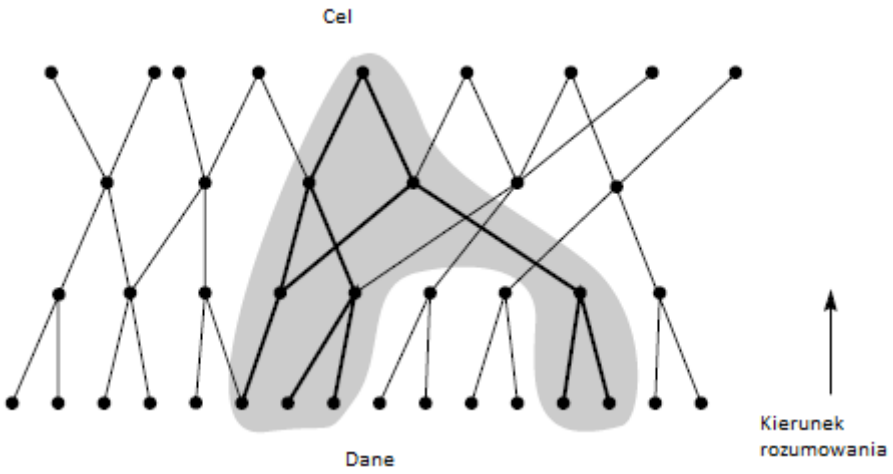
2. Istnieje wiele zasad, które pasują do faktów związanych z problemem, a tym samym generują coraz więcej wniosków lub celów. Wczesny wybór celu może wyeliminować większość tych gałęzi, dzięki czemu wyszukiwanie zorientowane na cel jest bardziej skuteczne w przycinaniu przestrzeni



Na przykład w dowodzie twierdzącym całkowita liczba reguł użytych do wytworzenia danego twierdzenia jest zwykle znacznie mniejsza niż liczba reguł, które można zastosować do całego zestawu aksjomatów.

3. Dane o problemie nie są podawane, ale muszą zostać pozyskane przez rozwiązanie problemu. W takim przypadku wyszukiwanie zorientowane na cel może pomóc w uzyskaniu danych. Na przykład w programie diagnostyki medycznej można zastosować szeroki zakres testów diagnostycznych. Lekarze zamawiają tylko tych, którzy są niezbędni do potwierdzenia lub odrzucenia określonej hipotezy. Wyszukiwanie zorientowane na cel wykorzystuje zatem wiedzę o pożądanym celu, aby poprowadzić wyszukiwanie przez odpowiednie reguły i wyeliminować gałęzie przestrzeni.

Wyszukiwanie oparte na danych



jest odpowiedni w przypadku problemów, w których:

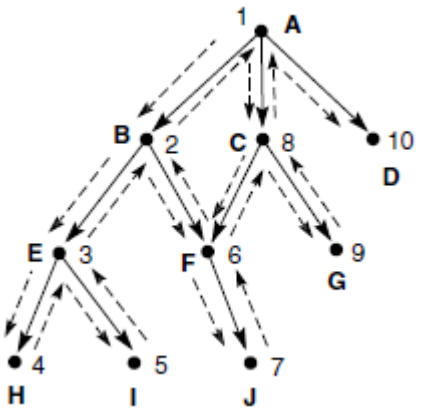
1. Wszystkie lub większość danych podano w początkowym opisie problemu. Problemy z interpretacją często pasują do tej formy, prezentując zbiór danych i prosząc system o zapewnienie interpretacji na wysokim poziomie. Systemy analizujące określone dane (np. PROSPECTOR lub programy Dipmeter, które interpretują dane geologiczne lub próbują znaleźć minerały, które mogą znajdować się w danym miejscu) pasują do podejścia opartego na danych.
2. Istnieje wiele potencjalnych celów, ale istnieje tylko kilka sposobów wykorzystania faktów i informacji o konkretnym wystąpieniu problemu. Przykładem jest program DENDRAL, system ekspercki, który znajduje strukturę molekularną związków organicznych na podstawie ich formuły, danych spektrografu masowego i wiedzy chemicznej. Dla każdego związku organicznego istnieje ogromna liczba możliwych struktur. Jednak dane spektrografu masowego związku pozwalają DENDRAL wyeliminować wszystkie oprócz kilku.

3. Trudno jest sformułować cel lub hipotezę. Na przykład przy stosowaniu DENDRAL początkowo niewiele wiadomo na temat możliwej struktury związku. Wyszukiwanie oparte na danych wykorzystuje wiedzę i ograniczenia znalezione w danych problemu, aby poprowadzić wyszukiwanie wzdłuż linii, o których wiadomo, że są prawdziwe.

Podsumowując, nie można zastąpić dokładnej analizy konkretnego problemu do rozwiązania, biorąc pod uwagę takie czynniki, jak czynnik rozgałęziający zastosowania reguł, dostępność danych i łatwość określenia potencjalnych celów.

Wdrożenie wyszukiwania graficznego

Przy rozwiązywaniu problemu za pomocą wyszukiwania opartego na celu lub danych rozwiązanie problemu musi znaleźć ścieżkę od stanu początkowego do celu poprzez wykres przestrzeni stanu. Sekwencja łuków na tej ścieżce odpowiada uporządkowanym krokom rozwiązania. Jeśli podano rozwiązanie problemu oracle lub inny nieomylny mechanizm wyboru ścieżki rozwiązania, wyszukiwanie nie będzie wymagane. Rozwiązujący problemy poruszałby się bezbłędnie przez przestrzeń do pożądanego celu, budując ścieżkę. Ponieważ wyroki nie istnieją dla interesujących problemów, narzędzie do rozwiązywania problemów musi rozważyć różne ścieżki w przestrzeni, dopóki nie znajdzie celu. Cofanie jest techniką systematycznego sprawdzania wszystkich ścieżek przez przestrzeń stanu. Zaczynamy od cofania, ponieważ jest to jeden z pierwszych algorytmów wyszukiwania badanych przez informatyków i ma naturalną implementację w środowisku rekurencyjnym zorientowanym na stos. Zaprezentujemy prostszą wersję algorytmu cofania z wyszukiwaniem w pierwszej kolejności. Wyszukiwanie zwrotne rozpoczyna się od stanu początkowego i podąża ścieżką, aż osiągnie cel lub „ślepy zaułek". Jeśli znajdzie cel, kończy pracę i zwraca ścieżkę rozwiązania. Jeśli osiągnie ślepy zaułek, „cofa się" do najnowszego węzła na ścieżce z niezbadanym rodzeństwem i kontynuuje w dół jednej z tych gałęzi, jak opisano w następującej regule rekurencyjnej: Jeśli obecny stan S nie spełnia wymagań opis celu, następnie wygeneruj swój pierwszy potomek Schild1 i zastosuj procedurę cofania rekursywnie do tego węzła. Jeśli powrót nie znajdzie węzła celu w podgrafie zrootowanym na Schild1, powtórz procedurę dla jego rodzeństwa, Schild2. Kontynuuj, aż jakiś potomek dziecka będzie węzłem celu lub wszystkie dzieci zostaną przeszukane. Jeśli żadne z dzieci S nie prowadzi do celu, wówczas powrót „nie powraca" do rodzica S, gdzie jest stosowany do rodzeństwa S i tak dalej. Algorytm trwa do momentu znalezienia celu lub wyczerpania przestrzeni stanów. Rysunek pokazuje algorytm cofania zastosowany do hipotetycznej przestrzeni stanów.



Kierunek przerywanych strzałek na drzewie wskazuje postęp wyszukiwania w górę i w dół przestrzeni. Liczba obok każdego węzła wskazuje kolejność odwiedzania. Teraz definiujemy algorytm, który wykonuje powrót, używając trzech list do śledzenia węzłów w przestrzeni stanu: SL, dla listy stanów, wymienia stany w bieżącej ścieżce, którą próbujesz. Jeśli cel zostanie

znaleziony, SL zawiera uporządkowaną listę stanów na ścieżce rozwiązania. NSL, dla nowej listy stanów, zawiera węzły oczekujące na ocenę, tj. Węzły, których potomkowie nie zostali jeszcze wygenerowani i przeszukani. DE dla ślepych zaułków wymienia stany, których potomkowie nie zawarli celu. Jeśli te stany wystąpią ponownie, zostaną one wykryte jako elementy DE i natychmiast wyeliminowane z rozpatrzenia. Podczas definiowania algorytmu cofania dla przypadku ogólnego (raczej wykresu niż drzewa) konieczne jest wykrycie wielu wystąpień dowolnego stanu, aby nie został ponownie wprowadzony i spowodować (nieskończone) pętle na ścieżce. Dokonuje się tego poprzez przetestowanie każdego nowo wygenerowanego stanu pod kątem członkostwa w którejkolwiek z tych trzech list. Jeśli nowy stan należy do którejkolwiek z tych list, to został już odwiedzony i może zostać zignorowany.

```
function backtrack;
begin
SL := [Start]; NSL := [Start]; DE := [ ]; CS := Start; % initialize:
while NSL ≠ [ ] do % while there are states to be tried
begin
if CS = goal (or meets goal description)
then return SL; % on success, return list of states in path.
if CS has no children (excluding nodes already on DE, SL, and NSL)
then begin
while SL is not empty and CS = the first element of SL do
begin
add CS to DE; % record state as dead end
remove first element from SL; %backtrack
remove first element from NSL;
CS := first element of NSL;
end
add CS to SL;
end
else begin
place children of CS (except nodes already on DE, SL, or NSL) on NSL;
CS := first element of NSL;
add CS to SL
end
end;
return FAIL;
end.
```

W przypadku cofania aktualnie rozważany stan nazywa się CS dla bieżącego stanu. CS jest zawsze równe stanowi ostatnio dodanemu do SL i reprezentuje „granicę” aktualnie badanej ścieżki rozwiązania. Reguły wnioskowania, ruchy w grze lub inne odpowiednie operatory rozwiązywania problemów są porządkowane i stosowane w CS. Rezultatem jest uporządkowany zestaw nowych stanów, dzieci CS. Pierwsze z tych dzieci ma nowy obecny stan, a pozostałe są uporządkowane w NSL do przyszłego badania. Nowy aktualny stan zostanie dodany do SL i wyszukiwanie będzie kontynuowane. Jeśli CS nie ma potomków, jest usuwany z SL (tutaj algorytm „wycofuje się”) i wszystkie pozostałe potomki jego poprzednika na SL są badane. Ślad cofnięcia na wykresie z powyższego rysunku jest podany przez:

Początkowe : SL = [A]; NSL = [A]; DE = [ ]; CS = A;

Po iteracji	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]

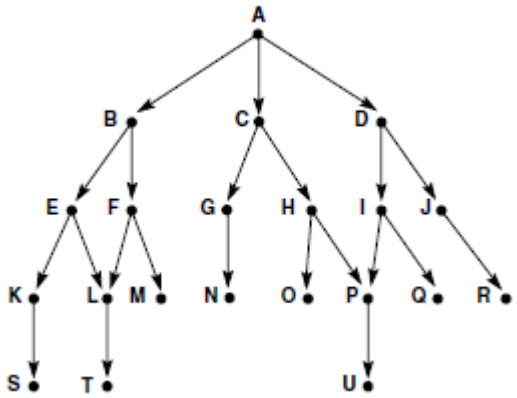
Jak przedstawiono tutaj, backtrack implementuje wyszukiwanie oparte na danych, przyjmując root jako stan początkowy i oceniając swoje dzieci pod kątem szukania celu. Algorytm może być postrzegany jako wyszukiwanie ukierunkowane na cel, pozwalając, aby cel był pierwiastkiem na wykresie, i oceniając potomków ponownie, próbując znaleźć stan początkowy. Jeśli opis celu jest typu 2, algorytm musi określić stan celu poprzez zbadanie ścieżki na SL. backtrack to algorytm przeszukiwania grafów przestrzeni stanów. Wykorzystywane są następujące algorytmy wyszukiwania wykresów, w tym wyszukiwanie według głębokości, szerokości i najlepszego wyszukiwania ,pomysły wykorzystane w backtracku, w tym:

1. Zastosowanie listy stanów nieprzetworzonych (NSL) w celu umożliwienia algorytmowi powrotu do dowolnego z tych stanów.
2. Lista stanów „złych” (DE), aby uniemożliwić algorytmowi ponawianie niepotrzebnych ścieżek.
3. Lista węzłów (SL) w bieżącej ścieżce rozwiązania, która jest zwracana w przypadku znalezienia celu.
4. Jawne kontrole członkostwa nowych stanów w tych listach, aby zapobiec zapętleniu.

W następnej sekcji przedstawiono algorytmy wyszukiwania, które podobnie jak ścieżka wsteczna używają list do śledzenia stanów w przestrzeni wyszukiwania. Algorytmy te, w tym wyszukiwanie w pierwszej kolejności, w pierwszej kolejności i w pierwszej kolejności, różnią się od ścieżki wstecznej, zapewniając bardziej elastyczną podstawę do wdrażania alternatywnych strategii wyszukiwania grafów

Przeszukiwanie w głąb i przeszukiwanie wszerz

Oprócz określenia kierunku wyszukiwania (na podstawie danych lub celu) algorytm wyszukiwania musi określić kolejność, w jakiej stany są badane w drzewie lub na wykresie. W tej sekcji omówiono dwie możliwości kolejności, w jakiej brane są pod uwagę węzły wykresu: wyszukiwanie od głębokości do pierwszej i szerokość do pierwszej. Rozważ wykres przedstawiony na rysunku



Stany są oznaczone (A, B, C, ...), aby można było do nich odwoływać się w dalszej dyskusji. W głębokich poszukiwaniach, gdy stan jest badany, wszystkie jego dzieci i ich potomkowie są badani przed jakimkolwiek rodzeństwem. Zawsze, gdy jest to możliwe, wyszukiwanie wgłębne idzie głębiej w przestrzeń wyszukiwania. Tylko wtedy, gdy nie można znaleźć dalszych potomków państwa, rozważa się jego rodzeństwo. Wyszukiwanie w pierwszej kolejności analizuje stany na powyższym wykresie w kolejności A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R. Algorytm cofania zaimplementował wyszukiwanie w pierwszej kolejności. Natomiast pierwsze wyszukiwanie szerokości bada przestrzeń w sposób poziom po poziomie. Tylko wtedy, gdy nie ma więcej stanów do zbadania na danym poziomie, algorytm przechodzi do następnego, głębszego poziomu. Pierwsze wyszukiwanie wykresu na powyższym rysunku uwzględnia stany w kolejności A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q , R, S, T, U. Wdrażamy szerokie wyszukiwanie za pomocą list, otwartych i zamkniętych, aby śledzić postęp w przestrzeni stanu. open, podobnie jak NSL w backtracku, zawiera listę stanów, które zostały wygenerowane, ale których dzieci nie zostały zbadane. Kolejność usuwania stanów z otwartych określa kolejność wyszukiwania. zamknięte zapisy stanów już sprawdzone. zamknięte jest połączenie list DE i SL algorytmu cofania.

```
function breadth_first_search;
begin
open := [Start]; % initialize
closed := [ ];
while open &ne; [ ] do % states remain
begin
remove leftmost state from open, call it X;
if X is a goal then return SUCCESS % goal found
else begin
generate children of X;
put X on closed;
discard children of X if already on open or closed; % loop check
put remaining children on right end of open % queue
end
end
return FAIL % no states left
end.
```

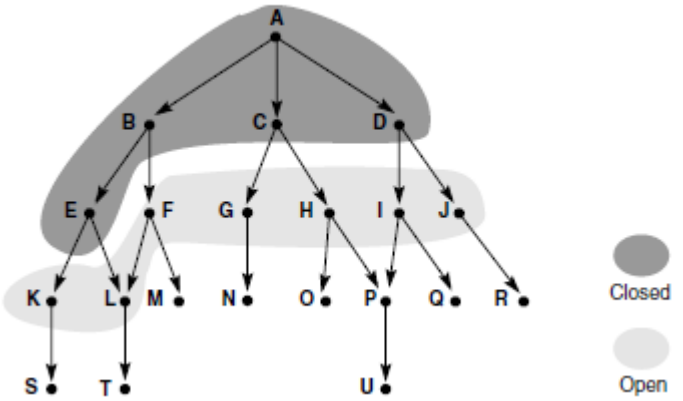
Stany potomne są generowane na podstawie reguł wnioskowania, legalnych ruchów gry lub innych operatorów zmiany stanu. Każda iteracja powoduje wygenerowanie wszystkich dzieci

stanu X i dodaje je do otwarcia. Należy pamiętać, że struktura otwarta jest utrzymywana jako kolejka lub struktura danych FIFO. Stany są dodawane po prawej stronie listy i usuwane z lewej strony. To tendencyjność wyszukiwania w kierunku stanów, które były najdłużej otwarte, powodując, że wyszukiwanie było pierwsze. Stany potomne, które zostały już odkryte (już pojawiają się albo jako otwarte, albo zamknięte) są odrzucane. Jeśli algorytm zakończy się, ponieważ warunek pętli „while" nie jest już spełniony (open = []), wówczas przeszukał cały wykres bez znalezienia pożądanego celu: wyszukiwanie nie powiodło się. Poniżej przedstawiono ślad analizy szerokości na pierwszym wykresie na powyższym rysunku. Każda kolejna liczba, 2,3,4,. . . , reprezentuje iterację pętli „while". U jest stanem docelowym.

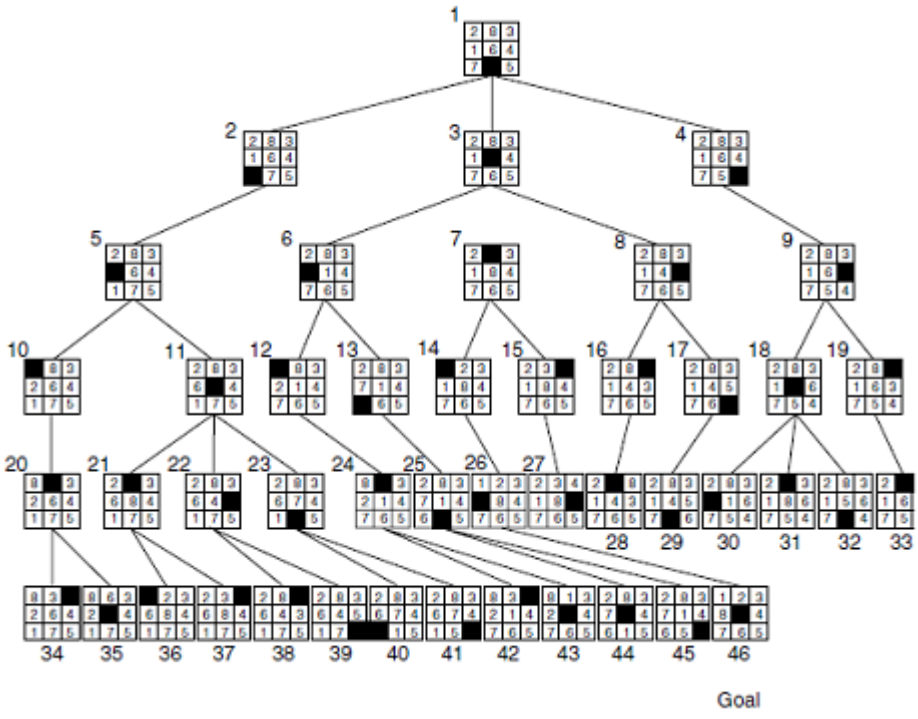
1. otwarte = [A]; zamknięte = []
2. otwarte = [B, C, D]; zamknięte = [A]
3. otwarte = [C, D, E, F]; zamknięte = [B, A]
4. otwarte = [D, E, F, G, H]; zamknięte = [C, B, A]
5. otwarte = [E, F, G, H, I, J]; zamknięte = [D, C, B, A]
6. otwarte = [F, G, H, I, J, K, L]; zamknięte = [E, D, C, B, A]
7. otwarte = [G, H, I, J, K, L, M] (ponieważ L jest już otwarte); zamknięte = [F, E, D, C, B, A]
8. otwarte = [H, I, J, K, L, M, N]; zamknięte = [G, F, E, D, C, B, A]
9. i tak dalej, dopóki nie zostanie znalezione U lub otwarte = [].

Rysunek poniżej ilustruje wykres. Powyżej po sześciu iteracjach breadth_first_search. Stany na otwartym i zamkniętym są podświetlone. Algorytmy nie wykryły stanów, które nie są zacienione. Zauważ, że open zapisuje stany na „granicy" wyszukiwania na dowolnym etapie oraz że zapisy stanów już odwiedzonych. Ponieważ pierwsze wyszukiwanie szerokości uwzględnia każdy węzeł na każdym poziomie wykresu przed przejściem głębiej w przestrzeń, wszystkie stany są najpierw osiągane najkrótszą ścieżką od stanu początkowego. Gwarantuje to, że pierwsze wyszukiwanie szerokości pozwoli znaleźć najkrótszą drogę od stanu początkowego do celu. Ponadto, ponieważ wszystkie stany są najpierw znalezione wzdłuż najkrótszej ścieżki, wszystkie stany napotkane po raz drugi znajdują się wzdłuż ścieżki o równej lub większej długości. Ponieważ nie ma szans, aby zduplikowane stany zostały znalezione na lepszej ścieżce, algorytm po prostu odrzuca wszelkie zduplikowane stany. Często przydatne jest utrzymywanie innych informacji otwartych i zamkniętych oprócz nazw stanów. Na przykład zwróć uwagę, że breadth_first_search nie utrzymuje listy stanów na bieżącej ścieżce do celu, tak jak cofnięcie na liście SL; wszystkie odwiedzane stany są zamknięte. Jeśli wymagana jest ścieżka rozwiązania, algorytm nie może jej zwrócić. Rozwiązanie można znaleźć, przechowując informacje o przodkach wraz z każdym stanem. Stan może zostać zapisany wraz z zapisem jego stanu nadrzędnego, np. Jako para (stan, nadrzędny). Jeśli zrobi się to w poszukiwaniu powyższego rysunku, zawartość otwartej i zamkniętej czwartej iteracji będzie następująca:

open = [(D, A), (E, B), (F, B), (G, C), (H, C)]; zamknięte = [(C, A), (B, A), (A, zero)]



Ścieżka (A, B, F), która prowadziła od A do F, można łatwo zbudować z tych informacji. Po znalezieniu celu algorytm może skonstruować ścieżkę rozwiązania, śledząc rodziców wzdłuż celu od stanu początkowego. Zauważ, że stan A ma wartość nadrzędną zero, co wskazuje, że jest to stan początkowy; to zatrzymuje rekonstrukcję ścieżki. Bo wyszukiwanie w pierwszej kolejności znajduje każdy stan na najkrótszej ścieżce i zachowuje pierwszą wersję każdego stanu, jest to najkrótsza ścieżka od początku do celu. Rysunek pokazuje stany usunięte z otwartej i zbadane w szerokiej analizie wykresu 8-puzzli.



Jak poprzednio, łuki odpowiadają ruchom pustego miejsca w górę, w prawo, w dół i w lewo. Liczba obok każdego stanu wskazuje kolejność, w jakiej został usunięty z otwartego. Stany pozostawione otwarte, gdy algorytm zatrzymany nie są wyświetlane. Następnie tworzymy algorytm wyszukiwania w pierwszej kolejności, uproszczenie algorytmu cofania, który został już wcześniej przedstawiony. W tym algorytmie dodawane są stany potomne i usunięte z lewego końca open: open jest utrzymywany jako stos lub struktura Last-First-First-Out (LIFO).

Organizacja open jako stosu kieruje wyszukiwanie w kierunku ostatnio wygenerowanych stanów, tworząc kolejność wyszukiwania od pierwszej głębokości:

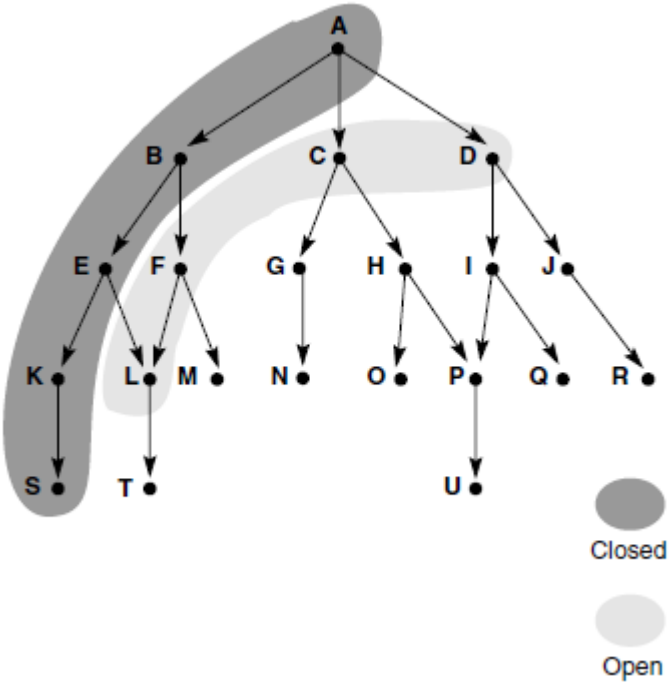
```
function depth_first_search;
begin
open := [Start]; % initialize
closed := [ ];
while open ≠ [ ] do % states remain
begin
remove leftmost state from open, call it X;
if X is a goal then return SUCCESS % goal found
else begin
generate children of X;
put X on closed;
discard children of X if already on open or closed; % loop check
put remaining children on left end of open % stack
end
end;
return FAIL % no states left
end.
```

Ślad głębi_pierwszego_wyszukiwania na wykresie na rysunku pojawia się poniżej. Każda kolejna iteracja pętli „while" jest oznaczona pojedynczą linią (2, 3, 4, ...). Początkowe stany otwarcia i zamknięcia podano w linii 1. Załóżmy, że U jest stanem celu.

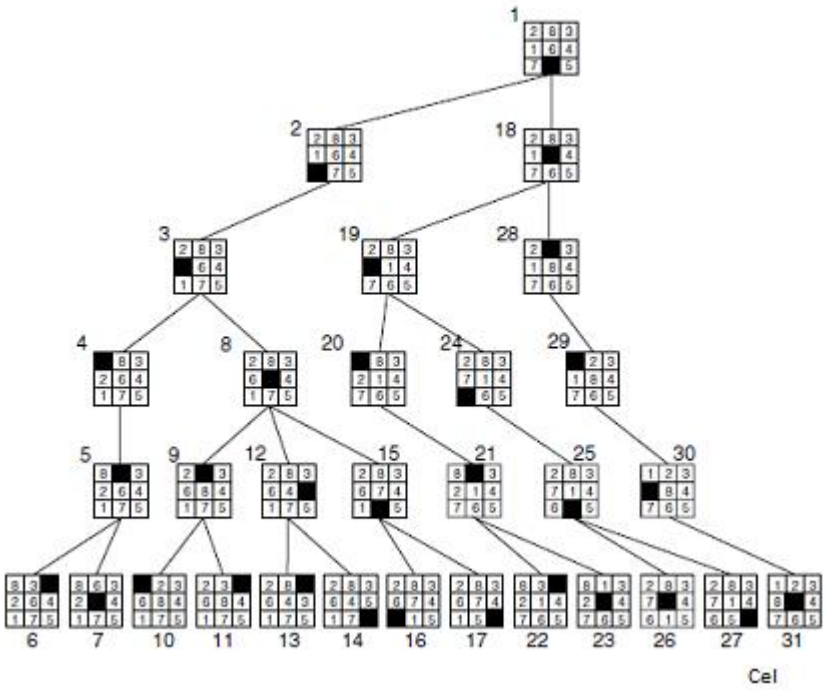
1. otwarte = [A]; zamknięte = []
2. otwarte = [B, C, D]; zamknięte = [A]
3. otwarte = [E, F, C, D]; zamknięte = [B, A]
4. otwarte = [K, L, F, C, D]; zamknięte = [E, B, A]
5. otwarte = [S, L, F, C, D]; zamknięte = [K, E, B, A]
6. otwarte = [L, F, C, D]; zamknięte = [S, K, E, B, A]
7. otwarte = [T, F, C, D]; zamknięte = [L, S, K, E, B, A]
8. otwarte = [F, C, D]; zamknięte = [T, L, S, K, E, B, A]
9. otwarte = [M, C, D], (ponieważ L jest już zamknięty); zamknięte = [F, T, L, S, K, E, B, A]
10. otwarte = [C, D]; zamknięte = [M, F, T, L, S, K, E, B, A]
11. otwarte = [G, H, D]; zamknięte = [C, M, F, T, L, S, K, E, B, A]

i tak dalej, dopóki albo U nie zostanie wykryte, albo otwarte = [].

Podobnie jak w przypadku breadth_first_search, open wyświetla wszystkie stany wykryte, ale jeszcze nie ocenione (obecna „granica" wyszukiwania), a stany zamkniętych rekordów zostały już uwzględnione. Rysunek poniżej pokazuje wykres .Wcześniej przy szóstej iteracji głębi_pierwszego_wyszukiwania.



Zawartość otwartych i zamkniętych jest podświetlona. Podobnie jak w przypadku analizy szerokości_pierwszego, algorytm może przechowywać rekord rodzica wraz z każdym stanem, umożliwiając algorytmowi odtworzenie ścieżki prowadzącej od stanu początkowego do celu. W przeciwieństwie do pierwszego wyszukiwania szerokości, pierwsze wyszukiwanie głębokości nie gwarantuje znalezienia najkrótszej ścieżki do stanu przy pierwszym napotkaniu tego stanu. W dalszej części wyszukiwania można znaleźć inną ścieżkę do dowolnego stanu. Jeśli długość ścieżki ma znaczenie w rozwiązaniu problemu, gdy algorytm napotka zduplikowany stan, algorytm powinien zapisać wersję osiągniętą wzdłuż najkrótszej ścieżki. Można to zrobić, przechowując każdy stan jako potrójny: (stan, rodzic długość_ścieżki). Gdy generowane są dzieci, wartość długości ścieżki jest po prostu zwiększana o jeden i zapisywana wraz z dzieckiem. Jeśli dziecko zostanie osiągnięte wieloma ścieżkami, informacje te można wykorzystać do zachowania najlepszej wersji. Jest to omówione bardziej szczegółowo w omówieniu algorytmu A wcześniej. Pamiętaj, że zachowanie najlepszej wersji stanu w prostym wyszukiwaniu z głębokością pierwszą nie gwarantuje osiągnięcia celu najkrótszą ścieżką. Rysunek poniższy daje wyszukiwanie z 8 puzzli w pierwszej kolejności.



Jak wspomniano wcześniej, przestrzeń jest generowana przez cztery reguły „przenieś puste” (w górę, w dół, w lewo i w prawo). Liczby obok stanów wskazują kolejność, w jakiej były one uwzględniane, tj. Usuwane z otwartych. Stany pozostawione otwarte po znalezieniu celu nie są wyświetlane. Na poszukiwania narzucono granicę głębokości 5, aby nie zgubić się głęboko w przestrzeni. Podobnie jak w przypadku wyboru między wyszukiwaniem opartym na danych i celu w celu oceny wykresu, wybór wyszukiwania w pierwszej kolejności lub w pierwszej kolejności zależy od konkretnego rozwiązanego problemu. Istotne cechy obejmują znaczenie znalezienia najkrótszej ścieżki do celu, czynnik rozgałęzienia przestrzeni, dostępny czas obliczeniowy i zasoby przestrzeni, średnią długość ścieżek do węzła celu oraz czy chcemy wszystkich rozwiązań, czy tylko pierwszego rozwiązania . Podejmowanie tych decyzji ma zalety i wady każdego podejścia.

Wszerz. Ponieważ zawsze bada wszystkie węzły na poziomie n przed przejściem do poziomu n 1, wyszukiwanie w pierwszej kolejności zawsze znajduje najkrótszą ścieżkę do węzła celu. W przypadku problemu, w którym wiadomo, że istnieje proste rozwiązanie, to rozwiązanie zostanie znalezione. Niestety, jeśli występuje zły czynnik rozgałęziający, tj. Stany mają wysoką średnią liczbę dzieci, eksplozja kombinatoryczna może uniemożliwić algorytmowi znalezienie rozwiązania wykorzystującego dostępną pamięć. Wynika to z faktu, że wszystkie nierozwinięte węzły dla każdego poziomu wyszukiwania muszą być otwarte. W przypadku głębokich wyszukiwań lub przestrzeni stanów o wysokim współczynniku rozgałęzienia może to być dość kłopotliwe. Wykorzystanie przestrzeni w wyszukiwaniu na szerokość, mierzone liczbą stanów w stanie otwartym, jest wykładniczą funkcją długości ścieżki w dowolnym momencie. Jeśli każdy stan ma średnio B dzieci, liczba stanów na danym poziomie jest B razy większa od liczby stanów na poprzednim poziomie. Daje to stany B^n na poziomie n. Poszerzenie po raz pierwszy umieściłoby je wszystkie w pozycji otwartej, gdy rozpocznie badanie poziomu n. Może to być przeszkodą, jeśli ścieżki rozwiązania są długie, na przykład w grze w szachy

W głąb. Wyszukiwanie w pierwszej kolejności szybko przechodzi w głęboką przestrzeń wyszukiwania. Jeśli wiadomo, że ścieżka rozwiązania będzie długa, wyszukiwanie w pierwszej głębokości nie marnuje czasu na wyszukiwanie dużej liczby „płytkich” stanów na wykresie. Z drugiej strony, wyszukiwanie w pierwszej głębokości może „zagubić się” głęboko na wykresie,

tracąc krótsze ścieżki do celu lub nawet utknąć na nieskończenie długiej ścieżce, która nie prowadzi do celu. Wyszukiwanie w głąb jest znacznie bardziej wydajne dla przestrzeni wyszukiwania z wieloma gałęziami, ponieważ nie musi utrzymywać wszystkich węzłów na danym poziomie na otwartej liście. Wykorzystanie przestrzeni do wyszukiwania w pierwszej głębokości jest liniową funkcją długości ścieżki. Na każdym poziomie open zachowuje tylko dzieci jednego stanu. Jeśli wykres ma średnią B potomków na stan, wymaga to całkowitego wykorzystania przestrzeni przez stany B × n, aby przejść n poziomów w głąb przestrzeni. Najlepszą odpowiedzią na pytanie „przede wszystkim głębokość kontra pierwsze" jest zbadanie przestrzeni problemów i skonsultowanie się z ekspertami w tej dziedzinie. Na przykład w szachach wyszukiwanie z pełną szerokością po prostu nie jest możliwe. W prostszych grach przeszukiwanie pierwszego miejsca jest nie tylko możliwe, ale ponieważ zapewnia najkrótszą ścieżkę, może być jedynym sposobem uniknięcia przegranej

Wyszukiwanie w głąb z iteracyjnym pogłębianiem

Dobrym kompromisem jest użycie głębokości związanej z wyszukiwaniem głębokościowym. Ograniczenie głębokości wymusza niepowodzenie na ścieżce wyszukiwania, gdy zejdzie poniżej określonego poziomu. Powoduje to, że obszar poszukiwań na tym poziomie głębokości zajmuje pierwsze miejsce. Kiedy wiadomo, że rozwiązanie leży w pewnej głębokości lub gdy ograniczenia czasowe, takie jak te występujące na bardzo dużej przestrzeni, takiej jak szachy, ogranicz liczbę stanów, które można rozważyć; wówczas wyszukiwanie najbardziej głębokie z ograniczeniem głębokości może być najbardziej odpowiednie. Ten wgląd prowadzi do algorytmu wyszukiwania, który eliminuje wiele wad wyszukiwania zarówno w pierwszej kolejności, jak i w pierwszej kolejności. Iteracyjne pogłębianie od pierwszej głębokości wykonuje pierwsze wyszukiwanie głębokości w przestrzeni z ograniczeniem głębokości 1. Jeśli nie uda się znaleźć celu, przeprowadza kolejne wyszukiwanie od głębokości z granicą głębokości wynoszącą 2. To trwa, zwiększając granicę głębokości po jednym za każdym razem. Przy każdej iteracji algorytm wykonuje pełne wyszukiwanie od głębokości do bieżącego ograniczenia głębokości. Pomiędzy iteracjami nie są zachowywane żadne informacje o przestrzeni stanów. Ponieważ algorytm przeszukuje przestrzeń w sposób poziom po poziomie, gwarantuje się znalezienie najkrótszej ścieżki do celu. Ponieważ wykonuje tylko wyszukiwanie w pierwszej kolejności w każdej iteracji, użycie miejsca na dowolnym poziomie n wynosi B × n, gdzie B jest średnią liczbą elementów potomnych węzła. Co ciekawe, chociaż wydaje się, że iteracyjne pogłębianie w pierwszej głębokości byłoby znacznie mniej wydajne niż wyszukiwanie w pierwszej lub w głębokości, jego złożoność czasowa jest w rzeczywistości tego samego rzędu wielkości co jedno z nich: $O(B^n)$. Intuicyjne wyjaśnienie tego pozornego paradoksu podaje Korf: Ponieważ liczba węzłów na danym poziomie drzewa rośnie wykładniczo wraz z głębokością, prawie cały czas spędza się na najgłębszym poziomie, mimo że płytsze poziomy generują arytmetycznie rosnącą liczbę czasów. Niestety, wszystkie strategie wyszukiwania omówione w tym rozdziale: głębokość pierwsza, szerokość pierwsza i głębokość iteracyjna pogłębianie - mogą wykazywać złożoność wykładniczą w najgorszym przypadku. Dotyczy to wszystkich niedoinformowanych algorytmów wyszukiwania. Jedyne podejścia do wyszukiwania, które zmniejszają tę złożoność, wykorzystują heurystykę do kierowania wyszukiwaniem. Wyszukiwanie według najlepszego wyniku jest algorytmem wyszukiwania podobnym do przedstawionych właśnie algorytmów wyszukiwania według głębokości i szerokości. Jednak wyszukiwanie według najlepszego według kolejności porządkuje stany na otwartej liście, będące aktualnym skrajem wyszukiwania, zgodnie z pewną

miarą ich heurystycznych zalet. Przy każdej iteracji nie bierze pod uwagę ani najgłębszego, ani płytszego stanu, ale stan „najlepszy".

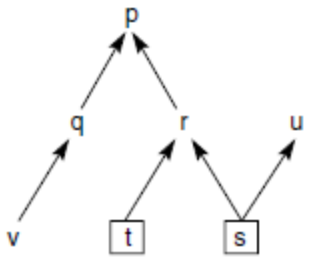
Wykorzystanie przestrzeni stanu do reprezentowania rozumowania za pomocą rachunku zdań i predykatów

Przestrzeń stanu. Opis układu logicznego

Kiedy zdefiniowaliśmy wykresy przestrzeni stanów, zauważyliśmy, że węzły muszą się od siebie odróżniać, przy czym każdy węzeł reprezentuje pewien stan procesu rozwiązania. Rachunek zdań i predykatów można wykorzystać jako formalny język specyfikacji do dokonywania tych rozróżnień, a także do mapowania węzłów wykresu na przestrzeń stanu. Ponadto reguły wnioskowania można wykorzystać do tworzenia i opisywania łuków między stanami. W ten sposób problemy w rachunku predykatów, takie jak ustalenie, czy dane wyrażenie jest logiczną konsekwencją danego zestawu twierdzeń, można rozwiązać za pomocą wyszukiwania. Rzetelność i kompletność reguł wnioskowania rachunku predykatów może zagwarantować poprawność wniosków wynikających z tej formy wnioskowania opartego na grafie. Ta zdolność do formalnego udowodnienia integralności rozwiązania za pomocą tego samego algorytmu, który tworzy rozwiązanie, jest unikalnym atrybutem wielu sztucznej inteligencji i rozwiązywania problemów opartych na twierdzeniach. Chociaż stany wielu problemów, np. Kółko i krzyżyk, można w bardziej naturalny sposób opisać innymi strukturami danych, takimi jak tablice, moc i ogólność logiki pozwalają na rozwiązywanie problemów AI przy użyciu opisów i wnioskowania rachunku zdań i predykatów zasady. Inne reprezentacje AI, takie jak reguły, sieci semantyczne lub ramki, również wykorzystują strategie wyszukiwania podobne do tych przedstawionych wcześniej

PRZYKŁAD 3.3.1: KALKULACJA WNIOSKÓW

Pierwszy przykład, w jaki sposób zestaw zależności logicznych można postrzegać jako definiujący wykres, pochodzi z rachunku zdań. Jeśli p, q, r, … są zdaniami, przyjmij twierdzenia:
$q \rightarrow p$
$r \rightarrow p$
$v \rightarrow q$
$s \rightarrow r$
$t \rightarrow r$
$s \rightarrow u$
s
t
Z tego zestawu twierdzeń i modus ponens reguły reguły wnioskowania można wywnioskować pewne zdania (p, r i u); inne (takie jak v i q) mogą nie być tak wywnioskowane i faktycznie nie logicznie wynikają z tych twierdzeń. Zależność między początkowymi twierdzeniami a tymi wnioskami jest wyrażona w ukierunkowanym wykresie na rysunku



Na powyższym rysunku łuki odpowiadają logicznym implikacjom (→). Twierdzenia podane jako prawda (s) odpowiadają danym danych problemu. Twierdzenia, które są logicznymi konsekwencjami tego zestawu twierdzeń, odpowiadają węzłom, które można osiągnąć wzdłuż ukierunkowanej ścieżki ze stanu reprezentującego prawdziwą propozycję; taka ścieżka odpowiada sekwencji zastosowań modus ponens. Na przykład ścieżka [s, r, p] odpowiada sekwencji wnioskowania:
s i s → r daje r.
r i r → p daje p.
Biorąc pod uwagę tę reprezentację, ustalenie, czy dana propozycja jest logiczną konsekwencją zbioru propozycji, staje się problemem znalezienia ścieżki od węzła pudełkowego (węzła początkowego) do propozycji (węzła celu). W związku z tym zadanie można rzutować jako problem z wyszukiwaniem wykresów. Stosowana tutaj strategia wyszukiwania opiera się na danych, ponieważ idzie od tego, co znane (prawdziwe propozycje) do celu. Alternatywnie, strategia zorientowana na cel może być zastosowana do tej samej przestrzeni stanu, zaczynając od twierdzenia, które ma zostać udowodnione (cel) i szukając wstecz wzdłuż łuków, aby znaleźć poparcie dla celu wśród prawdziwych zdań. Możemy również przeszukiwać tę przestrzeń wnioskowania w pierwszej kolejności lub w pierwszej kolejności.

Wykresy AND / OR

W przykładzie rachunku zdań wszystkie twierdzenia były implikacjami formy p → q. Nie dyskutowaliśmy o sposobie, w jaki operatory logiczne i i lub mogą być reprezentowane na takim wykresie. Wyrażenie zależności logicznych zdefiniowanych przez tych operatorów wymaga rozszerzenia podstawowego modelu grafu zdefiniowanego w rozdziale 3.1 na to, co nazywamy grafem i / lub I / lub wykresy są ważnym narzędziem do opisywania przestrzeni wyszukiwania generowanych przez wiele problemów AI, w tym rozwiązanych za pomocą logicznych twierdzeń dowodowych i systemów eksperckich.

W wyrażeniach postaci q ∧ r → p, zarówno q i r muszą być prawdziwe, aby p było prawdziwe. W wyrażeniach postaci q ∨ r → p, prawda q lub r jest wystarczająca do udowodnienia, że p jest prawdziwe. Ponieważ implikacje zawierające przesłanki rozłączne można zapisać jako osobne implikacje, to wyrażenie jest często pisane jako q →p, r → p. Aby przedstawić te różne relacje w formie graficznej i / lub wykresy, rozróżnij pomiędzy i węzły i / lub węzły. Jeśli przesłanki

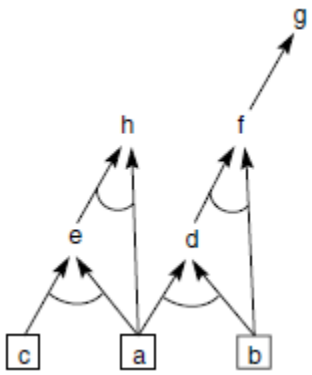
implikacji są połączone przez operatora ∧, są one wywoływane i węzły na wykresie, a łuki z tego węzła są połączone zakrzywionym łączem. Wyrażenie q ∧ r → p jest reprezentowane przez i / lub wykres na rycinie poniżej. Łącze łączące łuki na poniższym rysunku odzwierciedla ideę, że zarówno q i r muszą być prawdziwe, aby udowodnić p. Jeśli pomieszczenia są połączone przez operatora lub, są one traktowane jako lub węzły na wykresie. Łuki z lub węzły do ich węzła nadrzędnego nie są tak połączone tu

To oddaje pogląd, że prawda któregokolwiek z założeń jest niezależnie wystarczająca do ustalenia prawdziwości wniosku. Wykres i / lub wykres jest w rzeczywistości specjalizacją typu wykresu znanego jako hipergraf, który łączy węzły za pomocą zestawów łuków, a nie pojedynczych łuków. Hipergraf
definiuje się w następujący sposób:

DEFINICJA
HIPERGRAF

Hipergraph składa się z:
N, zbiór węzłów.
H, zestaw hiperkątów zdefiniowany przez uporządkowane pary, w którym pierwszy element pary jest pojedynczym węzłem od N, a drugi element jest podzbiorem N. Zwykły wykres jest szczególnym przypadkiem hipergrafu, w którym wszystkie zestawy potomków węzły mają liczność 1. Hiperkąty są również znane jako k-łączniki, gdzie k jest licznością zbioru węzłów potomnych. Jeśli k = 1, potomek jest uważany za węzeł lub. Jeśli k > 1, elementy zbioru potomków można traktować jako węzły i węzły. W takim przypadku łącznik jest rysowany między poszczególnymi krawędziami od elementu nadrzędnego do każdego z węzłów podrzędnych.

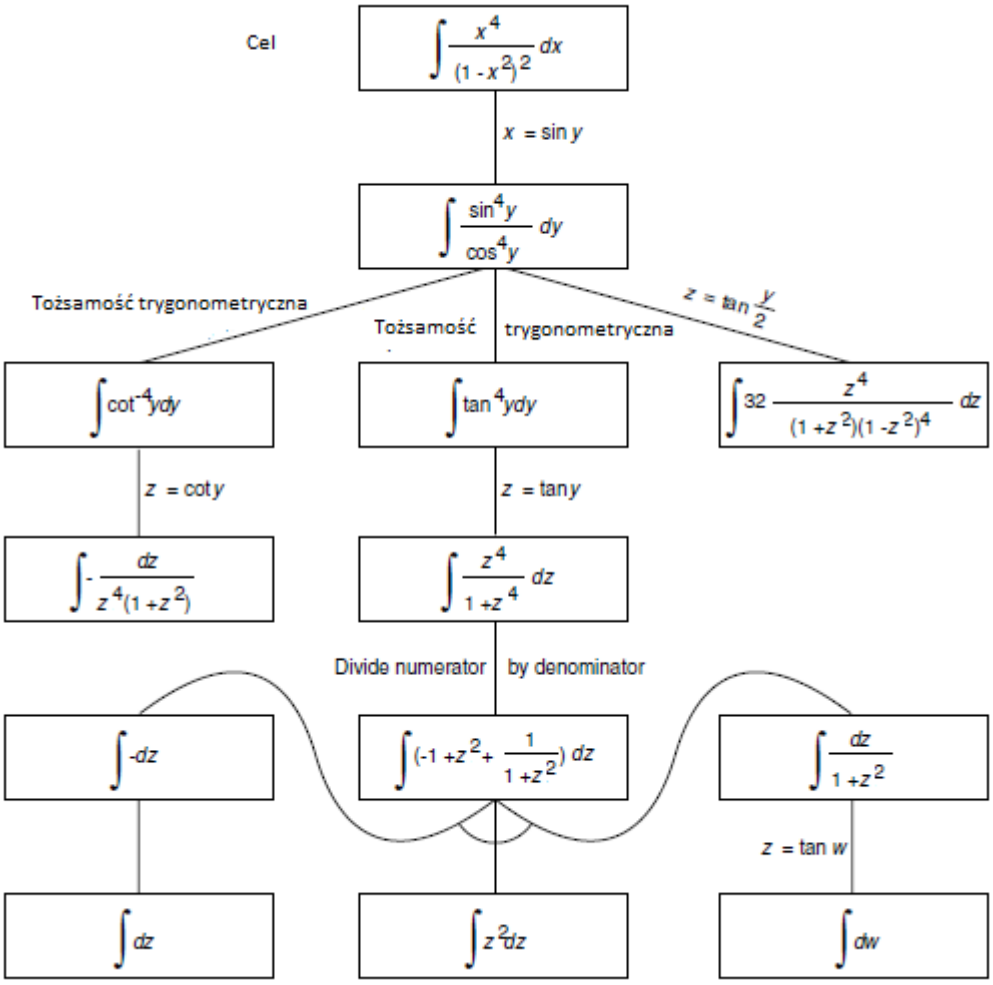
PRZYKŁAD 3.3.2: AND/ OR WYSZUKIWANIE
Przykład pochodzi z rachunku zdań, ale generuje wykres, który zawiera zarówno i, jak i potomków. Załóżmy, że prawdziwe są następujące zdania:
a
b
c
a ∧ b → d
a ∧ c → e
b ∧ d → f
f → g
a ∧ e → h
Ten zestaw twierdzeń generuje wykres i / lub wykres na rysunku



Pytania, które mogą być zadane (odpowiedzi wydedukowane podczas wyszukiwania tego wykresu) to:
1. Czy h jest prawdą?
2. Czy h jest prawdą, jeśli b nie jest już prawdą?
3. Jaka jest najkrótsza ścieżka (tj. Najkrótsza sekwencja wnioskowania), która pokazuje, że X (niektóre zdania) jest prawdziwe?
4. Pokaż, że twierdzenie p (zauważ, że p nie jest obsługiwane) jest fałszywe.
Co to znaczy? Co byłoby konieczne, aby dojść do tego wniosku? Wykres wyszukiwania AND/OR wymaga tylko nieco więcej przechowywania danych niż wyszukiwanie na zwykłych wykresach, czego przykładem był wcześniej omówiony algorytm cofania. Potomkowie lub są sprawdzani tak, jak byli na ścieżce powrotnej: po znalezieniu ścieżki łączącej cel z początkowym węzłem wzdłuż lub węzłami problem zostanie rozwiązany. Jeśli ścieżka prowadzi do awarii, algorytm może cofnąć się i wypróbować inną gałąź. Jednak przy wyszukiwaniu i węzłach wszyscy i potomkowie węzła muszą zostać rozwiązani (lub okazali się prawdą), aby rozwiązać węzeł nadrzędny. W powyższym przykładzie strategia zorientowana na cel w celu ustalenia prawdy pierwszych prób udowodnienia zarówno a, jak i e. Prawda a jest bezpośrednia, ale prawda e wymaga prawdy zarówno c, jak i a; są one podane jako prawdziwe. Po rozwiązaniu problemu przez wszystkie te łuki do prawdziwych zdań, prawdziwe wartości są ponownie łączone w węzłach i, aby zweryfikować prawdziwość h. Z drugiej strony, ukierunkowana na dane strategia określania prawdy h rozpoczyna się od znanych faktów (c, a i b) i zaczyna dodawać nowe propozycje do tego zestawu znanych faktów zgodnie z ograniczeniami i / lub wykres. e lub d może być pierwszą propozycją dodaną do zbioru faktów. Te dodatki umożliwiają wnioskowanie o nowych faktach. Proces ten trwa do momentu udowodnienia pożądanego celu h. Jednym ze sposobów patrzenia i / lub przeszukiwania wykresów jest to, że operator ((stąd i węzły wykresu) wskazuje rozkład problemu, w którym problem jest podzielony na podproblemy, tak że wszystkie podproblemy muszą zostać rozwiązane, aby rozwiązać pierwotny problem . Operator in w rachunku predykatów reprezentującym problem wskazuje wybór, punkt w rozwiązaniu problemu, w którym można dokonać wyboru między alternatywnymi ścieżkami lub strategiami rozwiązywania problemów, z których każda, jeśli się powiedzie, jest wystarczająca do rozwiązania problemu problem

PRZYKŁAD 3.3.3: MACSYMA
Jednym z naturalnych przykładów wykresu AND/OR wykresu jest program do symbolicznej integracji funkcji matematycznych. MACSYMA to dobrze znany program, który jest szeroko stosowany przez matematyków. Argumentacja MACSYMA może być reprezentowana jako wykres i / lub wykres. Podczas wykonywania integracji jedna ważna klasa strategii obejmuje rozbicie wyrażenia na podwyrażenia, które mogą być zintegrowane niezależnie od siebie, a

wynik jest łączony algebraicznie w wyrażenie rozwiązania. Przykłady tej strategii obejmują zasadę całkowania przez części i dekompozycji całki sumy na sumę całek poszczególnych terminów. Strategie te, reprezentujące rozkład problemu na niezależne podproblemy, mogą być reprezentowane przez i węzły na wykresie. Inna klasa strategii obejmuje uproszczenie wyrażenia poprzez różne podstawienia algebraiczne. Ponieważ każde wyrażenie może dopuszczać wiele różnych podstawień, z których każde reprezentuje niezależną strategię rozwiązania, strategie te są reprezentowane przez lub węzły wykresu. Rysunek ilustruje przestrzeń poszukiwaną przez takie narzędzie do rozwiązywania problemów.

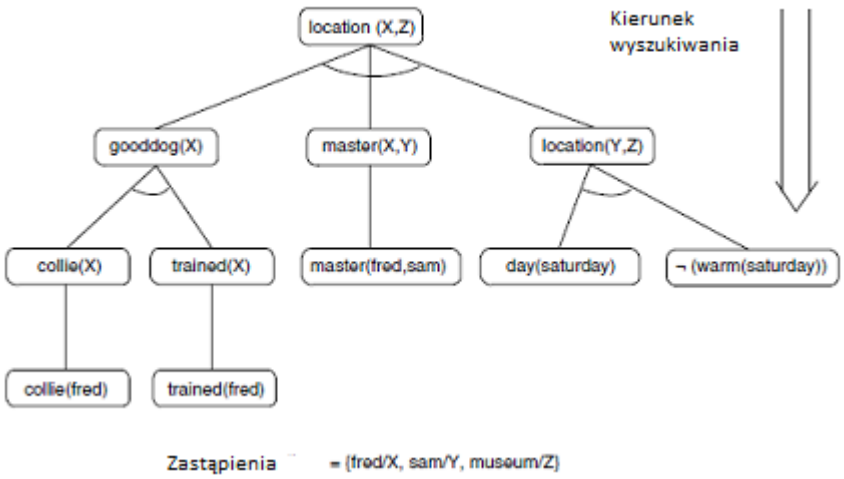


Wyszukiwanie tego wykresu jest ukierunkowane na cel, ponieważ zaczyna się od zapytania „znajdź całkę określonej funkcji” i powraca do wyrażeń algebraicznych, które ją definiują. Pamiętaj, że jest to przykład, w którym wyszukiwanie ukierunkowane na cel jest oczywistą strategią. Rozwiązanie problemu byłoby praktycznie niemożliwe, aby określić wyrażenia algebraiczne, które utworzyły pożądaną całkę, bez cofania się od zapytania.

PRZYKŁAD 3.3.4: WYKORZYSTANIE CELU I / LUB WYSZUKIWANIE

Ten przykład pochodzi z rachunku predykatów i reprezentuje wyszukiwanie wykresów zależne od celu, w którym celem, który ma zostać potwierdzony w tej sytuacji, jest wyrażenie rachunku predykatu zawierające zmienne. Aksjomaty to logiczne opisy relacji między psem Fredem i jego panem Samem. Zakładamy, że zimny dzień nie jest ciepłym dniem, pomijając kwestie takie jak złożoność dodana przez równoważne wyrażenia dla predykatów, zagadnienie omówione dalej.

Fakty i zasady tego przykładu podano w formie zdań angielskich, po których następuje ich odpowiednik rachunku predykatu:
1. Fred jest collie.
collie (fred).
2. Sam jest panem Freda.
master (fred, sam).
3. Dzień to saturday.
dzień (saturday).
4. W saturday jest zimno.
¬(ciepło (saturday)).
5. Fred jest trained.
trained (Fred).
6. Spaniele są dobrymi psami, podobnie jak wyszkolone psy.
∀ X [spaniel (X) ∨ (collie (X) ∧ trained (X)) → gooddog (X)]
7. Jeśli pies jest dobrym psem i ma mistrza, będzie z nim.
∀ (X, Y, Z) [gooddog (X) ∧ master (X, Y) ∧ location (Y, Z) → location (X, Z)]
8. Jeśli jest saturday i ciepło, Sam jest w parku.
(dzień (saturday) &and ciepło (saturday)) → location (sam, park).
9. Jeśli jest saturday i nie jest ciepło, to Sam jest w muzeum.
(dzień (saturday) ∧ ¬ (ciepło (saturday))) → location (sam, muzeum).
Celem jest wyrażenie location X location (fred, X), co oznacza „gdzie jest fred?" Algorytm wyszukiwania wstecznego analizuje alternatywne sposoby ustalenia tego celu: jeśli fred jest dobrym psem, a fred ma mistrza, a fred ma mistrza location, a następnie Fred jest w tej lokalizacji. Przesłanki tej reguły są następnie badane: co to znaczy być dobrym psem itp.? Proces ten trwa, konstruowanie wykresu AND/OR wykres z rysunku



Przeanalizujmy to wyszukiwanie bardziej szczegółowo, szczególnie dlatego, że jest to przykład wyszukiwania zorientowanego na cel przy użyciu rachunku predykatów i ilustruje rolę unifikacji w generowaniu przestrzeni wyszukiwania. Problemem do rozwiązania jest „gdzie jest fred?". Bardziej formalnie, można go postrzegać jako określenie podstawienia dla zmiennej X, jeśli takie zastąpienie istnieje, w którym położenie (fred, X) jest logiczną konsekwencją wstępnych twierdzeń . Gdy pożądane jest określenie lokalizacji Freda, badane są klauzule, które mają lokalizację jako swój wniosek, pierwszy to klauzula 7. Ten wniosek, location (X, Z), jest

następnie ujednolicony z lokalizacją (fred, X) przez podstawienia {fred / X, X / Z}. Przesłanki tej reguły, w ramach tego samego zestawu podstawień, tworzą i potomkowie głównego celu:
gooddog (fred) ∧ master (fred, Y) ∧ location (Y, X). Wyrażenie to można interpretować w ten sposób, że jednym ze sposobów na znalezienie Freda jest sprawdzenie, czy Fred jest dobrym psem, ustalenie, kto jest mistrzem Freda i ustalenie, gdzie jest mistrz. Początkowy cel został w ten sposób zastąpiony trzema podzadaniami. To są węzły i wszystkie z nich muszą zostać rozwiązane. Aby rozwiązać te podzadania, rozwiązywanie problemów najpierw określa, czy Fred jest dobrym pieskiem. Jest to zgodne z wnioskiem klauzuli 6 przy użyciu podstawienia {fred / X}. Założeniem klauzuli 6 jest lub dwa wyrażenia:
spaniel (fred) ∨ (collie (fred) ∧ trained (fred))
Pierwszym z nich jest spaniel (fred). Baza danych nie zawiera tego stwierdzenia, więc osoba rozwiązująca problemy musi założyć, że jest fałszywa. Drugi węzeł to (collie (fred) ∧ trained (fred)), tzn. Fred jest collie i jest trained przez Freda. Oba muszą być prawdziwe, tak jak w klauzulach 1 i 5. Dowodzi to, że gooddog (fred) jest prawdziwy. Narzędzie do rozwiązywania problemów sprawdza następnie drugą z przesłanek klauzuli 7: master (X, Y). Pod podstawieniem {fred / X} master (X, Y) staje się master (fred, Y), co łączy się z faktem (klauzula 2) master (fred, sam). Powoduje to ujednolicenie podstawienia {sam / Y}, co również nadaje wartość sam trzeciemu podrzędności klauzuli 7, tworząc nową lokalizację celu (sam, X). Aby rozwiązać ten problem, zakładając, że osoba rozwiązująca problemy próbuje uporządkować reguły, location celu (sam, X) najpierw ujednolici się z wnioskiem zawartym w klauzuli 7. Zauważ, że ta sama reguła jest wypróbowywana z różnymi powiązaniami dla X. Przypomnij, że X jest zmienną „obojętną" i może mieć dowolną nazwę (dowolny ciąg rozpoczynający się od dużej litery). Ponieważ zakres znaczenia dowolnej nazwy zmiennej jest zawarty w klauzuli, w której się pojawia, rachunek predykatów nie ma zmiennych globalnych. Innym sposobem powiedzenia tego jest to, że wartości zmiennych są przekazywane do innych klauzul jako parametry i nie mają ustalonych lokalizacji (pamięci). Tak więc wielokrotne występowanie X w różnych regułach w tym przykładzie wskazuje na różne parametry formalne. Podczas próby rozwiązania przesłanek reguły 7 za pomocą tych nowych powiązań rozwiązanie problemu zakończy się niepowodzeniem, ponieważ sam nie jest dobrym pieskiem. Tutaj wyszukiwanie cofnie się do miejsca docelowego (sam, X) i spróbuje następnego meczu, zakończenia reguły 8. To również się nie powiedzie, co spowoduje kolejny powrót i połączenie z wnioskiem z klauzuli 9, w (sam, muzeum). Ponieważ przesłanki klauzuli 9 są obsługiwane w zbiorze twierdzeń (klauzule 3 i 4), wynika z tego, że wniosek z 9 jest prawdziwy. To ostateczne zjednoczenie sięga z powrotem do drzewa, by w końcu odpowiedzieć ∃ X location (Fred, X) z lokalizacją (Fred, muzeum). Ważne jest dokładne zbadanie charakteru przeszukiwania opartego na celu wykresu i porównanie go z przeszukiwaniem danych w przykładzie wcześniej. Dalsza dyskusja na ten temat, w tym bardziej rygorystyczne porównanie tych dwóch metod wyszukiwania wykresu, jest kontynuowana w następnym przykładzie, ale jest szczegółowo opisana dopiero w dyskusji na temat systemów produkcyjnych później oraz w zastosowaniu do systemów eksperckich w. w tym przykładzie sugeruje się, że kolejność klauzul wpływa na kolejność wyszukiwania. W powyższym przykładzie wielokrotne klauzule lokalizacji zostały wypróbowane w kolejności, a wyszukiwanie za pomocą śledzenia wstecznego wyeliminowało te, których nie udało się potwierdzić

PRZYKŁAD 3.3.5: ZMIENIONY DORADCA FINANSOWY

W ostatnim przykładzie zastosowaliśmy rachunek predykatów do przedstawienia zestawu zasad udzielania porad inwestycyjnych. W tym przykładzie użyto modus ponens, aby wnioskować o właściwej inwestycji dla konkretnej osoby. Nie dyskutowaliśmy o tym, w jaki

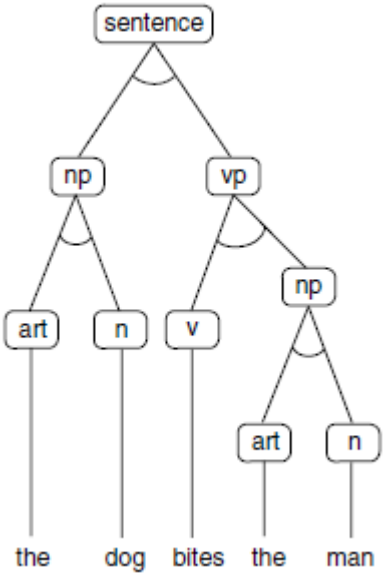
sposób program może określić odpowiednie wnioski. Jest to oczywiście problem z wyszukiwaniem; niniejszy przykład ilustruje jedno podejście do wdrożenia doradcy finansowego opartego na logice, wykorzystujące ukierunkowane na cel wyszukiwanie w pierwszej kolejności z cofaniem. W dyskusji wykorzystano znalezione wcześniej predykaty; te predykaty nie są tutaj duplikowane. Załóżmy, że dana osoba ma na utrzymaniu dwie osoby: 20 000 USD oszczędności i stały dochód w wysokości 30 000 USD. Jak omówiono wcześniej, możemy dodać wyrażenia rachunku predykatu opisujące te fakty do zbioru wyrażeń rachunku predykatu. Alternatywnie program może rozpocząć wyszukiwanie bez tych informacji i poprosić użytkownika o dodanie go w razie potrzeby. Ma to tę zaletę, że nie wymaga danych, które mogą okazać się konieczne dla rozwiązania. To podejście, często stosowane w systemach eksperckich, zostało zilustrowane tutaj. Podczas przeprowadzania konsultacji celem jest znalezienie inwestycji; jest to reprezentowane wyrażeniem rachunku predykatu investment X inwestycja (X), gdzie X w zmiennej celu chcemy powiązać. Istnieją trzy reguły (1, 2 i 3), które kończą się na inwestycjach, ponieważ zapytanie ujednolici się z wnioskiem tych reguł. Jeśli wybierzemy regułę 1 do wstępnej eksploracji, jej założenie oszczędności_konto (niewystarczające) stanie się podrzędnym, tj. Węzłem potomnym, który zostanie rozwinięty w następnej kolejności. Podczas generowania elementów potomnych konta oszczędnościowego (nieodpowiedniego) jedyną regułą, którą można zastosować, jest reguła 5. W ten sposób powstaje węzeł i:
kwota_zapisana (X) ∧ zależne (Y) ∧ ¬ większa (X, min. oszczędności (Y)).
Jeśli spróbujemy je spełnić w kolejności od lewej do prawej, kwota_zapisana (X) jest traktowana jako pierwszy podzadanie. Ponieważ system nie zawiera reguł, które kończą ten podzadanie, zapyta użytkownika. Po dodaniu kwoty_zapisanej (20000) pierwszy podzadanie zakończy się powodzeniem, z unifikacją podstawiająca X0000 na X. Zauważ, że ponieważ przeszukiwany jest węzeł i, awaria tutaj wyeliminowałaby potrzebę zbadania pozostałej części wyrażenia. Podobnie, zależności zależne od podrzędności (Y) prowadzą do zapytania użytkownika, a odpowiedź, zależności (2), jest dodawana do opisu logicznego. Podzadanie pasuje do tego wyrażenia z podstawieniem {2 / Y}. Przy tych podstawieniach wyszukiwanie następnie ocenia prawdę
¬ większy (20000, oszczędności min. (2)).
Wartość ta jest równa false, co powoduje awarię całego i węzła. Wyszukiwanie następnie cofa do węzła nadrzędnego, konto_oszczędności (nieodpowiednie) i próbuje znaleźć alternatywny sposób, aby udowodnić, że ten węzeł jest prawdziwy. Odpowiada to pokoleniu następnego dziecka w wyszukiwaniu. Ponieważ żadne inne reguły nie zawierają tego podzadania, wyszukiwanie nie powraca do celu najwyższego poziomu, inwestycji (X). Kolejną zasadą, której wnioski jednoczą się z tym celem, jest reguła 2, która tworzy nowe podzadania
konto_oszczędności (odpowiednie) ∧ dochód (odpowiedni). Kontynuując wyszukiwanie, rachunek oszczędnościowy (odpowiedni) okazał się prawdziwy, ponieważ wniosek z reguły 4, a dochód (odpowiedni) wynika z wniosku z reguły 6. Chociaż szczegóły pozostałej części wyszukiwania zostaną pozostawione czytelnikowi, i / lub pojawi się ostatecznie zbadany wykres

PRZYKŁAD 3.3.6: PARSER JĘZYKA ANGIELSKIEGO I GENERATOR ZDJĘĆ

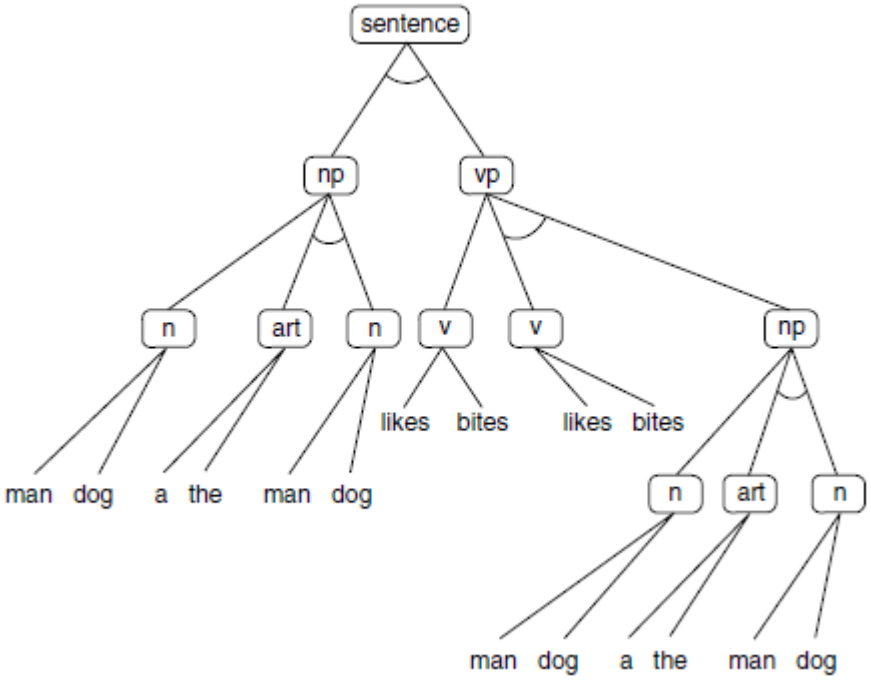
Nasz ostatni przykład nie pochodzi z rachunku predykatów, ale składa się z zestawu reguł przepisywania dla parsowania zdań w podzbiorze gramatyki angielskiej. Reguły przepisywania przyjmują wyrażenie i przekształcają je w inne, zastępując wzór po jednej stronie strzałki (↔) wzorem po drugiej stronie. Na przykład można zdefiniować zestaw reguł

przepisywania w celu zmiany wyrażenia w jednym języku, na przykład angielskim, na inny język (być może francuski lub predykatowy rachunek różniczkowy). Podane tutaj zasady przepisywania przekształcają podzbiór języka angielskiego zdania na konstrukcje gramatyczne wyższego poziomu, takie jak fraza rzeczownikowa, fraza czasownikowa i zdanie. Reguły te są używane do analizowania sekwencji słów, tj. Do ustalenia, czy są one poprawnie uformowanymi zdaniami (czy są poprawne gramatycznie, czy nie) oraz do modelowania struktury językowej zdań. Pięć zasad dla prostego podzbioru gramatyki angielskiej to:
1. sentence ↔ np vp
(Zdanie to fraza rzeczownikowa, po której następuje fraza czasownikowa.
2. np. ↔ n
(Fraza rzeczownikowa to rzeczownik.)
3. np. ↔ art n
(Fraza rzeczownikowa to artykuł, po którym następuje rzeczownik.)
4. vp ↔ v
(Fraza czasownika jest czasownikiem).
5. vp ↔ v np
(Fraza czasownika to czasownik, po którym następuje fraza rzeczownik.)
Oprócz tych reguł gramatycznych parser potrzebuje słownika słów w języku. Te słowa nazywane są terminami gramatyki. Są one zdefiniowane przez części mowy przy użyciu reguł przepisywania. W poniższym słowniku „a", „the", „man", „dog", „like" i „bites" to terminale naszej bardzo prostej gramatyki:
6. art
7. art
(„A" i „the" to artykuły)
8. n ↔ man
9. n ↔ dog
(„man" i „dog" to rzeczowniki)
10. Lubię to
11. Ugryzienia
(„likes" i „bites" są czasownikami)

Wiele reguł o takich samych wnioskach tworzy węzły OR. Zauważ, że liść OR węzła końcowego tego wykresu są angielskimi słowami w gramatyce (stąd nazywane są terminalami). Wyrażenie jest dobrze uformowane w gramatyce, jeśli składa się wyłącznie z symboli końcowych i występuje w nim szereg podstawień przy użyciu reguł przepisywania, które redukują je do symbolu zdania. Alternatywnie, może to być postrzegane jako konstruowanie parsowanego drzewa, które ma słowa wyrażenia jako liście i symbol zdania jako korzeń. Na przykład możemy przeanalizować zdanie, które pies gryzie mężczyznę, konstruując drzewo analizy z rysunku



To drzewo jest poddrzewem wykresu i / lub wykresu z rysunku 3.27 i jest budowane poprzez przeszukiwanie tego wykresu.



Algorytm parsowania oparty na danych implementowałby to, dopasowując prawe strony reguł przepisywania do wzorców w zdaniu, wypróbowując te dopasowania w kolejności, w jakiej reguły zostały zapisane. Po znalezieniu dopasowania część wyrażenia pasująca do prawej strony reguły jest zastępowana wzorcem po lewej stronie. Trwa to do momentu, gdy zdanie zostanie zredukowane do zdania symbolicznego (co oznacza udaną analizę) lub nie będzie można zastosować więcej reguł (wskazujących błąd). Ślad parsowania psa gryzie mężczyznę:
1. Pierwszą pasującą zasadą jest 7, przepisując jako art. To daje: pies sztuki gryzie mężczyznę.

2. Następna iteracja znalazłaby pasującą wartość dla 7, w wyniku której pies sztuki gryzie człowieka sztuki.
3. Zasada 8 będzie strzelać, wytwarzając sztukę gryzie psa sztuki n.
4. Zasada 3 zostanie wystrzelona, aby np. Ugryźć psa.
5. Zasada 9 tworzy sztuki n gryzie np.
6. Można zastosować zasadę 3, podając np. Ugryzienia np.
7. Reguła 11 daje np v np.
8. Reguła 5 daje np. Vp.
9. Zasada 1 ogranicza to do zdania, uznając wyrażenie za poprawne.

Powyższy przykład implementuje parsowaną w pierwszej kolejności analizę danych, ponieważ zawsze stosuje do wyrażenia regułę najwyższego poziomu; np. sztuka n redukuje się do np. zanim ukąszenia zredukują się do v. Parsowanie można również wykonać w sposób ukierunkowany na cel, przyjmując zdanie jako ciąg początkowy i znajdując serię zamian wzorców pasujących do lewej strony reguł prowadzących do seria terminali pasujących do zdania docelowego. Analiza jest ważna nie tylko dla języka naturalnego, ale także dla konstruowania kompilatorów i interpretatorów dla języków komputerowych. Literatura jest pełna algorytmów parsowania dla wszystkich klas języków. Na przykład wiele algorytmów analizowania ukierunkowanych na cele patrzy w przód w strumieniu wejściowym, aby określić, która reguła ma zostać zastosowana w następnej kolejności. W tym przykładzie przyjęliśmy bardzo proste podejście do przeszukiwania wykresu i / lub wykresu w niedoinformowany sposób. W tym przykładzie interesująca jest implementacja wyszukiwania. Takie podejście polegające na prowadzeniu rejestru bieżącego wyrażenia i próbowaniu dopasowania reguł w porządku jest przykładem użycia systemu produkcyjnego do implementacji wyszukiwania. Reguły zastępcze są również używane do generowania zdań prawnych zgodnie ze specyfikacją gramatyki. Zdania mogą być generowane przez wyszukiwanie ukierunkowane na cel, zaczynając od zdania jako celu najwyższego poziomu, a kończąc, gdy nie można zastosować więcej reguł. Daje to ciąg symboli końcowych, które są prawnym zdaniem w gramatyce. Na przykład:

Zdaniem jest np, po którym następuje vp (reguła 1).
np zastępuje się n (reguła 2), co daje n vp.
man jest pierwszym dostępnym n (reguła 8), co daje man vp.
Teraz np jest usatysfakcjonowany i podjęto próbę vp. Reguła 3 zastępuje vp v, man v.
Reguła 10 zastępuje v polubieniami.
człowiek lubi to pierwsze akceptowalne zdanie.

Jeśli pożądane jest utworzenie wszystkich akceptowalnych zdań, to wyszukiwanie może być systematycznie powtarzane do momentu wypróbowania wszystkich możliwości i wyczerpania przeszukiwania całej przestrzeni stanu. Generuje to zdania, w tym mężczyznę, który lubi, i tak dalej. Istnieje około 80 poprawnych zdań, które powstają w wyniku wyczerpującego wyszukiwania. Należą do nich takie semantyczne anomalie, jak człowiek gryzie psa. Przetwarzanie i generowanie może być używane razem na różne sposoby do rozwiązywania różnych problemów. Na przykład, jeśli pożądane jest znalezienie wszystkich zdań w celu uzupełnienia łańcucha „człowiek”, wówczas rozwiązującemu problem można podać niekompletny ciąg „człowiek”. Może działać w górę w sposób oparty na danych, aby osiągnąć cel wypełnienia reguły zdania (reguła 1), w której np jest zastąpiony przez człowieka, a następnie pracować w sposób ukierunkowany na cel, aby określić wszystkie możliwe vps, które zakończą zdanie. To stworzyłoby zdania takie jak mężczyzna, mężczyzna gryzie mężczyznę i tak dalej. Ponownie, ten przykład dotyczy tylko poprawności składniowej. Kwestia semantyki (czy łańcuch ma odwzorowanie na jakiś „świat” z „prawdą”) jest zupełnie inna. Wcześniej

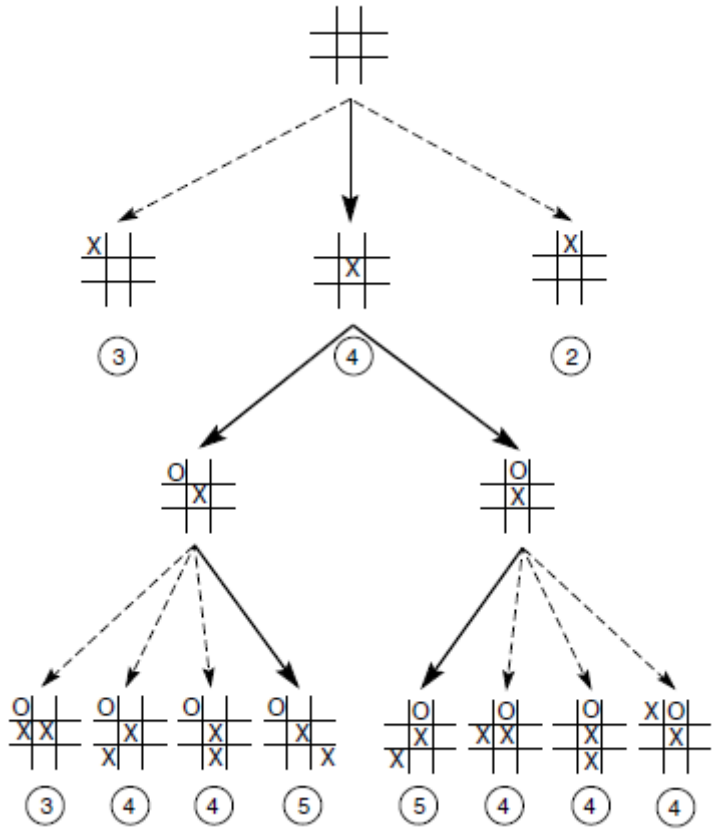
przeanalizowaliśmy kwestię konstruowania semantyki opartej na logice dla wyrażeń; do sprawdzania zdań w języku naturalnym.

Sztuczna inteligencja : wyszukiwanie heurystyczne

(IV / XVI)

ĆWICZENIA

1. Rozszerz heurystykę „najbardziej wygranych” dla dwóch kółek i krzyżyków o dwie warstwy głębiej w przestrzeni wyszukiwania na rysunku.



Jaka jest całkowita liczba badanych stanów za pomocą tej heurystyki? Czy w tej sytuacji działałby tradycyjny algorytm wspinania się na wzgórze? Dlaczego?

2. Użyj komponentu wstecznego algorytmu programowania dynamicznego, aby znaleźć inne optymalne wyrównanie znaków na rysunku.

	—	B	A	A	D	D	C	A	B	D	D	A
—	0	1	2	3	4	5	6	7	8	9	10	11
B	1	0	1	2	3	4	5	6	7	8	9	10
B	2	1	2	3	4	5	6	7	6	7	8	9
A	3	2	1	2	3	4	5	6	7	8	9	8
D	4	3	2	3	2	3	4	5	6	7	8	9
C	5	4	3	4	3	4	3	4	5	6	7	8
B	6	5	6	5	4	5	4	5	4	5	6	7
A	7	6	5	4	5	6	5	4	5	6	7	6

Ile jest optymalnych ustawień?

3. Za pomocą pomiaru Levenshteina, użyj programowania dynamicznego, aby określić minimalną odległość edycji od wykrywania ciągów źródłowych i wzbudzenia do wykonania ciągu docelowego.

4. Podaj heurystykę, której program do układania bloków może użyć do rozwiązania problemów w postaci „stos bloku X w bloku Y”. Czy to jest dopuszczalne? Monotoniczny?

5. Układanka z przesuwanymi kafelkami składa się z trzech czarnych kafelków, trzech białych kafelków i pustej przestrzeni w konfiguracji pokazanej na rysunku.

B	B	B		W	W	W

Łamigłówka ma dwa legalne ruchy z powiązanymi kosztami: Kafelek może przemieścić się na sąsiednie puste miejsce. Ma to koszt 1. Kafelek może przeskoczyć o jeden lub dwa inne kafelki w puste miejsce. Ma to koszt równy liczbie przeskakiwanych płytek. Celem jest umieszczenie wszystkich białych płytek po lewej stronie wszystkich czarnych płytek. Pozycja pustego miejsca nie jest ważna.

a. Przeanalizuj przestrzeń stanu pod względem złożoności i zapętlenia.

b. Zaproponuj heurystykę dla rozwiązania tego problemu i przeanalizuj ją pod względem dopuszczalności, monotoniczności i wiedzy.

6. Porównaj trzy heurystyki z 8 puzzli

	5	6	0
	3	4	0
	5	6	0
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals


Goal

z heurystyką dodania sumy odległości nie na miejscu do dwukrotności liczby bezpośrednich zwrotów. Porównaj je pod względem:

a. Dokładność w szacowaniu odległości do celu. Wymaga to najpierw uzyskania najkrótszej ścieżki i zastosowania jej jako standardu.

b. Informacje Który heurysta najskuteczniej przycina przestrzeń państwową?

c. Czy któraś z tych trzech heurystyk 8-łamigłówkowych jest monotoniczna?

d. Dopuszczalność Którą z tych heurystyk ogranicza z góry rzeczywisty koszt ścieżki do celu? Albo udowodnij swoje wnioski w sprawie ogólnej, albo podaj kontrprzykład.

7. a. Jak przedstawiono w tekście, wyszukiwanie w trybie „najlepsze w pierwszej kolejności" wykorzystuje zamkniętą listę do wdrożenia wykrywania pętli. Jaki byłby efekt wyeliminowania tego testu i polegania na teście głębokości g (n) w celu wykrycia pętli? Porównaj wydajność obu podejść.

b. best_first_search nie testuje stanu, aby sprawdzić, czy jest celem, dopóki nie zostanie usunięty z otwartej listy. Ten test można wykonać po wygenerowaniu nowych stanów. Jaki wpływ miałoby to na efektywność algorytmu? Dopuszczalność

8. Dowód A* jest dopuszczalny. Wskazówka: dowód powinien wykazać, że:

a. Wyszukiwanie * zostanie zakończone.

b. Podczas jego wykonywania zawsze jest otwarty węzeł, który leży na optymalnej ścieżce do celu.

c. Jeśli istnieje ścieżka do celu, A* zakończy się przez znalezienie optymalnej ścieżki.

9. Czy dopuszczalność oznacza monotoniczność heurystyki? Jeśli nie, czy możesz opisać, kiedy dopuszczalność oznaczałaby monotoniczność?

10. Wykazać, że zbiór stanów rozwinięty przez algorytm A * jest podzbiorem stanów badanych przez pierwsze wyszukiwanie szerokości.

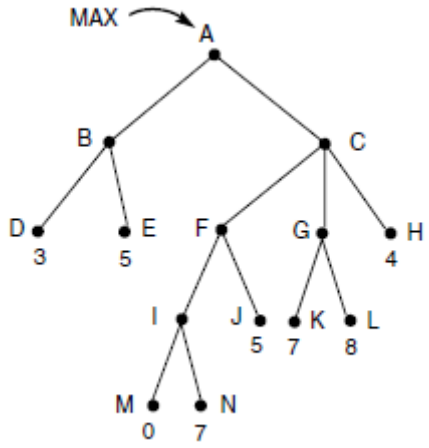
11. Udowodnij, że lepiej poinformowana heurystyka rozwija taką samą lub mniejszą przestrzeń poszukiwań.

12. Szyfr Cezara to schemat szyfrowania oparty na cyklicznych permutacjach alfabetu, z i-tą literą alfabetu zastąpioną (i + n) -tą literą alfabetu. Na przykład w szyfrie Cezara z przesunięciem o 4 „Cezar" będzie szyfrowany jako „Geiwev".

a. Podaj trzy heurystyki, które mogą być użyte do rozwiązania szyfrów Cezara.

b. W prostym szyfrze podstawienia każda litera jest zastępowana inną literą w ramach dowolnego arbitralnego odwzorowania jeden na jeden. Którą z heurystyk zaproponowanych dla szyfru Cezara można zastosować do rozwiązania szyfrów podstawienia? Wyjaśnić.

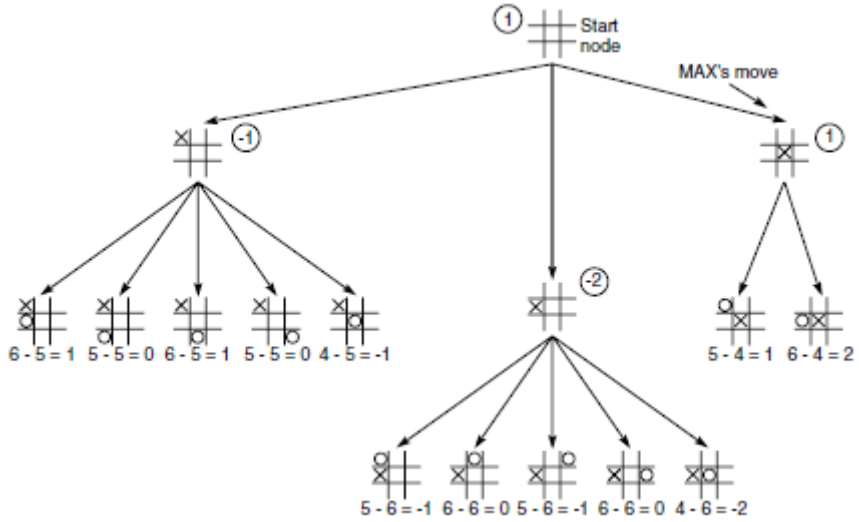
13. Wykonaj minimax na drzewie pokazanym na rysunku

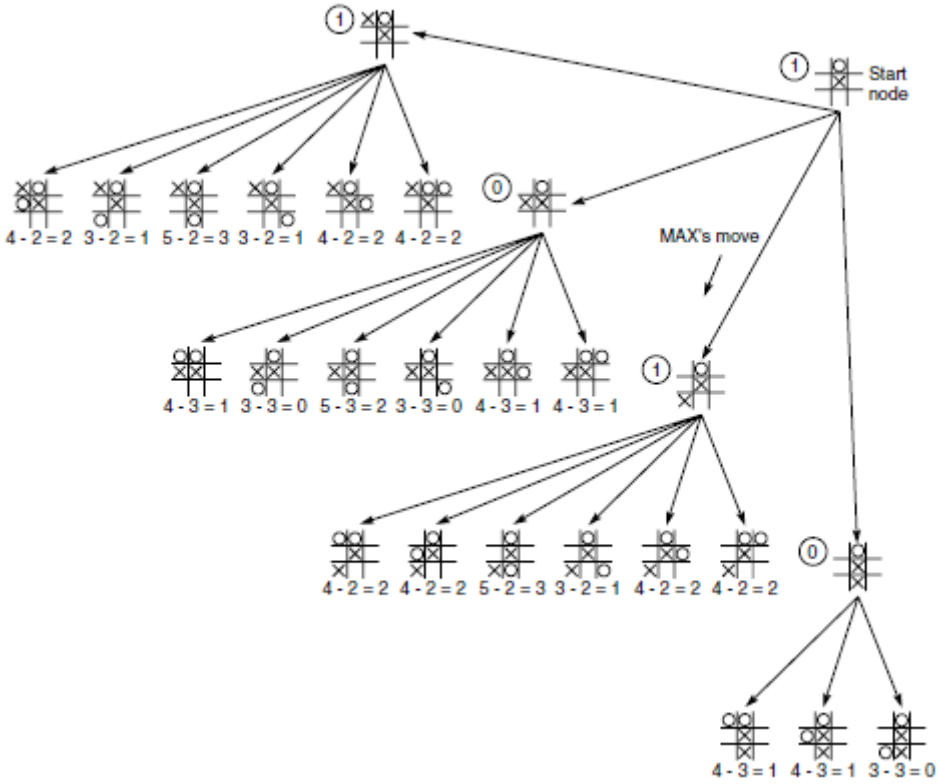


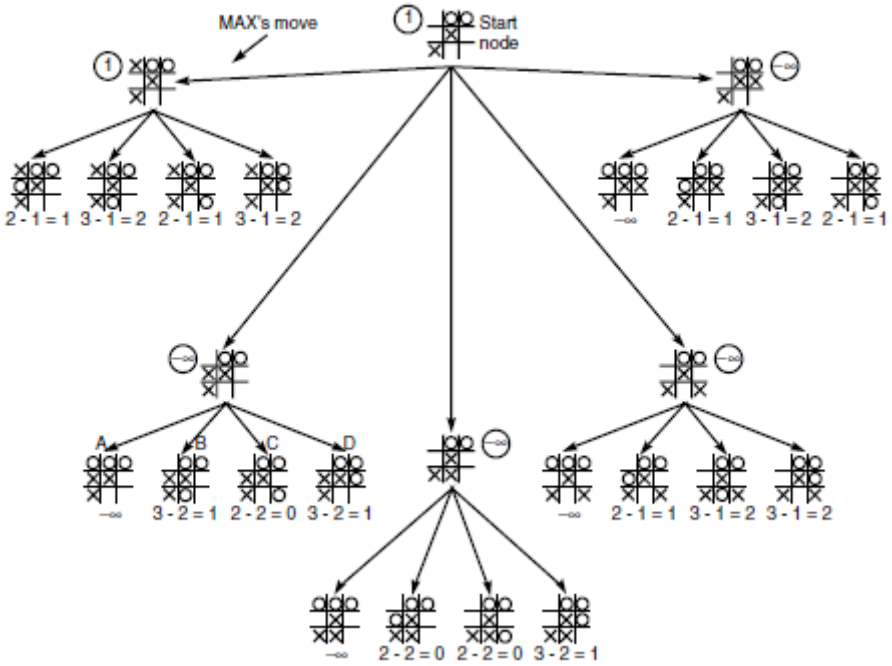
14. Wykonaj przycinanie od lewej do prawej alfa-beta na drzewie w ćwiczeniu 13. Wykonaj przycinanie od prawej do lewej na tym samym drzewie. Omów, dlaczego występuje inne przycinanie.

15. Rozważ trójwymiarowe kółko i krzyżyk. Omów kwestie reprezentacyjne; analizować złożoność przestrzeni stanu. Zaproponuj heurystykę w tej grze.

16. Wykonaj przycinanie alfa-beta podczas wyszukiwania kółko i krzyżyk na rysunkach:



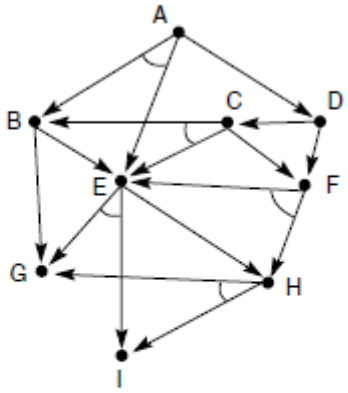




W jaki sposób w każdym przypadku można wyeliminować wiele węzłów liści?

17. a. Utwórz algorytm do heurystycznego wyszukiwania i / lub wykresów. Zauważ, że wszyscy potomkowie węzła i należy rozwiązać, aby rozwiązać element nadrzędny. Zatem w obliczeniach heurystycznych szacunki kosztów do celu, szacunkowy koszt rozwiązania i węzła musi wynosić przynajmniej suma szacunków do rozwiązania różnych gałęzi.

b. Użyj tego algorytmu, aby przeszukać wykres na rysunku.



h(A) = 1
h(B) = 2
h(C) = 5
h(D) = 3
h(E) = 2
h(F) = 5
h(G) = 1
h(H) = 3
h(I) = 1

WYSZUKIWANIE HEURYSTYCZNE

Zadaniem, z którym musi się zmierzyć system symboli, kiedy jest przedstawiany problem i przestrzeń problemowa, jest wykorzystanie jego ograniczonych zasobów przetwarzania do generowania możliwych rozwiązań, jeden po drugim, aż znajdzie takie, które spełni test definiujący problem . Gdyby system symboli miał pewną kontrolę nad kolejnością generowania potencjalnych rozwiązań, pożądane byłoby takie uporządkowanie kolejności generowania, aby rzeczywiste rozwiązania miały wysokie prawdopodobieństwo pojawienia się wcześnie. System symboli wykazywałby inteligencję w stopniu, w jakim udało się to zrobić. Inteligencja systemu z ograniczonymi zasobami przetwarzania polega na dokonywaniu mądrych wyborów, co dalej.

Wprowadzenie

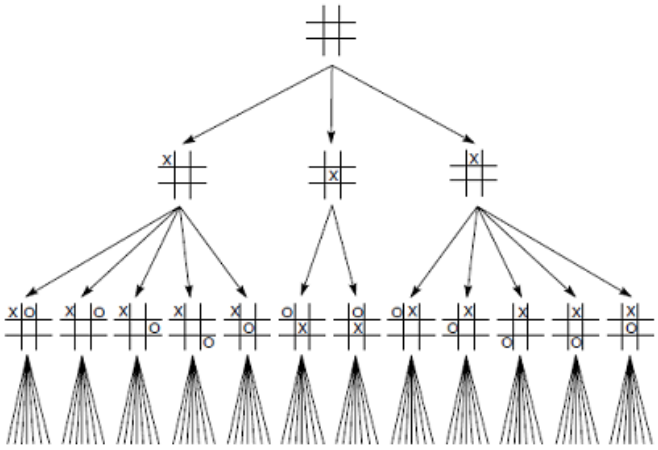
George Polya definiuje heurystykę jako „badanie metod i zasad odkrywania i wynalazków". Znaczenie to można przypisać greckiemu pierwiastkowi, czasownikowi eurisco, co oznacza „odkrywam". Kiedy Archimedes wyłonił się ze swojej słynnej łaźni, ściskając złotą koronę, krzyknął „Eureka!", Co znaczy „Znalazłem!". W wyszukiwaniu przestrzeni stanów heurystyka jest sformalizowana jako reguły wyboru gałęzi w przestrzeni stanów, które najprawdopodobniej doprowadzą do akceptowalnego rozwiązania problemu. Rozwiązujący problemy AI używają heurystyki w dwóch podstawowych sytuacjach:

1. Problem może nie mieć dokładnego rozwiązania z powodu nieodłącznych niejasności w opisie problemu lub dostępnych danych. Diagnoza medyczna jest tego przykładem. Dany zestaw objawów może mieć kilka możliwych przyczyn; lekarze używają heurystyki, aby wybrać najbardziej prawdopodobną diagnozę i sformułować plan leczenia. Wizja jest kolejnym przykładem niedokładnego problemu. Sceny wizualne są często niejednoznaczne, co pozwala na wiele interpretacji połączenia, zasięgu i orientacji obiektów. Złudzenia optyczne są przykładem tych dwuznaczności. Systemy wizyjne często wykorzystują heurystykę, aby wybrać najbardziej prawdopodobną z kilku możliwych interpretacji sceny.

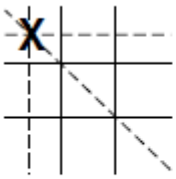
2. Problem może mieć dokładne rozwiązanie, ale obliczeniowy koszt jego znalezienia może być wygórowany. W wielu problemach (np. W szachach) wzrost przestrzeni stanów jest kombinatorycznie gwałtowny, a liczba możliwych stanów rośnie wykładniczo lub czynnikowo wraz z głębokością

poszukiwań. W takich przypadkach wyczerpujące techniki poszukiwania brutalnego, takie jak głębokość pierwsza lub pełne wyszukiwanie może nie znaleźć rozwiązania w jakimkolwiek praktycznym czasie. Heurystyka atakuje tę złożoność, prowadząc poszukiwania najbardziej „obiecującą" ścieżką w przestrzeni. Eliminując nierozważne stany i ich potomków, heurystyczny algorytm może (jego projektanci mają nadzieję) pokonać tę kombinatoryczną eksplozję i znaleźć akceptowalne rozwiązanie. Niestety, podobnie jak wszystkie zasady odkrywania i wynalazków, heurystyka jest omylna. Heurystyka jest tylko świadomym odgadnięciem następnego kroku, który należy podjąć w celu rozwiązania problemu. Często opiera się na doświadczeniu lub intuicji. Ponieważ heurystyka wykorzystuje ograniczone informacje, takie jak znajomość bieżącej sytuacji lub opisy stanów znajdujących się obecnie na otwartej liście, nie zawsze są one w stanie przewidzieć dokładne zachowanie przestrzeni stanów dalej podczas wyszukiwania. Heurystyka może doprowadzić algorytm wyszukiwania do rozwiązania nieoptymalnego lub w ogóle nie znaleźć żadnego rozwiązania. Jest to nieodłączne ograniczenie poszukiwań heurystycznych. Nie można go wyeliminować dzięki „lepszej" heurystyce lub bardziej wydajnym algorytmom wyszukiwania. Heurystyka i projektowanie algorytmów do implementacji wyszukiwania heurystycznego od dawna stanowi główny problem sztucznej inteligencji. Gra i dowodzenie twierdzeń to dwa z najstarszych zastosowań sztucznej inteligencji; oba wymagają heurystyki, aby przyciąć przestrzenie możliwych rozwiązań. Nie jest możliwe zbadanie każdego wniosku, który można dokonać w dziedzinie matematyki, ani każdego możliwego ruchu, jaki można wykonać na szachownicy. Wyszukiwanie heurystyczne jest często jedyną praktyczną odpowiedzią. Badania systemów ekspertowych potwierdziły znaczenie heurystyki jako istotnego elementu rozwiązywania problemów. Gdy ludzki ekspert rozwiązuje problem, bada dostępne informacje i podejmuje decyzję. „Podstawowe zasady", których ludzki ekspert używa do skutecznego rozwiązywania problemów, mają w dużej mierze charakter heurystyczny. Te heurystyki są wydobywane i formalizowane przez ekspertów projektantów systemów. Warto pomyśleć o wyszukiwaniu heurystycznym z dwóch perspektyw: miary heurystycznej i algorytmu wykorzystującego heurystykę do przeszukiwania przestrzeni stanu. Rozważmy heurystykę w grze kółko i krzyżyk

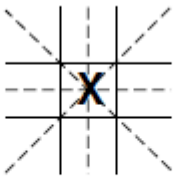
Kombinatoryka dla wyczerpujących poszukiwań jest wysoka, ale nie do pokonania. Każdy z dziewięciu pierwszych ruchów ma osiem możliwych odpowiedzi, które z kolei mają siedem ruchów kontynuujących i tak dalej. Prosta analiza stawia całkowitą liczbę stanów dla wyczerpującego wyszukiwania na $9 \times \times 7 \times \cdots$ lub $9!$. Redukcja symetrii zmniejsza przestrzeń wyszukiwania. Wiele konfiguracji problemów jest w rzeczywistości równoważnych przy symetrycznych operacjach planszy. Tak więc nie ma dziewięciu, ale tak naprawdę trzech początkowych ruchów: do rogu, do środka boku i do środka siatki. Zastosowanie symetrii na drugim poziomie dodatkowo zmniejsza liczbę ścieżek w przestrzeni do $12 \times 7!$, jak pokazano na rysunku



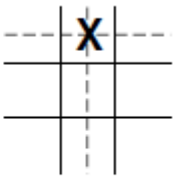
Symetrie w przestrzeni gry, takie jak ta, można opisać jako niezmienniki matematyczne, które, jeśli istnieją, mogą być często wykorzystywane do ogromnej przewagi w ograniczaniu wyszukiwania. Prosta heurystyka może jednak całkowicie wyeliminować wyszukiwanie: możemy przejść do stanu, w którym X ma największe szanse na wygraną.



Three wins through a corner square

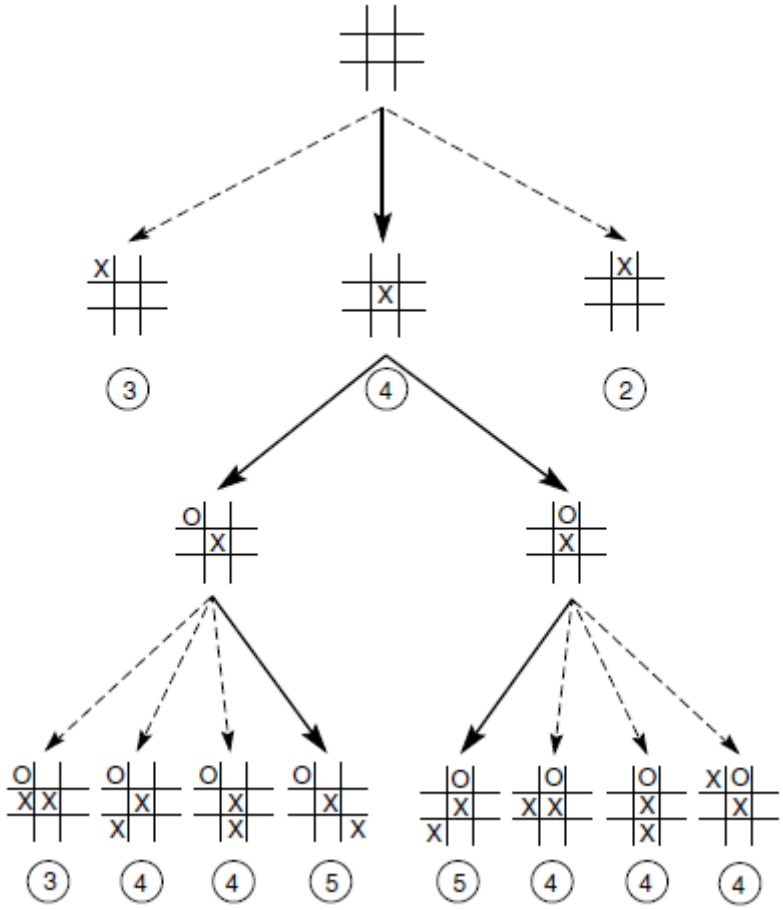


Four wins through the center square



Two wins through a side square

W przypadku stanów o równej liczbie potencjalnych wygranych, weź pierwszy znaleziony taki stan. Algorytm następnie wybiera i przechodzi do stanu o największej liczbie możliwości. W tym przypadku X zajmuje środek siatki. Zauważ, że nie tylko pozostałe dwie alternatywy zostały wyeliminowane, ale także ich potomkowie. Dwie trzecie pełnej przestrzeni jest przycinane przy pierwszym ruchu



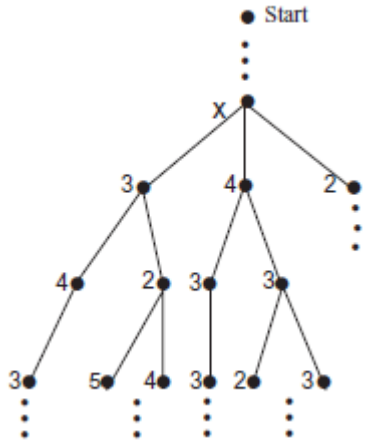
Po pierwszym ruchu przeciwnik może wybrać jeden z dwóch alternatywnych ruchów. Niezależnie od tego, co zostanie wybrane, heurystykę można zastosować do wynikowego stanu gry, ponownie używając „najbardziej wygranych okazji", aby wybrać spośród możliwych ruchów. W miarę kontynuowania wyszukiwania każdy ruch ocenia dzieci jednego węzła; wyczerpujące wyszukiwanie nie jest wymagane. Rysunek powyższy pokazuje ograniczone wyszukiwanie po trzech krokach w grze. Stany są naznaczone wartościami heurystycznymi. Chociaż nie jest to dokładne obliczenie wielkości wyszukiwania dla tej strategii „najwięcej zwycięstw" w kółko i krzyżyk, surową górną granicę można obliczyć, zakładając maksymalnie pięć ruchów w grze z pięcioma opcjami na ruch. W rzeczywistości liczba stanów jest mniejsza, ponieważ tablica wypełnia i zmniejsza opcje. Ta prymitywna granica 25 stanów jest poprawą o cztery rzędy wielkości w stosunku do 9 !.

Hill-climbing i programowanie dynamiczne

Wspinaczka górska

Najprostszym sposobem na wdrożenie wyszukiwania heurystycznego jest procedura zwana wspinaniem się na wzgórze. Strategie wspinaczki rozszerzają aktualny stan poszukiwań i oceniają ich dzieci. „Najlepsze" dziecko jest wybierane do dalszej ekspansji; ani rodzeństwo, ani rodzic nie są zatrzymani. Wspinaczka na wzgórze jest nazwana od strategii, która może być używana przez chętnego, ale niewidomego alpinistę: idź pod górę najostrzejszą możliwą ścieżką, aż nie będziesz mógł iść dalej. Ponieważ nie zachowuje historii, algorytm nie może odzyskać sprawności po niepowodzeniach strategii. Przykładem wspinaczki na kółkach w kółko i krzyżyk było „weź stan z jak największą liczbą wygranych". Głównym problemem strategii wspinaczkowych jest ich tendencja do

utknięcia w lokalnych maksimach. Jeśli osiągną stan, który ma lepszą ocenę niż którekolwiek z jego dzieci, algorytm załamuje się. Jeśli ten stan nie jest celem, a jedynie lokalnym maksimum, algorytm może nie znaleźć najlepszego rozwiązania. Oznacza to, że wydajność może się poprawić w ograniczonym otoczeniu, ale ze względu na kształt całej przestrzeni może nigdy nie osiągnąć najlepszego wyniku. Przykład lokalnych maksimów w grach występuje w 8-puzzlach. Często, aby przenieść konkretną płytkę do jej miejsca docelowego, inne płytki znajdujące się już w pozycji bramkowej muszą zostać przeniesione. Jest to konieczne do rozwiązania zagadki, ale chwilowo pogarsza stan planszy. Ponieważ „lepsze" nie musi być „najlepsze" w sensie absolutnym, metody wyszukiwania bez śledzenia wstecznego lub innego mechanizmu odzyskiwania nie są w stanie rozróżnić maksymalnych wartości lokalnych i globalnych. Rysunek poniżej jest przykładem lokalnego maksymalnego dylematu.



Załóżmy, że badając tę przestrzeń wyszukiwania, dochodzimy do stanu X, chcąc zmaksymalizować wartości stanu. Oceny dzieci X, wnuków i prawnuków pokazują, że wspinanie się na wzgórza może się zdezorientować, nawet jeśli spojrzysz w przyszłość na wiele poziomów. Istnieją metody na obejście tego problemu, takie jak przypadkowe zakłócanie funkcji oceny, ale ogólnie nie ma sposobu na zagwarantowanie optymalnej wydajności za pomocą technik wspinania się na wzgórze. Program sprawdzający Samuela oferuje interesującą odmianę algorytmu wspinaczki. Program Samuela był wyjątkowy jak na swój czas, 1959 r., Szczególnie biorąc pod uwagę ograniczenia komputerów z lat 50. Program Samuela nie tylko zastosował wyszukiwanie heurystyczne do gry w warcaby, ale także zaimplementował algorytmy optymalnego wykorzystania ograniczonej pamięci, a także prostej formy uczenia się. Rzeczywiście, był pionierem wielu technik wciąż używanych w grach i programach uczenia maszynowego. Program Samuela oceniał stany zarządu za pomocą ważonej sumy kilku różnych miar heurystycznych:

$$\sum_i a_i x_i$$

x_i w tej sumie reprezentuje cechy planszy, takie jak przewaga pionów, lokalizacja pionów, kontrola środkowej planszy, możliwości poświęcenia pionów dla korzyści, a nawet obliczenie momentów bezwładności pionków jednego gracza względem siekiery . Współczynniki a_i tych x_i były specjalnie wyregulowanymi wagami, które próbowały modelować znaczenie tego czynnika w ogólnej ocenie tablicy. Tak więc, jeśli przewaga w kawałku była ważniejsza niż kontrola środka, współczynnik przewagi w kawałku odzwierciedlałby to. Program Samuela szukałby w przestrzeni wyszukiwania pożądanej

liczby poziomów lub warstw (zwykle narzuconych przez przestrzeń i / lub ograniczenia czasowe komputera) i oceniał wszystkie stany na tym poziomie za pomocą wielomianu oceniającego. Wykorzystując wariację na temat minimax, propagował te wartości w górę wykresu. Gracz sprawdzający przejdzie wtedy do najlepszego stanu; po ruchu przeciwnika proces zostanie powtórzony dla nowego stanu planszy. Jeśli wielomian oceniający doprowadził do utraty serii ruchów, program dostosował swoje współczynniki, próbując poprawić wydajność. Większość obciążeń za straty ponosiły oceny o dużych współczynnikach, które zmniejszały ich wagi, a zwiększano mniejsze wagi, aby dać większy wpływ na te oceny. Jeśli program wygrał, sytuacja była odwrotna. Program szkolony przez grę przeciwko ludzkiemu partnerowi lub przeciwko innej wersji samego siebie. Program Samuela przyjął podejście wspinaczkowe do nauki, próbując poprawić wydajność poprzez lokalne ulepszenia wielomianu oceny. Samuela. Gracz w warcaby był w stanie poprawić swoją wydajność, dopóki nie zagrał w bardzo dobrą grę w warcaby. Samual zajął się niektórymi ograniczeniami wspinania się na wzgórze, sprawdzając skuteczność poszczególnych ważonych miar heurystycznych i zastępując mniej skuteczne. Program zachował również pewne interesujące ograniczenia. Na przykład z powodu ograniczonej globalnej strategii był podatny na funkcje oceny prowadzące do pułapek. Element edukacyjny programu był również podatny na niespójności w grze przeciwnika; na przykład, jeśli przeciwnik zastosował bardzo różne strategie lub po prostu grał głupio, wagi na wielomianie oceniającym mogą zacząć przyjmować wartości „losowe”, co prowadzi do ogólnego pogorszenia wydajności.

Programowanie dynamiczne

Programowanie dynamiczne (DP) jest czasem nazywane algorytmem Viterbi, przy użyciu prawdopodobieństwa do przodu lub do tyłu. Opracowane przez Richarda Bellmana programowanie dynamiczne rozwiązuje problem ograniczonego wyszukiwania pamięci w problemach złożonych z wielu interakcjei i powiązane ze sobą podproblemy. DP śledzi i ponownie wykorzystuje podproblemy, które zostały już przeszukane i rozwiązane w ramach rozwiązania większego problemu. Przykładem tego może być ponowne wykorzystanie rozwiązań podrzędnych w ramach rozwiązania serii Fibonacciego. Technika buforowania podproblemów w celu ponownego wykorzystania jest czasem nazywana zapamiętywaniem częściowych rozwiązań podrzędnych. Wynikiem jest ważny algorytm często używany do dopasowywania ciągów, sprawdzania pisowni i powiązanych obszarów w przetwarzaniu języka naturalnego. Demonstrujemy programowanie dynamiczne na dwóch przykładach, oba z przetwarzania tekstu; po pierwsze, znalezienie optymalnego globalnego wyrównania dwóch ciągów znaków, a po drugie, znalezienie minimalnej różnicy edycji między dwoma ciągami znaków. Załóżmy, że chcieliśmy znaleźć możliwie najlepsze wyrównanie znaków w ciągach BAADDCABDDA i BBADCBA. Jednym z optymalnych dopasowań, spośród kilku możliwych, byłoby:
BAADDCABDDA
BBA DC B A
Programowanie dynamiczne wymaga struktury danych do śledzenia podproblemów związanych z aktualnie przetwarzanym stanem. Używamy tablicy. Rozmiar wymiarów tej tablicy, ze względu na wymagania dotyczące inicjalizacji, jest o jeden większy niż długość każdego łańcucha, w naszym przykładzie 12 na 8, jak na rysunku.

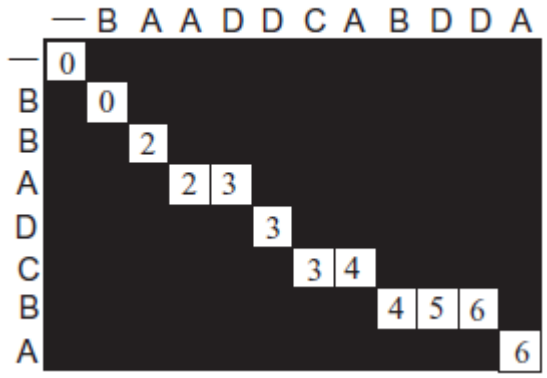
	—	B	A	A	D	D	C	A	B	D	D	A
—	0	1	2	3	4	5	6	7	8	9	10	11
B	1	0										
B	2											
A	3											
D	4											
C	5											
B	6											
A	7											

Wartość każdego elementu wiersz-kolumna tablicy odzwierciedla globalny sukces wyrównania do tego punktu w procesie dopasowywania. Istnieją trzy możliwe koszty utworzenia bieżącego stanu: jeśli postać zostanie przesunięta wzdłuż krótszego łańcucha w celu lepszego możliwego wyrównania, koszt wynosi 1 i jest rejestrowany na podstawie wyniku kolumny w tablicy. Jeśli wstawiona zostanie nowa postać, koszt wynosi 1 i jest odzwierciedlony w wyniku rzędu tablicy. Jeśli znaki do wyrównania są różne, przesuń i wstaw, koszt wynosi 2; lub jeśli są identyczne, koszt wynosi 0; znajduje to odzwierciedlenie w przekątnej tablicy. Powyższy rysunek pokazuje inicjalizację, rosnące + 1s w pierwszym rzędzie i kolumnie odzwierciedlają ciągłe przesuwanie lub wstawianie znaków do „_” lub pustego łańcucha. W przednim etapie algorytmu programowania dynamicznego wypełniamy tablicę od lewego górnego rogu, biorąc pod uwagę sukces częściowego dopasowania do bieżącego punktu rozwiązania. Oznacza to, że wartość przecięcia wiersza x i kolumny y (x, y) jest funkcją (dla problemu minimalnego wyrównania, kosztu minimalnego) jednej z trzech wartości w wierszu x - 1 kolumnie y, wierszu x - 1 kolumna y - 1 lub wiersz x kolumna y - 1. Te trzy lokalizacje tablic przechowują informacje o wyrównaniu do obecnego punktu rozwiązania. Jeśli występuje dopasowanie znaków w lokalizacji (x, y), dodaj 0 do wartości w lokalizacji (x - 1, y - 1), jeśli nie ma dopasowania, dodaj 2 (dla shift i insert). Dodajemy 1 poprzez przesunięcie krótszego ciągu znaków (dodanie do poprzedniej wartości z kolumny y) lub wstawienie znaku (dodanie do poprzedniej wartości z wiersza x). Kontynuacja tego podejścia tworzy wypełniony zestaw z rysunku.

	—	B	A	A	D	D	C	A	B	D	D	A
—	0	1	2	3	4	5	6	7	8	9	10	11
B	1	0	1	2	3	4	5	6	7	8	9	10
B	2	1	2	3	4	5	6	7	6	7	8	9
A	3	2	1	2	3	4	5	6	7	8	9	8
D	4	3	2	3	2	3	4	5	6	7	8	9
C	5	4	3	4	3	4	3	4	5	6	7	8
B	6	5	6	5	4	5	4	5	4	5	6	7
A	7	6	5	4	5	6	5	4	5	6	7	6

Można zauważyć, że dopasowanie minimalnego kosztu często ma miejsce w pobliżu górnej lewej do prawej dolnej przekątnej tablicy. Po zapełnieniu tablicy rozpoczynamy etap wstecz algorytmu, który daje określone rozwiązania. Oznacza to, że z najlepszego możliwego wyrównania wracamy do tablicy, aby wybrać konkretne wyrównanie rozwiązania. Proces ten rozpoczynamy od maksymalnego rzędu wartości kolumny, w naszym przykładzie 6 w wierszu 7 kolumna 12. Stamtąd przechodzimy z powrotem przez tablicę, na każdym kroku wybierając jednego z poprzedników stanu bezpośredniego, który wytworzył stan obecny (z etapu do przodu), tj. jeden z przekątna, rząd lub kolumna, które

wytworzyły ten stan. Ilekroć występuje wymuszona malejąca różnica, jak w 6 i 5 w pobliżu początku śladu wstecz, wybieramy poprzednią przekątną, ponieważ stąd pochodzi dopasowanie; w przeciwnym razie używamy wartości z poprzedniego wiersza lub kolumny. Ślad z tyłu rysunku poniżej, jeden z kilku możliwych, zapewnia optymalne wyrównanie ciągów poprzedniej strony.



W drugim przykładzie użycia DP rozważamy ideę minimalnej różnicy edycji między dwoma łańcuchami. Na przykład, jeśli budowaliśmy inteligentny moduł sprawdzania pisowni, moglibyśmy chcieć ustalić, które słowa z naszego zestawu poprawnie napisanych słów najlepiej przybliżają określony nierozpoznany ciąg znaków. Podobne podejście można zastosować do określenia, które znane (ciągi telefonów reprezentujących) słów najbardziej pasują do danego ciągu telefonów. Następny przykład ustalenia odległości edycji między dwoma ciągami znaków został zaadaptowany przez Jurafsky'ego i Martina. Załóżmy, że tworzysz nierozpoznany ciąg znaków. Zadanie zaklęcia Checker to utworzenie uporządkowanej listy najbardziej prawdopodobnych słów ze słownika, które chciałeś wpisać. Powstaje zatem pytanie, w jaki sposób można zmierzyć różnicę między parami ciągów znaków, tj. Właśnie wprowadzonym ciągiem znaków i ciągiem znaków w słowniku. Dla twojego ciągu chcemy stworzyć uporządkowaną listę możliwych poprawnych słów w słowniku. Dla każdego z tych słów chcemy liczbową miarę tego, jak „różne” jest każde słowo z łańcucha. Minimalną różnicę edycji między dwoma łańcuchami można określić jako liczbę wstawek, usunięć i zamian znaków, niezbędnych do przekształcenia pierwszego łańcucha, źródła, w drugi łańcuch, celu. Jest to czasem nazywane odległością Levenshteina. Wdrażamy teraz dynamiczne wyszukiwanie programowania w celu określenia minimalnej różnicy edycji między dwoma ciągami znaków. Pozwalamy intencji być ciągiem źródłowym, a realizacja celem. Koszt edycji przekształcenia pierwszego ciągu na drugi wynosi 1 za wstawienie znaku lub usunięcie i 2 za zastąpienie (usunięcie i wstawienie). Chcemy ustalić różnicę kosztów minimalnych między tymi dwoma ciągami. Nasza tablica dla podzbiorów będzie znowu o jeden znak dłuższa niż każdy z łańcuchów, w tym przypadku 10 na 10. Inicjalizacja jest jak na rysunku 4.8 (sekwencja wstawiania jest konieczna, zaczynając od łańcucha zerowego, aby dowolny łańcuch przypominał łańcuch inny.

	—	e	x	e	c	u	t	i	o	n
—	0	1	2	3	4	5	6	7	8	9
i	1	2								
n	2									
t	3									
e	4									
n	5									
t	6									
i	7									
o	8									
n	9									

Lokalizacja tablicy (2, 2) to 2, ponieważ do zamiany i w e jest wymagane zastąpienie (lub wykonanie operacji usuwania plus wstawka).

	—	e	x	e	c	u	t	i	o	n
—	0	1	2	3	4	5	6	7	8	9
i	1	**2**	3	4	5	6	7	8	9	10
n	2	3	**4**	5	6	7	8	9	10	11
t	3	4	**5**	6	7	8	9	10	11	12
e	4	5	6	**5**	**6**	7	8	9	10	11
n	5	6	7	6	7	**8**	9	10	11	12
t	6	7	8	7	8	9	**8**	9	10	11
i	7	8	9	8	9	10	9	**8**	9	10
o	8	9	10	9	10	11	10	9	**8**	9
n	9	10	11	10	11	12	11	10	9	**8**

Rysunek powyższy przedstawia pełny wynik zastosowania algorytmu programowania dynamicznego do przekształcenia zamiaru w wykonanie. Wartość w każdej lokalizacji (x, y) w tablicy to koszt minimalnej edycji do tego punktu plus (minimalny) koszt wstawienia, usunięcia lub wymiany. Zatem koszt (x, y) to minimalny koszt (x - 1, y) plus koszt wstawienia lub koszt (x - 1, y - 1) plus koszt wymiany lub koszt (x, y - 1) plus koszt usunięcia. Pseudokod dla jego algorytmu to funkcja przyjmująca dwa ciągi (źródło i cel) i ich długości oraz zwracająca minimalny koszt różnicy edycji:

```
function dynamic (source, sl, target, tl) return cost (i, j);
create array cost(sl + 1, tl + 1)
cost (0,0) := 0 % initialize
for i := 1 to sl + 1 do
for j := 1 to tl + 1 do
cost (i, j) := min [ cost (i - 1, j) + insertion cost targeti - 1 % add 1
cost (i - 1, j - 1) + replace cost sourcej - 1 targeti - 1 % add 2
cost(i, j - 1) + delete cost sourcej - 1 ] % add 1
```

Korzystając z wyników (pogrubionych) na rysunku powyższym, następujące zmiany przełożą zamiar na wykonanie z całkowitym kosztem edycji 8:
intention

ntention delete i, cost 1
etention replace n with e, cost 2
exention replace t with x, cost 2
exenution insert u, cost 1
execution replace n with c, cost 2

W sytuacji sprawdzania pisowni proponowania opartej na kosztach uporządkowanej listy słów do zastąpienia nierozpoznanego łańcucha, segment algorytmu programowania dynamicznego nie jest potrzebny. Po obliczeniu minimalnej miary edycji dla zestawu powiązanych ciągów proponowana jest kolejność priorytetów alternatywnych, z której użytkownik wybiera odpowiedni ciąg. Uzasadnieniem dla programowania dynamicznego jest koszt czasu / przestrzeni w obliczeniach. Programowanie dynamiczne, jak widać w naszych przykładach, ma koszt n2, gdzie n jest długością największego łańcucha; koszt w najgorszym przypadku wynosi n3, jeśli inne powiązane podproblemy muszą być wzięte pod uwagę (inne wartości wierszy / kolumn) w celu ustalenia bieżącego stanu. Wyczerpujące poszukiwanie dwóch ciągów ma charakter wykładniczy i kosztuje od 2n do 3n. Istnieje wiele oczywistych heurystyk, które można wykorzystać do przycinania wyszukiwania w programowaniu dynamicznym. Po pierwsze, przydatne rozwiązania będą zwykle znajdować się wokół górnej lewej lewej do prawej dolnej przekątnej tablicy; prowadzi to do ignorowania rozwoju ekstremów tablicowych. Po drugie, przydatne może być przycinanie wyszukiwania w miarę jego ewolucji, np. W przypadku edytowania odległości przekraczających określony próg, przecięcia ścieżki rozwiązania lub nawet porzucenia całego problemu, tzn. Łańcuch źródłowy będzie tak oddalony od łańcucha znaków docelowych jak być bezużytecznym.

Algorytm najlepszego wyszukiwania

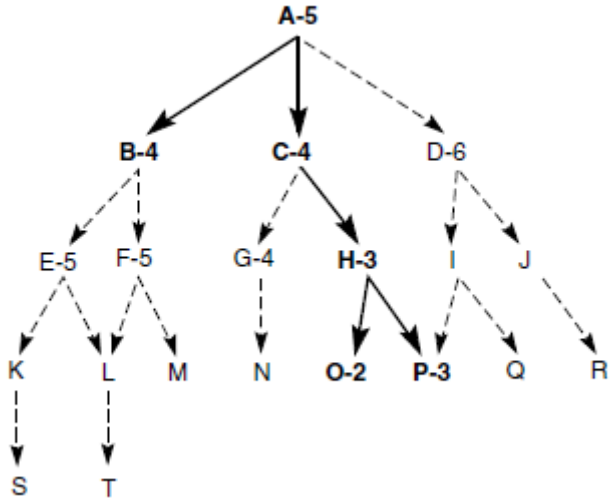
Wdrażanie najlepszego wyszukiwania

Pomimo ich ograniczeń, algorytmy takie jak powrót, wspinaczka pod górę i programowanie dynamiczne mogą być skutecznie stosowane, jeśli ich funkcje oceny są wystarczająco informacyjne, aby uniknąć lokalnych maksimów, ślepych zaułków i powiązanych anomalii w przestrzeni wyszukiwania. Zasadniczo jednak korzystanie z wyszukiwania heurystycznego wymaga bardziej elastycznego algorytmu: zapewnia to wyszukiwanie z najlepszym rozwiązaniem, w którym w przypadku kolejki priorytetowej możliwe jest odzyskanie po tych sytuacjach. Podobnie jak algorytmy przeszukiwania z głębokością pierwszą i szerokością z rozdziału 3, wyszukiwanie według najlepszego z nich wykorzystuje listy do utrzymywania stanów: otwarte, aby śledzić bieżącą granicę wyszukiwania i zamknięte dla rekordów, które już odwiedzono. Dodatkowy krok w algorytmie nakazuje stanom otwarcie zgodnie z niektórymi heurystycznymi szacunkami ich „bliskości" do celu. Dlatego każda iteracja pętli uwzględnia najbardziej „obiecujący" stan na otwartej liście. Pseudokod funkcji best_first_search pojawia się poniżej.

```
function best_first_search;
begin
open := [Start]; % initialize
closed := [ ];
while open ≠ [ ] do % states remain
begin
remove the leftmost state from open, call it X;
if X = goal then return the path from Start to X
else begin
```

```
generate children of X;
for each child of X do
case
the child is not on open or closed:
begin
assign the child a heuristic value;
add the child to open
end;
the child is already on open:
if the child was reached by a shorter path
then give the state on open the shorter path
the child is already on closed:
if the child was reached by a shorter path then
begin
remove the state from closed;
add the child to open
end;
end; % case
put X on closed;
re-order states on open by heuristic merit (best leftmost)
end;
return FAIL % open is empty
end.
```

Przy każdej iteracji best_first_search usuwa pierwszy element z otwartej listy. Jeśli spełnia warunki celu, algorytm zwraca ścieżkę rozwiązania, która doprowadziła do celu. Zauważ, że każdy stan zachowuje informacje o przodku, aby ustalić, czy wcześniej został osiągnięty krótszą ścieżką i aby umożliwić algorytmowi zwrócenie ostatecznej ścieżki rozwiązania. Jeśli pierwszy element przy otwartym nie jest celem, algorytm stosuje wszystkie pasujące reguły produkcji lub operatorów do generowania swoich potomków. Jeśli stan potomny nie jest otwarty lub zamknięty best_first_search stosuje ocenę heurystyczną do tego stanu, a otwarta lista jest sortowana zgodnie z wartościami heurystycznymi tych stanów. To stawia „najlepsze" stany na pierwszym planie. Zauważ, że ponieważ te szacunki mają charakter heurystyczny, następny „najlepszy" stan do zbadania może pochodzić z dowolnego poziomu przestrzeni stanu. Gdy otwarta jest utrzymywana jako posortowana lista, jest często nazywana kolejką priorytetową. Jeśli stan podrzędny jest już otwarty lub zamknięty, algorytm sprawdza, czy stan zapisuje krótszą z dwóch częściowych ścieżek rozwiązania. Zduplikowane stany nie są zachowywane. Poprzez aktualizację historii ścieżek węzłów w otwartym i zamkniętym, gdy zostaną one ponownie odkryte, algorytm znajdzie najkrótszą ścieżkę do celu (w rozważanych stanach). Rysunek 4.10 pokazuje hipotetyczną przestrzeń stanów z ocenami heurystycznymi dołączonymi do niektórych jej stanów.



Stany z dołączonymi ocenami są tymi, które zostały faktycznie wygenerowane best_first_search. Stany rozszerzone przez algorytm wyszukiwania heurystycznego zaznaczono pogrubioną czcionką; zauważ, że nie przeszukuje całej przestrzeni. Celem pierwszego wyszukiwania jest znalezienie stanu celu poprzez sprawdzenie jak najmniejszej liczby stanów; im bardziej poinformowany heurysta, tym mniej stanów jest przetwarzanych w celu znalezienia celu. Ślad wykonania best_first_search na tym wykresie znajduje się poniżej. Przypuszczenie P jest stanem docelowym na wykresie na powyższym rysunku. Ponieważ P jest celem, stany na drodze do P będą miały zwykle niższe wartości heurystyczne. Heurystyka jest omylna: stan O ma niższą wartość niż sam cel i jest najpierw badany. W przeciwieństwie do wspinaczki pod górę, która nie utrzymuje kolejki priorytetowej dla wyboru „kolejnych" stanów, algorytm odzyskuje ten błąd i znajduje właściwy cel.

1. otwarte = [A5]; zamknięte = []
2. ocenić A5; otwarte = [B4, C4, D6]; zamknięte = [A5]
3. ocenić B4; otwarte = [C4, E5, F5, D6]; zamknięte = [B4, A5]
4. ocenić C4; otwarte = [H3, G4, E5, F5, D6]; zamknięte = [C4, B4, A5]
5. ocenić H3; otwarte = [O2, P3, G4, E5, F5, D6]; zamknięte = [H3, C4, B4, A5]
6. ocenić O2; otwarte = [P3, G4, E5, F5, D6]; zamknięte = [O2, H3, C4, B4, A5]
7. ocenić P3; rozwiązanie zostało znalezione!

Rysunek pokazuje przestrzeń, jaka pojawia się po piątej iteracji pętli while.

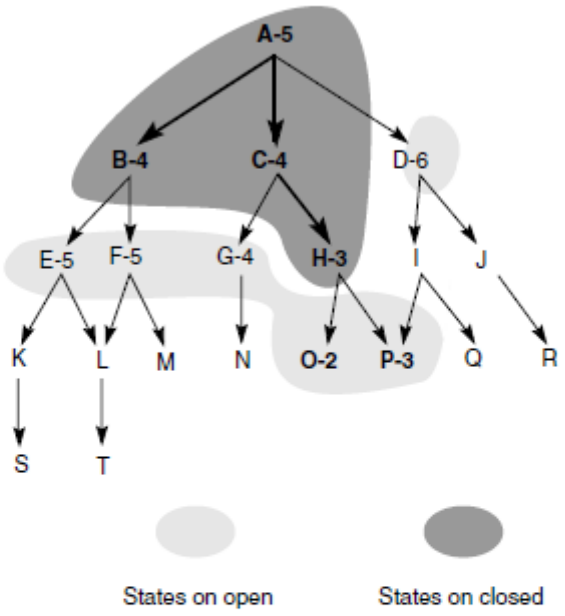


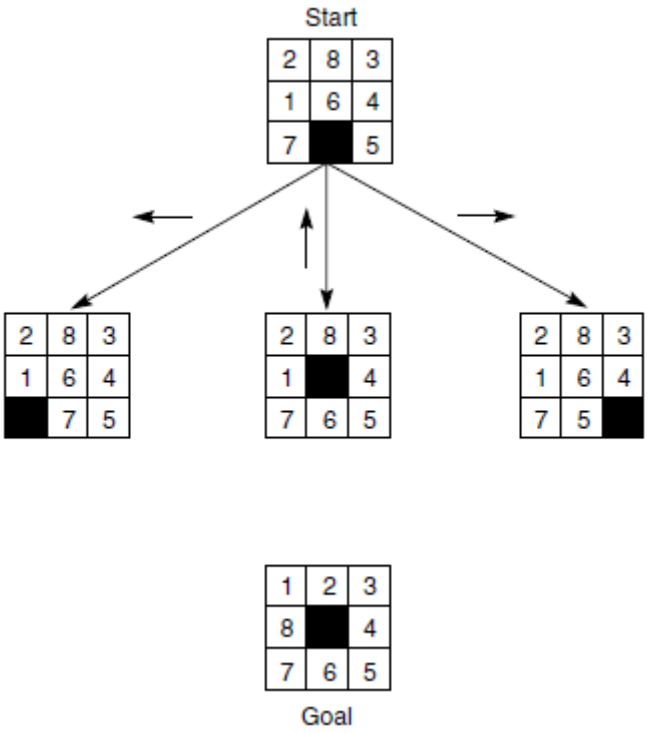
Figure 4.11 Heuristic search of a hypothetical state space with open and closed states highlighted.

3. evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]
4. evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]
5. evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]
6. evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!

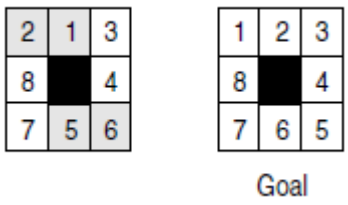
Wskazane są stany zawarte w otwartym i zamkniętym. otwórz rekordy aktualną granicę stanów wyszukiwania i zamkniętych rekordów, które zostały już uwzględnione. Zwróć uwagę, że granica wyszukiwania jest bardzo nierówna, co odzwierciedla oportunistyczny charakter najlepszego wyszukiwania. Algorytm najlepszego wyszukiwania najpierw wybiera najbardziej obiecujący stan przy otwartym do dalszego rozszerzenia. Ponieważ jednak używa heurystyki, która może okazać się błędna, nie porzuca wszystkich innych stanów, ale utrzymuje je otwarte. W przypadku, gdy heurystyka poprowadzi wyszukiwanie ścieżką, która okaże się niepoprawna, algorytm w końcu odzyska niektóre wcześniej wygenerowane, „następne najlepsze" stany z otwartego i przeniosą fokus na inną część przestrzeni. W przykładzie z ryciny wcześniej, gdy stwierdzono, że dzieci stanu B mają słabą ocenę heurystyczną, poszukiwania przeniosły swoje skupienie na stan C. Dzieci B pozostały otwarte, na wypadek gdyby algorytm musiał wrócić do nich później. W best_first_search, podobnie jak w algorytmach z Części 3, otwarta lista obsługuje cofanie ze ścieżek, które nie dają celu.

Wdrażanie heurystycznych funkcji oceny

Oceniamy wydajność kilku różnych heurystyk do rozwiązywania zagadek 8. Rysunek pokazuje stan początkowy i docelowy dla 8 puzzli, a także pierwsze trzy stany wygenerowane podczas wyszukiwania.



Najprostsza heurystyka liczy kafelki nie na miejscu w każdym stanie w porównaniu z celem. Jest to intuicyjnie atrakcyjne, ponieważ wydaje się, że mimo wszystko, że są równe, stan, który miał mniej płytek na swoim miejscu, jest prawdopodobnie bliższy pożądanemu celowi i najlepiej go zbadać w następnej kolejności. Jednak ta heurystyka nie wykorzystuje wszystkich informacji dostępnych w konfiguracji planszy, ponieważ nie bierze pod uwagę odległości, którą płytki należy przesunąć. „Lepsza" heurystyka zsumowałaby wszystkie odległości, na które płytki są nie na swoim miejscu, po jednym na każde pole płytka musi zostać przesunięta, aby osiągnąć swoją pozycję w stanie bramkowym. Obie te heurystyki można skrytykować za to, że nie uznają trudności w odwróceniu płytek. Oznacza to, że jeśli dwa kafelki znajdują się obok siebie, a cel wymaga, aby znajdowały się one w przeciwnych lokalizacjach, potrzeba (kilku) więcej niż dwóch ruchów, aby umieścić je z powrotem na swoim miejscu, ponieważ kafelki muszą się „ominąć"



Heurystyka, która bierze to pod uwagę, mnoży niewielką liczbę (na przykład 2) razy każde bezpośrednie odwrócenie płytki (gdzie dwie sąsiednie płytki muszą zostać wymienione, aby były w kolejności bramkowej). Rysunek pokazuje wynik zastosowania każdej z tych trzech heurystyk do trzech stanów potomnych na rycinie powyżej

2 8 3 1 6 4 ■ 7 5	5	6	0
2 8 3 1 ■ 4 7 6 5	3	4	0
2 8 3 1 6 4 7 5 ■	5	6	0
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

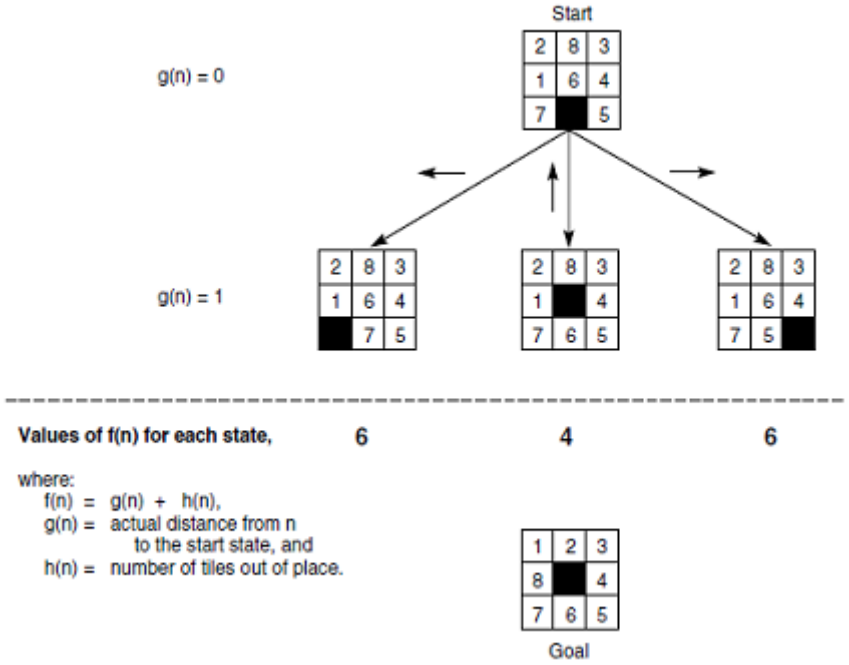
1 2 3
8 ■ 4
7 6 5

Goal

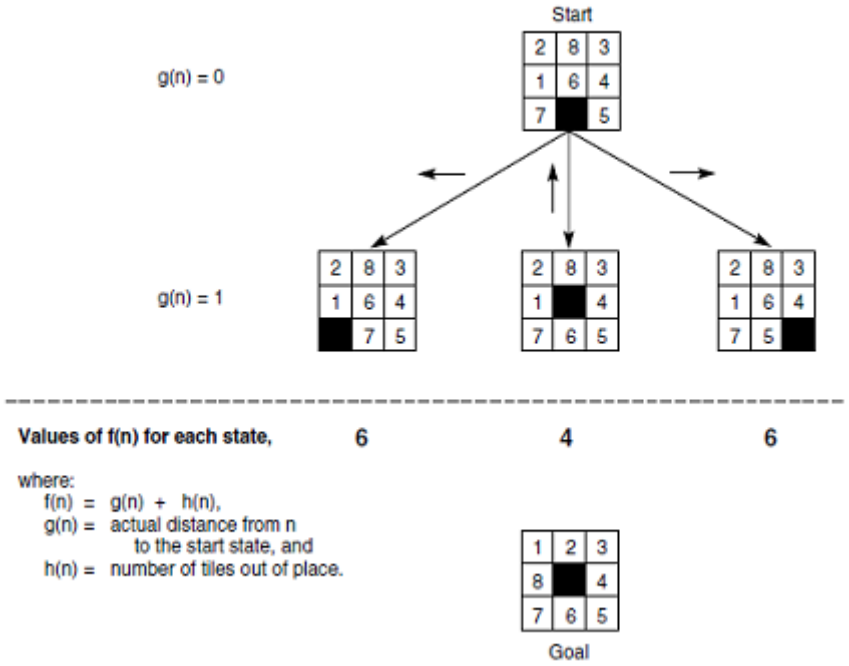
Na powyższym rysunku podsumowującym funkcje oceny suma odległości heurystycznych rzeczywiście wydaje się zapewniać dokładniejsze oszacowanie pracy do wykonania niż zwykłe zliczanie liczby płytek nie na miejscu. Również heurystyka odwracania płytek nie rozróżnia pomiędzy tymi stanami, dając każdemu ocenę 0. Chociaż jest to intuicyjnie atrakcyjna heurystyka, załamuje się, ponieważ żaden z tych stanów nie ma żadnych bezpośrednich zwrotów. Czwarta heurystyka, która może przezwyciężyć ograniczenia heurystyki odwracania płytek, dodaje sumę odległości, które płytki są nie na miejscu i 2-krotność liczby bezpośrednich zwrotów. Ten przykład ilustruje trudność opracowania dobrej heurystyki. Naszym celem jest wykorzystanie ograniczonych informacji dostępnych w jednym deskryptorze stanu do dokonywania inteligentnych wyborów. Każda z heurystyk zaproponowanych powyżej ignoruje niektóre krytyczne informacje i może ulec poprawie. Projektowanie dobrej heurystyki jest problemem empirycznym; osąd i intuicja pomagają, ale ostateczną miarą heurystyki musi być jej rzeczywiste działanie w przypadku wystąpienia problemów. Jeśli dwa stany mają takie same lub prawie takie same oceny heurystyczne, ogólnie lepiej jest zbadać stan najbliższy pierwiastkowi stanu wykresu. Ten stan będzie miał większe prawdopodobieństwo bycia na najkrótszej drodze do celu. Odległość od stanu początkowego do jego potomków można zmierzyć, utrzymując liczbę głębokości dla każdego stanu. Liczba ta wynosi 0 dla stanu początkowego i jest zwiększana o 1 dla każdego poziomu wyszukiwania. Tę miarę głębokości można dodać do oceny heurystycznej każdego stanu w celu wyszukiwania stronniczości na korzyść stanów znalezionych płytszych na wykresie. To sprawia, że nasza funkcja oceny, f, jest sumą dwóch składników:

f (n) = g (n) + h (n)

gdzie g (n) mierzy rzeczywistą długość ścieżki od dowolnego stanu n do stanu początkowego, a h (n) jest heurystycznym oszacowaniem odległości od stanu n do celu. Na przykład w 8-puzzlach możemy pozwolić h (n) być liczbą płytek nie na miejscu. Po zastosowaniu tej oceny do każdego ze stanów potomnych ich wartości f wynoszą odpowiednio 6, 4 i 6,



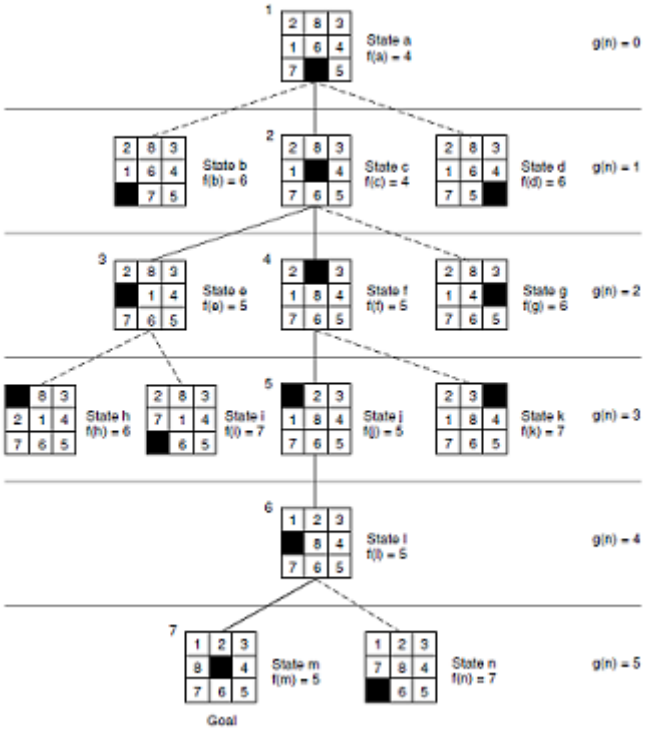
Pełne najlepsze wyszukiwanie wykresu z 8 łamigłówkami, przy użyciu f zdefiniowanego powyżej, pokazano na rysunku



Każdy stan jest oznaczony literą i jego heurystyczną wagą, f (n) = g (n) + h (n). Liczba u góry każdego stanu wskazuje kolejność, w jakiej została usunięta z otwartej listy. Niektóre stany (h, g, b, d, n, k i i) nie są tak ponumerowane, ponieważ nadal były otwarte po zakończeniu algorytmu. Kolejne etapy otwierania i zamykania, które generują ten wykres, to:

1. otwarte = [a4];
zamknięte = []
2. otwarte = [c4, b6, d6];
zamknięte = [a4]

3. otwarte = [e5, f5, b6, d6, g6];
zamknięte = [a4, c4]
4. otwarte = [f5, h6, b6, d6, g6, i7];
zamknięte = [a4, c4, e5]
5. otwarte = [j5, h6, b6, d6, g6, k7, i7];
zamknięte = [a4, c4, e5, f5]
6. otwarte = [15, h6, b6, d6, g6, k7, i7];
zamknięte = [a4, c4, e5, f5, j5]
7. otwarte = [m5, h6, b6, d6, g6, n7, k7, i7];
zamknięte = [a4, c4, e5, f5, j5, l5]
8. sukces, m = cel!
W kroku 3 zarówno e, jak i f mają heurystykę 5. Najpierw bada się stan e, uzyskując dzieci, hi hi. Chociaż h, dziecko e, ma taką samą liczbę płytek nie na miejscu, co f, jest o jeden poziom głębiej w przestrzeni. Pomiar głębokości g (n) powoduje, że algorytm wybiera f do oceny w kroku 4. Algorytm wraca do stanu płytszego i kontynuuje osiągnięcie celu. Wykres przestrzeni stanów na tym etapie wyszukiwania, z podświetlonymi otwartymi i zamkniętymi, pokazano na rysunku



Zwróć uwagę na oportunistyczną naturę najlepszego wyszukiwania. W efekcie składnik g (n) funkcji oceny nadaje poszukiwaniom bardziej pierwszy smak. Zapobiega to wprowadzeniu go w błąd w wyniku błędnej oceny: jeśli heurystyka stale zwraca „dobre" oceny dla stanów na ścieżce, która nie osiąga celu, wartość g wzrośnie do dominacji hi wymusi wyszukiwanie z powrotem do krótszej ścieżki rozwiązania. Gwarantuje to, że algorytm nie zostanie trwale utracony, schodząc w nieskończoną gałąź. W następnej części przeanalizowano warunki, w których można zagwarantować, że pierwsze wyszukiwanie za pomocą tej funkcji oceny prowadzi do uzyskania najkrótszej ścieżki do celu. Podsumowując:
1. Operacje na stanach generują dzieci badanego stanu.
2. Każdy nowy stan jest sprawdzany, aby sprawdzić, czy wystąpił wcześniej (jest otwarty lub zamknięty), zapobiegając w ten sposób pętlom.
3. Każdy stan n otrzymuje wartość f równą sumie jego głębokości w przestrzeni poszukiwań g (n) i heurystycznej ocenie jego odległości do celu h (n). Wartość h prowadzi wyszukiwanie w kierunku

stanów obiecujących heurystycznie, podczas gdy wartość g może uniemożliwić wyszukiwanie przez czas nieokreślony na bezowocnej drodze.
4. Stany przy otwartym są sortowane według ich wartości f. Utrzymując wszystkie stany otwarte, dopóki nie zostaną zbadane lub nie zostanie znaleziony cel, algorytm odzyskuje ślepe zaułki.
5. Jako punkt implementacyjny można poprawić wydajność algorytmu poprzez utrzymanie list otwartych i zamkniętych, na przykład jako hałdy lub drzewa lewicowe. Wyszukiwanie według najlepszego pierwszego to ogólny algorytm heurystycznego przeszukiwania dowolnego wykresu przestrzeni stanów (podobnie jak wcześniej przedstawione algorytmy szerokości i głębokości). Dotyczy to również wyszukiwań opartych na danych i celach oraz obsługuje szereg heurystycznych funkcji oceny. Będzie on kontynuowany (sekcja 4.3), aby zapewnić podstawę do zbadania zachowania wyszukiwania heurystycznego. Ze względu na swoją ogólność, wyszukiwanie typu „najlepsze w pierwszej kolejności" może być stosowane z różnymi heurystykami, od subiektywnych ocen „dobroci" stanu po wyrafinowane miary oparte na prawdopodobieństwie stanu prowadzącego do celu. Ważnym przykładem tego podejścia są bayesowskie miary statystyczne. Innym interesującym podejściem do wdrażania heurystyki jest stosowanie miar zaufania przez systemy eksperckie do ważenia wyników reguły. Kiedy eksperci ludzcy stosują heurystykę, zwykle są w stanie podać szacunek ich wiary w jej wnioski. Systemy eksperckie wykorzystują miary zaufania, aby wybierać wnioski o najwyższym prawdopodobieństwie sukcesu. Stany o wyjątkowo niskich ufnościach można całkowicie wyeliminować.

Heurystyczne systemy wyszukiwania i eksperckie

Proste gry, takie jak 8-puzzle, są idealnymi pojazdami do badania konstrukcji i zachowania heurystycznych algorytmów wyszukiwania z wielu powodów:
1. Przestrzenie poszukiwań są wystarczająco duże, aby wymagać heurystycznego przycinania.
2. Większość gier jest na tyle złożona, że sugeruje bogatą różnorodność ocen heurystycznych w celu porównania i analizy.
3. Gry zasadniczo nie wiążą się ze złożonymi problemami reprezentacyjnymi. Pojedynczy węzeł przestrzeni stanu jest tylko opisem płytki i zwykle można go uchwycić w prosty sposób. Pozwala to badaczom skupić się raczej na zachowaniu heurystycznym niż na problemach związanych z reprezentacją wiedzy.
4. Ponieważ każdy węzeł przestrzeni stanu ma wspólną reprezentację (np. Opis tablicy), w przestrzeni wyszukiwania można zastosować jedną heurystykę. Kontrastuje to z systemami takimi jak doradca finansowy, w których każdy węzeł reprezentuje inny podzakres z własnym odrębnym opisem.
Bardziej realistyczne problemy znacznie komplikują wdrażanie i analizę wyszukiwania heurystycznego, ponieważ wymagają wielu heurystyk do radzenia sobie z różnymi sytuacjami w przestrzeni problemów. Jednak spostrzeżenia uzyskane z prostych gier uogólniają się na takie problemy, jakie występują w aplikacjach systemów eksperckich, planowaniu, inteligentnym sterowaniu i uczeniu maszynowym. W przeciwieństwie do 8 puzzli, jedna heurystyka może nie mieć zastosowania do każdego stanu w tych domenach. Zamiast tego heurystyka rozwiązywania problemów specyficzna dla sytuacji jest zakodowana w składni i treści poszczególnych operatorów rozwiązywania problemów. Każdy krok rozwiązania zawiera własną heurystykę, która określa, kiedy należy go zastosować; dopasowywanie wzorca dopasowuje odpowiednią operację (heurystyczną) do odpowiedniego stanu w przestrzeni.

PRZYKŁAD: DORADCA FINANSOWY, ZMIENIONY

Zastosowanie środków heurystycznych do kierowania wyszukiwaniem jest ogólnym podejściem w sztucznej inteligencji. Zastanówmy się ponownie nad problemem doradcy finansowego z części 2 i 3,

w którym bazę wiedzy potraktowano jako zbiór logicznych implikacji, których wnioski są albo prawdziwe, albo fałszywe. W rzeczywistości reguły te mają charakter wysoce heurystyczny. Na przykład jedna reguła mówi, że osoba z odpowiednimi oszczędnościami i dochodami powinna inwestować w akcje:
rachunek_oszczędności (odpowiedni) ∧ dochód (odpowiedni) ⇒ inwestycje (zapasy).
W rzeczywistości możliwe jest, że taka osoba może preferować dodatkowe bezpieczeństwo strategii łączenia, a nawet bezpieczeństwo lokowania wszystkich pieniędzy inwestycyjnych w oszczędności. Zatem reguła jest heurystyczna, a osoba zajmująca się rozwiązywaniem problemów powinna próbować wyjaśnić tę niepewność. Możemy wziąć pod uwagę dodatkowe czynniki, takie jak wiek inwestora i długoterminowe perspektywy bezpieczeństwa i awansu w zawodzie inwestora, aby przepisy były bardziej świadome i zdolne do dokładniejszych rozróżnień. Nie zmienia to jednak zasadniczo heurystycznego charakteru doradztwa finansowego. Jednym ze sposobów rozwiązania tego problemu przez systemy eksperckie jest dołączenie wag lizb (zwana miarą ufności lub współczynnikiem pewności) do zakończenia każdej reguły. Mierzy to zaufanie, jakie można pokładać w ich wnioskach. Każda konkluzja reguły ma miarę ufności, powiedzmy liczbę rzeczywistą od -1 do 1, gdzie 1 odpowiada pewności (true) i -1 do określonej wartości false. Wartości pomiędzy odzwierciedlają zmienne zaufanie do wniosku. Na przykład powyższej regule można przypisać pewność, powiedzmy, 0,8, odzwierciedlając niewielką możliwość, że może nie być poprawna. Inne wnioski można wyciągać na podstawie różnych wag ufności:
rachunek_oszczędności (odpowiedni) ∧ dochód (odpowiedni) ⇒ inwestycja (akcje) z pewnością = 0,8.
rachunek_oszczędności (odpowiedni) ∧ dochód (odpowiedni) ⇒ inwestycja (połączenie) z pewnością = 0,5.
rachunek_oszczędności (odpowiedni) ∧ dochód (odpowiedni) ⇒ inwestycja (oszczędności) z pewnością = 0,1.
Reguły te odzwierciedlają wspólną radę inwestycyjną, zgodnie z którą chociaż osoba z odpowiednimi oszczędnościami i dochodami byłaby najbardziej zalecana do inwestowania w akcje, istnieje pewna możliwość, że należy zastosować strategię łączenia i istnieje niewielka szansa, że będą chcieli nadal inwestować w oszczędności . Heurystyczne algorytmy wyszukiwania mogą wykorzystywać te czynniki pewności na wiele sposobów. Na przykład można przedstawić wyniki wszystkich obowiązujących zasad wraz z powiązanymi z nimi zwierzeniami. To wyczerpujące poszukiwanie wszystkich możliwości może być odpowiednie w takich dziedzinach, jak medycyna. Alternatywnie program może zwrócić tylko wynik o największej wartości ufności, jeśli alternatywne rozwiązania nie są interesujące. Dzięki temu program może ignorować inne reguły, radykalnie oczyszczając przestrzeń wyszukiwania. Bardziej konserwatywna strategia przycinania mogłaby ignorować reguły, które wyciągają wnioski z pewnością mniejszą niż pewna wartość, na przykład 0,2. Należy zastosować szereg ważnych kwestii przy użyciu miar zaufania do ważenia wniosków dotyczących reguł. Co tak naprawdę oznacza „liczbowa miara zaufania”? Na przykład, w jaki sposób obsługiwane są zwierzenia, jeśli wniosek jednej reguły jest wykorzystywany jako przesłanka innych? Jak łączy się zwierzenia w przypadku, gdy więcej niż jedna reguła wyciąga ten sam wniosek? W jaki sposób właściwe miary zaufania są przypisywane regułom?

Dopuszczalność, monotoniczność i poinformowanie

Możemy ocenić zachowanie heurystyki w wielu wymiarach. Na przykład możemy chcieć rozwiązania i wymagać od algorytmu znalezienia najkrótszej ścieżki do celu. Może to być ważne, gdy aplikacja może mieć nadmierny koszt dodatkowych kroków rozwiązania, takich jak planowanie ścieżki autonomicznego robota przez niebezpieczne środowisko. Uważa się, że heurystyka, która znajduje najkrótszą drogę do celu, o ile istnieje, jest dopuszczalna. W innych aplikacjach minimalna ścieżka rozwiązania może nie być tak ważna jak ogólna wydajność rozwiązywania problemów. Możemy

zapytać, czy dostępna jest lepsza heurystyka. W jakim sensie jedna heurystyka jest „lepsza” od drugiej? To jest wiedza heurystyczna. Czy po wykryciu stanu za pomocą wyszukiwania heurystycznego istnieje jakakolwiek gwarancja, że ten sam stan nie zostanie znaleziony później podczas wyszukiwania przy niższym koszcie (przy krótszej ścieżce od stanu początkowego)? Jest to właściwość monotoniczności.

Środki dotyczące dopuszczalności

Algorytm wyszukiwania jest dopuszczalny, jeśli gwarantuje się znalezienie minimalnej ścieżki rozwiązania, ilekroć taka ścieżka istnieje. Wyszukiwanie na szerokość jest dopuszczalną strategią wyszukiwania. Ponieważ patrzy na każdy stan na poziomie n wykresu, zanim weźmie pod uwagę stan na poziomie n + 1, wszystkie węzły celu znajdują się wzdłuż możliwie najkrótszej ścieżki. Niestety, pierwsze wyszukiwanie jest często zbyt mało wydajne, aby można je było wykorzystać w praktyce. Używając funkcji oceny f (n) = g (n) + h (n), która została wprowadzona w ostatnim rozdziale, możemy scharakteryzować klasę dopuszczalnych heurystycznych strategii wyszukiwania. Jeśli n jest węzłem na wykresie przestrzeni stanów, g (n) mierzy głębokość, na której ten stan został znaleziony na wykresie, a h (n) jest heurystycznym oszacowaniem odległości od n do celu. W tym sensie f (n) szacuje całkowity koszt ścieżki od stanu początkowego przez n do stanu docelowego. Przy określaniu właściwości dopuszczalnej heurystyki definiujemy funkcję oceny f *:

f * (n) = g * (n) + h * (n)

gdzie g * (n) to koszt najkrótszej ścieżki od początku do węzła n, a h * zwraca rzeczywisty koszt najkrótszej ścieżki od n do celu. Wynika z tego, że f * (n) jest rzeczywistym kosztem optymalnej ścieżki od węzła początkowego do węzła docelowego, który przechodzi przez węzeł n. Jak zobaczymy, gdy zastosujemy funkcję best_first_search z funkcją oceny f *, wynikowa strategia wyszukiwania jest dopuszczalna. Chociaż wyrocznie takie jak f * nie istnieją dla większości rzeczywistych problemów, chcielibyśmy, aby funkcja oceny f była dokładnym oszacowaniem f *. W algorytmie A, g (n), koszt bieżącej ścieżki do stanu n jest rozsądnym oszacowaniem g *, ale mogą one nie być równe: g (n) ≥ g * (n). Są one równe tylko wtedy, gdy wyszukiwanie graficzne odkryło optymalną ścieżkę do stanu n. Podobnie, zastępujemy h * (n) przez h (n), heurystyczne oszacowanie minimalnego kosztu do stanu docelowego. Chociaż zwykle nie możemy obliczyć h *, często można ustalić, czy ocena heurystyczna, h (n), jest ograniczona od góry przez h * (n), tj. Zawsze jest mniejsza lub równa rzeczywistemu kosztowi minimalnej ścieżki. Jeśli algorytm A wykorzystuje funkcję oceny f, w której h (n) h * (n), nazywa się algorytmem A *.

DEFINICJA

ALGORYTM A, ADMISSIBILITY, ALGORYTM A *

Rozważ funkcję oceny f (n) = g (n) + h (n), gdzie n oznacza dowolny stan napotkany podczas wyszukiwania.

g (n) to koszt n ze stanu początkowego.

h (n) jest heurystycznym oszacowaniem kosztu przejścia od n do celu.

Jeśli ta funkcja oceny jest używana z algorytmem best_first_search wcześniejszym, wynik nazywa się algorytmem A. Algorytm wyszukiwania jest dopuszczalny, jeśli dla dowolnego wykresu zawsze kończy on optymalną ścieżkę rozwiązania, ilekroć ścieżka od początku do stanu celu istnieje. Jeśli algorytm A jest używany z funkcją oceny, w której h (n) jest mniejszy lub równy kosztowi minimalnej ścieżki od n do celu, wynikowy algorytm wyszukiwania nazywa się algorytmem A * (wymawiane „A STAR”). Można teraz określić właściwość algorytmów A *:

Wszystkie algorytmy A * są dopuszczalne. Dopuszczalność algorytmów A * jest twierdzeniem. Ćwiczenie na końcu części zawiera wskazówki dotyczące opracowania jego dowodu. Twierdzenie mówi, że każdy algorytm A *, tj. Taki, który używa heurystycznego h (n) takiego, że h (n) h * (n) dla wszystkich n, gwarantuje znalezienie minimalnej ścieżki od n do celu, jeśli taki ścieżka istnieje. Zauważ,

że przeszukiwanie z pełną szerokością można scharakteryzować jako algorytm A *, w którym f (n) = g (n) + 0. Decyzja o rozważeniu stanu zależy wyłącznie od jego odległości od stanu początkowego. Pokażemy, że zbiór węzłów rozpatrywanych przez algorytm A * jest podzbiorem stanów badanych w pierwszym wyszukiwaniu. Kilka heurystyk z 8 puzzli dostarcza przykładów algorytmów A *. Chociaż możemy nie być w stanie obliczyć wartości h * (n) dla 8-puzzli, możemy ustalić, kiedy heurystyka jest ograniczona z góry przez rzeczywisty koszt najkrótszej ścieżki do celu. Na przykład heurystyka liczenia liczby płytek, które nie znajdują się w pozycji bramkowej, jest z pewnością mniejsza lub równa liczbie ruchów wymaganych do przeniesienia ich do pozycji bramkowej. Tak więc ta heurystyka jest dopuszczalna i gwarantuje optymalną (lub najkrótszą) ścieżkę rozwiązania, gdy taka ścieżka istnieje. Suma bezpośrednich odległości płytek nie na miejscu jest również mniejsza lub równa minimalnej rzeczywistej ścieżce. Użycie małych mnożników do bezpośredniego odwrócenia płytki daje dopuszczalną heurystykę. Takie podejście do potwierdzania dopuszczalności heurystyk 8-łamigłówkowych można zastosować do każdego problemu wyszukiwania heurystycznego. Chociaż rzeczywisty koszt najkrótszej ścieżki do celu nie zawsze może być obliczony, często możemy udowodnić, że heurystyka jest ograniczona z góry przez tę wartość. Gdy będzie to możliwe, wynikowe wyszukiwanie zakończy się wraz z odkryciem najkrótszej ścieżki do celu, jeśli taka ścieżka istnieje.

Monotoniczność

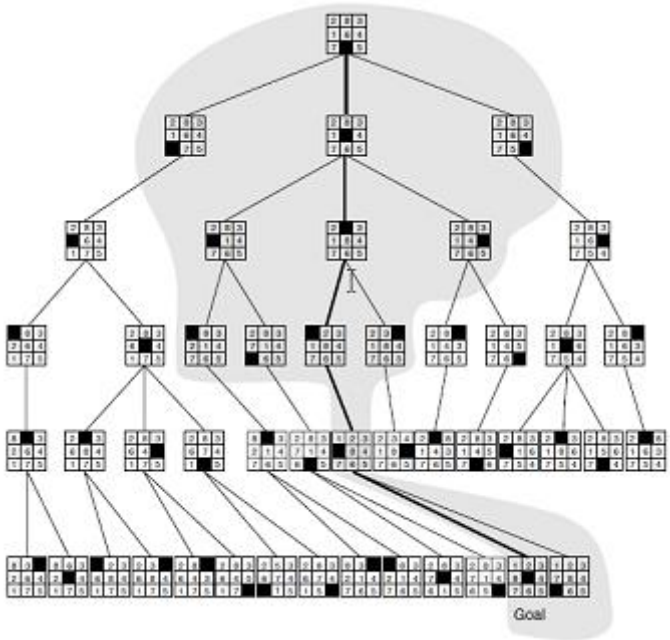
Przypomnijmy, że definicja algorytmów A* nie wymagała, aby g (n) = g*(n). Oznacza to, że dopuszczalna heurystyka może początkowo osiągnąć stany niecelowe wzdłuż ścieżki nieoptymalnej, o ile algorytm ostatecznie znajdzie optymalną ścieżkę do wszystkich stanów na drodze do celu. Naturalne jest pytanie, czy istnieją heurystyki, które są „lokalnie dopuszczalne", tj. Które konsekwentnie znajdują minimalną ścieżkę do każdego stanu napotkanego podczas wyszukiwania. Ta właściwość nazywa się monotonicznością.
DEFINICJA
MONOTONICZNOŚĆ
Funkcja heurystyczna h jest monotoniczna, jeśli
1. Dla wszystkich stanów n_i i n_j, gdzie n_j jest potomkiem n_i, h (n_i) - h (n_i) < koszt (n_i, n_j), gdzie koszt (n_i, n_j) to rzeczywisty koszt (liczba ruchów) przejścia ze stanu n_i do $n_{j>}$.
2. Heurystyczna ocena stanu celu wynosi zero lub h (cel) = 0.
Jednym ze sposobów opisania właściwości monotonicznej jest to, że przestrzeń wyszukiwania jest wszędzie lokalnie spójna z zastosowaną heurystyką. Różnica między miarą heurystyczną dla państwa a dowolnym z jego następców jest związana z faktycznym kosztem przejścia między tym stanem a jego następcą. To znaczy, że heurystyka jest wszędzie dopuszczalna, docierając do każdego stanu najkrótszą drogą od swoich przodków. Jeśli algorytm przeszukiwania wykresu dla najlepszego wyszukiwania jest używany z monotoniczną heurystyką, ważny krok można pominąć. Ponieważ heurystyka znajduje najkrótszą ścieżkę do dowolnego stanu przy pierwszym wykryciu tego stanu, po drugim napotkaniu stanu nie jest konieczne sprawdzanie, czy nowa ścieżka jest krótsza. Nie będzie! Umożliwia to natychmiastowe usunięcie dowolnego stanu odkrytego w przestrzeni bez aktualizacji informacji o ścieżce zachowanych w stanie otwartym lub zamkniętym. Podczas korzystania z monotonicznej heurystyki, gdy wyszukiwanie przemieszcza się w przestrzeni, miara heurystyczna dla każdego stanu n jest zastępowana faktycznym kosztem wygenerowania tego fragmentu ścieżki do n. Ponieważ rzeczywisty koszt jest w każdym przypadku równy lub większy niż heurystyka, f nie zmniejszy się; tj. f nie zmniejsza monotonicznie (stąd nazwa). Prosty argument może wykazać, że każda monotoniczna heurystyka jest dopuszczalna. Ten argument traktuje każdą ścieżkę w przestrzeni jako ciąg stanów s_1, s_2,…, s_g, gdzie s_1 jest stanem początkowym, a s_g jest celem. Aby uzyskać sekwencję ruchów na tej arbitralnie wybranej ścieżce:

s_1 to s_2	$h(s_1) - h(s_2) \leq cost(s_1,s_2)$	według właściwości monotonicznej
s_2 to s_3	$h(s_2) - h(s_3) \leq cost(s_2,s_3)$	według właściwości monotonicznej
s_3 to s_4	$h(s_3) - h(s_4) \leq cost(s_3,s_4)$	według właściwości monotonicznej
.	. . .	według właściwości monotonicznej
.	. . .	według właściwości monotonicznej
s_{g-1} to s_g	$h(s_{g-1}) - h(s_g) \leq cost(s_{g-1},s_g)$	według właściwości monotonicznej

Zsumowanie każdej kolumny i użycie właściwości monotonicznej h (s_g) = 0: ścieżka s_1 do s_g h (s_1) koszt (s_1, s_g) Oznacza to, że monotoniczna heurystyka h jest A* i dopuszczalna. Pozostawia się jako ćwiczenie, czy właściwość dopuszczalności heurystyki implikuje monotoniczność

Kiedy jedna heurystyka jest lepsza: bardziej świadoma heurystyka

Ostatni element tej sekcji porównuje zdolność dwóch heurystyk do znalezienia minimalnej ścieżki. Interesujący przypadek występuje, gdy heurystyka to A*.
DEFINICJA
INFORMACJA
W przypadku dwóch heurystyk A * h_1 i h_2, jeżeli h_1 (n) < h_2(n), dla wszystkich stanów n w przestrzeni poszukiwań uważa się, że heurystyka h_2 jest bardziej poinformowana niż h_1. Możemy użyć tej definicji do porównania heurystyki zaproponowanej do rozwiązania 8-puzzli. Jak wskazano poprzednio, pierwsze wyszukiwanie szerokości jest równoważne algorytmowi A * z heurystycznym h_1 takim, że h_1 (x) = 0 dla wszystkich stanów x. Jest to trywialnie mniej niż h*. Wykazaliśmy również, że h_2, liczba płytek nie na miejscu w stosunku do stanu bramki, jest dolną granicą dla h*. W tym przypadku h_1 < h_2 ≤ h*. Wynika z tego, że heurystyka „liczby kafelków nie na miejscu" jest bardziej świadoma niż przeszukiwanie wszerz. Rysunek porównuje przestrzenie wyszukiwane przez te dwie heurystyki.

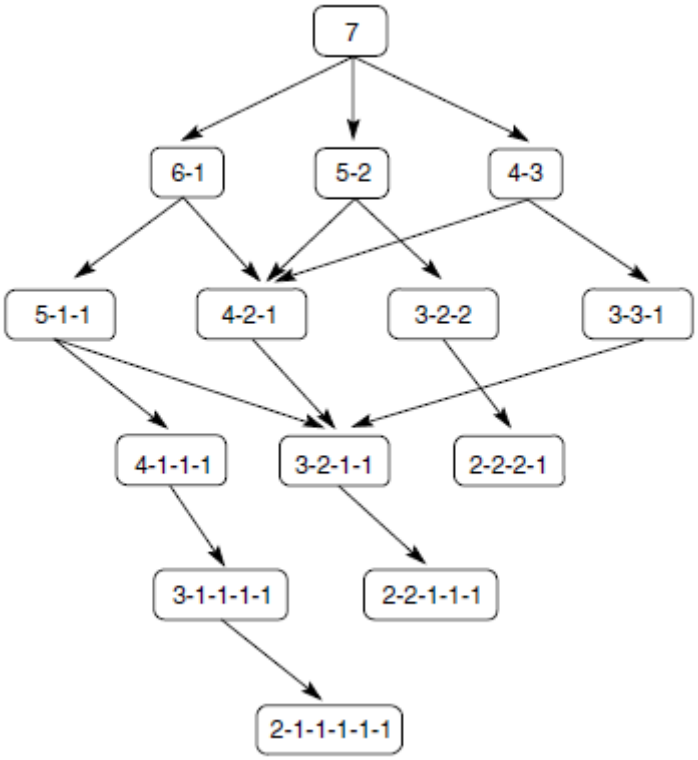


Zarówno h_1, jak i h_2 znajdują optymalną ścieżkę, ale $h_2$ocenia o wiele mniej stanów w tym procesie. Podobnie możemy argumentować, że heurystyka, która oblicza sumę bezpośrednich odległości, o które wszystkie płytki są nie na swoim miejscu, jest ponownie bardziej poinformowana niż obliczanie liczby płytek, które są nie na miejscu w odniesieniu do stanu celu, oraz rzeczywiście tak jest. Można

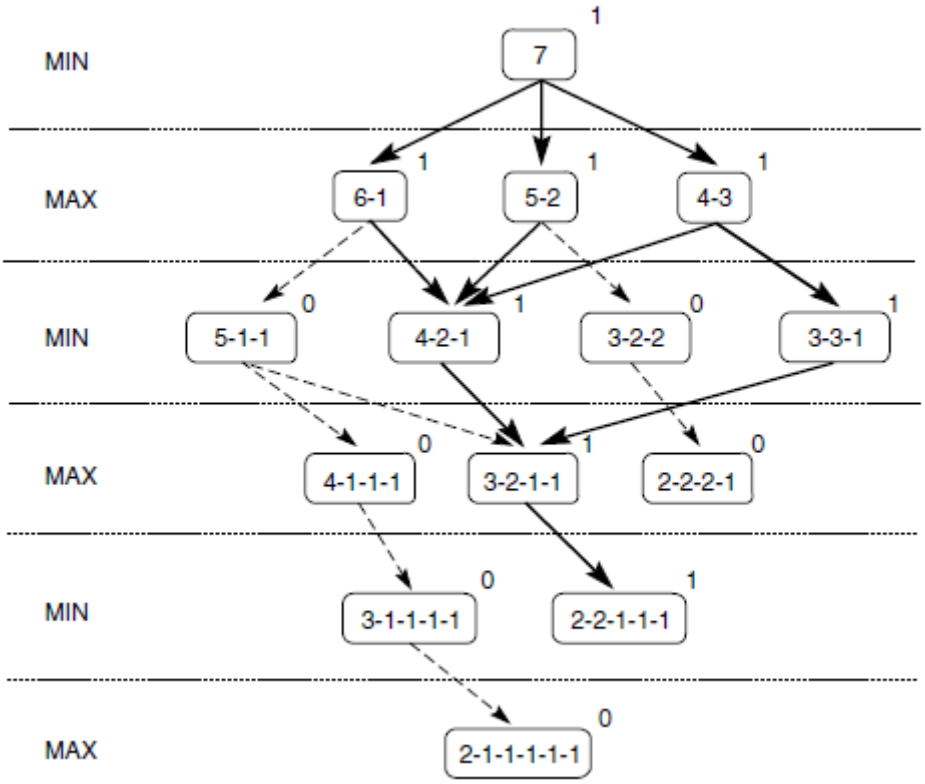
wizualizować sekwencję przestrzeni wyszukiwania, każdą mniejszą od poprzedniej, zbieżną na bezpośrednim optymalnym rozwiązaniu ścieżki. Jeśli heurystyka h_2 jest lepiej poinformowana niż h_1, wówczas zbiór stanów badanych przez h_2 jest podzbiorem stanów rozszerzonych o h_1. Można to zweryfikować, zakładając coś przeciwnego (że istnieje co najmniej jeden stan rozszerzony o h_2, a nie o h_1). Ale ponieważ h_2 jest lepiej poinformowany niż h_1, dla wszystkich n, $h_2(n)$ † $h_1(n)$ i oba są ograniczone przez h*, nasze założenie jest sprzeczne. Ogólnie rzecz biorąc, im bardziej poinformowany algorytm A*, tym mniej miejsca potrzebuje na rozszerzenie, aby uzyskać optymalne rozwiązanie. Musimy jednak uważać, aby obliczenia niezbędne do zastosowania bardziej świadomej heurystyki nie były tak nieefektywne, aby zrównoważyć korzyści wynikające ze zmniejszenia liczby przeszukiwanych stanów. Komputerowe programy szachowe stanowią ciekawy przykład tego kompromisu. Jedna szkoła myśli wykorzystuje prostą heurystykę i polega na szybkości komputera do głębokiego przeszukiwania przestrzeni wyszukiwania. Programy te często używają specjalistycznego sprzętu do oceny stanu w celu zwiększenia głębokości wyszukiwania. Inna szkoła opiera się na bardziej wyrafinowanej heurystyce w celu zmniejszenia liczby przeszukiwanych stanów zarządu. Te heurystyki obejmują obliczenia przewagi pionowej, kontrolę położenia planszy, możliwe strategie ataku, przekazane pionki i tak dalej. Samo obliczanie heurystyki może pociągać za sobą złożoność wykładniczą. Ponieważ całkowity czas pierwszych 40 ruchów w grze jest ograniczony, ważne jest, aby zoptymalizować kompromis między wyszukiwaniem a oceną heurystyczną. Optymalne połączenie ślepych poszukiwań i heurystyki pozostaje otwartym pytaniem empirycznym w szachach komputerowych, jak widać w bitwie Gary'ego Kasparowa Deep Blue

Wykorzystanie heurystyki w grach

Procedura Minimax dotycząca gier z wyczerpującym przeszukiwaniem graficznym zawsze była ważnym obszarem zastosowania algorytmów heurystycznych. Gry dwuosobowe są bardziej skomplikowane niż proste łamigłówki z powodu istnienia „wrogiego" i zasadniczo nieprzewidywalnego przeciwnika. Dają zatem interesujące możliwości rozwoju heurystyki, a także większe trudności w opracowaniu algorytmów wyszukiwania. Najpierw rozważymy gry, których przestrzeń stanu jest na tyle mała, że można ją dokładnie przeszukać; tutaj problemem jest systematyczne przeszukiwanie przestrzeni możliwych ruchów i przeciwdziała przeciwnikowi. Następnie patrzymy na gry, w których wyczerpujące przeszukiwanie wykresu gry jest niemożliwe lub niepożądane. Ponieważ tylko część przestrzeni stanów może być generowana i przeszukiwana, gracz musi użyć heurystyki, aby poprowadzić grę na drodze do zwycięskiego stanu. Najpierw rozważamy wariant gry nim, którego przestrzeń stanu można dokładnie przeszukać. Aby zagrać w tę grę, między dwoma przeciwnikami umieszcza się wiele żetonów; przy każdym ruchu gracz musi podzielić stos żetonów na dwa niepuste stosy o różnych rozmiarach. Tak więc 6 żetonów można podzielić na stosy 5 i 1 lub 4 i 2, ale nie 3 i 3. Pierwszy gracz, który nie może wykonać ruchu, przegrywa. W przypadku rozsądnej liczby tokenów przestrzeń stanu można dokładnie przeszukać. Poniższy rysunek ilustruje miejsce na grę z 7 żetonami.



Podczas grania w gry, w których przestrzeń stanu może być wyczerpująco określona, podstawową trudnością jest rozliczanie działań przeciwnika. Prosty sposób, aby sobie z tym poradzić, zakłada, że przeciwnik wykorzystuje tę samą wiedzę o przestrzeni stanów, z której korzystasz, i stosuje tę wiedzę w konsekwentny sposób, aby wygrać grę. Chociaż założenie to ma swoje ograniczenia, stanowi ono rozsądną podstawę do przewidywania zachowania przeciwnika. Przy tym założeniu Minimax przeszukuje przestrzeń gry. Przeciwnicy w grze są określani jako MIN i MAX. Chociaż częściowo wynika to z przyczyn historycznych, znaczenie tych nazw jest proste: MAX reprezentuje gracza, który próbuje wygrać, lub maksymalizować swoją przewagę. MIN to przeciwnik, który próbuje zminimalizować wynik MAX. Zakładamy, że MIN używa tych samych informacji i zawsze próbuje przejść do stanu najgorszego dla MAX. Wdrażając minimax, oznaczamy każdy poziom w przestrzeni wyszukiwania zgodnie z tym, czyim jest ruchem w tym momencie gry, MIN lub MAX. W przykładzie z rysunku MIN może się poruszać jako pierwszy.

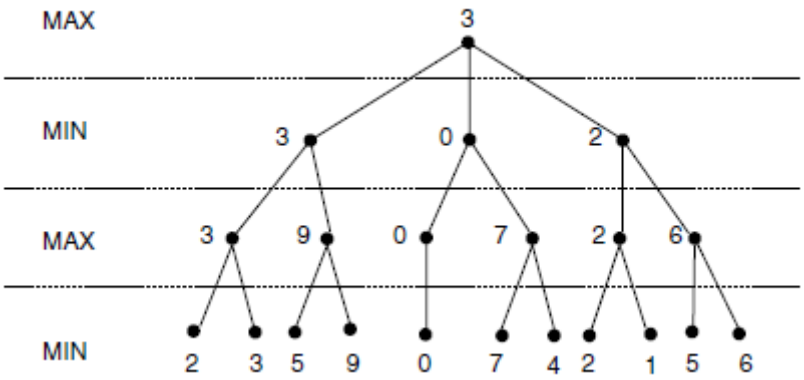


Każdy węzeł liścia ma wartość 1 lub 0, w zależności od tego, czy jest to wygrana dla MAX, czy dla MIN. Minimax propaguje te wartości w górę wykresu przez kolejne węzły nadrzędne zgodnie z regułą: Jeśli stanem nadrzędnym jest węzeł MAX, podaj mu maksymalną wartość wśród jego potomków. Jeśli rodzic jest węzłem MIN, podaj mu minimalną wartość jego potomków. Wartość, która jest w ten sposób przypisana do każdego stanu, wskazuje wartość najlepszego stanu, jaki ten gracz może osiągnąć (zakładając, że przeciwnik gra zgodnie z przewidywaniami algorytmu minimax). Te pochodne wartości służą do wybierania spośród możliwych ruchów. Wynik zastosowania minimax do wykresu przestrzeni stanu dla nim pokazano na powyższym rysunku. Wartości węzłów liści są propagowane w górę wykresu przy użyciu minimax. Ponieważ wszystkie możliwe pierwsze ruchy MIN prowadzą do węzłów o wartości pochodnej 1, drugi gracz, MAX, zawsze może zmusić grę do wygranej, niezależnie od pierwszego ruchu MIN. MIN może wygrać tylko wtedy, gdy MAX gra głupio. Na powyższym rysunku MIN może wybrać dowolną z alternatyw pierwszego ruchu, z wynikowymi ścieżkami wygranej dla MAKSU w pogrubionych strzałkach. Chociaż istnieją gry, w których można wyczerpująco przeszukać przestrzeń stanu, najbardziej interesujące gry tego nie robią. Następnie sprawdzamy wyszukiwanie stałej głębokości

Minimaksowanie do stałej głębokości warstwy

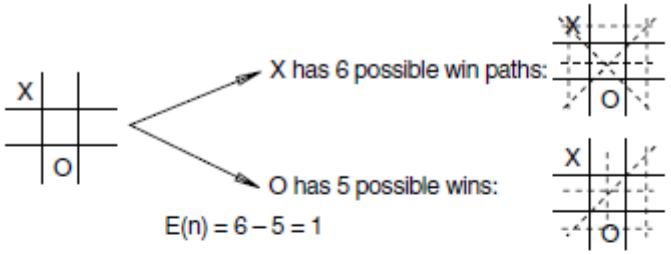
Stosując minimax do bardziej skomplikowanych gier, rzadko jest możliwe rozwinięcie wykresu przestrzeni stanu do węzłów liści. Zamiast tego przestrzeń stanu jest przeszukiwana do z góry określonej liczby poziomów, określonych przez dostępne zasoby czasu i pamięci. Strategia ta nazywana jest n-warstwową prognozą, gdzie n to liczba zbadanych poziomów. Ponieważ liście tego podgrafu nie są końcowymi stanami gry, nie jest możliwe podanie im wartości odzwierciedlających wygraną lub przegraną. Zamiast tego każdy węzeł otrzymuje wartość zgodnie z jakąś heurystyczną funkcją oceny. Wartość propagowana z powrotem do węzła głównego nie wskazuje, czy można osiągnąć wygraną (jak w poprzednim przykładzie), ale jest po prostu heurystyczną wartością najlepszego stanu, jaki można

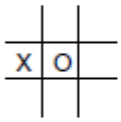
osiągnąć w n ruchach z korzenia . Spojrzenie w przyszłość zwiększa moc heurystyki, umożliwiając jej zastosowanie na większym obszarze przestrzeni stanu. Minimax konsoliduje te oddzielne oceny dla stanu przodka. W grze konfliktowej każdy gracz próbuje pokonać drugiego, więc wiele heurystyk gry mierzy bezpośrednio przewagę jednego gracza nad drugim. W warcabach lub szachach ważna jest przewaga pionowa, więc prosty heurysta może wziąć różnicę w liczbie pionów należących do MAX i MIN i spróbować zmaksymalizować różnicę między tymi miarami pionów. Bardziej wyrafinowana strategia może przypisywać różne wartości do pionów, w zależności od ich wartości (np. Królowa kontra pionek lub król kontra zwykły pion) lub lokalizacji na planszy. Większość gier zapewnia nieograniczone możliwości projektowania heurystyki. Wykresy gry są wyszukiwane według poziomu lub warstwy. Jak widzieliśmy na rysunku , MAX i MIN na przemian wybierają ruchy. Każdy ruch gracza określa nową warstwę wykresu. Gra odtwarzające programy zazwyczaj patrzą w przyszłość na ustaloną głębokość warstwy, określoną przez ograniczenia przestrzeni / czasu komputera. Stany na tej warstwie są mierzone heurystycznie, a wartości są propagowane z powrotem na wykresie za pomocą minimax. Algorytm wyszukiwania wykorzystuje następnie te pochodne wartości, aby wybrać spośród możliwych następnych ruchów. Po przypisaniu oceny do każdego stanu na wybranej warstwie program się propaguje do wartości do każdego stanu nadrzędnego. Jeśli rodzic jest na poziomie MIN, tworzona jest kopia zapasowa minimalnej wartości dzieci. Jeśli rodzic jest węzłem MAX, minimax przypisuje mu maksymalną wartość swoich potomków. Maksymalizując dla rodziców MAX i minimalizując dla MIN, wartości wracają w górę wykresu do dzieci bieżącego stanu. Wartości te są następnie używane przez bieżący stan do wybierania spośród jego potomków. Rysunek pokazuje minimax w hipotetycznej przestrzeni stanów za czterowarstwowe spojrzenie w przyszłość.

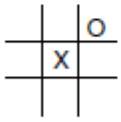


Możemy przedstawić kilka końcowych uwag na temat procedury minimax. Po pierwsze, i najważniejsze, oceny dowolnej (wcześniej ustalonej) ustalonej głębokości warstwy mogą być bardzo mylące. Gdy zastosowana jest heurystyka z ograniczonym wyprzedzeniem, możliwe, że głębokość przewidywania może nie wykryć, że heurystycznie obiecująca ścieżka prowadzi później do złej sytuacji w grze. Jeśli twój przeciwnik w szachach oferuje wieżę jako przynętę, aby zabrać twoją królową, a ocena ocenia tylko warstwę, w której oferowana jest wieża, ocena będzie stronnicza w stosunku do tego stanu. Niestety wybór stanu może spowodować utratę całej gry! Nazywa się to efektem horyzontalnym. Zwykle przeciwdziała się temu, przeszukując kilka warstw głębiej ze stanów, które wyglądają wyjątkowo dobrze. To selektywne pogłębienie poszukiwań w ważnych obszarach nie spowoduje jednak zniknięcia efektu horyzontu. Poszukiwanie musi gdzieś się zatrzymać i będzie ślepe na stany poza tym punktem. Jest jeszcze inny efekt, który występuje w minimaksowaniu na podstawie ocen heurystycznych. Oceny, które odbywają się głęboko w przestrzeni kosmicznej, mogą być stronnicze ze względu na ich głębokość (Pearl 1984). W ten sam sposób, w jaki średnia produktów różni się od iloczynu średnich, oszacowanie minimax (czego pragniemy) różni się od minimax oszacowań (co

robimy). W tym sensie głębsze wyszukiwanie z oceną i minimax zwykle oznacza, ale nie zawsze, oznacza lepsze wyszukiwanie. Dalszą dyskusję na temat tych problemów i możliwe rozwiązania można znaleźć w Pearl. Kończąc dyskusję na temat minimax, prezentujemy aplikację na kółko i krzyżyk, zaadaptowaną przez Nilssona. Stosowana jest nieco bardziej złożona heurystyka, która próbuje zmierzyć konflikt w grze. Heurystyka przyjmuje stan, który należy zmierzyć, zlicza całkowitą liczbę zwycięskich linii otwartych do MAX, a następnie odejmuje całkowitą liczbę zwycięskich linii otwartych do MIN. Wyszukiwanie próbuje zmaksymalizować tę różnicę. Jeśli stan jest wymuszoną wygraną dla MAX, jest on oceniany jako + ∞; wymuszone zwycięstwo dla MIN jako ∞ Rysunek poniższy pokazuje tę heurystykę zastosowaną do kilku stanów próby.

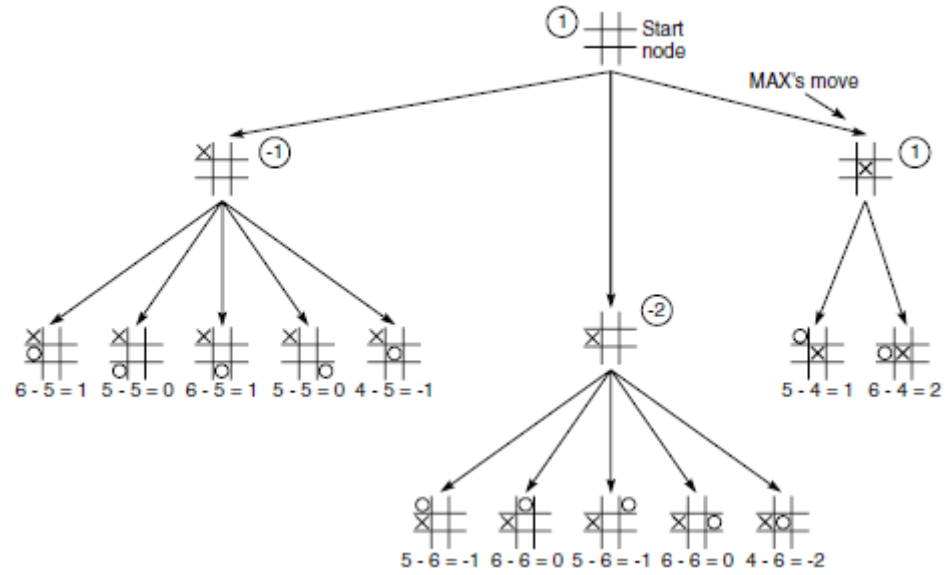


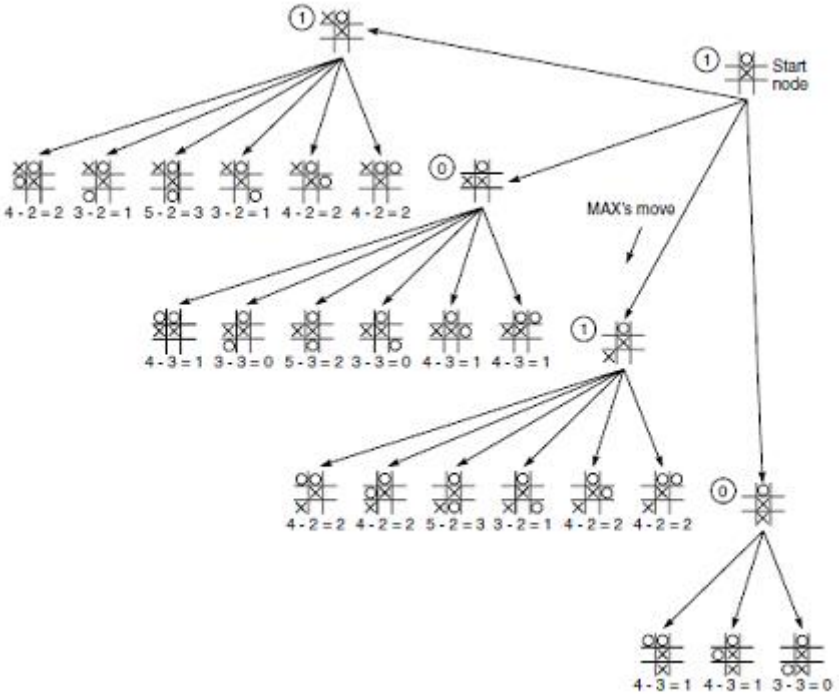


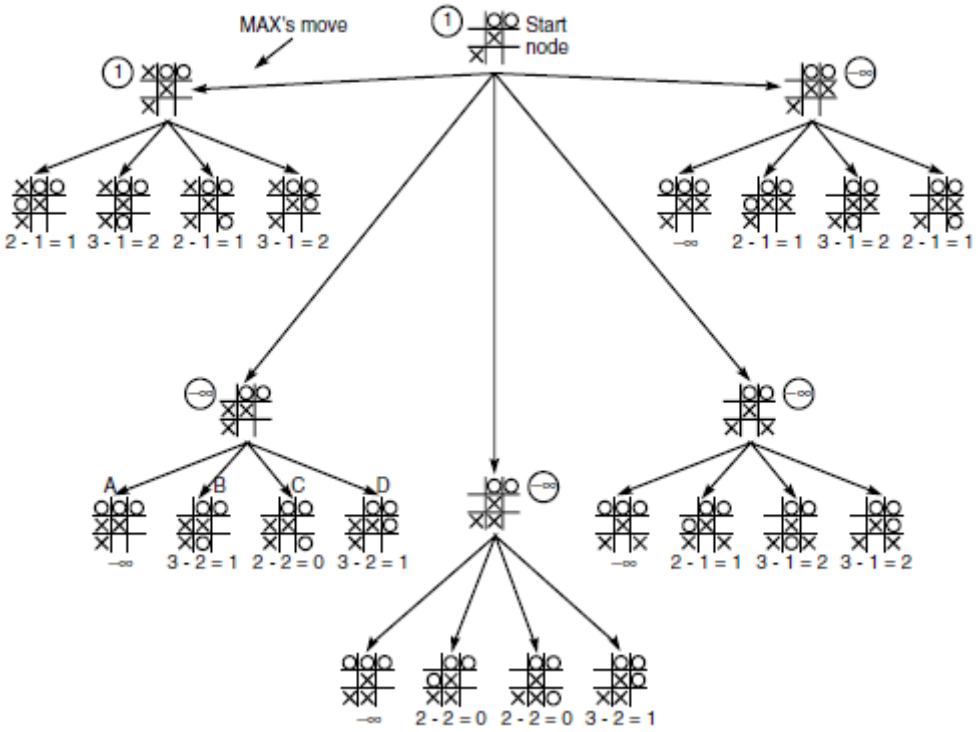


Heuristic is E(n) = M(n) – O(n)
where M(n) is the total of My possible winning lines
O(n) is total of Opponent's possible winning lines
E(n) is the total Evaluation for state n

Kolejne trzy rysunki pokazują heurystykę powyższego rysunku w dwuwarstwowej minimaksie. Te liczby pokazują ocenę heurystyczną, tworzenie kopii zapasowych minimax i ruch MAX, z pewnym rodzajem przerywnika powiązanego zastosowanym do ruchów o równej wartości, z Nilsson.


1
Start
node
MAX's move
-1
1
-2
6 - 5 = 1
5 - 5 = 0
6 - 5 = 1
5 - 5 = 0
4 - 5 = -1
5 - 4 = 1
6 - 4 = 2
5 - 6 = -1
6 - 6 = 0
5 - 6 = -1
6 - 6 = 0
4 - 6 = -2

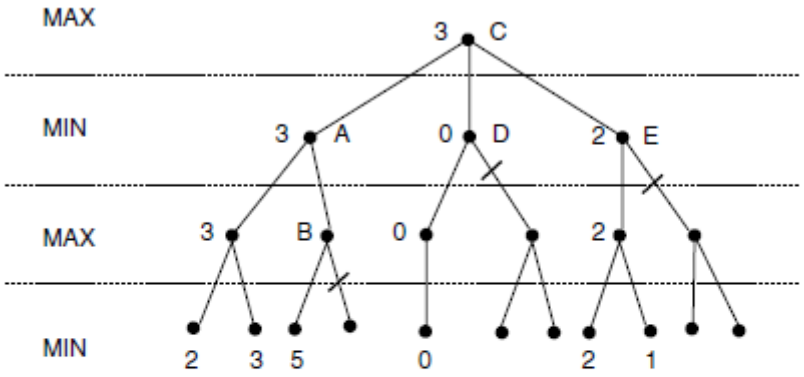

1
1
Start
node
0
MAX's move
1
0
4 - 2 = 2
3 - 2 = 1
5 - 2 = 3
3 - 2 = 1
4 - 2 = 2
4 - 2 = 2
4 - 3 = 1
3 - 3 = 0
5 - 3 = 2
3 - 3 = 0
4 - 3 = 1
4 - 3 = 1
4 - 2 = 2
4 - 2 = 2
5 - 2 = 3
3 - 2 = 1
4 - 2 = 2
4 - 2 = 2
4 - 3 = 1
4 - 3 = 1
3 - 3 = 0



Prosty minimax wymaga dwuprzebiegowej analizy przestrzeni poszukiwań, z których pierwsza schodzi do głębokości warstwy, a następnie stosuje heurystykę, a druga propaguje wartości z powrotem do drzewa. Minimax realizuje wszystkie gałęzie w przestrzeni, w tym wiele, które można zignorować lub przyciąć za pomocą bardziej inteligentnego algorytmu. Naukowcy grający w gry opracowali klasę technik wyszukiwania o nazwie przycinanie alfa-beta, po raz pierwszy zaproponowaną pod koniec lat 50. XX wieku, aby poprawić wydajność wyszukiwania w grach dwuosobowych. Pomysł wyszukiwania alfa-beta jest prosty: zamiast przeszukiwać całą przestrzeń na głębokość warstwy, wyszukiwanie alfa-beta przebiega w pierwszej kolejności. Podczas wyszukiwania tworzone są dwie wartości, zwane alfa i beta. Wartość alfa, powiązana z węzłami MAX, nigdy nie może się zmniejszyć, a wartość beta, powiązana z węzłami MIN, nigdy nie może wzrosnąć. Załóżmy, że wartość alfa węzła MAX wynosi 6. Wówczas MAX nie musi brać pod uwagę żadnej kopii zapasowej wartości mniejszej lub równej 6, która jest powiązana z dowolnym węzłem MIN poniżej. Alfa jest najgorsza jak ,że MAX może „zdobyć", biorąc pod uwagę, że MIN również zrobi wszystko, co w jego mocy. Podobnie, jeśli MIN ma wartość beta 6, nie musi uwzględniać żadnego węzła MAX poniżej, który ma wartość 6 lub więcej. Aby rozpocząć wyszukiwanie alfa-beta, schodzimy do pełnej głębokości warstwy w pierwszej kolejności i stosujemy naszą heurystyczną ocenę do stanu i całego jego rodzeństwa. Załóżmy, że są to węzły MIN. Kopie zapasowe maksimum z tych wartości MIN są następnie tworzone dla elementu nadrzędnego (węzeł MAX, podobnie jak w minimax). Ta wartość jest następnie oferowana dziadkom tych MIN jako potencjalna wartość graniczna beta. Następnie algorytm zstępuje na inne wnuki i kończy badanie ich rodzica, jeśli którakolwiek z ich wartości jest równa lub większa niż ta wartość beta. Podobne procedury można opisać dla przycinania alfa nad wnukami węzła MAX. Dwie zasady zakończenia wyszukiwania, oparte na wartościach alfa i beta, to:

1. Wyszukiwanie można zatrzymać poniżej dowolnego węzła MIN o wartości beta mniejszej lub równej wartości alfa dowolnego z jego przodków MAX.
2. Wyszukiwanie może zostać zatrzymane poniżej dowolnego węzła MAX o wartości alfa większej lub równej wartości beta dowolnego z jego przodków węzła MIN.

Przycinanie alfa-beta wyraża zatem związek między węzłami na warstwie n i węzłami na warstwie n + 2, pod którymi można wykluczyć całe poddrzewa zakorzenione na poziomie n + 1. Jako przykład, rysunek stosuje przycinanie alfa-beta.



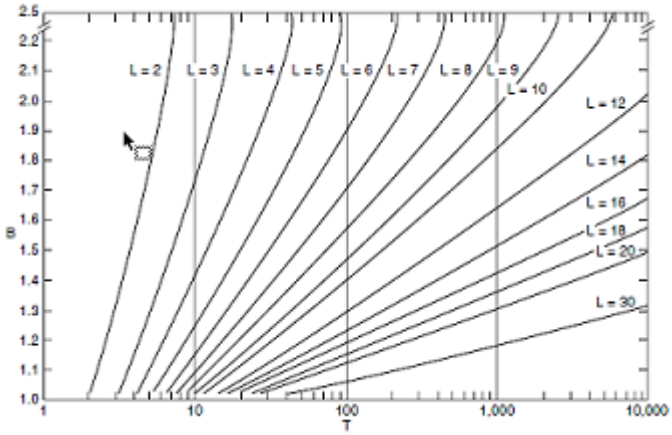
A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

Zauważ, że wynikowa wartość kopii zapasowej jest identyczna z wynikiem minimax, a oszczędność wyszukiwania w porównaniu z minimax jest znaczna. Przy przypadkowym uporządkowaniu stanów w przestrzeni wyszukiwania, alfa-beta może skutecznie podwoić głębokość przestrzeni wyszukiwania uwzględnianą przy stałym zaangażowaniu komputera czasoprzestrzennego. Jeśli występuje szczególne niefortunne zamówienie, alfa-beta przeszukuje nie więcej miejsca niż normalna minimax; wyszukiwanie odbywa się jednak tylko w jednym przebiegu.

Problemy ze złożonością

Najtrudniejszym aspektem problemów kombinatorycznych jest to, że „eksplozja” często ma miejsce, a projektanci programów nie zdają sobie z tego sprawy. Ponieważ większość działań ludzkich, obliczeniowych i innych, ma miejsce w świecie czasu liniowego, trudno nam docenić wzrost wykładniczy. Słyszymy skargę: „Gdybym tylko miał większy (lub szybszy lub bardzo równoległy) komputer, mój problem zostałby rozwiązany”. Takie twierdzenia, często zgłaszane po wybuchu, są zwykle śmieciami. Problem nie został właściwie zrozumiany i / lub nie podjęto odpowiednich kroków, aby rozwiązać kombinatorykę sytuacji. Pełny zakres kombinatorycznego wzrostu budzi wyobraźnię. Oszacowano, że liczba stanów wytworzonych przez pełne przeszukiwanie przestrzeni możliwych ruchów szachowych wynosi około 10^{120}. Nie jest to „kolejna duża liczba”; jest ona porównywalna z liczbą cząsteczek we wszechświecie lub liczbą nanosekundy od „wielkiego wybuchu”. Opracowano kilka miar, aby pomóc obliczyć złożoność. Jednym z nich jest czynnik rozgałęziający przestrzeń. Definiujemy współczynnik rozgałęziania jako średnią liczbę rozgałęzień (dzieci), które są rozwinięte z dowolnego stanu w przestrzeni. Liczba stanów na głębokości n wyszukiwania jest równa współczynnikowi rozgałęzienia podniesionemu do n-tej potęgi. Po obliczeniu współczynnika rozgałęzienia dla przestrzeni można oszacować koszt wyszukiwania w celu wygenerowania ścieżki o

dowolnej długości. Rysunek poniższy przedstawia zależność między B (rozgałęzienie), L (długość ścieżki) i T (stany całkowite w wyszukiwaniu) dla małych wartości.



Liczba jest logarytmiczna w T, więc L nie jest „prostą" linią, którą wygląda na wykresie. Kilka przykładów wykorzystujących ten rysunek pokazuje, jak złe mogą być rzeczy. Jeśli czynnik rozgałęziający to 2, potrzeba przeszukania około 100 stanów w celu zbadania wszystkich ścieżek, które sięgają sześciu poziomów w głąb przestrzeni wyszukiwania. Przeszukanie około 10 000 stanów wymaga rozważenia ścieżek o głębokości 12 ruchów. Jeśli rozgałęzienie można zmniejszyć do 1,5 (za pomocą heurystyki), wówczas ścieżkę dwukrotnie dłuższą można zbadać dla tej samej liczby przeszukiwanych stanów. Wzór matematyczny, który wytworzył relacje z rysunku, jest następujący:

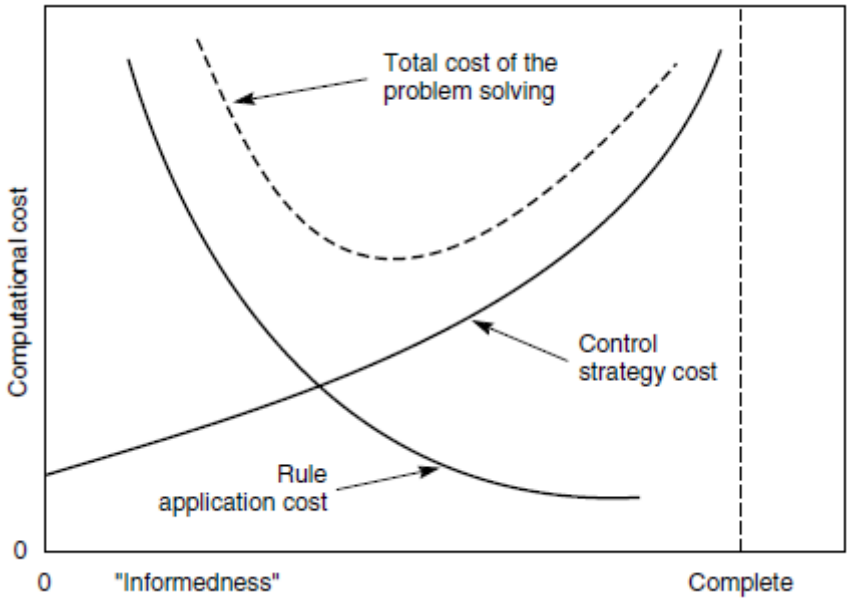
$T = B + B^2 + B^3 + ... + BL$

o stanach całkowitych T, długości ścieżki L i współczynniku rozgałęzienia B. To równanie ogranicza się do:

$T = B (B^L - 1) / (B - 1)$

Pomiar przestrzeni poszukiwań jest zwykle procesem empirycznym, polegającym na znacznej zabawie z problemem i testowaniu jego wariantów. Załóżmy na przykład, że chcemy ustalić współczynnik rozgałęzienia 8-łamigłówki. Obliczamy całkowitą liczbę możliwych ruchów: 2 z każdego rogu w sumie 8 ruchów w rogu, 3 od środka każdej strony w sumie 12 i 4 od środka siatki w sumie 24. Podzielony przez 9, różna liczba możliwych lokalizacji pustych miejsc, daje średni współczynnik rozgałęzienia wynoszący 2,67. Jak widać na rysunku, nie jest to zbyt dobre do głębokiego poszukiwania. Jeśli wyeliminujemy ruchy bezpośrednio z powrotem do stanu nadrzędnego (już wbudowanego w algorytmy wyszukiwania w tym rozdziale), jest jeden ruch mniej z każdego stanu. Daje to współczynnik rozgałęzienia wynoszący 1,67, znaczną poprawę, która może (w niektórych przestrzeniach stanu) umożliwić wyczerpujące wyszukiwanie. Jak wspomniano wcześniej, koszt złożoności algorytmu można również zmierzyć na podstawie wielkości list otwartych i zamkniętych. Jedną z metod utrzymywania rozsądnego rozmiaru otwartego jest oszczędzanie na otwartym tylko kilku (heurystycznie) najlepszych stanów. Może to prowadzić do bardziej ukierunkowanego wyszukiwania, ale wiąże się z ryzykiem wyeliminowania najlepszej, a nawet jedynej ścieżki rozwiązania. Ta technika utrzymywania rozmiaru związanego z otwartym, często będąca funkcją liczby kroków oczekiwanych dla wyszukiwania, nazywa się wyszukiwaniem wiązki. Próbując ograniczyć rozgałęzienia wyszukiwania lub w inny sposób ograniczyć przestrzeń wyszukiwania, przedstawiliśmy pojęcie bardziej świadomej heurystyki. Im bardziej świadome wyszukiwanie, tym mniej miejsca należy przeszukać, aby uzyskać minimalne rozwiązanie ścieżki. Jak wskazaliśmy wcześniej, koszty obliczeniowe dodatkowych informacji potrzebnych do dalszego ograniczenia przestrzeni wyszukiwania mogą nie zawsze być do przyjęcia. Aby rozwiązać problemy na komputerze, nie wystarczy znaleźć minimalną ścieżkę. Musimy również

zminimalizować całkowite koszty procesora. Rysunek poniższy, zaczerpnięty z analizy Nilssona, jest nieformalną próbą rozwiązania tych problemów.



Współrzędna „poinformowania" oznacza kwotę kosztów informacji uwzględnionych w heurystyce oceny, które mają na celu poprawę wydajności. Współrzędna procesora oznacza koszty procesora do wdrożenia oceny stanu i innych aspektów wyszukiwania. Wraz ze wzrostem informacji zawartych w heurystyce wzrasta koszt jednostki centralnej heurystyki. Podobnie, gdy heurystyka staje się bardziej informowana, koszt jednostki centralnej oceny stanów zmniejsza się, ponieważ rozważanych jest mniej stanów. Kosztem krytycznym jest jednak całkowity koszt obliczenia heurystycznych stanów oceny PLUS i zwykle pożądane jest zminimalizowanie tego kosztu. Wreszcie, heurystyczne wyszukiwanie i / lub wykresy są ważnym obszarem zainteresowania, ponieważ przestrzenie stanowe dla systemów eksperckich często mają taką formę. Całkowite ogólne poszukiwanie tych struktur składa się z wielu komponentów omówionych już w tym i poprzednim rozdziale. Ponieważ wszyscy i dzieci muszą zostać przeszukani, aby znaleźć cel, heurystyczna ocena kosztu przeszukania i węzła jest sumą oszacowań przeszukiwania dzieci. Istnieje jednak wiele innych zagadnień heurystycznych, oprócz numerycznej wyceny jednostek i stanów, w badaniach i / lub grafach, takich jak stosowane w systemach opartych na wiedzy. Na przykład, jeżeli do rozwiązania stanu rodzicielskiego wymagana jest satysfakcja zbioru i dzieci, które dziecko należy rozważyć jako pierwsze? Stan najbardziej kosztowny do oceny? Stan najbardziej narażony na upadek? Stan, który ludzki ekspert uważa za pierwszy? Decyzja jest ważna zarówno dla wydajności obliczeniowej, jak i dla całkowitego kosztu, np. W medycznych lub innych testach diagnostycznych, systemu wiedzy.

Epilog

Przestrzenie poszukiwania interesujących problemów zwykle rosną wykładniczo; Wyszukiwanie heurystyczne jest podstawowym narzędziem zarządzania tą złożonością kombinatoryczną. W tym rozdziale zaprezentowano różne strategie kontroli dla wdrażania wyszukiwania heurystycznego. Rozpoczęliśmy rozdział dwoma tradycyjnymi algorytmami, oba odziedziczone po dyscyplinie badań operacyjnych, wspinaczki i programowania dynamicznego. Zalecamy przeczytanie artykułu Arthura Samuela omawiającego jego program do gry w warcaby i jego wyrafinowane wykorzystanie do wspinaczki i wyszukiwania minimax. Samuel przedstawia również wczesne interesujące przykłady zaawansowanego systemu zarządzania pamięcią i programu, który jest w stanie się uczyć. Opracowane

przez Bellmana algorytmy programowania dynamicznego pozostają ważne w takich obszarach, jak przetwarzanie języka naturalnego, w których konieczne jest porównywanie ciągów znaków, słów lub fonemów. Programowanie dynamiczne jest często nazywane algorytmami do przodu / do tyłu lub Viterbi. Dla bezsilnych przykładów zastosowania programowania dynamicznego do analizy języka patrz Jurafsky i Martin. Następnie przedstawiliśmy heurystykę w kontekście tradycyjnego wyszukiwania w przestrzeni państwowej. Zaprezentowaliśmy algorytmy A i A* do implementacji najlepszego wyszukiwania. Wyszukiwanie heurystyczne wykazano za pomocą prostych gier, takich jak 8-puzzle, i rozszerzono na bardziej złożone przestrzenie problemowe generowane przez oparte na regułach systemy eksperckie. Zastosowaliśmy wyszukiwanie heurystyczne do gier dwuosobowych, używając przyszłości z przycinaniem minimax i alphabeta, aby spróbować przewidzieć zachowanie przeciwnika. Po omówieniu algorytmów A * przeanalizowaliśmy ich zachowanie, biorąc pod uwagę właściwości, w tym dopuszczalność, monotoniczność i wiedzę. Dyscyplina teorii złożoności ma istotne konsekwencje dla praktycznie każdej gałęzi informatyki, szczególnie dla analizy wzrostu przestrzeni stanu i heurystycznego przycinania. Teoria złożoności bada naturalną złożoność problemów (w przeciwieństwie do algorytmów zastosowanych do tych problemów). Kluczowym przypuszczeniem w teorii złożoności jest to, że istnieje klasa nierozwiązywalnych problemów. Ta klasa, określana jako NP-twarda (niedeterministycznie wielomianowa), składa się z problemów, których nie można rozwiązać w czasie krótszym niż wykładniczy bez uciekania się do zastosowania heurystyki. Prawie wszystkie interesujące problemy z wyszukiwaniem należą do tej klasy.

Sztuczna inteligencja : metody stochastyczne (V / XVI)

ĆWICZENIA

1. Rzutka sześciokątna kostka jest rzucana pięć razy, a kolejne liczby są rejestrowane w sekwencji. Ile jest różnych sekwencji?
2. Znajdź liczbę możliwych do odróżnienia permutacji liter w słowie MISSISSIPPI, liter wyrazu ASSOCIATIVE.
3. Załóżmy, że urna zawiera 15 kulek, z których osiem jest czerwonych, a siedem czarnych. Na ile sposobów można wybrać pięć piłek, aby:
a. wszystkie pięć jest czerwonych? wszystkie pięć jest czarnych?
b. dwa są czerwone, a trzy czarne
co. co najmniej dwa są czarne
4. Na ile sposobów można wybrać komitet złożony z trzech członków wydziału i dwóch studentów z grupy pięciu wykładowców i siedmiu studentów?
5. W badaniu przeprowadzonym wśród 250 widzów telewizyjnych 88 lubi oglądać wiadomości, 98 lubi oglądać sport, a 94 lubi oglądać komedie. 33 osoby lubią oglądać wiadomości i sport, 31 lubią oglądać sport i komedię, a 35 lubi oglądać wiadomości i komedie. 10 osób lubi oglądać wszystkie trzy. Załóżmy, że osoba z tej grupy jest wybierana losowo:
za. Jakie jest prawdopodobieństwo, że oglądają wiadomości, ale nie sport?
b. Jakie jest prawdopodobieństwo, że oglądają wiadomości lub sport, ale nie komedię?
do. Jakie jest prawdopodobieństwo, że nie oglądają sportu ani wiadomości?
6. Jakie jest prawdopodobieństwo, że czterocyfrowa liczba całkowita bez zer początkowych:
a. Ma 3, 5 lub 7 jako cyfrę?
b. Zaczyna się od 3, kończy na 5, czy ma 7 jako cyfrę?
7. Dwie kości są rzucane. Znajdź prawdopodobieństwo, że ich suma wynosi:
a. 4
b. 7 lub liczba parzysta
c. 10 lub więcej
8. Karta jest dobierana ze zwykłej talii 52 kart. Jakie jest prawdopodobieństwo:
a. losowanie karty twarzy (walet, królowa, król lub as)
b. rysunek królowej lub pika
c. losowanie karty twarzy lub maczugi
9. Jakie jest prawdopodobieństwo, że zostaną rozdane następujące układy w pokera na pięć kart (z normalnej talii 52 kart)?
a. „Kolor” lub wszystkie karty tego samego koloru.
b. „Full house” lub dwie karty o tej samej wartości i trzy karty o innej wartości.
c. „Poker królewski” lub dziesiątka, walet, królowa, król i as tego samego koloru.
10. Oczekiwanie jest średnią lub średnią wartości zmiennej losowej. Na przykład rzucając kostką, można ją obliczyć, sumując uzyskane wartości z dużej liczby rzutów, a następnie dzieląc przez liczbę rzutów.
a. Jakie są oczekiwania po rzucie uczciwą kostką?

b. Wartość koła ruletki z 37 równie prawdopodobnymi wynikami?
c. Wartość losowania z zestawu kart (as jest wyceniany na 1, wszystkie pozostałe karty na 10)?
11. Załóżmy, że gramy w grę, w której rzucamy kostką, a następnie otrzymujemy kwotę równą wartości kości. Na przykład, jeśli pojawi się 3, otrzymamy 3 USD. Czy zagranie w tę grę kosztuje nas 4 USD, czy jest to uzasadnione?
12. Rozważ sytuację, w której losowo generowane są ciągi bitów o długości cztery. Zademonstruj, czy zdarzenie wytworzenia ciągów bitów zawierających parzystą liczbę 1 jest niezależne od zdarzenia wytworzenia ciągów bitów, które kończą się na 1.
13. Pokaż, że zdanie p (A, B | C) = p (A | C) p (B | C) jest równoważne zarówno p (A | B, C) = p (A | C) i p (B | A, C) = p (B | C).
14. Przy wytwarzaniu produktu 85% wytwarzanych produktów nie jest wadliwych. Spośród skontrolowanych produktów 10% dobrych jest postrzeganych jako wadliwe i nie jest wysyłanych, podczas gdy tylko 5% wadliwych produktów jest zatwierdzonych i wysłanych. Jeśli produkt jest wysłane, jakie jest prawdopodobieństwo, że jest uszkodzony?
15. Badanie krwi jest w 90% skuteczne w wykrywaniu choroby. Fałszywie diagnozuje również, że u zdrowej osoby choroba występuje w 3% przypadków. Jeśli 10% badanych ma chorobę, jakie jest prawdopodobieństwo, że osoba, która uzyska pozytywny wynik, faktycznie będzie miała tę chorobę?
16. Załóżmy, że towarzystwo ubezpieczeń samochodowych klasyfikuje kierowcę jako dobry, średni lub zły. Spośród wszystkich ubezpieczonych kierowców 25% jest sklasyfikowanych jako dobre, 50% to średnie, a 25% to złe. Załóżmy, że na nadchodzący rok dobry kierowca ma 5% szansy na wypadek, a przeciętny kierowca ma 15% szansy na wypadek, a zły kierowca ma 25% szansy. Jeśli miałeś wypadek w ubiegłym roku, jakie jest prawdopodobieństwo, że jesteś dobrym kierowcą?
17. Trzej więźniowie, A, B, C są w swoich celach. Powiedziano im, że jeden z nich zostanie stracony następnego dnia, a inni zostaną ułaskawieni. Tylko gubernator wie, kto zostanie stracony. Więzień A prosi strażnika o przysługę. „Zapytaj gubernatora, kto zostanie stracony, a następnie powiedz więźniowi B lub C, że zostaną ułaskawieni". Strażnik robi to, o co go poproszono, a następnie wraca i mówi więźniowi A, że powiedział więźniowi B, że on (B) zostanie ułaskawiony. Jakie są szanse na egzekucję więźnia A, biorąc pod uwagę tę wiadomość? Czy jest więcej informacji niż przed jego prośbą do strażnika?

METODY STOCHASTYCZNE

Wprowadzenie

Wprowadziliśmy wyszukiwanie heurystyczne jako podejście do rozwiązywania problemów w domenach, w których albo problem nie ma dokładnego rozwiązania, albo gdzie pełna przestrzeń stanu może być zbyt kosztowna do obliczenia. W tym rozdziale proponujemy metodologię stochastyczną jako odpowiednie dla tych sytuacji. Argumentacja probabilistyczna jest również odpowiednia w sytuacjach, w których informacje o stanie znajdują się na podstawie próbkowania bazy informacji, a modele przyczynowe są wyciągane z danych. Jedną z ważnych dziedzin zastosowań dla stochastycznej metodologii jest rozumowanie diagnostyczne, w którym relacje przyczyna / skutek nie zawsze są ujmowane w sposób czysto deterministyczny, jak to często jest możliwe w podejściach opartych na wiedzy do rozwiązywania problemów, które widzieliśmy w rozdziałach 2, 3 , 4 i zobaczymy ponownie w rozdziale 8. Sytuacja diagnostyczna zwykle przedstawia dowody, takie jak gorączka lub ból głowy, bez dalszego przyczynowego uzasadnienia. W rzeczywistości dowody mogą często wskazywać na kilka różnych przyczyn, np. Gorączka może być spowodowana grypą lub infekcją. W takich sytuacjach informacje probabilistyczne mogą często wskazywać i uszeregować pod względem ważności możliwe wyjaśnienia dowodów. Inną interesującą aplikacją dla stochastycznej metodologii jest hazard, w którym podobno przypadkowe zdarzenia, takie jak rzut kostkami, rozdawanie kart potasowanych lub obrót koła ruletki, powodują potencjalną wypłatę gracza. W rzeczywistości w XVIII wieku próba

stworzenia matematycznych podstaw hazardu była ważną motywacją dla Pascala (a później Laplace'a) do opracowania rachunku probabilistycznego. Wreszcie, jak zauważono w sekcji 1.1.4, „umiejscowienie" rachunkowości inteligencji sugeruje, że decyzje ludzkie często pojawiają się w złożonych, krytycznych czasowo i ucieleśnionych środowiskach, w których rachunek w pełni mechanistyczny może po prostu nie być możliwy do zdefiniowania lub, jeśli jest zdefiniowany, może nie obliczać odpowiedzi w użytecznym czasie. W takich sytuacjach inteligentne działania najlepiej można postrzegać jako stochastyczne reakcje na przewidywane koszty i korzyści. Następnie opisujemy kilka obszarów problemowych, wśród których wiele, w których obliczeniowa implementacja inteligencji jest często wykorzystywana metodologią stochastyczną; obszary te będą głównymi tematami w dalszych częściach.

1. Uzasadnienie diagnostyczne. Na przykład w diagnozie medycznej nie zawsze istnieje oczywisty związek przyczynowo-skutkowy między zestawem objawów przedstawionych przez pacjenta a przyczynami tych objawów. W rzeczywistości te same zestawy objawów często sugerują wiele możliwych przyczyn. Ważne są również modele probabilistyczne , złożone sytuacje mechaniczne, takie jak monitorowanie lotów samolotów lub helikopterów. Systemy oparte na regułach i probabilistyczne zostały zastosowane w tych i innych domenach diagnostycznych.

2. Rozumienie języka naturalnego. Jeśli komputer ma rozumieć i używać ludzkiego języka, musi on być w stanie scharakteryzować sposób, w jaki sami ludzie używają tego języka. Wyrazy, wyrażenia i metafory są przyswajane, ale zmieniają się i ewoluują wraz z upływem czasu. Obsługuje metodologię stochastyczną rozumienia języka; na przykład, gdy system obliczeniowy jest szkolony w bazie danych specyficznych zastosowań języka (zwanej językoznawstwem korpusowym). Rozważamy te problemy językowe w dalszej części.

3. Planowanie i harmonogramowanie. Gdy agent tworzy plan, na przykład samochód na wakacje, często zdarza się, że żadna deterministyczna sekwencja operacji nie jest gwarantowana. Co się stanie, jeśli samochód się zepsuje, jeśli prom samochodowy zostanie odwołany w określonym dniu, jeśli hotel jest w pełni zarezerwowany, mimo że dokonano rezerwacji? Szczegółowe plany dotyczące ludzi lub robotów są często wyrażane w języku probabilistycznym.

4. Uczenie się. Trzy poprzednie wspomniane obszary można również postrzegać jako dziedziny automatycznego uczenia się. Ważnym elementem wielu systemów stochastycznych jest to, że mają one możliwość próbkowania sytuacji i uczenia się w czasie. Niektóre zaawansowane systemy są w stanie zarówno próbkować dane i przewidywać wyniki, jak i uczyć się nowych relacji probabilistycznych na podstawie danych i wyników.

Metodologia stochastyczna opiera się na właściwościach liczenia. Prawdopodobieństwo zdarzenia w sytuacji opisuje się jako stosunek liczby sposobów wystąpienia zdarzenia do całkowitej liczby możliwych wyników tego zdarzenia. Zatem prawdopodobieństwo, że liczba parzysta wynika z rzutu rzetelną kością, jest całkowitą liczbą parzystych wyników (tutaj 2, 4 lub 6) w stosunku do całkowitej liczby wyników (1, 2, 3, 4, 5 lub 6) lub 1/2. Ponownie, prawdopodobieństwo wyciągnięcia marmuru określonego koloru z torby marmuru jest stosunkiem liczby kulek tego koloru do całkowitej liczby kulek w torbie. W sekcji 1 wprowadzamy podstawowe techniki liczenia, w tym zasady sumowania i produktu. Ze względu na ich znaczenie w liczeniu, prezentujemy również permutacje i kombinacje dyskretnych zdarzeń. Jest to opcjonalna sekcja, którą czytelnicy mogą pominąć z wystarczającym tłem w dyskretnej matematyce. W sekcji 2 wprowadzamy język formalny do rozumowania przy użyciu metody stochastycznej. Obejmuje to definicje niezależności i różne typy zmiennych losowych. Na przykład, w przypadku twierdzeń probabilistycznych, zmienne losowe mogą być logiczne (prawda lub fałsz), dyskretne, liczby całkowite od 1 do 6 jak w rzucie rzetelną lub ciągłą, funkcją zdefiniowaną na liczbach rzeczywistych. W części dalszej przedstawiamy twierdzenie Bayesa, które popiera większość podejść do modelowania stochastycznego i uczenia się. Zasada Bayesa jest ważna dla interpretacji nowych dowodów w kontekście wcześniejszej wiedzy lub doświadczenia. Później przedstawiamy dwa

zastosowania metod stochastycznych, w tym probabilistyczne automaty skończone i metodologię przewidywania wzorców słów angielskich na podstawie próbkowanych danych.

5.1 Elementy liczenia (opcjonalnie)

Podstawą metodologii stochastycznej jest umiejętność zliczania elementów domeny aplikacji. Podstawą gromadzenia i liczenia elementów jest oczywiście teoria mnogości, w której musimy być w stanie jednoznacznie ustalić, czy element jest, czy nie jest członkiem zbioru elementów. Po ustaleniu tego, istnieją metodologie zliczania elementów zbiorów, dopełnienia zbioru oraz połączenia i zamiany wielu zbiorów. Przeglądniemy te techniki

Zasady dodawania i mnożenia

Jeśli mamy zestaw A, liczba elementów w zestawie A jest oznaczona przez | A |, zwaną licznością A. Oczywiście, A może być puste (liczba elementów wynosi zero), skończone, w nieskończoność lub nieskończenie nieskończone. Każdy zestaw jest zdefiniowany w kategoriach dziedziny zainteresowań lub wszechświata U elementów, które mogą być w tym zestawie. Na przykład zbiór mężczyzn w klasie może być zdefiniowany w kontekście lub we wszechświecie wszystkich ludzi w tym pokoju. Podobnie rzut 3 na uczciwej kości może być postrzegany jako jeden z zestaw sześciu możliwych wyników. Domena lub wszechświat zbioru A jest zatem również zbiorem i służy do określenia dopełnienia tego zbioru, A. Na przykład dopełnieniem zbioru wszystkich mężczyzn w klasie właśnie wspomnianej jest zbiór wszystkich kobiet, i uzupełnieniem rzutu {3} rzetelnej kości jest {1, 2, 4, 5, 6}. Jeden zestaw A jest podzbiorem drugiego zestawu B, ab.png , jeśli każdy element zestawu A jest również elementem zestawu B. Zatem, trywialnie, każdy zestaw jest podzbiorem samego siebie, każdy zestaw A jest podzbiorem jego wszechświata, a pusty zestaw, oznaczony {} lub ϕ, jest podzbiorem każdego zestawu. Połączenie dwóch zbiorów A i B, A ∪ B, można opisać jako zbiór wszystkich elementów w każdym zestawie. Liczba elementów w połączeniu dwóch zbiorów jest sumą wszystkich elementów w każdym zestawie minus liczba elementów znajdujących się w obu zestawach. Uzasadnieniem tego jest oczywiście fakt, że każdy odrębny element w zestawie można policzyć tylko raz. Trywialnie, jeśli dwa zbiory nie mają wspólnych elementów, liczba elementów w ich połączeniu jest sumą liczby elementów w każdym zestawie. Przecięcie dwóch zbiorów A i B, A ∩ B, jest zbiorem wszystkich elementów wspólnych dla obu zbiorów. Podajemy teraz przykłady szeregu właśnie zdefiniowanych pojęć.

Załóżmy, że wszechświat, U, jest zbiorem {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Niech A będzie zbiorem {1, 3, 5, 7, 9}

Niech B będzie zbiorem {0, 2, 4, 6, 8}

Niech C będzie zbiorem {4, 5, 6}

Następnie | A | wynosi 5, | B | wynosi 5, | C | wynosi 3, a | U | wynosi 10.

Ponadto, abu.png | A| ∪ B = U, | B | = | A |, $A = B^{-}$

Ponadto, | A | ∪ B | = | A | + | B | = 10, ponieważ A ∩ B = {}, ale

| A | ∪ C | = | A | + | C | - | A ∩ C | = 7, ponieważ A ∩ C = {5}

Właśnie przedstawiliśmy główne składniki reguły dodawania do łączenia dwóch zestawów. Dla dowolnych dwóch zbiorów A i C liczba elementów w połączeniu tych zbiorów wynosi:

| A ∪ C | = | A | + | C | - | A ∩ C |

Zauważ, że ta reguła dodawania utrzymuje, czy dwa zestawy są rozłączne, czy mają wspólne elementy. Podobna reguła dodawania obowiązuje dla trzech zestawów A, B i C, ponownie, niezależnie od tego, czy mają one wspólne elementy:

$$| A \cup B \cup C | = | A | + | B | + | C | - | A \cap B | - | A \cap C | - | B \cap C | + | A \cap B \cap C |$$

Argument podobny do przedstawionego wcześniej można wykorzystać do uzasadnienia tego równania. Podobne równania włączenia / wyłączenia są dostępne i łatwe do wykazania w przypadku dodawania zestawów elementów z więcej niż trzech zbiorów. Zasada mnożenia do zliczania mówi, że jeśli mamy dwa zestawy elementów A i B odpowiednio o rozmiarach a i b, to istnieje x x unikalne sposoby łączenia elementów tych zbiorów razem. Uzasadnieniem tego jest oczywiście to, że dla każdego z elementów A istnieją pary b dla tego elementu. Zasada mnożenia obsługuje wiele technik stosowanych w zliczaniu, w tym iloczyn kartezjański zbiorów, a także permutacje i kombinacje zbiorów. Iloczyn kartezjański dwóch zbiorów A i B, oznaczony jako A X B, jest zbiorem wszystkich uporządkowanych par (a, b), gdzie a jest elementem zbioru A, a b jest elementem zbioru B; lub bardziej formalnie:

$$A \text{ X } B = \{(a, b) \mid (a \in A) \land (b \in B)\}$$

oraz przez zasadę mnożenia liczenia:

$$| A \text{ X } B | = | A | \text{ x } | B |$$

Produkt kartezjański można oczywiście zdefiniować w dowolnej liczbie zestawów. Produktem dla n zestawów będzie zestaw n-krotek, w którym pierwszy składnik n-krotki jest dowolnym elementem pierwszego zestawu, drugim składnikiem n-krotki jest dowolny element drugiego zestawu i tak dalej. Ponownie liczba unikalnych n-krotek, które powstają, jest iloczynem liczby elementów w każdym zestawie.

Permutacje i kombinacje

Permutacja zestawu elementów jest uporządkowaną sekwencją elementów tego zestawu. W takim układzie elementów każdy może być użyty tylko raz. Przykładem permutacji jest uporządkowanie lub zamówienie zestawu dziesięciu książek na półce, które mogą pomieścić wszystkie dziesięć. Innym przykładem jest przydzielenie określonych zadań czterem z grupy sześciorga dzieci.

Często chcemy wiedzieć, ile (unikalnych) permutacji ma zbiór n elementów. Aby to ustalić, używamy mnożenia. Jeśli w zestawie A znajduje się n elementów, to zestaw permutacji tych elementów jest sekwencją długości n, gdzie pierwszy element sekwencji jest dowolnym z n elementów A, drugim elementem sekwencji jest dowolny z (n - 1) pozostałych elementów A, trzeci element sekwencji to dowolny z pozostałych (n - 2) elementów, a więc na. Kolejność umieszczania elementów w sekwencji permutacji jest nieistotna, tj. Dowolny z n elementów zestawu może być umieszczony najpierw w dowolnym miejscu w sekwencji, dowolny z pozostałych pozostałych elementów n - 1 może być umieszczony jako drugi w dowolnym z n - 1 pozostałych lokalizacji sekwencji permutacji i tak dalej. Wreszcie, zgodnie z zasadą mnożenia, istnieje n! sekwencje permutacji dla tego zestawu n elementów. Możemy ograniczyć liczbę elementów w permutacji zestawu A do dowolnej liczby większej lub równej zero i mniejszej lub równej liczbie elementów n w oryginalnym zestawie A. Na przykład, chcielibyśmy wiedzieć ile jest różnych zamówień z dziesięciu możliwych książek na półce, które mogą pomieścić tylko sześć z nich na raz. Gdybyśmy chcieli ustalić liczbę permutacji n elementów A wziętych r jednocześnie, gdzie $0 \leq r \leq n$, używamy mnożenia jak poprzednio, z tym wyjątkiem, że teraz mamy tylko r miejsc w każdej sekwencji permutacji:

$$n \text{ x } (n - 1) \text{ x } (n - 2) \text{ x } (n - 3) \text{ x } ... \text{ x } (n - (r - 1))$$

Alternatywnie możemy przedstawić to równanie jako:

nx (n - 1) x (n - 2) x (n - 3) x ... x (n - (r - 1)) x (n - r) x (n - r -1) x ... x 2 x 1 / (n - r) x (n - r -1) x ... x 2 x 1

lub równoważnie, liczba permutacji n elementów wziętych r w czasie, która jest symbolizowana jako nPr, wynosi:

nPr = n! / (n - r)!

Kombinacja zestawu n elementów jest dowolnym podzbiorem tych elementów, który można utworzyć. Podobnie jak w przypadku permutacji, często chcemy policzyć liczbę kombinacji elementów, które można utworzyć, biorąc pod uwagę zestaw elementów. Zatem istnieje tylko jedna kombinacja n elementów zestawu n elementów. Kluczową ideą jest to, że liczba kombinacji reprezentuje liczbę podzbiorów pełnego zestawu elementów, które można utworzyć. W przykładzie półki na książki kombinacje reprezentują różne podzbiory sześciu książek, książek na półce, które mogą być utworzone z pełnego zestawu dziesięciu książek. Innym przykładem kombinacji jest zadanie tworzenia czteroosobowych komitetów z grupy piętnastu osób. Każda osoba jest w komitecie lub nie, i nie ma znaczenia, czy jest on pierwszym, czy ostatnim wybranym członkiem. Kolejnym przykładem jest pięciokartowa ręka w grze w pokera. Kolejność rozdawania kart nie ma znaczenia dla ostatecznej wartości ręki. (Może to mieć ogromną różnicę w wartości zakładów, jeśli ostatnie cztery karty zostaną odkryte, jak w tradycyjnym pokerze stud, ale ostateczna wartość układu jest niezależna od rozłożonego zamówienia). Liczba kombinacji n elementów wziętych r jednocześnie, gdzie $0 \leq r \leq n$, jest symbolizowane przez nCr. Prostą metodą określania liczby tych kombinacji jest wzięcie liczby permutacji, nPr, jak już to zrobiliśmy, a następnie podzielenie liczby duplikatów. Ponieważ jakikolwiek podzbiór elementu r n elementów ma r! permutacje, aby uzyskać liczbę kombinacji n elementów wziętych r na raz, dzielimy liczbę permutacji n elementów wziętych r na raz przez r !. Mamy zatem:

nCr = nPr = n! /! (n - r)! r!

Istnieje wiele innych wariantów właśnie przedstawionych zasad liczenia, niektóre z nich zostaną przedstawione w ćwiczeniach z rozdziału 5. W celu dalszego rozwoju tych technik liczenia zalecamy dowolny dyskretny podręcznik matematyki

Elementy teorii prawdopodobieństwa

Opierając się na zasadach liczenia przedstawionych wcześniej, możemy teraz wprowadzić teorię prawdopodobieństwa. Po pierwsze, w sekcji 1 rozważamy kilka podstawowych definicji, takich jak pojęcie, czy dwa lub więcej zdarzeń jest od siebie niezależnych. W sekcji 2 pokazujemy, jak wnioskować o wyjaśnieniach dla poszczególnych zestawów danych. To pozwoli nam rozważyć kilka przykładów wnioskowania probabilistycznego w sekcji 3 i twierdzenia Bayesa w sekcji 4.

Przykładowa przestrzeń, prawdopodobieństwa i niezależność

Następujące definicje, będące podstawą teorii prawdopodobieństwa, zostały po raz pierwszy sformalizowane przez francuskiego matematyka Laplace'a (1816) na początku XIX wieku. Jak wspomniano we wstępie do rozdziału 5, Laplace był w trakcie tworzenia rachunku różniczkowego do hazardu!

DEFINICJA

WYDARZENIE ELEMENTARNE

Wydarzenie elementarne lub atomowe to wydarzenie lub wydarzenie, które nie może składać się z innych zdarzeń.

WYDARZENIE, E

Wydarzenie to zestaw elementarnych zdarzeń.

PRZESTRZEN PRÓBNA, S

Zbiór wszystkich możliwych wyników zdarzenia E to przykładowa przestrzeń S lub wszechświat dla tego zdarzenia.

PRAWDOPODOBIEŃSTWO,

Prawdopodobieństwo zdarzenia E w przestrzeni próbnej S jest stosunkiem liczby elementów w E do całkowitej liczby możliwych wyników w przestrzeni próbnej S z E.

p (E) = | E | / | S |.

Na przykład, jakie jest prawdopodobieństwo, że 7 lub 11 są wynikiem rzutu dwoma uczciwymi kośćmi? Najpierw określamy przestrzeń próbki dla tej sytuacji. Stosując zasadę mnożenia liczenia, każda kość ma 6 wyników, więc całkowity zestaw wyników dwóch kości wynosi 36. Liczba kombinacji dwóch kości, które mogą dać 7, wynosi 1,6; 2,5; 3,4; 4,3; 5,2; oraz 6,1–6 łącznie. Prawdopodobieństwo wyrzucenia 7 wynosi zatem 6/36 = 1/6. Liczba kombinacji dwóch kości, które mogą dać 11, wynosi 5,6; 6,5 - lub 2, a prawdopodobieństwo wyrzucenia 11 wynosi 2/36 = 1/18. Wykorzystując właściwość addytywną różnych wyników, istnieje prawdopodobieństwo 1/6 + 1/18 lub 2/9 wyrzucenia 7 lub 11 z dwoma uczciwymi kośćmi. W tym przykładzie z 7/11 dwa zdarzenia otrzymują 7, a 11. 11. Elementarnymi zdarzeniami są odmienne wyniki rzutu dwiema kostkami. Zatem zdarzenie 7 składa się z sześciu zdarzeń atomowych (1,6), (2,5), (3,4), (4,3), (5,2) i (6,1) . Pełna przestrzeń próbki to połączenie wszystkich trzydziestu sześciu możliwych zdarzeń atomowych, zestawu wszystkich par, które wynikają z rzutu kostką. Jak wkrótce zobaczymy, ponieważ zdarzenia uzyskania 7 i otrzymania 11 nie mają wspólnych zdarzeń atomowych, są one niezależne, a prawdopodobieństwo ich sumy (unii) jest tylko sumą ich indywidualnych prawdopodobieństw. W drugim przykładzie, ile czterech rodzajów kart można rozdać we wszystkich możliwych układach pokera na kartę fivecard? Po pierwsze, zestaw wydarzeń atomowych, które składają się na pełną przestrzeń wszystkich rąk pokera na kartę fivecard, to kombinacja 52 kart pobranych po 5 na raz. Aby uzyskać łączną liczbę czterech kart tego samego rodzaju, stosujemy zasadę mnożenia. Mnożymy liczbę kombinacji 13 kart branych po 1 na raz (liczbę różnych rodzajów kart: as, 2, 3 ..., król) razy liczbę sposobów wybrania wszystkich czterech kart tego samego rodzaju (kombinacja 4 kart pobranych 4 na raz) razy liczba możliwych innych kart, które wypełniają układ 5 kart (pozostało 48 kart). Zatem prawdopodobieństwo czterokartowego układu pokera wynosi:

$({}_{13}C_1 \times {}_4C_4 \times {}_{48}C_1) / {}_{52}C_5 = 13 \times 1 \times 48/2\,598{,}960 \approx 0{,}00024$

Kilka wyników wynika bezpośrednio z właśnie wykonanych definicji. Po pierwsze, prawdopodobieństwo dowolnego zdarzenia E z przestrzeni próbki S wynosi:

$$0 \le p(E) \le 1, \text{ where } E \subseteq S$$

Drugim wynikiem jest to, że suma prawdopodobieństw wszystkich możliwych wyników w S wynosi 1. Aby to zobaczyć, należy zauważyć, że definicja przestrzeni próby S wskazuje, że składa się ona z połączenia wszystkich pojedynczych zdarzeń E w problemie. Jako trzeci wynik definicji należy zauważyć, że prawdopodobieństwo dopełnienia zdarzenia wynosi:

$$p(\overline{E}) = (|S| - |E|) \ / \ |S| \ = \ (|S| \ / \ |S|) - \ (|E| \ / \ |S|) = 1 - \ p(E)$$

Uzupełnieniem wydarzenia jest ważny związek. Wiele razy łatwiej jest ustalić prawdopodobieństwo wystąpienia zdarzenia w zależności od jego nieistnienia, na przykład określenie prawdopodobieństwa, że co najmniej jeden element losowo generowanego ciągu bitów o długości n wynosi 1. Dopełnieniem jest to, że wszystkie bity w ciągu są równe 0, z prawdopodobieństwem 2-n, przy n długości łańcucha. Zatem prawdopodobieństwo pierwotnego zdarzenia wynosi 1 - 2-n. Wreszcie, na podstawie prawdopodobieństwa uzupełnienia zestawu zdarzeń obliczamy prawdopodobieństwo, gdy nie wystąpi żadne zdarzenie, czasami określane jako sprzeczne lub fałszywe zdanie:

$$p(\{\}) = 1 - p(\overline{\{\}}) = 1 - p(S) = 1 - 1 = 0, \text{ or alternatively,}$$
$$= |\{\}| / |S| = 0 / |S| = 0$$

Ostateczną istotną zależność, prawdopodobieństwo połączenia dwóch zbiorów zdarzeń, można ustalić na podstawie zasady liczenia przedstawionej wcześniej, mianowicie dla dowolnych dwóch zbiorów A i B: $| A \cup B | = | A | + | B | - | A \cap B |$. Z tej zależności możemy określić prawdopodobieństwo zjednoczenia dowolnych dwóch zbiorów pobranych z przestrzeni próbnej S:

$p(A \cup B) = |A \cup B| / |S| = (|A| + |B| - |A \cap B|) / |S| = |A| / |S| + |B| / |S| - |A \cap B| / |S| = p(A) + p(B) - p(A \cap B)$

Oczywiście wynik ten można rozszerzyć na sumę dowolnej liczby zbiorów, zgodnie z przedstawioną wcześniej zasadą włączenia / wyłączenia. Już przedstawiliśmy przykład określania prawdopodobieństwa połączenia dwóch zestawów: Prawdopodobieństwo wyrzucenia 7 lub 11 za pomocą dwóch uczciwych kości. W tym przykładzie właśnie zaprezentowana formuła została zastosowana z prawdopodobieństwem par kostek, które dały 7 rozłączenia z par kostek, które dały 11. Możemy również użyć tej formuły w bardziej ogólnym przypadku, gdy zestawy nie są rozłączne . Załóżmy, że chcieliśmy ustalić, rzucając dwiema rzetelnymi kośćmi, prawdopodobieństwo rzutu 8 lub parą tej samej liczby. Po prostu obliczalibyśmy prawdopodobieństwo tego związku, gdy istnieje jedno zdarzenie elementarne - (4,4) - czyli przecięcie obu pożądanych zdarzeń końcowych.

Następnie rozważamy prawdopodobieństwo wystąpienia dwóch niezależnych zdarzeń. Załóżmy, że jesteś graczem w czteroosobową grę karcianą, w której wszystkie karty są równo rozłożone. Jeśli nie masz królowej pik, możesz dojść do wniosku, że każdy z pozostałych graczy ma ją z prawdopodobieństwem 1/3. Podobnie można stwierdzić, że każdy gracz ma asa kier z prawdopodobieństwem 1/3 i że każdy gracz ma obie karty z prawdopodobieństwem 1/3 × 1/3 lub 1/9. W tej sytuacji założyliśmy, że zdarzenia związane z uzyskaniem tych dwóch kart są niezależne, chociaż jest to tylko w przybliżeniu prawda. Formalizujemy tę intuicję za pomocą definicji.

DEFINICJA

NIEZALEŻNE WYDARZENIA

Dwa zdarzenia A i B są niezależne wtedy i tylko wtedy, gdy prawdopodobieństwo ich wystąpienia jest równe iloczynowi ich wystąpienia indywidualnie. Ta zależność niezależności jest wyrażona:

$p (A \cap B) = p (A) * p (B)$

Czasami używamy równoważnego zapisu p (s, d) dla p (s ∩ d). Wyjaśniamy pojęcie niezależności w kontekście prawdopodobieństw warunkowych. Ponieważ opis przedstawionej właśnie niezależności zdarzeń jest relacją tylko wtedy i tylko wtedy, możemy ustalić, czy dwa zdarzenia są niezależne, opracowując ich relacje probabilistyczne. Rozważ sytuację, w której losowo generowane są ciągi bitów o długości cztery. Chcemy wiedzieć, czy zdarzenie ciągu bitów zawierającego parzystą liczbę 1s jest niezależne od zdarzenia, w którym ciąg bitów kończy się na 0. Przy zastosowaniu zasady mnożenia,

każdy bit ma 2 wartości, w sumie 24 = 16 ciągi bitów z długość 4. Istnieje 8 ciągów bitów o długości cztery, które kończą się cyfrą 0: {1110, 1100, 1010, 1000, 0010, 0100, 0110, 0000}. Istnieje również 8 ciągów bitowych, które mają parzystą liczbę 1: {1111, 1100, 1010, 1001, 0110, 0101, 0011, 0000}. Liczba ciągów bitów, które mają zarówno parzystą liczbę 1, jak i kończą 0, wynosi 4: {1100, 1010, 0110, 0000}. Od tego czasu te dwa wydarzenia są niezależne

p ({parzysta liczba 1s} ∩ {koniec z 0}) = p ({parzysta liczba 1s}) x p ({koniec z 0}) 4/16 = 8/16 x 8/16 = 1/4

Rozważ ten sam przykład losowo generowanych ciągów bitów o długości cztery. Czy dwa następujące zdarzenia są niezależne: ciągi bitów mają parzystą liczbę 1, a ciągi bitów kończą się na 1? Gdy dwa lub więcej zdarzeń nie jest niezależnych, tzn. Prawdopodobieństwo, że jedno zdarzenie wpłynie na prawdopodobieństwo innych, wymaga pojęcia warunkowego prawdopodobieństwa wypracowania ich relacji. Przed zamknięciem tego rozdziału zauważamy, że możliwe są inne systemy aksjomatów wspierające podstawy teorii prawdopodobieństwa, na przykład jako rozszerzenie rachunku zdań. Jako przykład podejścia opartego na zestawie, rosyjski matematyk Kołmogorow (1950) zaproponował wariant następujących aksjomatów: równoważne z naszymi definicjami. Z tych trzech aksjomatów Kołmogorow systematycznie konstruował wszystkie teorie prawdopodobieństwa.

1. Prawdopodobieństwo zdarzenia E w przestrzeni próbki S wynosi od 0 do 1, tj. 0 ≤ p (E) ≤ 1.

2. Gdy suma wszystkich E = S, p (S) = 1, a p (S^{-}) = 0.

3. Prawdopodobieństwo połączenia dwóch zbiorów zdarzeń A i B wynosi:

p (A ∪ B) = p (A) + p (B) - p (A ∩ B)

Wnioskowanie probabilistyczne: przykład

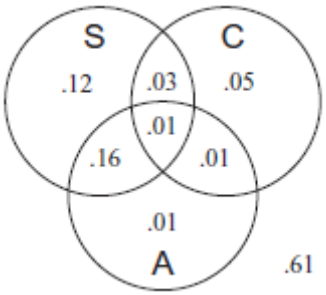
Pokazujemy teraz przykłady uzasadnienia z właśnie przedstawionymi pomysłami. Załóżmy, że jedziesz autostradą międzystanową i zdajesz sobie sprawę, że stopniowo zwalniasz z powodu zwiększonego natężenia ruchu. Zaczynasz szukać możliwych wyjaśnień spowolnienia. Czy to może być budowa dróg? Czy był wypadek? Wszystko, co wiesz, to to, że zwalniasz. Ale poczekaj! Masz dostęp do statystyk stanu autostrad, a dzięki nowemu samochodowemu GUI i systemowi wnioskowania możesz pobrać na komputer swojego samochodu odpowiednie informacje statystyczne. Okej, więc masz dane; co możesz z nimi zrobić? W tym przykładzie zakładamy, że mamy trzy parametry prawda lub fałsz (zdefiniujemy ten parametr jako boolowską zmienną losową w rozdziale 5.2.4). Po pierwsze, czy ruch - a ty - zwalniasz, czy nie. Ta sytuacja będzie oznaczona S, z przypisaniem t lub f. Po drugie, istnieje prawdopodobieństwo, że nastąpi wypadek, A, z przypisaniami t lub f. Wreszcie, prawdopodobieństwo, że w tym czasie istnieje budowa drogi, C; ponownie albo t albo f. Możemy wyrazić te relacje dla ruchu na autostradzie międzystanowej, dzięki naszemu samochodowemu systemowi pobierania danych, w tabeli

S	C	A	p
t	t	t	0.01
t	t	f	0.03
t	f	t	0.16
t	f	f	0.12
f	t	t	0.01
f	t	f	0.05
f	f	t	0.01
f	f	f	0.61

Pozycje w tabeli są oczywiście interpretowane podobnie jak tabele prawdy, z tą różnicą, że w prawej kolumnie podano prawdopodobieństwo wystąpienia sytuacji po lewej stronie. Zatem trzeci rząd tabeli podaje prawdopodobieństwo spowolnienia ruchu i wypadku, ale bez konstrukcji wynoszącej 0,16:

$$S \cap \overline{C} \cap A = 0.16$$

Należy zauważyć, że opracowujemy nasz rachunek probabilistyczny w kontekście zbiorów zdarzeń. Rysunek pokazuje, w jaki sposób prawdopodobieństwa z tabeli można przedstawić za pomocą tradycyjnego diagramu Venna.



Równie dobrze moglibyśmy przedstawić tę sytuację jako probabilistyczne przypisanie prawdy zdań, w którym to przypadku ∩ zostałby zastąpiony przez ∧, a Tabela byłaby interpretowana jako wartości prawdy w połączeniu zdań. Następnie zauważamy, że suma wszystkich możliwych wyników wspólnego rozkładu tabeli wynosi 1,0; jest to, jak można się spodziewać, z aksjomatami prawdopodobieństwa przedstawionymi w rozdziale. Należy zauważyć, że rozwijamy nasz rachunek probabilistyczny w kontekście zbiorów zdarzeń.

Równie dobrze moglibyśmy przedstawić tę sytuację jako probabilistyczne przypisanie prawdy zdań, w którym to przypadku ∩ zostałoby zastąpione przez ∧, a Tabela byłaby zinterpretowana jako wartości prawdy w połączeniu zdań. Następnie zauważamy, że suma wszystkich możliwych wyników wspólnego rozkładu tabeli 5.1 wynosi 1,0; jest to zgodne z oczekiwaniami z aksjomatami prawdopodobieństwa przedstawionymi. Możemy również obliczyć prawdopodobieństwo dowolnego prostego lub złożonego zestawu zdarzeń. Na przykład możemy obliczyć prawdopodobieństwo spowolnienia ruchu S. Wartość dla wolnego ruchu wynosi 0,32, sumę pierwszych czterech wierszy tabeli 5.1; to znaczy wszystkie sytuacje, w których S = t. Czasami nazywa się to bezwarunkowym lub krańcowym prawdopodobieństwem powolnego ruchu, S. Proces ten nazywa się marginalizacją, ponieważ wszystkie prawdopodobieństwa inne niż powolny ruch są sumowane. Oznacza to, że rozkład zmiennej

można uzyskać, sumując wszystkie pozostałe zmienne ze wspólnego rozkładu zawierającego tę zmienną

W podobny sposób możemy obliczyć prawdopodobieństwo budowy C bez spowolnienia es.png zjawisko niezbyt częste w stanie Nowy Meksyk! Sytuację tę ujmuje p (C ∩ S) = t, jako suma piątej i szóstej linii tabeli 5.1 lub 0,06. Jeśli weźmiemy pod uwagę negację sytuacji C ∩ S, otrzymalibyśmy (stosując prawa deMorgan) p (C ∪ S). Obliczając prawdopodobieństwo zjednoczenia dwóch zbiorów, otrzymujemy:

0,16 + 0,12 + 0,01 + 0,61 + 0,01 + 0,03 + 0,16 + 0,12 - (0,16 + 0,12) = 0,94

I znowu całkowite prawdopodobieństwo C ∩ S i jego dopełniacza (negacji) wynosi 1,0.

Zmienne losowe

W teorii prawdopodobieństwa poszczególne prawdopodobieństwa są obliczane analitycznie za pomocą metod kombinatorycznych lub empirycznie przez próbkowanie populacji zdarzeń. Do tego momentu większość naszych prawdopodobieństw została ustalona analitycznie. Na przykład kostka ma sześć boków i dwie monety. Kiedy „kość" lub „moneta" jest „sprawiedliwa", mówimy, że każdy wynik, z rzutu lub rzutu, jest równie prawdopodobny. W rezultacie łatwo jest określić przestrzeń zdarzeń dla tych problematycznych sytuacji, które później nazywamy parametrycznymi. Bardziej interesujące rozumowanie probabilistyczne wynika jednak z próbkowania analizy sytuacji w rzeczywistym świecie wydarzeń. W tych sytuacjach często brakuje dobrze zdefiniowanej specyfikacji, która wspiera analityczne obliczanie prawdopodobieństw. Jest tak również, że niektóre sytuacje, nawet jeśli istnieją podstawy analityczne, są tak złożone, że czas i koszty obliczeń nie są wystarczające do deterministycznego obliczenia wyników probabilistycznych. W takich sytuacjach zwykle przyjmujemy empiryczną metodologię próbkowania. Co najważniejsze, zakładamy, że wszystkie wyniki eksperymentu nie są jednakowo prawdopodobne. Zachowujemy jednak podstawowe aksjomaty lub założenia, które przyjęliśmy w poprzednich sekcjach; mianowicie, że prawdopodobieństwo zdarzenia jest liczbą pomiędzy (i włącznie) 0 i 1 oraz, że zsumowane prawdopodobieństwa wszystkich wyników wynoszą 1. Zachowujemy również naszą zasadę dotyczącą prawdopodobieństwa zjednoczonych zbiorów zdarzeń. Definiujemy zmienną losową jako metodę uściślenia tego rachunku.

DEFINICJA

ZMIENNA LOSOWA

Zmienna losowa jest funkcją, której domeną jest przestrzeń próbki i zawiera ona zestaw wyników, najczęściej liczb rzeczywistych. Zamiast korzystać z przestrzeni zdarzeń specyficznych dla problemu, zmienna losowa pozwala mówić o prawdopodobieństwach jako wartościach liczbowych związanych z przestrzenią zdarzeń.

BOOLEAN, DISCRETE i CIĄGŁE ZMIENNE LOSOWE

Logiczna zmienna losowa jest funkcją od przestrzeni zdarzeń do {true, false} lub do podzbioru liczb rzeczywistych {0,0, 1,0}. Zmienna losowa typu boolean jest czasem nazywana próbą Bernoulliego. Dyskretna zmienna losowa, która zawiera boolowskie zmienne losowe jako podzbiór, to funkcja z przestrzeni próbnej na (policzalny podzbiór) liczb rzeczywistych w [0,0, 1,0]. Ciągła zmienna losowa ma w swoim zakresie zbiór liczb rzeczywistych. Przykład z wykorzystaniem dyskretnej zmiennej losowej w domenie Season, w której zdarzeniami atomowymi w sezonie są {wiosna, lato, jesień, zima}, przypisuje 0,75, powiedzmy, elementowi domeny Season = wiosna. W tej sytuacji mówimy p (Sezon = wiosna) = 0,75.

Przykładem losowej zmiennej typu boolean w tej samej domenie byłoby mapowanie p (Season = spring) = true. Większość przykładów probabilistycznych, które rozważamy, będzie dotyczyć dyskretnych zmiennych losowych. Innym przykładem zastosowania losowej zmiennej typu boolean byłoby obliczenie prawdopodobieństwa uzyskania 5 głów w 7 rzutach uczciwej monety. Byłaby to kombinacja 5 z 7 przewrotów będących główkami razy prawdopodobieństwo 1/2 główki do 5. potęgi razy prawdopodobieństwo 1/2 trafienia głów do 2. potęgi lub:

$_7C_5 \times (1/2)^5 \times (1/2)^2$

Ta sytuacja z rzutem monetą jest przykładem tak zwanego rozkładu dwumianowego. W rzeczywistości wynik każdej sytuacji, w której chcemy mierzyć sukcesy rw próbach n, gdzie p jest znanym prawdopodobieństwem sukcesu, można przedstawić jako:

$_nC_r \times p^r \times (1 - p)^{(n - r)}$

Ważnym naturalnym rozszerzeniem powiązania miar probabilistycznych ze zdarzeniami jest pojęcie oczekiwanego kosztu lub wypłaty za ten wynik. Na przykład możemy obliczyć przyszły zwrot z obstawiania określonych wartości pieniężnych z losowania karty lub zakręcenia kołem ruletki. Definiujemy oczekiwanie zmiennej losowej lub zdarzenia, np. (E):

DEFINICJA

OCZEKIWANIE ZDARZENIA

Jeżeli nagrodą za wystąpienie zdarzenia E, z prawdopodobieństwem p (E), jest r, a koszt nieistnienia zdarzenia, 1 - p (E), wynosi c, to oczekiwanie wynikające ze zdarzenia, np. (E), jest:

ex (E) = r x p (E) + c x (1 - p (E))

Załóżmy na przykład, że uczciwe koło ruletki ma liczby całkowite od 0 do 36 równomiernie rozmieszczone na rowkach koła. W grze każdy gracz kładzie 1 USD na dowolną wybraną przez siebie liczbę: jeśli koło zatrzyma się na wybranej liczbie, wygrywa 35 USD; w przeciwnym razie traci dolara. Nagroda za zwycięstwo wynosi 35 $; koszt straty 1 USD. Ponieważ prawdopodobieństwo wygranej wynosi 1/37, przegranej 36/37, oczekiwana wartość tego zdarzenia, np. (E), wynosi:

ex (E) = 35 (1/37) + (-1) (36/37) ≈ -0,027

W ten sposób gracz traci średnio około 0,03 USD na grę!

Tą część kończymy krótką dyskusją i podsumowaniem początków wartości prawdopodobieństw stosowanych w rozumowaniu stochastycznym. Jak wspomniano powyżej, w większości naszych dotychczasowych przykładów rozważaliśmy wartości probabilistyczne, które można ustalić na podstawie znanych sytuacji, takich jak przewrócenie uczciwej monety lub obrót uczciwego koła ruletki. Kiedy mamy takie sytuacje, możemy wyciągnąć wiele wniosków na temat aspektów przestrzeni prawdopodobieństwa, takich jak jej średnia, statystycznie miara prawdopodobieństwa i jak daleko od tej średniej wartości próbkowane zwykle się różnią, odchylenie standardowe wyników w tym domena. Tę dobrze rozumianą sytuację nazywamy parametrycznym podejściem do generowania przykładowej przestrzeni wyników. Podejście parametryczne jest uzasadnione w sytuacjach stochastycznych, w których istnieją a priori oczekiwania dotyczące struktury wyników prób eksperymentalnych. Naszym zadaniem jest „uzupełnienie” parametrów tej dobrze rozumianej sytuacji. Przykładem jest rzucie uczciwą monetą z wynikiem jako rozkład dwumianowy. Następnie możemy zbudować nasze oczekiwania dotyczące sytuacji na podstawie modelu dwumianowego dla możliwych wyników. Istnieje wiele zalet podejść parametrycznych. Po pierwsze, do skalibrowania oczekiwanych wyników potrzeba mniej punktów danych, ponieważ kształt krzywej wyników jest znany z góry. Kolejną zaletą jest to, że

często możliwe jest ustalenie z góry liczby wyników lub ilości danych szkoleniowych wystarczających do oszacowania prawdopodobieństwa jakości. W rzeczywistości w sytuacji parametrycznej, oprócz obliczenia średniej i odchylenia standardowego oczekiwań, możemy dokładnie określić, kiedy pewne punkty danych wykraczają poza normalne oczekiwania. Oczywiście wiele, jeśli nie większość, interesujących sytuacji nie przynosi wyraźnych oczekiwanych rezultatów. Jednym z przykładów jest na przykład rozumowanie diagnostyczne w medycynie. Drugim przykładem jest użycie i interpretacja wyrażenia w języku naturalnym. W przypadku języka dość często przyjmuje się nieparametryczne podejście do oczekiwań, próbkując dużą liczbę sytuacji, jak na przykład w korpusie językowym. Analizując zebrane przykłady użycia języka w gazetach, powiedzmy lub w rozmowach z działem pomocy technicznej dla wsparcia komputerowego, można wywnioskować znaczenie niejednoznacznych wyrażeń w tych domenach. Metodologię tę analizujemy, analizując możliwe relacje fonemów w rozdziale 5.3. Przy wystarczającej liczbie punktów danych wynikowy dyskretny rozkład w środowiskach nieparametrycznych można często wygładzić przez interpolację w celu zachowania ciągłości. Następnie można wywnioskować nowe sytuacje w kontekście utworzonej dystrybucji. Główną wadą metod nieparametrycznych jest to, że przy braku ograniczeń wynikających z wcześniejszych oczekiwań, często wymagana jest duża ilość danych szkoleniowych w celu kompensacji.

Warunkowe prawdopodobieństwo

Miary prawdopodobieństwa omówione do tego momentu w rozdziale 5 są często nazywane wcześniejszymi prawdopodobieństwami, ponieważ są one opracowywane przed uzyskaniem jakichkolwiek nowych informacji o oczekiwanych skutkach zdarzeń w konkretnej sytuacji. W niniejszej sekcji rozważamy warunkowe prawdopodobieństwo wystąpienia zdarzenia, to znaczy prawdopodobieństwo zdarzenia, biorąc pod uwagę pewne nowe informacje lub ograniczenie tego zdarzenia. Jak widać wcześniej w tym rozdziale, wcześniejsze prawdopodobieństwo uzyskania 2 lub 3 w rzucie rzetelnej kości jest sumą tych dwóch indywidualnych wyników podzieloną przez całkowitą liczbę możliwych wyników rzutu rzetelną kością lub 2 / 6. Wcześniejsze prawdopodobieństwo wystąpienia choroby to liczba osób z chorobą podzielona przez liczbę osób w danej dziedzinie. Przykładem warunkowego lub późniejszego prawdopodobieństwa jest sytuacja, gdy pacjent wchodzi do gabinetu lekarskiego z, na przykład, zestawem objawów, bólów głowy i nudności. Doświadczony lekarz będzie znał zestaw wcześniejszych oczekiwań dotyczących różnych chorób w oparciu o objawy, ale będzie chciał ustalić konkretną diagnozę dla tego pacjenta, który obecnie cierpi na bóle głowy i nudności. Aby doprecyzować te pomysły, tworzymy dwie ważne definicje.

DEFINICJA

PRZED PRAWDOPODOBIEŃSTWO

Wcześniejsze prawdopodobieństwo, zwykle bezwarunkowe prawdopodobieństwo zdarzenia, jest prawdopodobieństwem przypisywanym na podstawie całej wiedzy potwierdzającej jego wystąpienie lub nieobecność, to znaczy prawdopodobieństwo zdarzenia przed jakimkolwiek nowym dowodem. Wcześniejsze prawdopodobieństwo zdarzenia jest symbolizowane: p (zdarzenie).

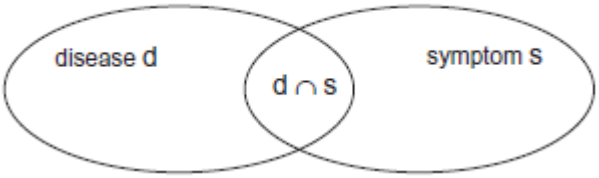
PRAWDOPODOBIEŃSTWO

Prawdopodobieństwo późniejsze (po fakcie), na ogół prawdopodobieństwo warunkowe zdarzenia, to prawdopodobieństwo zdarzenia, biorąc pod uwagę pewne nowe dowody. Późniejsze prawdopodobieństwo zdarzenia, biorąc pod uwagę niektóre dowody, jest symbolizowane: p (zdarzenie | dowód). Następnie rozpoczynamy prezentację twierdzenia Bayesa, którego ogólna postać znajduje się dalej. Idea popierająca Bayesa polega na tym, że prawdopodobieństwo nowej (tylnej) sytuacji hipotezy przy danym dowodzie może być postrzegane jako funkcja znanych prawdopodobieństw dla dowodów przy tej hipotezie. Można powiedzieć, że chcemy wyznaczyć

funkcję f, tak aby p (h | e) = f (p (e | h)). Zwykle chcemy ustalić wartość po lewej stronie tego równania biorąc pod uwagę, że często łatwiej jest obliczyć wartości po prawej stronie. Udowadniamy teraz twierdzenie Bayesa o jednym objawie i jednej chorobie. Opierając się na poprzednich definicjach, prawdopodobieństwo tylne osoby z chorobą d, z zestawu chorób D, z objawem lub dowodem, s, z zestawu objawów S, wynosi:

p (d s) = d ∩ s ⁄ s

Jak w sekcji 5.1, „|" otaczające zbiór to liczność lub liczba elementów w tym zbiorze. Prawa strona tego równania to liczba osób mających zarówno (przecięcie) chorobę d, jak i objawy podzielona przez całkowitą liczbę osób mających objawy. Rycina 5.2 przedstawia schemat Venna tej sytuacji. Rozwijamy prawą stronę



tego równania. Ponieważ przestrzeń próbki do wyznaczania prawdopodobieństwa licznika i mianownika jest taka sama, otrzymujemy:

p (d | s) = p (d ∩ s) / p (s).

Istnieje równoważny związek dla p (s | d); ponownie, patrz rysunek 5.2:

p (s | d) = p (s ∩ d) / p (d).

Następnie rozwiązujemy równanie p (s | d), aby określić wartość p (s ∩ d):

p (s ∩ d) = p (s | d) p (d).

Podstawiając ten wynik w poprzednim równaniu na p (d | s) tworzy regułę Bayesa dla jednej choroby i jednego objawu:

p (d s) = p (s d) p (d) / p (s)

Tak więc prawdopodobieństwo choroby z danym objawem w późniejszym okresie jest iloczynem prawdopodobieństwa objawu przy danej chorobie i prawdopodobieństwa choroby, znormalizowanym przez prawdopodobieństwo tego objawu. Następnie przedstawiamy regułę łańcucha, ważną technikę stosowaną w wielu domenach rozumowanie stochastyczne, szczególnie w przetwarzaniu języka naturalnego. Właśnie opracowaliśmy równania dla dowolnych dwóch zestawów A1 i A2:

p (A1 ∩ A2) = p (A1 | A2) p (A2) = p (A2 | A1) p (A1).

a teraz uogólnienie na wiele zbiorów Ai, zwane regułą łańcucha:

p (A1 ∩ A2 ∩ ... ∩ An) = p (A1) p (A2 | A1) p (A3 | A1 ∩ A2) .... p (An | ∩ Ai)

Robimy argument indukcyjny, aby udowodnić regułę łańcucha, rozważmy n-ty przypadek:

p (A1 ∩ A2 ∩ ... ∩ An-1 ∩ An) = p ((A1 ∩ A2 ∩ ... ∩ An-1) ∩ An),

Stosujemy regułę przecięcia dwóch zestawów, aby uzyskać:

p ((A1 ∩ A2 ∩ ... ∩ An-1) ∩ An) = p (A1 ∩ A2 ∩ ... ∩ An-1) p (An | A1 ∩ A2 ∩ ... ∩ An-1)

a następnie zmniejsz ponownie, biorąc pod uwagę, że:

p (A1 ∩ A2 ∩ ... An-1) = p ((A1 ∩ A2 ∩ ... ∩ An-2) ∩ An-1)

aż do osiągnięcia p (A1∩ A2), przypadek podstawowy, który już wykazaliśmy. Zamykamy tę sekcję kilkoma definicjami opartymi na zastosowaniu relacji reguły łańcucha. Najpierw redefiniujemy zdarzenia niezależne w kontekście prawdopodobieństw warunkowych, a następnie definiujemy zdarzenia warunkowo niezależne lub pojęcie, w jaki sposób zdarzenia mogą być od siebie niezależne, biorąc pod uwagę jakieś trzecie zdarzenie. Warunkowe prawdopodobieństwo. Miary prawdopodobieństwa omówione do tego momentu w rozdziale 5 są często nazywane wcześniejszymi prawdopodobieństwami, ponieważ są one opracowywane przed uzyskaniem jakichkolwiek nowych informacji o oczekiwanych skutkach zdarzeń w konkretnej sytuacji. W niniejszej sekcji rozważamy warunkowe prawdopodobieństwo wystąpienia zdarzenia, to znaczy prawdopodobieństwo zdarzenia, biorąc pod uwagę pewne nowe informacje lub ograniczenie tego zdarzenia. Jak widać wcześniej w tym rozdziale, wcześniejsze prawdopodobieństwo uzyskania 2 lub 3 w rzucie rzetelnej kości jest sumą tych dwóch indywidualnych wyników podzieloną przez całkowitą liczbę możliwych wyników rzutu rzetelną kością lub 2 / 6. Wcześniejsze prawdopodobieństwo wystąpienia choroby to liczba osób z chorobą podzielona przez liczbę osób w danej dziedzinie. Przykładem warunkowego lub późniejszego prawdopodobieństwa jest sytuacja, gdy pacjent wchodzi do gabinetu lekarskiego z, na przykład, zestawem objawów, bólów głowy i nudności. Doświadczony lekarz będzie znał zestaw wcześniejszych oczekiwań dotyczących różnych chorób w oparciu o objawy, ale będzie chciał ustalić konkretną diagnozę dla tego pacjenta, który obecnie cierpi na bóle głowy i nudności. Aby doprecyzować te pomysły, tworzymy dwie ważne definicje.

DEFINICJA

PRZED PRAWDOPODOBIEŃSTWO

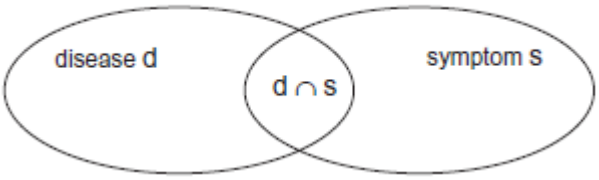
Wcześniejsze prawdopodobieństwo, zwykle bezwarunkowe prawdopodobieństwo zdarzenia, jest prawdopodobieństwem przypisywanym na podstawie całej wiedzy potwierdzającej jego wystąpienie lub nieobecność, to znaczy prawdopodobieństwo zdarzenia przed jakimkolwiek nowym dowodem. Wcześniejsze prawdopodobieństwo zdarzenia jest symbolizowane: p (zdarzenie).

PRAWDOPODOBIEŃSTWO

Prawdopodobieństwo późniejsze (po fakcie), na ogół prawdopodobieństwo warunkowe zdarzenia, to prawdopodobieństwo zdarzenia, biorąc pod uwagę pewne nowe dowody. Późniejsze prawdopodobieństwo zdarzenia, biorąc pod uwagę niektóre dowody, jest symbolizowane: p (zdarzenie | dowód). Następnie rozpoczynamy prezentację twierdzenia Bayesa, którego ogólna postać znajduje się w rozdziale 5.4. Idea popierająca Bayesa polega na tym, że prawdopodobieństwo nowej (tylnej) sytuacji hipotezy przy danym dowodzie może być postrzegane jako funkcja znanych prawdopodobieństw dla dowodów przy tej hipotezie. Można powiedzieć, że chcemy wyznaczyć funkcję f, tak aby p (h | e) = f (p (e | h)). Zwykle chcemy ustalić wartość po lewej stronie tego równania biorąc pod uwagę, że często łatwiej jest obliczyć wartości po prawej stronie. Udowadniamy teraz twierdzenie Bayesa o jednym objawie i jednej chorobie. Opierając się na poprzednich definicjach, prawdopodobieństwo tylne osoby z chorobą d, z zestawu chorób D, z objawem lub dowodem, s, z zestawu objawów S, wynosi:

p (d s) = d ∩ s ⁄ s

„|” otaczające zbiór to liczność lub liczba elementów w tym zbiorze. Prawa strona tego równania to liczba osób mających zarówno (przecięcie) chorobę d, jak i objawy podzielona przez całkowitą liczbę osób mających objawy. Rysunej przedstawia schemat Venna tej sytuacji. Rozwijamy prawą stronę



tego równania. Ponieważ przestrzeń próbki do wyznaczania prawdopodobieństwa licznika i mianownika jest taka sama, otrzymujemy:

$p(d \mid s) = p(d \cap s) / p(s)$.

Istnieje równoważny związek dla $p(s \mid d)$; ponownie, patrz rysunek powyżej:

$p(s \mid d) = p(s \cap d) / p(d)$.

Następnie rozwiązujemy równanie $p(s \mid d)$, aby określić wartość $p(s \cap d)$:

$p(s \cap d) = p(s \mid d)\, p(d)$.

Podstawiając ten wynik w poprzednim równaniu na $p(d \mid s)$ tworzy regułę Bayesa dla jednej choroby i jednego objawu:

$p(d\, s) = p(s\, d)\, p(d) / p(s)$

Tak więc prawdopodobieństwo choroby z danym objawem w późniejszym okresie jest iloczynem prawdopodobieństwa objawu przy danej chorobie i prawdopodobieństwa choroby, znormalizowanym przez prawdopodobieństwo tego objawu. Następnie przedstawiamy regułę łańcucha, ważną technikę stosowaną w wielu domenach rozumowanie stochastyczne, szczególnie w przetwarzaniu języka naturalnego. Właśnie opracowaliśmy równania dla dowolnych dwóch zestawów A_1 i A_2:

$p(A_1 \cap A_2) = p(A_1 \mid A_2)\, p(A_2) = p(A_2 \mid A_1)\, p(A_1)$.

a teraz uogólnienie na wiele zbiorów Ai, zwane regułą łańcucha:

$p(A_1 \cap A_2 \cap ... \cap A_n) = p(A_1)\, p(A_2 \mid A_1)\, p(A_3 \mid A_1 \cap A_2) .... p(A_n \mid \cap A_i)$

Robimy argument indukcyjny, aby udowodnić regułę łańcucha, rozważmy n-ty przypadek:

$p(A_1 \cap A_2 \cap ... \cap A_{n-1} \cap A_n) = p((A_1 \cap A_2 \cap ... \cap A_{n-1}) \cap A_n)$,

Stosujemy regułę przecięcia dwóch zestawów, aby uzyskać:

$p((A_1 \cap A_2 \cap ... \cap A_{n-1}) \cap A_n) = p(A_1 \cap A_2 \cap ... \cap A_{n-1})\, p(A_n \mid A_1 \cap A_2 \cap ... \cap A_{n-1})$

a następnie zmniejsz ponownie, biorąc pod uwagę, że:

$p(A_1 \cap A_2 \cap ... A_{n-1}) = p((A_1 \cap A_2 \cap ... \cap A_{n-2}) \cap A_{n-1})$

aż do osiągnięcia $p(A_1 \cap A_2)$, przypadek podstawowy, który już wykazaliśmy. Zamykamy tę sekcję kilkoma definicjami opartymi na zastosowaniu relacji reguły łańcucha. Najpierw redefiniujemy zdarzenia niezależne w kontekście prawdopodobieństw warunkowych, a następnie definiujemy zdarzenia warunkowo niezależne lub pojęcie, w jaki sposób zdarzenia mogą być od siebie niezależne, biorąc pod uwagę jakieś trzecie zdarzenie.

DEFINICJA

NIEZALEŻNE WYDARZENIA

Dwa zdarzenia A i B są od siebie niezależne wtedy i tylko wtedy, gdy $p(A \cap B) = p(A)\, p(B)$. Kiedy $p(B) \neq 0$ jest to takie samo, jak powiedzenie, że $p(A) = p(A \mid B)$. To znaczy, wiedząc, że B jest prawdziwe, nie wpływa na prawdopodobieństwo, że A jest prawdziwe.

ZDARZENIA WARUNKOWO NIEZALEŻNE

Mówi się, że dwa zdarzenia A i B są warunkowo niezależne od siebie, biorąc pod uwagę zdarzenie C wtedy i tylko wtedy, gdy $p((A \cap B) \mid C) = p(A \mid C)\, p(B \mid C)$.

W wyniku uproszczenia ogólnej reguły łańcucha oferowanej przez zdarzenia warunkowo niezależne można zbudować większe systemy stochastyczne przy mniejszych kosztach obliczeniowych; to znaczy, warunkowo niezależne zdarzenia upraszczają wspólne rozkłady. Przykład z naszej sytuacji na wolnym ruchu: załóżmy, że gdy zwalniamy, zauważamy pomarańczowe beczki kontroli ruchu wzdłuż drogi. Oprócz sugerowania, że przyczyną naszego spowolnienia jest teraz bardziej prawdopodobne budownictwo drogowe niż wypadek drogowy, obecność pomarańczowych beczek będzie miała własną miarę probabilistyczną. W rzeczywistości zmienne przedstawiające spowolnienie ruchu i obecność pomarańczowych beczek są warunkowo niezależne, ponieważ oba są spowodowane przez budowę drogi. Mówimy zatem, że zmienna konstrukcja drogi oddziela spowolnienie ruchu od pomarańczowych beczek. Ze względu na statystyczną efektywność uzyskaną dzięki warunkowej niezależności głównym zadaniem w budowie dużych stochastycznych systemów obliczeniowych jest rozbicie złożonego problemu na słabiej powiązane podproblemy. Relacje interakcji podproblemów są następnie kontrolowane przez różne relacje separacji warunkowej. Widzimy ten pomysł sformalizowany wraz z definicją d-separacji w rozdziale 9.3. Następnie, w rozdziale 5.3, przedstawiamy ogólną formę twierdzenia Bayesa i pokazujemy, w jaki sposób w złożonych sytuacjach obliczenia niezbędne do poparcia pełnego wnioskowania bayesowskiego mogą stać się trudne. Wreszcie w sekcji 5.4 przedstawiamy przykłady rozumowania opartego na bayesowskich miarach prawdopodobieństwa

Twierdzenie Bayesa

Wielebny Thomas Bayes był matematykiem i pastorem. Jego słynne twierdzenie zostało opublikowane w 1763 roku, cztery lata po jego śmierci. Jego artykuł zatytułowany „Esej na temat rozwiązania problemu w doktrynie szans" został opublikowany w Philosophical Transactions of Royal Society of London. Twierdzenie Bayesa wiąże przyczynę i skutek w taki sposób, że poprzez zrozumienie efektu możemy poznać prawdopodobieństwo jego przyczyn. W rezultacie twierdzenie Bayesa jest ważne zarówno dla ustalenia przyczyn chorób, takich jak rak, jak i przydatne do określenia wpływu niektórych konkretnych leków na tę chorobę.

Wprowadzenie

Jednym z najważniejszych wyników teorii prawdopodobieństwa jest ogólna postać twierdzenia Bayesa. Równanie Bayesa dla jednej choroby i jednego objawu. Aby pomóc utrzymać nasze relacje diagnostyczne w kontekście, zmieniamy nazwy zmiennych używanych wcześniej do wskazania poszczególnych hipotez, cześć, z zestawu hipotez, H i zestawu dowodów, E. Ponadto, rozważymy teraz zbiór indywidualnych hipotez cześć jako rozłączny i mając związek wszystkich cześć równy H.

$p(h_i \mid E) = (p(E \mid h_i) \times p(h_i)) / p(E)$

To równanie można odczytać: „Prawdopodobieństwo hipotezy przy danym zbiorze dowodów E wynosi… " Po pierwsze, zauważ, że mianownik p(E) po prawej stronie równania jest bardzo czynnikiem

normalizującym dla każdej hipotezy h_i z zestawu H. Często zdarza się, że twierdzenie Bayesa jest używane do ustalenia, która hipoteza z zestawu możliwych hipotez jest najsilniejszy, biorąc pod uwagę konkretny zestaw dowodów E. W tym przypadku często upuszczamy mianownik p(E), który jest identyczny dla wszystkich h_i, z oszczędnością potencjalnie dużego kosztu obliczeniowego. Bez mianownika stworzyliśmy maksymalną wartość a posteriori dla hipotezy:

$\arg\max (h_i)\ p(E \mid h_i)\ p(h_i)$

Odczytujemy to wyrażenie jako „Maksymalna wartość w całym h_i $p(E \mid h_i)$ $p(h_i)$". Opisane właśnie uproszczenie jest bardzo ważne dla rozumowania diagnostycznego, a także w przetwarzaniu języka naturalnego. Oczywiście, arg max lub hipoteza maksymalnego prawdopodobieństwa, nie jest już zmienną losową, jak zdefiniowano wcześniej. Następnie rozważ obliczenie mianownika p (E) w sytuacji, gdy cała przestrzeń próbki jest podzielona przez zestaw hipotez hi. Podział zestawu jest zdefiniowany jako podział tego zestawu na rozłączne nieprzekraczające się podzbiory, których połączenie tworzy cały zestaw. Zakładając, że zestaw hipotez podzieli całą przestrzeń próbki, otrzymujemy:

$p(E) = \Sigma_i\ p(E \mid h_i)\ p(h_i)$

Zależność tę wykazano, biorąc pod uwagę fakt, że zestaw hipotez hi tworzy podział pełnego zestawu dowodów E wraz z regułą prawdopodobieństwa przecięcia dwóch zbiorów. Więc:

$E = (E \cap h_1) \cup (E \cap h_2) \cup ... \cup (E \cap h_n)$

Ale przez uogólnioną zasadę unii zbiorów:

$p(E) = p((E \cap h_1) \cup (E \cap h2) \cup ... \cup (E \cap h_n))$

$= p(E \cap h_1) + p(E \cap h_2) + ... + p(E \cap h_n) - p(E \cap h_1 \cap E \cap h_2 \cap ... \cap h_n)$

$= p(E \cap h_1) + p(E \cap h_2) + ... + p(E \cap h_n)$

ponieważ zestaw hipotez dotyczy podziału E, a ich przecięcie jest puste. To obliczenie p (E) daje ogólną postać twierdzenia Bayesa, w której zakładamy, że zbiór hipotez, hi dzielenie zbioru dowodów E:

phie.png

$p(h_i|E)$ to prawdopodobieństwo, że h_i jest prawdziwe, biorąc pod uwagę dowody E.

$p(h_i)$ jest prawdopodobieństwem, że hi jest ogólnie prawdziwe.

$p(E|h_i)$ to prawdopodobieństwo zaobserwowania dowodu E, gdy hi jest prawdziwe.

n jest liczbą możliwych hipotez.

Twierdzenie Bayesa zapewnia sposób obliczenia prawdopodobieństwa hipotezy hi, biorąc pod uwagę konkretny dowód, biorąc pod uwagę tylko prawdopodobieństwa, z którymi dowód wynika z faktycznych przyczyn (hipotez). Jako przykład załóżmy, że chcemy zbadać dowody geologiczne w pewnym miejscu, aby sprawdzić, czy nadaje się ono do znalezienia miedzi. Musimy wiedzieć z góry prawdopodobieństwo znalezienia każdego zestawu minerałów i prawdopodobieństwo istnienia pewnych dowodów obecnych, gdy zostanie znaleziony konkretny minerał Następnie możemy użyć twierdzenia Bayesa, z dowodami znalezionymi w konkretnej lokalizacji, aby określić prawdopodobieństwo miedzi. Takie podejście stosuje PROSPECTOR, zbudowany na Uniwersytecie Stanforda i SRI International i wykorzystywany w badaniach minerałów (miedź, molibden i inne minerały). POSZUKIWACZ znalazł komercyjnie znaczące złoża minerałów w kilku lokalizacjach. Następnie przedstawiamy prosty numeryczny przykład demonstrujący twierdzenie Bayesa. Załóżmy, że wychodzisz na zakup samochodu. Prawdopodobieństwo, że udasz się do dilera 1, d_1, wynosi 0,2.

Prawdopodobieństwo przejścia do rozdającego 2, d_2, wynosi 0,4. Jesteś tylko trzema dealerami biorąc pod uwagę, a prawdopodobieństwo przejścia do trzeciego, d3, wynosi również 0,4. Przy d_1 prawdopodobieństwo zakupu określonego samochodu, a_1, wynosi 0,2; u dealera d_2 prawdopodobieństwo zakupu a_1 wynosi 0,4. Wreszcie u dilera d3 prawdopodobieństwo zakupu a1 wynosi 0,3. Załóżmy, że kupujesz samochód A1. Jakie jest prawdopodobieństwo, że kupiłeś go u dilera d_2? Po pierwsze, chcemy wiedzieć, biorąc pod uwagę, że kupiłeś samochód a1, że kupiłeś go od sprzedawcy d_2, tj. W celu ustalenia p (d_2 | a_1). Prezentujemy twierdzenie Bayesa w postaci zmiennej do wyznaczania p (d_2 | a_1), a następnie ze zmiennymi związanymi z sytuacją w przykładzie.

p (d_2 | a_1) = (p (a_1 | d_2) p (d_2)) / (p (a_1 | d_1) p (d_1) + p (a_1 | d_2) p (d_2) + p (a_1 | d_3) p (d_3))

= (0,4) (0,4) / ((0,2) (0,2) + (0,4) (0,4) + (0,4) (0,3))

= 0,16 / 0,32

= 0,5

Stosowanie twierdzenia Bayesa wiąże się z dwoma głównymi zobowiązaniami: po pierwsze, muszą być znane wszystkie prawdopodobieństwa dotyczące powiązania dowodów z różnymi hipotezami, a także relacje prawdopodobieństwa między dowodami. Po drugie, a czasem trudniejsze do ustalenia, wszystkie relacje między dowodami a hipotezami lub p (E | h_k) muszą być oszacowane lub próbkowane empirycznie. Przypomnijmy, że obliczenie p (E) dla ogólnej postaci twierdzenia Bayesa wymagało również, aby zestaw hipotez hi podzielił zbiór dowodów E. Zasadniczo, a zwłaszcza w takich dziedzinach, jak medycyna i przetwarzanie języka naturalnego, założenia tego podziału nie można z góry uzasadnić. Warto jednak zauważyć, że wiele sytuacji, które naruszają to założenie (że poszczególne dowody dzielą zestaw dowodów) zachowują się całkiem dobrze! Korzystanie z tego założenia partycji, nawet w sytuacjach, gdy jest to nieuzasadnione, nazywa się przy użyciu naiwnego Bayesa lub klasyfikatora Bayesa. W przypadku naiwnego Bayesa przyjmuje się, że dla hipotezy h_j:

$$p(E|h_j) \approx \prod_{i=1}^{n} p(e_i| h_j)$$

tzn. zakładamy, że dowody są niezależne, biorąc pod uwagę szczególną hipotezę. Wykorzystując twierdzenie Bayesa do ustalenia prawdopodobieństwa pewnej hipotezy hi, biorąc pod uwagę zbiór dowodów E, p (h_i | E), liczby po prawej stronie równania są często łatwo dostępne. Jest to szczególnie prawdziwe w porównaniu z uzyskaniem wartości dla lewej strony równania, tj. Bezpośrednim wyznaczeniem p (h_i | E). Na przykład, ponieważ populacja jest mniejsza, o wiele łatwiej jest ustalić liczbę pacjentów z zapaleniem opon mózgowych, którzy mają bóle głowy, niż określić odsetek osób cierpiących na bóle głowy z zapaleniem opon mózgowych. Co ważniejsze, w przypadku prostego przypadku pojedynczej choroby i pojedynczego objawu nie jest potrzebnych bardzo wiele liczb. Problemy zaczynają się jednak, gdy rozważymy wiele chorób h_i z dziedziny chorób H i wiele objawów en z zestawu E możliwych objawów. Kiedy rozważymy każdą chorobę z H i każdy objaw z E pojedynczo, mamy m x n środków do zebrania i zintegrowania. (Właściwie m x n prawdopodobieństw późniejszych plus m + n wcześniejszych prawdopodobieństw.) Niestety, nasza analiza wkrótce stanie się znacznie bardziej złożona. Do tego momentu każdy objaw rozpatrywaliśmy indywidualnie. W rzeczywistych sytuacjach rzadko występują pojedyncze objawy. Na przykład, gdy lekarz rozważa pacjenta, musi wziąć pod uwagę wiele kombinacji objawów. Potrzebujemy formy twierdzenia Bayesa, aby rozważyć każdą pojedynczą hipotezę h_i w kontekście połączenia wielu możliwych objawów e_i.

p (h_i | $e_1 \cup e_2 \cup ... \cup e_n$) = (p ($h_i$) p ($e_1 \cup e_2 \cup ... \cup e_n$ | h_i)) / p ($e_1 \cup e_2 \cup ... \cup e_n$)

Przy jednej chorobie i jednym objawie potrzebowaliśmy tylko m x n pomiarów. Teraz, dla każdej pary objawów e_i i ej oraz konkretnej hipotezy choroby cześć, musimy znać zarówno p ($e_i \cup e_j \mid h_i$), jak i p ($e_i \cup e_j$). Liczba takich par wynosi nx (n - 1), lub w przybliżeniu n^2, gdy n ma objawy w E. Teraz, jeśli chcemy użyć Bayesa, będzie około (mx n^2 prawdopodobieństwa warunkowe) + (n^2 prawdopodobieństwo symptomu) + (m prawdopodobieństwa choroby) lub około mx n^2 + n^2 + m informacji do zebrania. W realistycznym systemie medycznym z 200 chorobami i 2000 objawami wartość ta wynosi ponad 800 000 000! Jest jednak nadzieja. Jak omówiono, kiedy przedstawiliśmy warunkową niezależność, wiele z tych par symptomów będzie niezależnych, to znaczy p ($e_i \mid e_j$) = p (e_i). Niezależność oznacza oczywiście, że obecność e_j nie ma wpływu na prawdopodobieństwo e_i. Na przykład w medycynie większość objawów nie jest powiązanych, np. Wypadanie włosów i ból łokcia. Ale nawet jeśli tylko dziesięć procent naszych przykładowych objawów nie jest niezależnych, pozostaje jeszcze około 80 000 000 relacji do rozważenia. W wielu sytuacjach diagnostycznych musimy również radzić sobie z negatywnymi informacjami, np. Gdy pacjent nie ma objawów takich jak złe ciśnienie krwi. Wymagamy zarówno: p (e_i) = 1 - p (e_i) i p ($h_i \mid e_i$) = 1 - p ($h_i \mid e_i$).

Zauważamy również, że p ($e_i \mid h_i$) i p ($h_i \mid e_i$) nie są takie same i prawie zawsze będą miały różne wartości. Zależności te oraz unikanie kołowego rozumowania są ważne przy projektowaniu bayesowskich sieci wierzeń. Ostatnim problemem, który ponownie sprawia, że utrzymywanie statystyk złożonych systemów bayesowskich jest praktycznie niemożliwe, jest potrzeba odbudowania tabel prawdopodobieństwa, gdy zostaną odkryte nowe związki między hipotezami a zestawami dowodów. W wielu aktywnych obszarach badawczych, takich jak medycyna, nowe odkrycia zachodzą nieustannie. Argumentacja bayesowska wymaga pełnych i aktualnych prawdopodobieństw, w tym prawdopodobieństw łącznych, jeśli wnioski z nich mają być prawidłowe. W wielu domenach tak obszerne gromadzenie i weryfikacja danych nie są możliwe, a jeśli to możliwe, dość drogie. Tam, gdzie te założenia są spełnione, podejścia bayesowskie oferują korzyść matematycznie uzasadnionego rozwiązania problemu niepewności. Większość domen systemowych ekspertów nie spełnia tych wymagań i musi opierać się na podejściach heurystycznych, jak przedstawiono w Rozdziale 8. Ponadto, z powodu problemów ze złożonością wiemy, że nawet dość potężne komputery nie mogą stosować pełnych technik bayesowskich do skutecznego rozwiązywania problemów w czasie rzeczywistym. Kończymy ten rozdział dwoma przykładami, które pokazują, jak podejście bayesowskie może działać w celu uporządkowania relacji hipotezy / dowodów. Ale najpierw musimy zdefiniować probabilistyczne maszyny / akceptory stanów skończonych

Zastosowania metodologii stochastycznej

W tej sekcji przedstawiamy dwa przykłady, które wykorzystują miary prawdopodobieństwa do uzasadnienia interpretacji niejednoznacznych informacji. Po pierwsze, definiujemy ważne narzędzie do modelowania oparte na maszynie stanu skończonego z Sekcji 3.1, probabilistycznej maszynie stanu skończonego.

DEFINICJA

PROBABILISTYCZNA MASZYNA SKOŃCZONA

Probabilistyczna maszyna stanów skończonych jest maszyną stanów skończonych, w której następną funkcją stanu jest rozkład prawdopodobieństwa dla pełnego zestawu stanów maszyny.

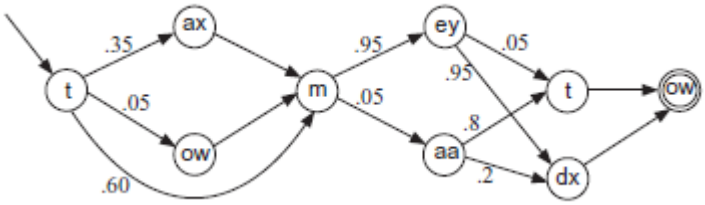
PROBABILISTYCZNY AKCEPTOR STANU SKOŃCZONEGO

Probabilistyczna maszyna stanów skończonych jest akceptorem, gdy jeden lub więcej stanów jest wskazanych jako stany początkowe, a jeden lub więcej jako stany akceptacji. Można zauważyć, że te dwie definicje są prostymi rozszerzeniami stanu skończonego i obecnością maszyn Moore. Dodatkiem

do niedeterminizmu jest to, że funkcja następnego stanu nie jest już funkcją w ścisłym tego słowa znaczeniu. Oznacza to, że nie ma unikalnego stanu zakresu dla każdej wartości wejściowej i każdego stanu domeny. Zamiast tego, dla każdego stanu, funkcja następnego stanu jest rozkładem prawdopodobieństwa we wszystkich możliwych kolejnych stanach

Jak zatem wymawia się „Tomato”?

Rysunek poniżej przedstawia probabilistyczny akceptor stanu skończonego, który reprezentuje różne wymowy słowa „tomato”. Szczególnie dopuszczalna wymowa słowa tomato to



charakteryzuje się ścieżką od stanu początkowego do stanu akceptacji. Wartości dziesiętne, które oznaczają łuki na wykresie, reprezentują prawdopodobieństwo, że głośnik wykona to konkretne przejście w maszynie stanu. Na przykład 60% wszystkich głośników w tym zestawie danych idzie bezpośrednio z t na m, bez produkowania żadnej samogłoski pomiędzy. Oprócz scharakteryzowania różnych sposobów, w jakie ludzie w bazie wymowy wymawiają słowo „pomidor”, model ten może być wykorzystany do interpretacji niejednoznacznych zbiorów fonemów. Odbywa się to poprzez sprawdzenie, jak dobrze fonemy dopasowują możliwe ścieżki przez automaty stanów powiązanych słów. Ponadto, biorąc pod uwagę częściowo utworzone słowo, maszyna może spróbować ustalić ścieżkę probabilistyczną, która najlepiej uzupełnia to słowo. W drugim przykładzie, również zaadaptowanym przez Jurafsky'ego i Martina, rozważamy problem rozpoznawania fonemów, często nazywany dekodowaniem. Załóżmy algorytm rozpoznawanie fonemów który zidentyfikował telefon ni (jak w „kolanie”), który pojawia się tuż po rozpoznanym słowie (telefon) I, i chcemy skojarzyć ni ze słowem lub pierwszą częścią słowa. W tym przypadku mamy do pomocy korpusy językowe, korpusy Brown i Switchboard. Korpus Browna to zbiór słów z 500 napisanych słów teksty, w tym gazety, powieści i pisma akademickie zebrane na Uniwersytecie Browna w latach 60. XX wieku (Kucera i Francis 1967, Francis 1979). Korpus centrali to 1,4 miliona słów w rozmowach telefonicznych. Korpusy te zawierają łącznie około 2 500 000 słów, które pozwalają nam próbkować zarówno informacje pisane, jak i mówione. Istnieje wiele sposobów identyfikowania najbardziej prawdopodobnego słowa kojarzonego z telefonem ni. Najpierw możemy ustalić, które słowo, z tym telefonem jako pierwsze, jest najczęściej używane. Tabela pierwszA przedstawia surowe częstotliwości tych słów wraz z prawdopodobieństwem ich wystąpienia, to znaczy częstotliwości słowa podzielonej przez całkowitą liczbę słów w tych połączonych ciałach. (Ta tabela została zaadaptowana przez Jurafsky'ego i Martina (2009); patrz ich książka dla uzasadnienia, że „the” należy do tej kolekcji). Z tych danych, słowo „the” wydaje się być pierwszym wyborem do dopasowania ni. Następnie zastosujemy formę twierdzenia Bayesa, którą przedstawiliśmy w poprzednim rozdziale. Nasza druga próba analizy telefonu po I polega na uproszczeniu tej formuły, która ignoruje jej mianownik:

$$p(\text{word} \mid [\text{ni}]) \propto p([\text{ni}] \mid \text{word}) \times p(\text{word})$$

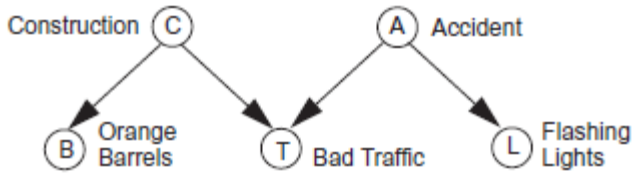
word	frequency	probability
knee	61	.000024
the	114834	.046
neat	338	.00013
need	1417	.00056
new	2625	.001

word	p([ni] | word)	p(word)	p([ni] | word) x p(word)
new	0.36	0.001	0.00036
neat	0.52	0.00013	0.000068
need	0.11	0.00056	0.000062
knee	1.0	0.000024	0.000024
the	0.0	0.046	0.0

Wyniki tego obliczenia, uporządkowane od najbardziej zalecanych do najmniej, znajdują się w tabeli drugiej i wyjaśniają, dlaczego p (ni | the) jest niemożliwe. Wyniki tabeli drugiej sugerują również, że new jest najbardziej prawdopodobnym słowem do dekodowania ni. Ale nowa kombinacja dwóch słów wydaje się nie mieć większego sensu, podczas gdy inne kombinacje, takie jak potrzebuję. Częścią problemu w tej sytuacji jest to, że wciąż rozważamy na poziomie telefonu, to znaczy określając prawdopodobieństwo p (ni | new). W rzeczywistości istnieje prosty sposób rozwiązania tego problemu, a mianowicie poszukiwanie wyraźnych kombinacji dwóch słów w korpusie. Zgodnie z tym tokiem rozumowania okazuje się, że potrzebuję o wiele bardziej prawdopodobnej pary następujących po sobie słów niż jestem nowy lub jakakolwiek inna kombinacja I-słów. Metodologia wyprowadzania prawdopodobieństw z par lub potrójnych kombinacji słów w ciałach nazywa się analizą n-gramową. Używając dwóch słów, używaliśmy bigramsów, z trzema trygramami.

Rozszerzenie przykładu drogi / ruchu

Ponownie przedstawiamy i rozszerzamy przykład z sekcji wcześniejszej. Załóżmy, że jedziesz autostradą międzystanową i zdajesz sobie sprawę, że stopniowo zwalniasz z powodu zwiększonego natężenia ruchu. Zaczynasz szukać możliwych wyjaśnień spowolnienia. Czy to może być budowa dróg? Czy był wypadek? Być może istnieją inne możliwe wyjaśnienia. Po kilku minutach natkniesz się na pomarańczowe beczki na poboczu drogi, które zaczynają odcinać zewnętrzny pas ruchu. W tym momencie stwierdzasz, że najlepszym wyjaśnieniem spowolnienia jest budowa dróg. Jednocześnie alternatywna hipoteza wypadku została wyjaśniona. Podobnie, gdybyś widział migające światła w odległości, na przykład z samochodu policyjnego lub karetki pogotowia, najlepszym wyjaśnieniem, biorąc pod uwagę ten dowód, byłby wypadek drogowy, a budowa drogi byłaby wyjaśniona. Wyjaśnienie hipotezy nie oznacza, że nie jest to już możliwe. Raczej w kontekście nowych dowodów jest to po prostu mniej prawdopodobne. Rycina 5.4 przedstawia opis bayesowski tego, co właśnie widzieliśmy. budowa dróg jest skorelowana z pomarańczowymi beczkami i złym ruchem.



Podobnie wypadek koreluje z migającymi światłami i złym ruchem. Analizujemy rysunek 5.4 i budujemy wspólny rozkład prawdopodobieństwa dla budowy drogi i złych relacji drogowych. Upraszczamy obie te zmienne, aby były prawdziwe (t) lub fałszywe (f) i przedstawiają rozkład prawdopodobieństwa w tabeli

C is true = .5

C	T	p
t	t	.3
t	f	.2
f	t	.1
f	f	.4

T is true = .4

Zauważ, że jeśli konstrukcja jest f, prawdopodobnie nie będzie dużego ruchu, a jeśli jest t, to prawdopodobnie jest zły ruch. Należy również zauważyć, że prawdopodobieństwo budowy drogi na autostradzie międzystanowej C = true wynosi 0,5, a prawdopodobieństwo złego ruchu, T = true wynosi 0,4 (to jest Nowy Meksyk!). Następnie rozważamy zmianę prawdopodobieństwa budowy drogi, biorąc pod uwagę fakt, że mamy zły ruch, lub p (C | T) lub p (C = t | T = t).

p (C | T) = p (C = t, T = t) / (p (C = t, T = t) + p (C = f, T = t)) = .3 / (.3 +. 1) = 0,75

Zatem teraz, z prawdopodobieństwem budowy drogi wynoszącym 0,5, biorąc pod uwagę fakt, że ruch jest rzeczywiście zły, prawdopodobieństwo budowy drogi wzrasta do 0,75. Prawdopodobieństwo to wzrośnie jeszcze bardziej dzięki obecności pomarańczowych beczek, co wyjaśnia hipotezę wypadku. Oprócz wymogu posiadania wiedzy lub pomiarów dla któregokolwiek z naszych średnic znajdujących się w określonym stanie, musimy również zająć się, jak wspomniano w sekcji 5.4.1, kwestiami złożoności. Rozważ obliczenie prawdopodobieństwa łącznego wszystkich parametrów z powyższego rysunku (przy użyciu reguły łańcucha i uporządkowanej topologicznie kolejności zmiennych):

p (C, A, B, T, L) = p (C) xp (A | C) xp (B | C, A) xp (T | C, A, B) xp (L | C, A, B , T)

Ten wynik jest ogólnym rozkładem miar prawdopodobieństwa, który jest zawsze prawdziwy. Koszt wytworzenia tej wspólnej tabeli prawdopodobieństwa jest wykładniczy pod względem liczby zaangażowanych parametrów, w tym przypadku wymaga tabeli o wielkości 25 lub 32. Rozważamy oczywiście problem zabawki z tylko pięcioma parametrami. Sytuacja o interesujących rozmiarach, z trzydziestoma lub więcej parametrami, wymaga wspólnej tabeli dystrybucji zawierającej około miliarda elementów. Jak zobaczymy w rozdziale 9.3, bayesowskie sieci przekonań i separacja d dają nam dalsze narzędzia do rozwiązania tej złożoności reprezentacyjnej i obliczeniowej

Epilog

Gry losowe pochodzą przynajmniej z cywilizacji greckiej i rzymskiej. Jednak dopiero w czasach europejskiego renesansu rozpoczęła się matematyczna analiza teorii prawdopodobieństwa. Jak zauważono w rozdziale, rozumowanie probabilistyczne rozpoczyna się od ustalenia zasad liczenia i

kombinatoryki. W rzeczywistości jedną z pierwszych „maszyn" kombinatorycznych przypisuje się Ramonowi Llullowi, hiszpańskiemu filozofowi i franciszkańskiemu mnichowi, który stworzył swoje urządzenie, które ma automatycznie wyliczać atrybuty Boga z zamiarem nawrócenia pogan . Pierwszą publikację o prawdopodobieństwach, De Ratiociniis Ludo Aleae, napisał Christian Huygens. Huygens opisuje wcześniejsze wyniki Blaise Pascal, w tym metodykę obliczania prawdopodobieństw, a także warunkowe prawdopodobieństwa. Pascal skupił się zarówno na „obiektywnej" analizie świata gier, jak i na bardziej „subiektywnej" analizie systemów przekonań, w tym na istnieniu Boga. Definicje prawdopodobieństw opierają się na formalizmie zaproponowanym przez Francuski matematyk Pierre Simon Laplace. Książka Laplace'a Theorie Analytique des Probabilitees (1816) dokumentuje to podejście. Praca Laplace'a była oparta na wcześniejszych wynikach opublikowanych przez Gotloba Leibnitza i Jamesa Bernoulli. Thomas Bayes był matematykiem i pastorem. Jego słynne twierdzenie zostało opublikowane w 1764 roku, po jego śmierci. Jego artykuł zatytułowany Esej na temat rozwiązania problemu w doktrynie szans został opublikowany w Philosophical Transactions of Royal Society of London. Jak na ironię, twierdzenie Bayesa nigdy nie jest wyraźnie określone w tym artykule, chociaż tak jest tam! Bayes prowadzi również obszerną dyskusję na temat „rzeczywistości" miar statystycznych. Badania Bayesa były częściowo motywowane do odpowiedzi na filozoficzny sceptycyzm szkockiego filozofa Davida Hume'a. Odrzucenie przez Hume'a związku przyczynowego zniszczyło wszelkie podstawy do argumentów poprzez cuda o istnieniu Boga. W rzeczywistości, w artykule z 1763 r. Przedstawionym Brytyjskiemu Towarzystwu Królewskiemu, minister Richard Price wykorzystał twierdzenie Bayesa, aby wykazać, że istnieją dobre dowody na korzyść cudów opisanych w Nowym Testamencie. Matematycy z początku XX wieku, w tym Fisher (1922), Popper (1959) i Carnap (1948), ukończyli podstawy współczesnej teorii prawdopodobieństwa, kontynuując „subiektywne / obiektywne" debaty na temat natury prawdopodobieństw. Kołmogorow (1950, 1965) aksjatyzował podstawy rozumowania probabilistycznego (patrz rozdział 5.2.1). Możliwe jest przedstawienie tematu wnioskowania probabilistycznego z kilku punktów widzenia. Dwa najbardziej popularne podejścia opierają się na rachunku zdań i teorii mnogości. Dla rachunku zdań, rozdział 2.1, twierdzeniom przypisuje się wartość ufności lub probabilistyczną wartość prawdy w przedziale [0,0, 1,0]. Podejście to oferuje naturalne rozszerzenie semantyki rachunku zdań. Wybraliśmy drugie podejście do probabilistycznego wnioskowania, czując, że ta orientacja jest nieco bardziej intuicyjna, przenosząc wszystkie techniki liczenia i inne techniki teorii mnogości, jak pokazano w rozdziale 5.1. W rozdziałach 9 i 13 rozszerzamy naszą prezentację systemów stochastycznych na reprezentację pierwszego rzędu (opartą na zmiennych) schematy. Twierdzenie Bayesa stanowiło podstawę dla kilku systemów eksperckich z lat 70. i 80. XX wieku, w tym do szczegółowej analizy ostrego bólu brzucha w University of Glasgow Hospital oraz PROSPECTOR, system ze Stanford Uniwersytet wspierający poszukiwania minerałów. Naiwne podejście Bayesa zastosowano w wielu problemach z klasyfikacją, w tym w rozpoznawaniu wzorców (Duda i Hart 1973), przetwarzaniu języka naturalnego (Mooney 1996) i innych. Domingos a Pazzani (1997) uzasadniają sukcesy naiwnych klasyfikatorów Bayesa w sytuacjach, które nie spełniają założeń Bayesowskiej niepodległości. Obecnie prowadzonych jest wiele badań dotyczących probabilistycznego wnioskowania w sztucznej inteligencji. Niepewne rozumowanie stanowi element konferencji AI, w tym AAAI, IJCAI, NIPS i ZEA. Istnieje kilka doskonałych tekstów wprowadzających w rozumowaniu probabilistycznym (Ross 1988, DeGroot 1989), a także kilka wprowadzeń do stosowania metod stochastycznych w aplikacjach sztucznej inteligencji (Russell i Norvig 2003, Jurafsky i Martin 2009, Manning i Schutze 1999).

Sztuczna inteligencja : Budowanie algorytmów sterowania do wyszukiwania w przestrzeni stanów

(VI / XVI)

ĆWICZENIA

1.

a. Napisz algorytm sprawdzania członków, aby rekurencyjnie określać, czy dany element należy do listy.

b. Napisz algorytm zliczający liczbę elementów na liście.

c. Napisz algorytm zliczający liczbę atomów na liście.

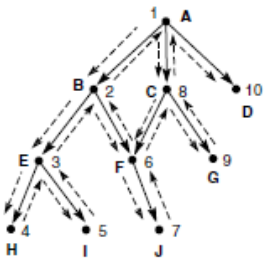
(Różnica między atomami a pierwiastkami polega na tym, że element może sam być listą).

2. Napisz algorytm rekurencyjny (przy użyciu list otwartych i zamkniętych), aby zaimplementować wyszukiwanie wszerz. Czy rekurencja pozwala pominąć otwartą listę podczas wdrażania pierwszego wyszukiwania? Wyjaśnić.

3. Śledź wykonanie rekurencyjnego algorytmu wyszukiwania w pierwszej kolejności (wersja, która nie korzysta z otwartej listy) w przestrzeni stanu na rysunku

Initialize: SL = [A]; NSL = [A]; DE = [ ]; CS = A;

AFTER ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]



4. W starożytnej hinduskiej ceremonii parzenia herbaty bierze udział trzech uczestników: starszy, sługa i dziecko. Cztery zadania, które wykonują: karmienie ogniem, serwowanie ciastek, nalewanie herbaty

i czytanie poezji; ta kolejność odzwierciedla malejące znaczenie zadań. Na początku ceremonii dziecko wykonuje wszystkie cztery zadania. Są one przekazywane pojedynczo służącemu i starszemu, aż pod koniec ceremonii starszy wykona wszystkie cztery zadania. Nikt nie może podjąć mniej ważnego zadania niż te, które już wykonują. Wygeneruj sekwencję ruchów, aby przenieść wszystkie zadania z dziecka na starszego. Napisz algorytm rekurencyjny, aby wykonać sekwencję ruchu.

5. Korzystając z definicji ruchu i ścieżki dla trasy rycerza, prześledź wykonanie wzorca_wyszukiwania na celach:

a. ścieżka (1,9).

b. ścieżka (1,5).

c. ścieżka (7,6).

Podczas próby wykonania predykatów ruchu, wyszukiwanie jest często zapętlone. Porozmawiaj o wykrywaniu pętli i nawracaniu w tej sytuacji.

6. Napisz definicję pseudokodu dla pierwszej wersji wzorca_wyszukiwania. Omów efektywność czasową i przestrzenną tego algorytmu.

7. Używając reguły z przykładu 6.2.3 jako modelu, napisz osiem reguł ruchu wymaganych dla pełnej wersji 8 × 8 trasy rycerza.

8. Korzystając ze stanów celu i początkowego z rysunku 6.3, ręcznie uruchom rozwiązanie systemu produkcyjnego 8-puzzle:

a. W sposób zorientowany na cel.

b. W sposób oparty na danych.

9. Rozważ problem doradcy finansowego omówiony w rozdziałach 2, 3 i 4. Używanie rachunku predykatu jako języka reprezentacji:

a. Napisz problem wprost jako system produkcyjny.

b. Wygeneruj przestrzeń stanu i etapy pamięci roboczej dla rozwiązania opartego na danych z

przykładu w rozdziale 3.

10. Powtórz problem 9.b, aby uzyskać rozwiązanie ukierunkowane na cel.

11. W sekcji 6.2.3 przedstawiono ogólne strategie rozwiązywania konfliktów załamania, aktualności i specyficzności. Zaproponuj i uzasadnij jeszcze dwie takie strategie.

12. Zaproponuj dwie aplikacje odpowiednie do rozwiązania z wykorzystaniem architektury tablicowej. Krótko scharakteryzuj organizację tablicy i źródeł wiedzy dla każdego wdrożenia.

Wprowadzenie

Do tego momentu przedstawiono rozwiązywanie problemów jako przeszukiwanie zestawu sytuacji lub stanów problemowych. Przedstawiono rachunek predykatów jako medium do opisu stanów problemu i wnioskowania dźwiękowego jako metodę tworzenia nowych stanów. Wprowadzono wykresy do reprezentowania i łączenia sytuacji problemowych. Algorytmy cofania, a także algorytmy wyszukiwania w pierwszej kolejności i w pierwszej kolejności mogą eksplorować te wykresy. W Części

4 przedstawiono algorytmy wyszukiwania heurystycznego. W rozdziale 5, biorąc pod uwagę probabilistyczne stany świata, wnioskowanie stochastyczne wykorzystano do wytworzenia nowych stanów. Podsumowując, wcześniejsza część:

1. Rozwiązanie problemu jako ścieżkę na wykresie od stanu początkowego do celu.

2. Wykorzystano wyszukiwanie do systematycznego testowania alternatywnych ścieżek do celów.

3. Zastosowano backtracking lub jakiś inny mechanizm, aby umożliwić algorytmom odzyskanie ze ścieżek, które nie znalazły celu.

4. Używane listy do prowadzenia wyraźnych rejestrów rozważanych stanów.

a. Otwarta lista pozwala algorytmowi zbadać niesprawdzone stany, jeśli to konieczne.

b. Zamknięta lista odwiedzonych stanów umożliwia algorytmowi wdrożenie wykrywania pętli i uniknięcie powtarzania bezowocnych ścieżek.

5. Zaimplementowano otwartą listę jako stos dla pierwszego wyszukiwania w głębokości, kolejkę dla pierwszego wyszukiwania i kolejkę priorytetową dla najlepszego wyszukiwania.

W tej sekcji wprowadzimy dalsze techniki budowania algorytmów wyszukiwania. Wyszukiwanie rekurencyjne implementuje wyszukiwanie głębokości, szerokości i najlepsze pierwsze w bardziej zwięzły i naturalny sposób. Ponadto rekurencja jest wzbogacona o ujednolicenie w celu przeszukiwania przestrzeni stanu generowanej przez twierdzenia rachunku predykatu. Ten algorytm wyszukiwania ukierunkowanego jest podstawą PROLOGU i kilka systemów eksperckich W dalszej sekcji wprowadzamy systemy produkcyjne, ogólną architekturę rozwiązywania problemów kierowanych wzorcami, która była szeroko stosowana do modelowania człowieka rozwiązywanie problemów, , a także w innych aplikacjach AI, w tym systemach eksperckich. Przedstawiamy kolejną architekturę sterowania rozwiązywaniem problemów AI, tablicę.

Wyszukiwanie na podstawie rekurencji (opcjonalnie)

Wyszukiwanie rekurencyjne

W matematyce definicja rekurencyjna wykorzystuje termin zdefiniowany jako część własnej definicji. W informatyce rekurencja służy do definiowania i analizowania zarówno struktur danych, jak i procedur. Procedura rekurencyjna składa się z:

1. Krok rekurencyjny: procedura wzywa się do powtórzenia sekwencji działań.

2. Warunek zakończenia, który zatrzymuje powtarzanie się procedury w nieskończoność (rekurencyjna wersja nieskończonej pętli).

Oba te komponenty są niezbędne i pojawiają się we wszystkich rekurencyjnych definicjach i algorytmach. Rekurencja to naturalny konstrukt kontrolny dla struktur danych, które mają regularną strukturę i nie mają określonego rozmiaru, takich jak listy, drzewa i wykresy, i jest szczególnie odpowiedni do wyszukiwania w przestrzeni stanów. Bezpośrednie tłumaczenie algorytmu przeszukiwania w pierwszej kolejności na formę rekurencyjną ilustruje równoważność rekurencji i iteracji. Algorytm wykorzystuje zmienne globalne zamknięte i otwarte do utrzymywania list stanów. Wyszukiwanie według szerokości i najlepszego wyniku można zaprojektować praktycznie z tym samym algorytmem, tzn. Zachowując zamkniętą jako globalną strukturę danych i implementując opcję otwartą jako kolejkę lub kolejkę priorytetową, a nie jako stos (stos kompilacji staje się kolejką kompilacji lub zbuduj kolejkę priorytetową):

```
unction depthsearch; % open & closed global
begin
if open is empty
then return FAIL;
current_state := the first element of open;
if current_state is a goal state
then return SUCCESS
else
begin
open := the tail of open;
closed := closed with current_state added;
for each child of current_state
if not on closed or open % build stack
then add the child to the front of open
end;
depthsearch % recur
end.
```

Wyszukiwanie w pierwszej kolejności, tak jak właśnie przedstawione, nie wykorzystuje pełnej mocy rekurencji. Możliwe jest dalsze uproszczenie procedury poprzez użycie samej rekurencji (zamiast jawnej otwartej listy) do organizowania stanów i ścieżek w przestrzeni stanów. W tej wersji algorytmu globalna zamknięta lista jest używana do wykrywania duplikatów stanów i zapobiegania pętlom, a otwarta lista jest uwikłana w rekordy aktywacji środowiska rekurencyjnego. Ponieważ nie można już jawnie manipulować otwartą listą, wyszukiwanie w pierwszej kolejności i wyszukiwanie w pierwszej kolejności nie są już naturalnymi rozszerzeniami następującego algorytmu:

```
function depthsearch (current_state); % closed is global
begin
if current_state is a goal
then return SUCCESS;
add current_state to closed;
while current_state has unexamined children
begin
child := next unexamined child;
if child not member of closed
```

then if depthsearch(child) = SUCCESS

then return SUCCESS

end;

return FAIL % search exhausted

end

Algorytm ten nie generuje wszystkich potomków stanu i umieszcza je na otwartej liście, ale generuje stany potomne pojedynczo i rekurencyjnie przeszukuje potomków każdego dziecka przed wygenerowaniem jego rodzeństwa. Zauważ, że algorytm zakłada zamówienie do operatorów generujących stan. Podczas rekurencyjnego przeszukiwania stanu potomnego, jeśli jakiś potomek tego stanu jest celem, wywołanie rekurencyjne zwraca sukces, a algorytm ignoruje rodzeństwo.

Jeśli wywołanie rekurencyjne w stanie potomnym nie znajdzie celu, generowane jest następne rodzeństwo i przeszukiwane są wszystkie jego potomki. W ten sposób algorytm przeszukuje cały wykres w pierwszej kolejności głębokości. Czytelnik powinien zweryfikować, czy faktycznie przeszukuje wykres w tej samej kolejności, co algorytm przeszukiwania z głębokością pierwszą .

Pominięcie jawnej otwartej listy jest możliwe poprzez rekurencję. Mechanizmy, za pomocą których język programowania realizuje rekurencję, obejmują osobny rekord aktywacyjny (Aho i Ullman 1977) dla każdego wywołania rekurencyjnego. Każdy rekord aktywacji przechwytuje zmienne lokalne i stan wykonania każdego wywołania procedury. Gdy procedura jest wywoływana rekurencyjnie z nowym stanem, nowy rekord aktywacyjny przechowuje jego parametry (stan), dowolne zmienne lokalne i bieżący stan wykonania. W algorytmie wyszukiwania rekurencyjnego serie stanów na bieżącej ścieżce są zapisywane w sekwencji rekordów aktywacyjnych wywołań rekurencyjnych. Zapis każdego połączenia wskazuje również ostatnią operację użytą do wygenerowania stanu potomnego; pozwala to na wygenerowanie następnego rodzeństwa w razie potrzeby. Cofanie następuje, gdy wszyscy potomkowie stanu nie uwzględniają celu, co powoduje niepowodzenie wywołania rekurencyjnego. To zwraca błąd procedury rozwijania stanu nadrzędnego, który następnie generuje i powtarza się przy następnym rodzeństwie. W tej sytuacji wewnętrzne mechanizmy rekurencji działają na otwartej liście stosowanej w iteracyjnej wersji algorytmu. Rekursywna implementacja pozwala programiście ograniczyć jego punkt widzenia do jednego stanu i jego dzieci, bez konieczności jawnego utrzymywania otwartej listy stanów. Zdolność rekurencji do wyrażania globalnych koncepcji w formie zamkniętej jest głównym źródłem jej siły. Jak pokazują te dwa algorytmy, wyszukiwanie w przestrzeni stanów jest z natury procesem rekurencyjnym. Aby znaleźć ścieżkę od bieżącego stanu do celu, przejdź do stanu potomnego i powtórz. Jeśli ten stan dziecka nie prowadzi do celu, spróbuj po kolei rodzeństwa. Rekurencja dzieli duży i trudny problem (przeszukiwanie całej przestrzeni) na mniejsze, prostsze części (generuje dzieci z jednego stanu) i stosuje tę strategię (rekurencyjnie) do każdego z nich. Proces ten trwa do momentu wykrycia stanu celu lub wyczerpania przestrzeni. W następnej sekcji to rekurencyjne podejście do rozwiązywania problemów zostało rozszerzone na kontroler logicznego rozwiązywania problemów, który wykorzystuje unifikację i wnioskowanie do generowania i przeszukiwania przestrzeni relacji logicznych. Algorytm obsługuje wiele celów, a także łańcuchy wstecz od celu do lokalu.

Przykład wyszukiwania rekurencyjnego: Uzasadnienie oparte na wzorcach

Stosujemy wyszukiwanie rekurencyjne w przestrzeni logicznych wniosków; wynikiem jest ogólna procedura wyszukiwania specyfikacji problemów opartych na rachunku predykatu. Załóżmy, że chcemy napisać algorytm, który określa, czy wyrażenie rachunku predykatu jest logiczną konsekwencją

pewnego zestawu twierdzeń. Sugeruje to wyszukiwanie ukierunkowane z początkowym zapytaniem tworzącym cel i modus ponens definiujący przejścia między stanami. Biorąc pod uwagę cel (taki jak p (a)), algorytm używa unifikacji, aby wybrać implikacje, których wnioski pasują do celu (np. Q (X) → p (X)). Ponieważ algorytm traktuje implikacje jako potencjalne reguły rozwiązywania zapytania, często są one po prostu nazywane regułami. Po ujednoliceniu celu z wnioskiem o implikacji (lub regule) i zastosowaniu wynikających z tego podstawień w regule, założenie reguły staje się nowym celem (q (a)). To się nazywa subgoal. Algorytm następnie powtarza się w podzadaniu. Jeśli podskala jest zgodna z faktem w bazie wiedzy, wyszukiwanie kończy się. Szereg wnioskowania, które doprowadziły od pierwotnego celu do podanych faktów, potwierdzają prawdziwość pierwotnego celu.

function pattern_search (current_goal);

begin

if current_goal is a member of closed % test for loops

then return FAIL

else add current_goal to closed;

while there remain in data base unifying facts or rules do

begin

case

current_goal unifies with a fact:

return SUCCESS;

current_goal is a conjunction (p

∧ ...):

begin

for each conjunct do

call pattern_search on conjunct;

if pattern_search succeeds for all conjuncts

then return SUCCESS

else return FAIL

end;

current_goal unifies with rule conclusion (p in q

→ p):

begin

apply goal unifying substitutions to premise (q);

call pattern_search on premise;

if pattern_search succeeds

then return SUCCESS

else return FAIL

end;

end; % end case

end;

return FAIL

end.

W funkcji wzorzec_wyszukiwania wyszukiwanie jest wykonywane przez zmodyfikowaną wersję algorytmu wyszukiwania rekurencyjnego, który korzysta z unifikacji, aby określić, kiedy dwa wyrażenia pasują do siebie, i modus ponens w celu wygenerowania potomków stanów. Bieżący fokus wyszukiwania jest reprezentowany przez zmienną current_goal. Jeśli current_goal pasuje do faktu, algorytm zwraca sukces. W przeciwnym razie algorytm próbuje dopasować wartość bieżąca do wniosku z jakąś regułą, rekurencyjnie próbując rozwiązać przesłankę. Jeśli wartość bieżąca nie pasuje do żadnego z podanych twierdzeń, algorytm zwraca błąd. Algorytm ten obsługuje również cele łączone. Dla uproszczenia algorytm nie rozwiązuje problemu utrzymania spójności między podstawieniami zmiennych wytwarzanymi przez unifikację. Jest to ważne przy rozwiązywaniu spójnych zapytań ze wspólnymi zmiennymi (jak w p (X) ∧ q (X)). Obie koniunkcje się powiodły, ale muszą również odnieść sukces z tym samym ujednoliconym wiązaniem dla X. Główną zaletą stosowania ogólnych metod, takich jak unifikacja i modus ponens do generowania stanów, jest to, że wynikowy algorytm może przeszukiwać dowolną przestrzeń wnioskowania logicznego, w której specyfika problem opisano za pomocą twierdzeń rachunku predykatu. Mamy zatem sposób na oddzielenie wiedzy na temat rozwiązywania problemów od jej kontroli i wdrażania na komputerze. pattern_search stanowi nasz pierwszy przykład oddzielenia wiedzy problemowej od kontroli wyszukiwania. Chociaż początkowa wersja wzorca_wyszukiwania określa zachowanie algorytm wyszukiwania dla wyrażeń rachunku predykatu, należy jeszcze zająć się kilkoma subtelnościami. Obejmują one kolejność, z jaką algorytm próbuje alternatywnych dopasowań oraz poprawne działanie pełnego zestawu operatorów logicznych (∧, ∨ i ¬). Logika jest deklaratywna i bez ustalonej strategii wyszukiwania: określa przestrzeń możliwych wnioskowań, ale nie mówi rozwiązującemu problemu, jak zrobić użyteczne. Aby uzasadnić za pomocą rachunku predykatu, potrzebujemy systemu kontroli, który systematycznie przeszukuje przestrzeń, unikając bezsensownych ścieżek i pętli. Algorytm kontroli, taki jak wzorzec_wyszukiwania, musi wypróbować alternatywne dopasowania w pewnej kolejności. Znając tę kolejność, projektant programu może kontrolować wyszukiwanie poprzez prawidłowe porządkowanie reguł w bazie wiedzy. Prostym sposobem zdefiniowania takiego porządku jest wymaganie od algorytmu wypróbowania reguł i faktów w kolejności, w jakiej pojawiają się w bazie wiedzy. Drugim zagadnieniem jest istnienie logicznych łączników w przesłankach reguł: np. Implikacje postaci „p ← q ∧ r ”lub„ p ← q ∨ (r ∧ s). ” Jak zostanie przypomniane z dyskusji i / lub wykresów, operator indicates wskazuje, że oba wyrażenia muszą być prawdziwe, aby cała przesłanka była prawdziwa. Ponadto spójniki wyrażenia należy rozwiązać za pomocą spójnych wiązań zmiennych. Zatem, aby rozwiązać p (X) ∧ q (X), nie wystarczy rozwiązać p (X) z podstawieniem {a / X} i q (X) z podstawieniem {b / X}. Oba muszą zostać rozwiązane przy użyciu tego samego jednoznacznego wiązania dla X. An lub operator, z drugiej strony, i wskazuje, że jedno z wyrażeń musi zostać uznane za prawdziwe. Algorytm wyszukiwania musi to uwzględnić. Ostatnim dodatkiem do algorytmu jest umiejętność rozwiązywania celów z logiczną negacją (¬). pattern_search radzi sobie z zanegowanymi celami, rozwiązując operand z ¬. Jeśli ten podzadanie powiedzie się, wówczas wyszukiwanie wzorzec_z powodzeniem nie powiedzie się. Jeśli operand się nie

powiedzie, wówczas attern_search zwraca pusty zestaw podstawień, co wskazuje na sukces. Zauważ, że nawet jeśli podzadanie może zawierać zmienne, wynik rozwiązania jego negacji może nie zawierać żadnych podstawień. Jest tak, ponieważ ¬ może odnieść sukces tylko wtedy, gdy operand zawiedzie; dlatego nie może zwrócić żadnych powiązań dla operandu.

Na koniec algorytm nie powinien zwracać sukcesu, ale powinien zwracać powiązania związane z rozwiązaniem. Pełna wersja wzorca_wyszukiwania, który zwraca zestaw ujednoliceń, który spełnia każde podzadanie, to:

```
function pattern_search(current_goal);

begin

if current_goal is a member of closed % test for loops

then return FAIL

else add current_goal to closed;

while there remain unifying facts or rules do

begin

case

current_goal unifies with a fact:

return unifying substitutions;

current_goal is negated (¬ p):

begin

call pattern_search on p;

if pattern_search returns FAIL

then return {}; % negation is true

else return FAIL;

end;

current_goal is a conjunction (p ∧…):

begin

for each conjunct do

begin

call pattern_search on conjunct;

if pattern_search returns FAIL

then return FAIL;

else apply substitutions to other conjuncts;

end;
```

if pattern_search returns SUCCESS for all conjuncts

then return composition of unifications;

else return FAIL;

end;

current_goal is a disjunction (p ∨ …):

begin

repeat for each disjunct

call pattern_search on disjunct

until no more disjuncts or SUCCESS;

if pattern_search returns SUCCESS

then return substitutions

else return FAIL;

end;

current_goal unifies with rule conclusion (p in p ← q):

begin

apply goal unifying substitutions to premise (q);

call pattern_search on premise;

if pattern_search returns SUCCESS

then return composition of p and q substitutions

else return FAIL;

end;

end; %end case

end %end while

return FAIL

end.

Ten algorytm wzorca wyszukiwania dla przeszukiwania przestrzeni reguł i faktów rachunku predykatów stanowi podstawę Prologu (gdzie stosowana jest forma predykatów z klauzuli Horn, Rozdział 14.3) oraz w wielu powłokach systemu ekspertów ukierunkowanych na cel (Rozdział 8). Alternatywna struktura sterowania dla wyszukiwania ukierunkowanego na wzorzec jest zapewniona przez system produkcyjny, omówiony w następnym rozdziale.

Systemy produkcyjne

Definicja i historia

System produkcyjny jest modelem obliczeniowym, który okazał się szczególnie ważny w sztucznej inteligencji, zarówno przy wdrażaniu algorytmów wyszukiwania, jak i modelowaniu rozwiązywania ludzkich problemów. System produkcyjny zapewnia ukierunkowane na wzorce sterowanie procesem rozwiązywania problemów i składa się z zestawu reguł produkcji, pamięci roboczej i cyklu kontroli rozpoznawania.

DEFINICJA

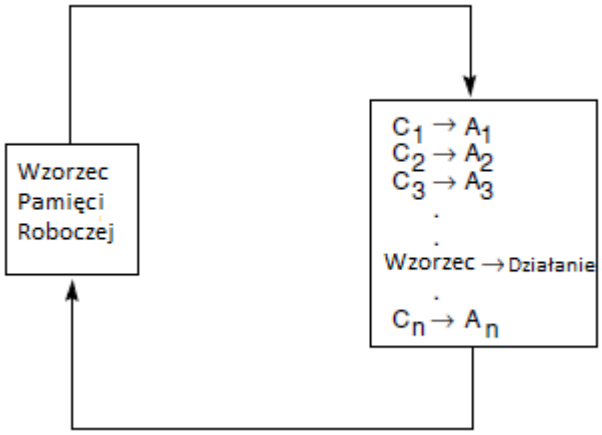
SYSTEM PRODUKCYJNY

System produkcji jest zdefiniowany przez:

1. Zestaw reguł produkcji. Są to często po prostu produkcje. Produkcja jest parą warunek-działanie i definiuje pojedynczy fragment wiedzy na temat rozwiązywania problemów. Warunek stanowi część reguły, która określa, kiedy reguła może być zastosowana do wystąpienia problemu. Część czynności określa powiązane kroki rozwiązywania problemów.

2. Pamięć robocza zawiera opis aktualnego stanu świata w procesie wnioskowania. Opis ten jest wzorcem dopasowanym do warunku części produkcji, aby wybrać odpowiednie działania rozwiązywania problemów. Gdy element warunku reguły jest dopasowany do zawartości pamięci roboczej, wówczas można wykonać działania powiązane z tym warunkiem. Działania reguł produkcji są specjalnie zaprojektowane w celu zmiany zawartości pamięci roboczej.

3. Cykl rozpoznawania-działania. Struktura sterowania dla systemu produkcyjnego jest prosta: pamięć robocza jest inicjowana początkowym opisem problemu. Obecny stan rozwiązywania problemów jest utrzymywany jako zestaw wzorców w pamięci roboczej. Wzory te są dopasowane do warunków reguł produkcji; tworzy to podzbiór reguł produkcji, zwany zestawem konfliktów, którego warunki są zgodne z wzorcami w pamięci roboczej. Mówi się, że produkcje w zestawie konfliktów są włączone. Następnie wybiera się jedną z produkcji w zestawie konfliktów (rozwiązywanie konfliktów) i uruchamia produkcję. Aby uruchomić regułę, wykonuje się jej działanie, zmieniając zawartość pamięci roboczej. Po uruchomieniu wybranej produkcji cykl sterowania powtarza się ze zmodyfikowaną pamięcią roboczą. Proces kończy się, gdy zawartość pamięci roboczej nie spełnia warunków żadnej reguły. Rozwiązywanie konfliktów wybiera regułę z zestawu konfliktów do strzelania. Strategie rozwiązywania konfliktów mogą być proste, takie jak wybranie pierwszej reguły, której stan odpowiada stanowi świata, lub może obejmować złożoną heurystykę wyboru reguł. Jest to ważny sposób, w jaki system produkcyjny pozwala na dodanie kontroli heurystycznej do algorytmu wyszukiwania. Model czystego systemu produkcyjnego nie ma mechanizmu odzyskiwania z ślepych zaułków podczas poszukiwań; po prostu trwa, dopóki nie zostaną włączone żadne produkcje i nie zatrzyma się. Większość praktycznych wdrożeń systemów produkcyjnych pozwala na powrót do poprzedniego stanu pamięci roboczej w takich sytuacjach. Schemat systemu produkcyjnego przedstawiono na rysunku



Bardzo prosty przykład wykonania systemu produkcyjnego pokazano na rysunku.

Iteracja #	Pamięć robocza	Zbiór konfliktu	Reguła Wyzwolona
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	Ø	Halt

Jest to program systemu produkcyjnego do sortowania łańcucha złożonego z liter a, b i c. W tym przykładzie produkcja jest włączona, jeśli jej stan odpowiada części ciągu w pamięci roboczej. Po uruchomieniu reguły podciąg pasujący do warunku reguły jest zastępowany łańcuchem po prawej stronie reguły. Systemy produkcyjne to ogólny model obliczeń, który można zaprogramować do robienia wszystkiego, co można zrobić na komputerze. Ich prawdziwą siłą jest jednak architektura systemów opartych na wiedzy. Pomysł projektowania opartego na produkcji dla komputerów powstał pierwotnie z pism Posta, który zaproponował model reguł produkcji jako formalną teorię obliczeń. Głównym konstruktem tej teorii był zestaw reguł przepisywania ciągów znaków na wiele sposobów podobnych do reguł parsowania w przykładzie 3.3.6. Jest również ściśle związany z podejściem przyjętym przez algorytmy Markowa (Markov 1954) i, podobnie jak one, ma moc równoważną maszynie Turinga. Interesujące zastosowanie reguł produkcji do modelowania ludzkiego poznania znajduje się w pracach Newella i Simona w Carnegie Institute of Technology (obecnie Carnegie Mellon University) w latach 60. i 70. XX wieku. Opracowane przez nich programy, w tym ogólne rozwiązywanie problemów, są w dużej mierze odpowiedzialne za znaczenie systemów produkcyjnych w AI. W tych badaniach ludzi monitorowano w różnych czynnościach rozwiązywania problemów, takich jak rozwiązywanie problemów w logice predykatów i granie w gry takie jak szachy. Protokół (wzorce zachowań, w tym słowne opisy procesu rozwiązywania problemów, ruchy gałek ocznych itp.) Osób rozwiązujących problemy zostały zapisane i rozbite na podstawowe elementy. Składniki te zostały uznane za podstawowe elementy wiedzy na temat rozwiązywania problemów u ludzi i zostały utworzone jako przeszukanie wykresu (zwanego wykresem zachowań problemowych). Następnie zastosowano system produkcyjny do wdrożenia earch tego wykresu. Reguły produkcji reprezentowały

zestaw umiejętności rozwiązywania problemów przez człowieka. Obecne skupienie uwagi było reprezentowane jako obecny stan świata. Podczas uruchamiania systemu produkcyjnego „uwaga" lub „bieżąca koncentracja" rozwiązania problemu pasowałaby do reguły produkcyjnej, która zmieniałaby stan „uwagi" w celu dopasowania do innej zakodowanej w produkcji umiejętności i tak dalej. Należy zauważyć, że w tej pracy Newell i Simon wykorzystali system produkcyjny nie tylko jako narzędzie do wdrażania wyszukiwania grafów, ale także jako rzeczywisty model postępowania człowieka w rozwiązywaniu problemów. Produkcje odpowiadały umiejętności rozwiązywania problemów w długotrwałej pamięci człowieka. Podobnie jak umiejętności w zakresie pamięci długoterminowej, produkcje te nie ulegają zmianie w wyniku działania systemu; są przywoływane przez „wzorzec" konkretnej instancji problemu, a nowe umiejętności mogą być dodawane bez konieczności „przekodowywania" wcześniejszej wiedzy. Pamięć robocza systemu produkcyjnego odpowiada pamięci krótkotrwałej lub bieżącej koncentracji uwagi u człowieka i opisuje aktualny etap rozwiązania problemu. Zawartość pamięci roboczej zasadniczo nie jest zachowywana po rozwiązaniu problemu. Te początki technologii systemu produkcji są dalej opisane w Human Problem Solving przez Newella i Simona oraz w Luger. Newell, Simon i inni nadal używają reguł produkcji do modelowania różnicy między nowicjuszami a ekspertami w takich obszarach, jak rozwiązywanie problemów ze słowami algebry i problemów fizyki. Systemy produkcyjne stanowią również podstawę do nauki uczenia się zarówno na ludziach, jak i na komputerach. ACT i SOAR bazują na tej tradycji. Systemy produkcyjne zapewniają model kodowania ludzkiej wiedzy specjalistycznej w formie reguł i projektowania algorytmów wyszukiwania opartych na wzorach, zadań, które są kluczowe w projektowaniu systemu eksperckiego opartego na regułach. W systemach eksperckich niekoniecznie zakłada się, że system produkcyjny faktycznie modeluje ludzkie zachowanie podczas rozwiązywania problemów; jednak aspekty systemów produkcyjnych, które czynią je użytecznymi jako potencjalny model rozwiązywania problemów człowieka (modułowość reguł, rozdział wiedzy i kontroli, rozdział pamięci roboczej i wiedzy na temat rozwiązywania problemów), czynią je idealnym narzędziem do projektowania i budowania ekspertów systemy. Ważna rodzina języków AI pochodzi bezpośrednio z badań języka systemu produkcyjnego w Carnegie Mellon. To są języki OPS; OPS oznacza oficjalny system produkcji. Chociaż ich początki polegają na modelowaniu rozwiązywania problemów ludzkich, języki te okazały się bardzo skuteczne w programowaniu systemów eksperckich i innych aplikacji AI. OPS5 był językiem implementacyjnym dla konfiguratora VAX XCON i innych wczesnych systemów eksperckich opracowanych w Digital Equipment Corporation. Tłumacze OPS są szeroko dostępni na komputery PC i stacje robocze. CLIPS, zaimplementowany w języku programowania C, jest szeroko stosowaną, obiektową wersją systemu produkcyjnego zbudowanego przez NASA. JESS, system produkcyjny wdrożony w Javie, został stworzony przez Sandia National Laboratories. W następnej sekcji podajemy przykłady wykorzystania systemu produkcyjnego do rozwiązywania różnych problemów związanych z wyszukiwaniem.

Przykłady systemów produkcyjnych

PRZYKŁAD: ZMIENIONO 8-PUZZLE

Przestrzeń wyszukiwania wygenerowana przez 8-puzzle, , jest zarówno wystarczająco złożona, aby była interesująca, jak i wystarczająco mała, aby była możliwa do przeszukiwania, dlatego często jest używana do eksploracji różnych strategii wyszukiwania, takich jak wyszukiwanie od pierwszej do głębokości i od pierwszego do drugiego , a także strategie heurystyczne. Prezentujemy teraz rozwiązanie systemu produkcyjnego. Przypomnijmy, że zyskujemy ogólność, myśląc raczej o „przesunięciu pustego miejsca", a nie o przeniesieniu numerowanego kafelka. Legalne ruchy są zdefiniowane przez produkcje poniżej:

Stan startowy :

2	8	3
1	6	4
7		5

Stan docelowy :

1	2	3
8		4
7	6	5

Zestaw produkcyjny:

stan celu w pamięci roboczej -> postój

puste miejsce nie znajduje się na lewej krawędzi -> przesuń puste miejsce w lewo

puste miejsce nie znajduje się na górnej krawędzi ->przesuń puste miejsce w górę

puste miejsce nie znajduje się na prawej krawędzi -> przesuń puste miejsce w prawo

puste miejsce nie znajduje się na dole -> przesuń puste miejsce w dół

Pamięć robocza to obecny stan płyty i stan celu

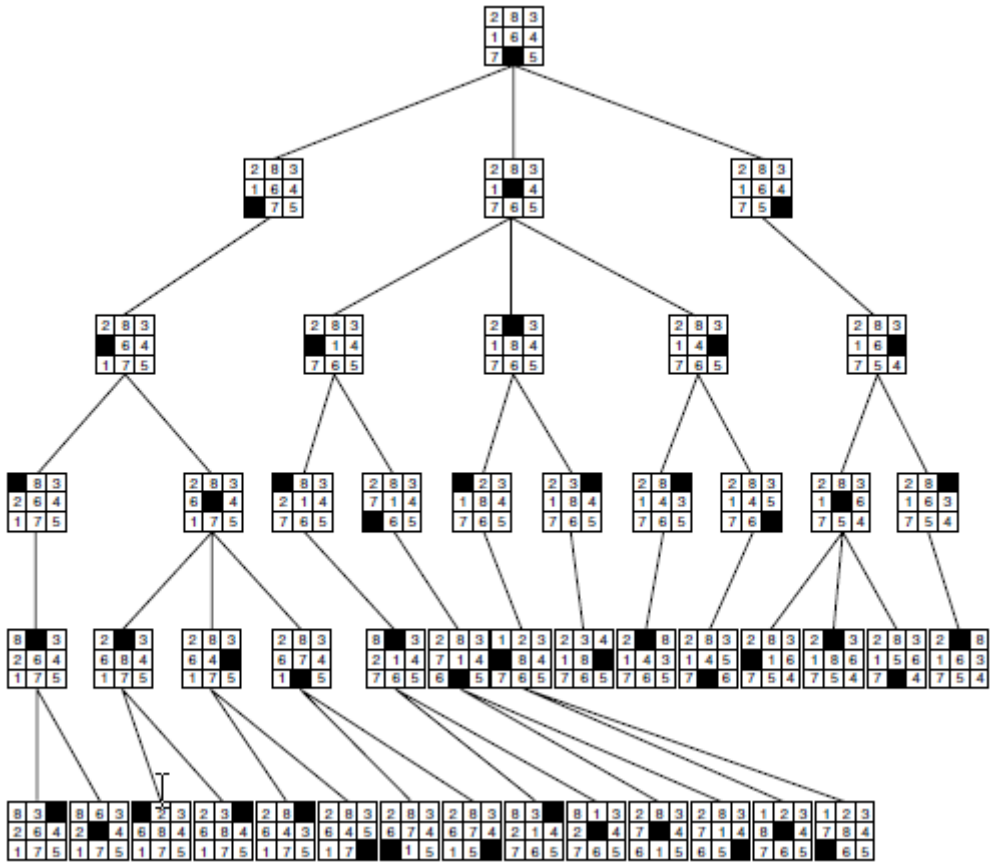
System kontroli:

1. Wypróbuj każdą produkcję w kolejności.

2. Nie zezwalaj na pętle.

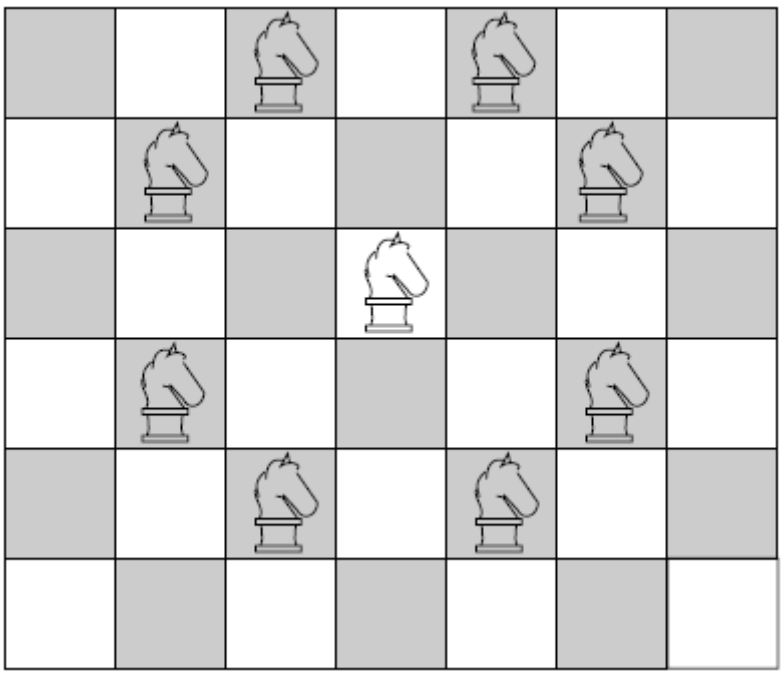
3. Zatrzymaj się, gdy cel zostanie znaleziony

Oczywiście wszystkie cztery te ruchy mają zastosowanie tylko wtedy, gdy puste miejsce znajduje się na środku; gdy jest w jednym z rogów, możliwe są tylko dwa ruchy. Jeśli określono teraz stan początkowy i stan celu dla 8 puzzli, możliwe jest, aby system produkcyjny rozliczał przestrzeń wyszukiwania problemu. Rzeczywista implementacja tego problemu może reprezentować każdą konfigurację planszy z predykatem „stanowym” z dziewięcioma parametrami (dla dziewięciu możliwych lokalizacji ośmiu płytek i pustego pola); reguły można zapisać jako implikacje, których założenie wykonuje wymaganą kontrolę stanu. Alternatywnie, dla stanów tablic można zastosować tablice lub struktury list. Przykład zaczerpnięty z Nilssona (1980) dotyczący przestrzeni poszukiwanej w celu znalezienia rozwiązania problemu podanego na rysunku znajduje się na rysunku poniżej. Ponieważ ta ścieżka rozwiązania może być bardzo głęboka, jeśli nie jest ograniczona, do wyszukiwania dodano granicę głębokości. (Prostym sposobem na dodanie granicy głębokości jest śledzenie długości / głębokości bieżącej ścieżki i wymuszenie cofania, jeśli granica ta zostanie przekroczona.) W rozwiązaniu zastosowano granicę głębokości 5 rysunku poniej. Zauważ, że liczba możliwych stanów pamięci roboczej rośnie wykładniczo wraz z głębokością wyszukiwania.



PRZYKŁAD: PROBLEM Z WYCIECZKĄ Rycerza

W grze w szachy rycerz może poruszać się o dwa pola w poziomie lub w pionie, a następnie o jedno pole w kierunku prostopadłym, o ile nie zejdzie z planszy. Istnieje więc co najwyżej osiem możliwych ruchów, które rycerz może wykonać.



Zgodnie z tradycyjną definicją problem trasy rycerza polega na znalezieniu serii legalnych ruchów, w których rycerz ląduje dokładnie na każdym polu szachownicy. Problem ten stanowił podstawę opracowania i prezentacji algorytmów wyszukiwania. Przykład, którego używamy w tym rozdziale, to uproszczona wersja problemu rycerza. Pyta, czy istnieje seria legalnych ruchów, które przeniosą rycerza z jednego pola na drugie na szachownicy o zmniejszonych rozmiarach (3 × 3). Rysunek 6.6 pokazuje szachownicę 3 × 3 z każdym kwadratem oznaczonym liczbami całkowitymi od 1 do 9. Ten schemat znakowania jest stosowany zamiast bardziej ogólnego podejścia do nadawania każdej spacji numeru wiersza i kolumny w celu dalszego uproszczenia przykładu. Z powodu zmniejszonej wielkości problemu po prostu wyliczamy ruchy alternatywne, zamiast opracowywać ogólny operator ruchów. Legalne ruchy na planszy są następnie opisywane w rachunku predykatów przy użyciu predykatu zwanego ruchem, którego parametrami są początek i koniec kwadratu legalnego ruchu. Na przykład ruch (1,8) przenosi rycerza z lewego górnego rogu na środek dolnego rzędu. Predykaty na rysunku poniżej wyliczają wszystkie możliwe ruchy szachownicy 3 × 3.

move(1,8) move(6,1)
move(1,6) move(6,7)
move(2,9) move(7,2)
move(2,7) move(7,6)
move(3,4) move(8,3)
move(3,8) move(8,1)
move(4,9) move(9,2)
move(4,3) move(9,4)

1	2	3
4	5	6
7	8	9

Problem trasy rycerskiej 3 × 3 można rozwiązać za pomocą systemu produkcyjnego. Każdy ruch może być reprezentowany jako reguła, której warunkiem jest lokalizacja rycerza na danym polu i którego działanie przenosi rycerza na inne pole. Przedstawiono szesnaście produkcji poniżej przedstawiają wszystkie możliwe ruchy rycerza.

ZASADA# WARUNEK DZIAŁANIE

1 rycerz na kwadracie 1 → przenieś rycerza na kwadrat 8

2 rycerz na kwadracie 1 → przenieś rycerza na kwadrat 6

3 rycerz na kwadracie 2 → przenieś rycerza na kwadrat 9

4 rycerz na kwadracie 2 → przenieś rycerza na kwadrat 7

5 rycerz na kwadracie 3 → przenieś rycerza na kwadrat 4

6 rycerzy na kwadracie 3 → przenieś rycerza na kwadrat 8

7 rycerz na kwadracie 4 → przenieś rycerza na kwadrat 9

8 rycerzy na kwadracie 4 → przenieś rycerza na kwadrat 3

9 rycerz na kwadracie 6 → przenieś rycerza na kwadrat 1

10 rycerzy na polu 6 → przenieś rycerza na pole 7

11 rycerz na kwadracie 7 → przenieś rycerza na kwadrat 2

12 rycerz na kwadracie 7 → przenieś rycerza na kwadrat 6

13 rycerz na kwadracie 8 → przenieś rycerza na kwadrat 3

14 rycerz na kwadracie 8 → przenieś rycerza na kwadrat 1

15 rycerz na kwadracie 9 → przenieś rycerza na kwadrat 2

16 rycerz na kwadracie 9 → przenieś rycerza na kwadrat 4

Następnie określamy procedurę rekurencyjną w celu zaimplementowania algorytmu sterowania dla systemu produkcyjnego. Ponieważ ścieżka (X, X) ujednolici się tylko z predykatami, takimi jak ścieżka (3,3) lub ścieżka (5,5), określa pożądany warunek zakończenia. Jeśli ścieżka (X, X) nie powiedzie się, sprawdzamy reguły produkcji dla możliwego następnego stanu, a następnie powtarzamy się. Ogólna definicja ścieżki rekurencyjnej jest następnie podawana w dwóch predykatowych formułach rachunku różniczkowego:

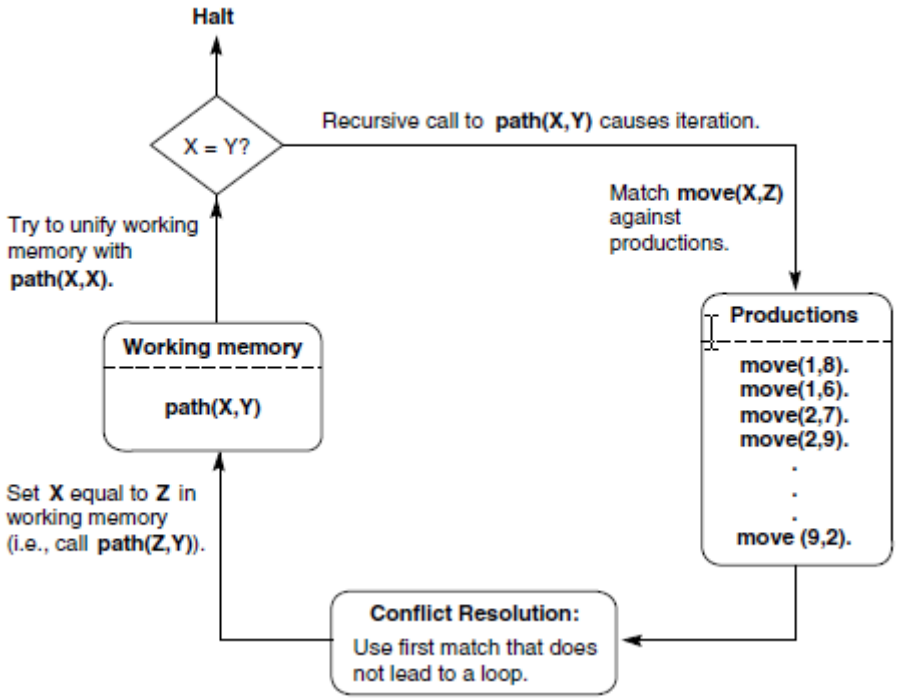
Path Ścieżka X (X, X)

∀ X, Y [ścieżka (X, Y) ← ∃ Z [ruch (X, Z) ∧ ścieżka (Z, Y)

Pamięć robocza, parametry predykatu ścieżki rekurencyjnej, zawiera zarówno bieżący stan płytki, jak i stan celu. System kontroli stosuje reguły, dopóki obecny stan nie zrówna się z celem, a następnie się zatrzyma. Prosty schemat rozwiązywania konfliktów uruchomiłby pierwszą regułę, która nie spowodowała zapętlenia wyszukiwania. Ponieważ wyszukiwanie może prowadzić do ślepych zaułków (z których każdy możliwy ruch prowadzi do poprzednio odwiedzonego stanu, a tym samym pętli), reżim kontroli musi również umożliwiać powrót do poprzedniego stanu; wykonanie tego systemu produkcyjnego, który określa, czy istnieje ścieżka od kwadratu 1 do kwadratu 2, pokazano na rysunku

Iteration #	Working memory		Conflict set (rule #'s)	Fire rule
	Current square	Goal square		
0	1	2	1, 2	1
1	8	2	13, 14	13
2	3	2	5, 6	5
3	4	2	7, 8	7
4	9	2	15, 16	15
5	2	2		Halt

Ta charakterystyka definicji ścieżki jako systemu produkcyjnego jest podana na rysunku .



Systemy produkcyjne są w stanie generować nieskończone pętle podczas przeszukiwania wykresu przestrzeni stanów. Pętle te są szczególnie trudne do wykrycia w systemie produkcyjnym, ponieważ reguły mogą być uruchamiane w dowolnej kolejności. Oznacza to, że podczas wykonywania systemu może pojawić się zapętlenie, ale nie można go łatwo znaleźć na podstawie kontroli składni zestawu reguł. Na przykład przy regułach „ruchu" problemu trasy rycerza uporządkowanych jak w tabeli 1 i strategii rozwiązywania konfliktu wyboru pierwszego dopasowania, ruch wzorca (2, X) pasowałby do ruchu (2,9), wskazując przejdź do kwadratu 9. Podczas następnej iteracji ruch wzoru (9, X) będzie pasował do ruchu (9,2), przenosząc wyszukiwanie z powrotem na kwadrat 2, powodując pętlę. Aby zapobiec zapętleniu, wzorzec_wyszukiwania sprawdził globalną listę (zamkniętych) odwiedzonych stanów. Rzeczywista strategia rozwiązywania konfliktów była zatem: wybierz pierwszy pasujący ruch, który prowadzi do niezapowiedzianego stanu. W systemie produkcyjnym właściwym miejscem do rejestrowania danych specyficznych dla przypadku, takich jak lista wcześniej odwiedzonych stanów, nie jest globalna zamknięta lista, ale sama pamięć robocza. assert służy do umieszczania „znacznika" w pamięci roboczej, aby wskazać, kiedy stan został odwiedzony. Ten znacznik jest reprezentowany jako jednoargumentowy predykat, był (X), który jako argument przyjmuje kwadrat na planszy. been (X) jest dodawany do pamięci roboczej, gdy nowy stan X jest odwiedzany. Rozwiązanie konfliktu może wtedy wymagać, aby wcześniej (Z) nie mógł znajdować się w pamięci roboczej, zanim ruch (X, Z) będzie mógł zostać uruchomiony. W przypadku określonej wartości Z można to sprawdzić, dopasowując wzorzec do pamięci roboczej. Zmodyfikowany kontroler ścieżki rekurencyjnej dla systemu produkcyjnego to:

Path Ścieżka X (X, X)

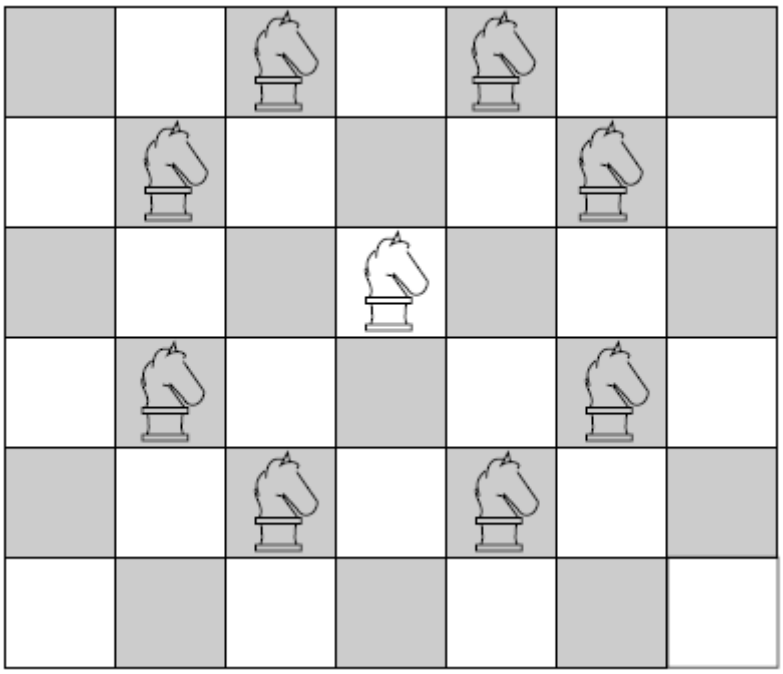
∀ X, Y [ścieżka (X, Y) ← ∃ Z [ruch (X, Z) ∧ ¬ (były (Z)) ∧ potwierdzają (były (Z)) ∧ ścieżka (Z, Y)]]

W tej definicji ruch (X, Z) kończy się pierwszym meczem z predykatem ruchu. Wiąże to wartość z Z. Jeśli wcześniej (Z) pasuje do wpisu w pamięci roboczej, ¬ (był (Z)) spowoduje awarię (tj. Będzie fałszem). wzorzec_wyszukiwania następnie cofa się i próbuje innego dopasowania dla ruchu (X, Z). Jeśli kwadratowy Z jest nowym stanem, wyszukiwanie będzie kontynuowane, z uwzględnieniem (Z) w pamięci roboczej, aby zapobiec przyszłym pętlom. Faktyczne uruchomienie produkcji ma miejsce, gdy

algorytm ścieżki się powtarza. Zatem obecność wcześniejszych predykatów w pamięci roboczej implementuje wykrywanie pętli w tym systemie produkcyjnym.

PRZYKŁAD [6.2.3]: WYCIECZKA PEŁNEGO RYCERZA

Możemy uogólnić rozwiązanie wycieczki rycerza do pełnej szachownicy 8 × 8. Ponieważ nie ma sensu wyliczanie ruchów dla tak złożonego problemu, zastępujemy 16 faktów ruchów zestawem 8 reguł, aby wygenerować legalne ruchy rycerza. Te ruchy (produkcje) odpowiadają 8 możliwym sposobom ruchu rycerza.



Jeśli zindeksujemy szachownicę według numerów wierszy i kolumn, możemy zdefiniować regułę produkcji dla przesunięcia rycerza w dół o dwa pola i prawo o jedno pole:

WARUNEK: bieżący wiersz 6 ∧ bieżąca kolumna 7

DZIAŁANIE: nowy wiersz = bieżący wiersz + 2 ∧ nowa kolumna = bieżąca kolumna + 1

Jeśli użyjemy rachunku predykatu do przedstawienia produkcji, kwadrat kwadratu może być zdefiniowany przez kwadrat predykatu (R, C), reprezentujący rząd R i kolumnę C. Powyższą zasadę można przepisać w rachunku predykatów jako:

przesuń (kwadrat (wiersz, kolumna), kwadrat (Newrow, Newcolumn)) ←

less_than_or_equals (Row, 6) ∧

równa się (Newrow, plus (Row, 2)) ∧

less_than_or_equals (kolumna, 7) ∧

równa się (Newcolumn, plus (Column, 1))

plus to funkcja dodawania; less_than_or_equals i equals mają oczywiste interpretacje arytmetyczne. Można zaprojektować siedem dodatkowych reguł, które obliczą pozostałe możliwe ruchy. Te zasady zastępują fakty dotyczące przenoszenia w wersji problemu 3 × 3. Definicja ścieżki z przykładu rycerza 3 × 3 oferuje również pętlę sterowania dla tego problemu. Jak widzieliśmy, kiedy opisy rachunku

predykatu są interpretowane proceduralnie, na przykład za pomocą algorytmu wzorzec_wyszukiwania, subtelne zmiany są wprowadzane do semantyki rachunku predykatu. Jedną z takich zmian jest sekwencyjny sposób rozwiązywania celów. Narzuca to uporządkowanie lub semantykę proceduralną interpretacji wyrażeń rachunku predykatu. Kolejną zmianą jest wprowadzenie predykatów meta-logicznych, takich jak aser, które wskazują działania wykraczające poza interpretację wartości prawdowych wyrażeń rachunku predykatu.

PRZYKŁAD: DORADCA FINANSOWY JAKO SYSTEM PRODUKCJI

Wcześniej opracowaliśmy małego doradcę finansowego, używając rachunku predykatów do reprezentowania wiedzy finansowej i wyszukiwania grafów w celu dokonania odpowiednich wniosków podczas konsultacji. System produkcyjny stanowi naturalny nośnik do jego wdrożenia. Implikacje logicznego opisu tworzą produkcje. Informacje dotyczące konkretnego przypadku (wynagrodzenie danej osoby, osoby pozostające na utrzymaniu itp.) Są ładowane do pamięci roboczej. Reguły są włączane, gdy ich przesłanki są spełnione. Reguła jest wybierana z tego zestawu konfliktów i uruchamiana, dodając swoje zakończenie do pamięci roboczej. Trwa to do momentu dodania wszystkich możliwych wniosków najwyższego poziomu do pamięci roboczej. Rzeczywiście, wiele „powłok" systemu eksperckiego, w tym JESS i CLIPS, to systemy produkcyjne z dodatkowymi funkcjami do obsługi interfejsu użytkownika, obsługi niepewności w uzasadnieniu, edycji wiedzy wykonanie bazy i śledzenia

[6.2.3]

Kontrola wyszukiwania w systemach produkcyjnych

Model systemu produkcyjnego oferuje szereg możliwości dodania kontroli heurystycznej do algorytmu wyszukiwania. Należą do nich wybór strategii wyszukiwania opartych na danych lub celu, struktura samych reguł oraz wybór różnych opcji rozwiązywania konfliktów.

Kontrola poprzez wybór strategii wyszukiwania opartej na danych lub celu Wyszukiwanie oparte na danych rozpoczyna się od opisu problemu (takiego jak zbiór logicznych aksjomatów, objawów choroby lub zbioru danych wymagających interpretacji) i wyciąga z nich nową wiedzę. Odbywa się to poprzez zastosowanie reguł wnioskowania, legalnych ruchów w grze lub innych operacji generujących stan do bieżącego opisu świata i dodanie wyników do opisu problemu. Proces ten trwa do momentu osiągnięcia celu. Ten opis wnioskowania opartego na danych podkreśla jego ścisłe dopasowanie do modelu obliczeniowego systemu produkcyjnego. „Bieżący stan świata" (dane, które albo uznano za prawdziwe, albo wydedukowano jako prawdziwe przy wcześniejszym użyciu reguł produkcji) są umieszczane w pamięci roboczej. Cykl rozpoznawania-działania dopasowuje następnie aktualny stan do (uporządkowanego) zestawu produkcji. Kiedy te dane pasują (są zunifikowane) do warunków jednej z reguł produkcji, działanie produkcji dodaje (modyfikując pamięć roboczą) nową informację do aktualnego stanu świata. Wszystkie produkcje mają postać WARUNEK → DZIAŁANIE. Kiedy WARUNEK pasuje do niektórych elementów pamięci roboczej, wykonywana jest AKCJA. Jeśli reguły produkcji są sformułowane jako logiczne implikacje, a AKCJA dodaje twierdzenia do pamięci roboczej, wówczas odpalenie reguły może odpowiadać zastosowaniu modus ponens reguły reguły wnioskowania. To tworzy nowy stan wykresu. Rycina 6.9 przedstawia proste wyszukiwanie oparte na danych na zestawie produkcji wyrażonych jako implikacje rachunku zdań.

Production set:

1. p ∧ q → goal
2. r ∧ s → p
3. w ∧ r → q
4. t ∧ u → q
5. v → s
6. start → v ∧ r ∧ q

Trace of execution:

Iteration #	Working memory	Conflict set	Rule fired
0	start	6	6
1	start, v, r, q	6, 5	5
2	start, v, r, q, s	6, 5, 2	2
3	start, v, r, q, s, p	6, 5, 2, 1	1
4	start, v, r, q, s, p, goal	6, 5, 2, 1	halt

Space searched by execution:

Strategia rozwiązywania konfliktów polega na wybraniu włączonej reguły, która została uruchomiona co najmniej niedawno (lub wcale); w przypadku powiązań wybierana jest pierwsza zasada. Realizacja zostaje zatrzymana, gdy cel zostanie osiągnięty. Na rysunku przedstawiono również sekwencję odpalenia reguł i etapy pamięci roboczej w wykonaniu wraz z wykresem przeszukiwanej przestrzeni. Do tego momentu potraktowaliśmy systemy produkcyjne w sposób oparty na danych; mogą jednak być również wykorzystywane do wyszukiwania ukierunkowanego na cel. Jak zdefiniowano wcześniej, wyszukiwanie goaldriven rozpoczyna się od celu i działa wstecz do faktów problemu, aby go osiągnąć. Aby zaimplementować to w systemie produkcyjnym, cel jest umieszczany w pamięci roboczej i dopasowywany do AKCJI reguł produkcji. Te AKCJE są dopasowane (na przykład przez ujednolicenie), podobnie jak WARUNKI produkcji zostały dopasowane w rozumowaniu opartym na danych. Wszystkie reguły produkcji, których wnioski (AKCJE) są zgodne z celem, stanowią zbiór konfliktów. Gdy AKCJA reguły jest dopasowana, WARUNKI są dodawane do pamięci roboczej i stają się nowymi podzbiorami (stanami) wyszukiwania. Nowe stany są następnie dopasowywane do AKCJI innych reguł produkcji. Proces ten trwa do momentu znalezienia faktów, zwykle w początkowym opisie problemu lub, jak to często bywa w systemach eksperckich, poprzez bezpośrednie poproszenie użytkownika o określone informacje. Wyszukiwanie kończy się, gdy okaże się, że WARUNKI obsługujące produkcje wystrzelone w ten sposób wstecz, które prowadzą do celu, są prawdziwe. WARUNKI i łańcuch ostrzeżeń prowadzących do celu stanowią dowód jego prawdy poprzez kolejne wnioski, takie jak modus ponens. Zobacz rysunek, gdzie znajduje się przykład wnioskowania ukierunkowanego na cel na tym samym zestawie produkcji, co na rycinie powyżej.

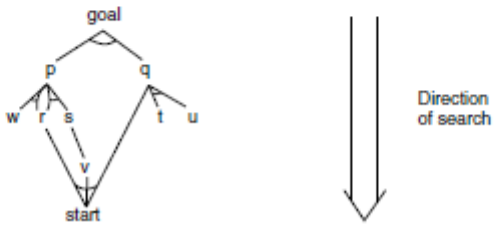
Production set:

1. p ∧ q → goal
2. r ∧ s → p
3. w ∧ r → p
4. t ∧ u → q
5. v → s
6. start → v ∧ r ∧ q

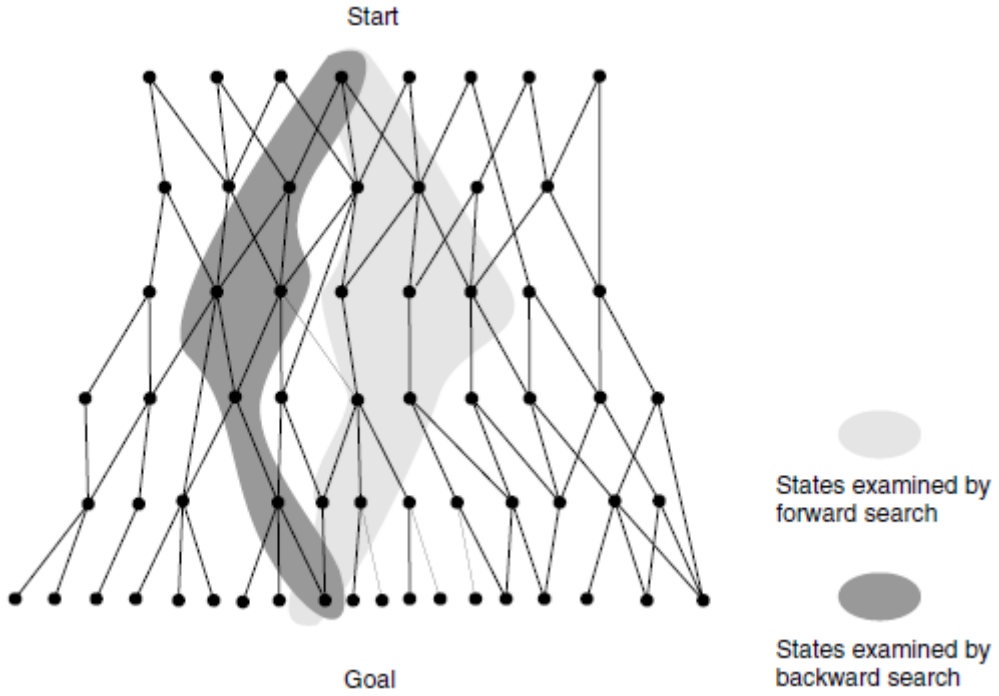
Trace of execution:

Iteration #	Working memory	Conflict set	Rule fired
0	goal	1	1
1	goal, p, q	1, 2, 3, 4	2
2	goal, p, q, r, s	1, 2, 3, 4, 5	3
3	goal, p, q, r, s, w	1, 2, 3, 4, 5	4
4	goal, p, q, r, s, w, t, u	1, 2, 3, 4, 5	5
5	goal, p, q, r, s, w, t, u, v	1, 2, 3, 4, 5, 6	6
6	goal, p, q, r, s, w, t, u, v, start	1, 2, 3, 4, 5, 6	halt

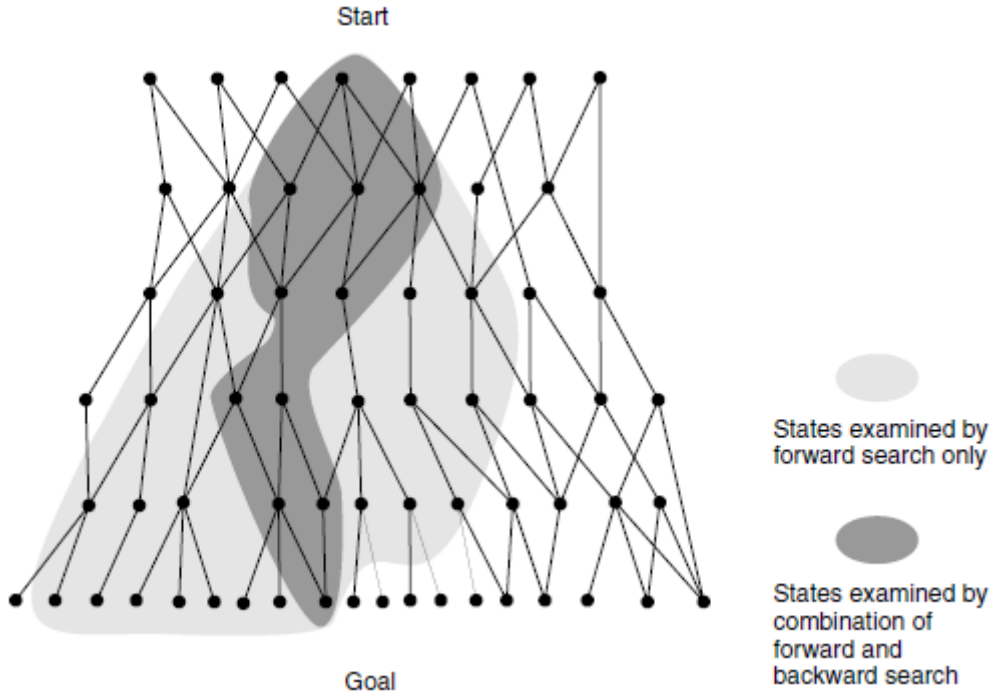
Space searched by execution:



Pamiętaj, że wyszukiwanie zorientowane na cel uruchamia inną serię produkcji i szuka innej przestrzeni niż wersja oparta na danych. Jak pokazuje ta dyskusja, system produkcyjny oferuje naturalną charakterystykę wyszukiwania ukierunkowanego na cel i danych. Reguły produkcji to zakodowany zestaw wniosków („wiedza” w systemie eksperckim opartym na regułach) do zmiany stanu na wykresie. Kiedy aktualny stan świata (zbiór prawdziwych stwierdzeń opisujących świat) odpowiada WARUNKOM reguł produkcji i to dopasowanie powoduje, że część AKCJA reguły tworzy kolejny (prawdziwy) deskryptor dla świata, jest to określane jako wyszukiwanie oparte na danych. Alternatywnie, gdy cel jest dopasowany do części AKCJA reguł w zestawie reguł produkcji, a ich WARUNKI są następnie ustawiane jako podzadania, które mają być „prawdziwe” (przez dopasowanie AKCJI reguł w następnym cyklu system produkcji), wynikiem jest ukierunkowane na rozwiązywanie problemów. Ponieważ zestaw reguł może być wykonywany w sposób oparty na danych lub celu, możemy porównywać i kontrastować skuteczność każdego podejścia w kontrolowaniu wyszukiwania. Złożoność poszukiwania którejkolwiek ze strategii jest mierzona takimi pojęciami, jak czynnik rozgałęziający lub penetracja. Te miary złożoności wyszukiwania mogą zapewnić oszacowanie kosztów zarówno dla wersji rozwiązania problemów, jak i danych, a tym samym pomóc w wyborze najbardziej skutecznej strategii. Możemy również stosować kombinacje strategii. Na przykład możemy wyszukiwać w kierunku do przodu, aż liczba stanów stanie się duża, a następnie przełączyć się na wyszukiwanie ukierunkowane na cel, aby użyć możliwych podzadań do wyboru spośród alternatywnych stanów. Niebezpieczeństwo w tej sytuacji polega na tym, że w przypadku wyszukiwania heurystycznego lub najlepszego wyszukiwania (rozdział 4) części faktycznie przeszukiwanych wykresów mogą się wzajemnie „ominąć” i ostatecznie wymagać więcej wyszukiwania niż prostszego podejścia, jak pokazano na rysunku



Jednak gdy rozgałęzienie przestrzeni jest stałe i stosowane jest wyczerpujące wyszukiwanie, łączona strategia wyszukiwania może drastycznie zmniejszyć ilość przeszukiwanego miejsca, jak pokazano na rysunku.



Kontrola wyszukiwania poprzez strukturę reguł

Struktura reguł w systemie produkcyjnym, w tym rozróżnienie między warunkiem a działaniem oraz kolejnością, w jakiej warunki są wypróbowywane, określa sposób przeszukiwania przestrzeni.

Wprowadzając rachunek predykatów jako język reprezentacyjny, podkreśliliśmy deklaratywny charakter jego semantyki. Oznacza to, że wyrażenia rachunku predykatu po prostu definiują prawdziwe relacje w dziedzinie problemów i nie twierdzą o ich kolejności interpretacji. Zatem indywidualną regułą może być ∀ X (foo (X) ∧ goo (X) → moo (X)). Zgodnie z regułami rachunku predykatów alternatywną formą tej samej reguły jest ∀ X (foo (X) → moo (X) ∨ ¬ goo (X)). Równoważność tych dwóch klauzul pozostawia się jako ćwiczenie. Chociaż te formuły są logicznie równoważne, nie prowadzą do takich samych rezultatów, gdy są interpretowane jako produkcje, ponieważ system produkcyjny narzuca porządek w dopasowywaniu i uruchamianiu reguł. Tak więc określona forma reguł określa łatwość (lub możliwość) dopasowania reguły do wystąpienia problemu. Jest to wynikiem różnic w sposobie, w jaki system produkcyjny interpretuje reguły. System produkcji narzuca semantykę proceduralną na język deklaratywny używany do tworzenia reguł. Ponieważ system produkcyjny wypróbowuje każdą z reguł w określonej kolejności, programista może kontrolować wyszukiwanie w strukturze i kolejności reguł w zestawie produkcyjnym. Chociaż logicznie równoważne, ∀ X (foo (X) ∧ goo (X) → moo (X)) i ∀ X (foo (X) → moo (X) ∨ ¬ goo (X)) nie zachowują się tak samo wdrożenie wyszukiwania. Ludzcy eksperci kodują kluczowe heurystyki w ramach swoich zasad ekspertyzy. Kolejność przesłania koduje często krytyczne informacje proceduralne w celu rozwiązania problemu. Ważne jest, aby zachować tę formę przy tworzeniu programu, który „rozwiązuje problemy jak ekspert". Kiedy mechanik mówi: „Jeśli silnik się nie przekręci, a światła się nie zapalą, sprawdź akumulator", sugeruje określoną sekwencję działań. Informacje te nie są przechwytywane przez logicznie równoważne stwierdzenie „silnik się obraca lub zapala się lampka lub sprawdź akumulator". Ta forma reguł ma kluczowe znaczenie w kontrolowaniu wyszukiwania, dzięki czemu system zachowuje się logicznie, a porządek uruchamiania reguł jest bardziej zrozumiały.

Kontrola wyszukiwania poprzez rozwiązywanie konfliktów

Chociaż systemy produkcyjne (jak wszystkie architektury systemów opartych na wiedzy) pozwalają na zakodowanie heurystyki w treści wiedzy o samych regułach, oferują inne możliwości kontroli heurystycznej poprzez rozwiązywanie konfliktów. Chociaż najprostszą taką strategią jest wybranie pierwszej reguły pasującej do zawartości pamięci roboczej, każda strategia może potencjalnie zostać zastosowana do rozwiązywania konfliktów. Na przykład strategie rozwiązywania konfliktów obsługiwane przez OPS5 (Brownston i in. 1985) obejmują:

1. Refrakcja. Załamanie określa, że po uruchomieniu reguły reguła może nie zostać ponownie uruchomiona, dopóki elementy pamięci roboczej odpowiadające jej warunkom nie zostaną zmodyfikowane. To zniechęca do zapętlania.

2. Niedawność. Strategia aktualności preferuje reguły, których warunki są zgodne z wzorcami ostatnio dodanymi do pamięci roboczej. Skupia to wyszukiwanie na jednej linii rozumowania.

3. Specyfika. Strategia ta zakłada, że należy zastosować bardziej szczegółową zasadę rozwiązywania problemów, a nie bardziej ogólną. Jedna reguła jest bardziej szczegółowa niż inna, jeśli ma więcej warunków, co oznacza, że będzie pasować do mniejszej liczby wzorców pamięci roboczej.

Zalety systemów produkcyjnych dla AI

Jak pokazano w poprzednich przykładach, system produkcyjny oferuje ogólne ramy wdrażania wyszukiwania. Ze względu na swoją prostotę, modyfikowalność i elastyczność w stosowaniu wiedzy na temat rozwiązywania problemów, system produkcji okazał się ważnym narzędzie do budowy systemów eksperckich i innych aplikacji AI. Główne zalety systemów produkcji sztucznej inteligencji obejmują:

Rozdział wiedzy i kontroli. System produkcyjny to elegancki model rozdziału wiedzy i kontroli w programie komputerowym. Kontrolę zapewnia cykl rozpoznawania-działania w pętli systemu produkcyjnego, a wiedza na temat rozwiązywania problemów jest zakodowana w samych regułach. Zalety tego rozdzielenia obejmują łatwość modyfikacji bazy wiedzy bez konieczności zmiany kodu do sterowania programem i, przeciwnie, możliwość zmiany kodu do sterowania programem bez zmiany zestawu reguł produkcji.

Naturalne mapowanie na wyszukiwanie w przestrzeni stanu. Komponenty systemu produkcyjnego mapują się naturalnie w konstrukcje przeszukiwania przestrzeni stanów. Kolejne stany pamięci roboczej tworzą węzły wykresu przestrzeni stanów. Reguły produkcji to zbiór możliwych przejść między stanami, przy czym rozwiązywanie konfliktów implementuje wybór gałęzi w przestrzeni stanu. Reguły te upraszczają implementację, debugowanie i dokumentację algorytmów wyszukiwania.

Modułowość reguł produkcji. Ważnym aspektem modelu systemu produkcyjnego jest brak jakichkolwiek interakcji składniowych między regułami produkcji. Reguły mogą wpływać na odpalanie innych reguł tylko poprzez zmianę wzorca w pamięci roboczej; nie mogą „dzwonić" kolejną regułą bezpośrednio, tak jakby to była podprogram innych reguł produkcji. Zakres zmiennych tych reguł jest ograniczony do jednostki reguły. Ta niezależność składniowa wspiera stopniowy rozwój systemów eksperckich poprzez sukcesywne dodawanie, usuwanie lub zmienianie wiedzy (zasad) systemu.

Sterowanie według wzorca. Problemy rozwiązywane przez programy AI często wymagają szczególnej elastyczności w wykonywaniu programu. Celowi temu służy fakt, że reguły w systemie produkcyjnym mogą strzelać w dowolnej kolejności. Opisy problemu, które składają się na bieżący stan świata, determinują zestaw konfliktów, aw konsekwencji konkretną ścieżkę wyszukiwania i rozwiązanie, ani nie mogą ustawiać wartości zmiennych w innych regułach produkcji. Zakres zmiennych tych reguł jest ograniczony do reguły indywidualnej. Ta niezależność składniowa wspiera stopniowy rozwój systemów eksperckich poprzez sukcesywne dodawanie, usuwanie lub zmienianie wiedzy (zasad) systemu.

Sterowanie według wzorca. Problemy rozwiązywane przez programy AI często wymagają szczególnej elastyczności w wykonywaniu programu. Celowi temu służy fakt, że reguły w systemie produkcyjnym mogą strzelać w dowolnej kolejności. Opisy problemu, które składają się na bieżący stan świata, determinują zestaw konfliktów, aw konsekwencji konkretną ścieżkę wyszukiwania i rozwiązanie

Możliwości heurystycznej kontroli wyszukiwania. (W poprzedniej części opisano kilka technik kontroli heurystycznej).

Śledzenie i objaśnienie. Modułowość reguł i iteracyjny charakter ich wykonywania ułatwiają śledzenie wykonania systemu produkcyjnego. Na każdym etapie cyklu rozpoznawania czynności można wyświetlić wybraną regułę. Ponieważ każda reguła odpowiada pojedynczemu „fragmentowi" wiedzy związanej z rozwiązywaniem problemów, treść reguły może zapewnić sensowne wyjaśnienie bieżącego stanu i działania systemu. Ponadto łańcuch reguł zastosowanych w procesie rozwiązania odzwierciedla zarówno ścieżkę na wykresie, jak i „linię rozumowania" eksperta. Natomiast pojedynczy wiersz kodu lub procedury w tradycyjny język aplikacji, taki jak Pascal, FORTRAN lub Java, jest praktycznie bez znaczenia.

Niezależność językowa. Model sterowania systemem produkcyjnym jest niezależny od reprezentacji wybranej dla reguł i pamięci roboczej, o ile reprezentacja ta obsługuje dopasowanie wzorca. Opisaliśmy reguły produkcji jako implikacje rachunku predykatu postaci $A \Rightarrow B$, gdzie prawda A i modus ponens reguły reguły wnioskowania pozwalają nam dojść do wniosku B. Chociaż istnieje wiele korzyści z używania logiki jako podstawy reprezentacji wiedzy i źródła reguł wnioskowania dźwiękowego, produkcja model systemu może być używany z innymi reprezentacjami, np. CLIPS i JESS.

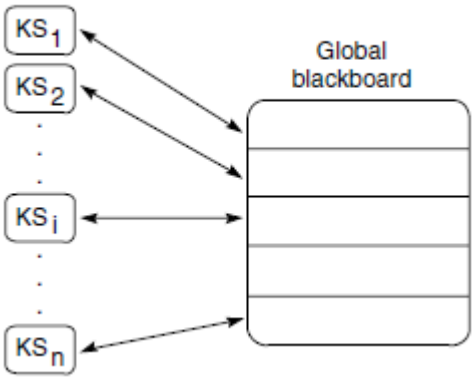
Chociaż rachunek predykatów oferuje przewagę logicznego wnioskowania, wiele problemów wymaga rozumowania, które nie jest logiczne. Zamiast tego dotyczą rozumowania probabilistycznego, użycia niepewnych dowodów i domyślnych założeń. Późniejsze rozdziały (7, 8 i 9) omawiają alternatywne reguły wnioskowania, które zapewniają te możliwości. Niezależnie od rodzaju zastosowanych reguł wnioskowania system produkcyjny zapewnia pojazd do przeszukiwania przestrzeni stanu

Możliwy model rozwiązywania problemów u ludzi. Modelowanie rozwiązywania problemów ludzkich było jednym z pierwszych zastosowań systemów produkcyjnych, patrz Newell i Simon (1972). Są one nadal stosowane jako model wydajności człowieka w wielu obszarach badań kognitywistyki (rozdział 16). Wyszukiwanie ukierunkowane na wzorzec daje nam możliwość zbadania przestrzeni wnioskowania logicznego w rachunku predykatów. Wiele problemów opiera się na tej technice, wykorzystując rachunek predykatów do modelowania określonych aspektów świata, takich jak czas i zmiana. W następnej sekcji systemy tablicowe są przedstawione jako odmiana metodologii systemu produkcyjnego, w której określone reguły zadań produkcji są łączone w źródła wiedzy i współpracują przy rozwiązywaniu problemów poprzez komunikację poprzez globalną pamięć roboczą lub tablicę.

Architektura tablicy do rozwiązywania problemów

Tablica jest ostatnim mechanizmem kontrolnym przedstawionym w tym rozdziale. Kiedy chcemy badać stany w przestrzeni wnioskowania w bardzo deterministyczny sposób, systemy produkcyjne zapewniają dużą elastyczność, umożliwiając nam reprezentowanie wielu częściowych rozwiązań jednocześnie w pamięci roboczej i wybieranie następnego stanu poprzez rozwiązywanie konfliktów. Tablice rozszerzają systemy produkcyjne, pozwalając nam organizować pamięć produkcyjną w osobne moduły, z których każdy odpowiada innemu podzbiorowi produkcji . Tablice integrują te oddzielne zestawy reguł produkcji i koordynują działania wielu agentów rozwiązywania problemów, zwanych czasami źródłami wiedzy, w ramach jednej globalnej struktury - tablicy. Wiele problemów wymaga koordynacji wielu różnych rodzajów agentów. Na przykład, program do rozumienia mowy może najpierw zmanipulować wypowiedź przedstawioną w postaci cyfrowej fali. W trakcie procesu dekodowania musi znaleźć słowa w tej wypowiedzi, uformować je w frazy i zdania, a na końcu stworzyć semantyczną reprezentację znaczenia wypowiedzi.

Powiązany problem występuje, gdy wiele procesów musi współpracować, aby rozwiązać jeden problem. Przykładem tego jest problem zespolenia czujnika. Załóżmy, że mamy sieć czujników, z których każdy jest monitorowany przez osobny proces. Załóżmy również, że procesy mogą się komunikować i że właściwa interpretacja danych każdego czujnika zależy od danych otrzymanych przez inne czujniki w sieci. Problem ten powstaje w sytuacjach tak różnorodnych, jak śledzenie samolotów w wielu lokalizacjach radarowych po łączenie odczytów wielu czujników w procesie produkcyjnym. Architektura tablicowa jest modelem kontroli zastosowanym do tych i innych problemów wymagających koordynacji wielu procesów lub źródeł wiedzy. Tablica jest centralną globalną bazą danych do komunikacji z niezależnymi źródłami wiedzy asynchronicznej, koncentrującymi się na powiązanych aspektach konkretnego problemu. Rysunek 6.13 przedstawia schemat projektu tablicy.



Na rysunku każde źródło wiedzy KS_i pobiera dane z tablicy, przetwarza dane i zwraca wyniki do tablicy, aby mogły zostać wykorzystane przez inne źródła wiedzy. Każdy KSi jest niezależny, ponieważ jest oddzielnym procesem działającym zgodnie z nim własne specyfikacje, a gdy stosowany jest system wieloprocesorowy lub wieloprocesorowy, jest on niezależny od innych procesów przetwarzania w problemie. Jest to system asynchroniczny, w którym każdy KSi rozpoczyna działanie, gdy tylko znajdzie odpowiednie dane wejściowe umieszczone na tablicy. Po zakończeniu przetwarzania publikuje wyniki na tablicy i oczekuje na nowe odpowiednie dane wejściowe. Podejście tablicowe do organizowania dużego programu zostało po raz pierwszy przedstawione w badaniu HEARSAY-II. HEARSAY-II był programem do rozumienia mowy; został pierwotnie zaprojektowany jako interfejs do bibliotecznej bazy danych artykułów informatycznych. Użytkownik biblioteki zwracałby się do komputera w mowie po angielsku, pytając: „Czy są to Feigenbaum i Feldman?" a komputer odpowiedziałby na pytanie informacjami z bazy danych biblioteki. Zrozumienie wymaga zintegrowania wielu różnych procesów, z których wszystkie wymagają bardzo różnej wiedzy i algorytmów, z których wszystkie mogą być wykładniczo złożone. Przetwarzanie sygnałów; rozpoznawanie fonemów, sylab i słów; parsowanie składniowe; oraz analiza semantyczna wzajemnie się ograniczają w interpretacji mowy. Architektura tablicy umożliwiła HEARSAY-II koordynację kilku różnych źródeł wiedzy wymaganych do tego złożonego zadania. Tablica jest zwykle zorganizowana w dwóch wymiarach. W HEARSAY-II tymi wymiarami były czas, w którym powstał akt mowy i poziom analizy wypowiedzi. Każdy poziom analizy przetwarzany był przez inną klasę źródeł wiedzy. Te poziomy analizy to:

KS_1 Kształt fali sygnału akustycznego.

KS_2 Fonemy lub możliwe segmenty dźwiękowe sygnału akustycznego.

KS_3 Sylaby, które fonemy mogłyby wytworzyć.

KS_4 Możliwe słowa analizowane przez jeden KS.

KS_5 Możliwe słowa analizowane przez drugi KS (zwykle biorąc pod uwagę słowa z różnych części danych).

KS_6 KS, aby spróbować wygenerować możliwe sekwencje słów.

KS_7 KS, który umieszcza sekwencje słów w możliwych frazach.

Możemy wizualizować te procesy jako elementy na rysunku 6.13. Podczas przetwarzania mowy mówionej fala sygnału mówionego jest wprowadzana na najniższym poziomie. Źródła wiedzy na temat przetwarzania tego wpisu są włączone i zamieszczają swoje interpretacje na tablicy, aby je zebrać w odpowiednim procesie. Ze względu na dwuznaczności języka mówionego na każdym poziomie tablicy może występować wiele konkurencyjnych hipotez. Źródła wiedzy na wyższych poziomach próbują

ujednoznacznić te konkurencyjne hipotezy. Analiza HEARSAY-II nie powinna być postrzegana jako po prostu jeden niższy poziom generujący dane, które wyższe poziomy mogą następnie analizować. To jest o wiele bardziej skomplikowane. Jeśli KS na jednym poziomie nie może przetworzyć (zrozumieć) przesłanych do niego danych, KS może poprosić KS, który wysłał mu dane, o powrót do kolejnej próby lub postawienie kolejnej hipotezy na temat danych. Co więcej, różne KS mogą jednocześnie pracować nad różnymi częściami wypowiedzi. Wszystkie procesy, jak wspomniano wcześniej, są asynchroniczne i sterowane danymi; działają, gdy mają dane wejściowe, kontynuują działanie do momentu zakończenia zadania, a następnie publikują wyniki i czekają na następne zadanie.

Jeden z KS, zwany harmonogramem, obsługuje komunikację „zużycie danych po wyniku" między KS. Ten harmonogram ma oceny wyników każdej KS i jest w stanie dostarczyć, za pomocą kolejki priorytetowej, pewien kierunek w rozwiązywaniu problemów. Jeśli żaden KS nie jest aktywny, program planujący stwierdza, że zadanie zostało zakończone i wyłącza się. Gdy program HEARSAY miał bazę około 1000 słów, działał całkiem dobrze, choć nieco wolno. Gdy baza danych została dodatkowo rozszerzona, dane dla źródeł wiedzy stały się bardziej złożone, niż mogły sobie z tym poradzić. HEARSAY-III jest uogólnieniem podejścia przyjętego przez HEARSAY-II. Wymiar czasowy HEARSAY-II nie jest już potrzebny, ale zachowano wiele KS dla poziomów analizy. Tablica HEARSAY-III ma na celu interakcję z ogólnym systemem relacyjnych baz danych. Rzeczywiście, HEARSAY-III to powłoka do projektowania systemów eksperckich. Ważną zmianą w HEARSAY-III było rozdzielenie programu planującego KS (jak opisano powyżej dla HEARSAY-II) i uczynienie go oddzielnym kontrolerem tablicy dla pierwszej (lub domeny problemowej) tablicy. Ta druga tablica pozwala rozbić proces planowania, podobnie jak dziedzina problemu, na osobne KS zajmujące się różnymi aspektami procedury rozwiązania (na przykład, kiedy i jak zastosować wiedzę w tej dziedzinie). Druga tablica może zatem porównać i zrównoważyć różne rozwiązania dla każdego problemu (Nii i Aiello 1979, Nii 1986a, 1986b). Alternatywny model tablicy zachowuje ważne części bazy wiedzy w tablicy, zamiast rozprowadzać je między źródłami wiedzy

Epilog

Omówiliśmy wdrażanie strategii wyszukiwania . Przedstawiono rekurencję jako ważne narzędzie do programowania wyszukiwania wykresów, implementujące algorytmy głębokości pierwszej i wstecznej w formie rekurencyjnej. Wyszukiwanie ukierunkowane na wzorce z regułami unifikacji i wnioskowania, , upraszcza implementację wyszukiwania w przestrzeni logicznych wniosków. Pokazano system produkcyjny jako naturalną architekturę do modelowania rozwiązywania problemów i implementacji algorytmów wyszukiwania. Sekcja zakończyła się przykładami implementacji systemu produkcyjnego wyszukiwania opartego na danych i celu. W rzeczywistości system produkcyjny zawsze był ważnym paradygmatem programowania AI, poczynając od pracy Newella i Simona oraz ich kolegów z Carnegie Mellon University. System produkcyjny był także ważną architekturą wspierającą badania w dziedzinie kognitywistyki. Odniesienia do budowania systemów produkcyjnych, zwłaszcza języków OPS, obejmują programowanie systemów ekspertowych w OPS5 autorstwa Lee Brownstona i in. oraz Pattern Directed Inference Systems autorstwa Donalda Watermana i Fredericka Hayes-Rotha. Aby zapoznać się z nowoczesnymi systemami produkcyjnymi w języku C i Java, przejdź do stron internetowych dla CLIPS i JESS. Wczesna praca w modelach tablicowych została opisana w badaniach HEARSAY-II. Późniejsze zmiany w tablicach opisano w pracy HEARSAY-III oraz Blackboard Systems, pod redakcją Roberta Engelmore'a i Tony'ego Morgana. Badania systemów produkcji, planowania i architektury tablic pozostają aktywną częścią sztucznej inteligencji. Zalecamy, aby zainteresowany czytelnik zapoznał się z najnowszymi obradami Konferencji Amerykańskiego Stowarzyszenia dla Sztucznej Inteligencji i Międzynarodowej Wspólnej Konferencji na temat Sztucznej Inteligencji. Morgan Kaufmann i AAAI Press publikują materiały z konferencji, a także zbiory lektur na tematy AI

Sztuczna inteligencja : Reprezentacja Wiedzy (VII / XVI)

ĆWICZENIA

1. Rozumowanie zdroworozsądkowe posługuje się takimi pojęciami, jak przyczynowość, analogia i równoważność, ale używa ich w inny sposób niż języki formalne. Na przykład, jeśli powiemy: „Inflacja spowodowała, że Jane poprosiła o podwyżkę", sugerujemy bardziej skomplikowany związek przyczynowy niż ten, który można znaleźć w prostych prawach fizycznych. Jeśli mówimy „Użyj noża lub dłuta, aby przyciąć drewno", sugerujemy ważne pojęcie równoważności. Omów problemy związane z tłumaczeniem tych i innych takich pojęć na język formalny.

2. W sekcji 7.2.1 przedstawiliśmy niektóre argumenty przeciwko używaniu logiki do reprezentowania wiedzy zdroworozsądkowej. Argumentuj za użyciem logiki w przedstawianiu tej wiedzy. McCarthy i Hayes (1969) prowadzą interesującą dyskusję na ten temat.

3. Przetłumacz każde z poniższych zdań na rachunek predykatów, zależności pojęciowe i grafy pojęciowe:

„Jane dała Tomowi lody w rożku".

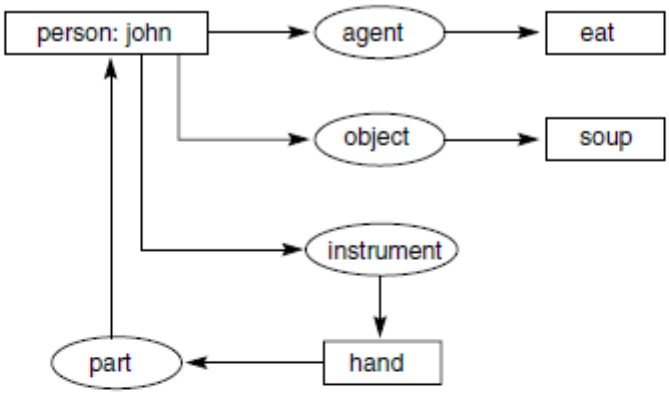
„Koszykarze są wysocy".

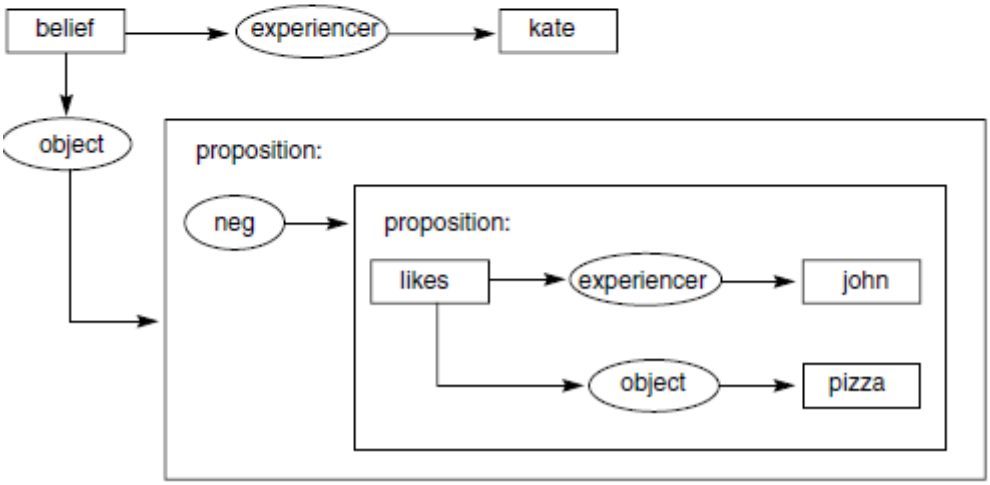
„Paul ściął drzewo siekierą".

„Umieść wszystkie składniki w misce i dokładnie wymieszaj".

4. Przeczytaj „What's in a Link" autorstwa Woodsa (1985). W sekcji IV tego artykułu wymieniono szereg problemów związanych z reprezentacją wiedzy. Zaproponuj rozwiązanie każdego z tych problemów za pomocą logiki, grafów koncepcyjnych i notacji ramek.

5. Przetłumacz koncepcyjne wykresy z rysunku na angielskie zdania





6. Operacje łączą i ograniczają definiując uporządkowanie uogólnień na grafach pojęciowych. Pokaż, że relacja uogólnienia jest przechodnia.

7. Specjalizacja grafów konceptualnych za pomocą łączenia i ograniczania nie jest operacją zachowującą prawdę. Podaj przykład, który pokazuje, że ograniczenie prawdziwego wykresu niekoniecznie jest prawdziwe. Jednak uogólnienie prawdziwego wykresu jest zawsze prawdziwe; udowodnij to.

8. Zdefiniuj specjalistyczny język reprezentacji do opisania działalności biblioteki publicznej. Ten język będzie zbiorem pojęć i relacji za pomocą grafów pojęciowych. Zrób to samo dla handlu detalicznego. Jakie pojęcia i relacje miałyby wspólne te dwa języki? Która istniałaby w obu językach, ale miałaby inne znaczenie?

9. Przetłumacz wykresy pojęciowe z rysunku 7.28 na rachunek predykatów.

10. Przetłumacz bazę wiedzy doradców finansowych, sekcja 2.4, na koncepcyjny wykres.

11. Przedstaw dowody z własnego doświadczenia, które sugerują organizację pamięci ludzkiej w stylu skryptu lub ramki.

12. Korzystając z zależności koncepcyjnych, zdefiniuj skrypt dla:

a. Restauracja typu fast-food.

b. Interakcja ze sprzedawcą używanych samochodów.

c. Idąc do opery.

13. Skonstruować hierarchię podtypów dla pojazdu koncepcyjnego; na przykład podtypy pojazdu mogą być land_vehicle lub ocean-vehicle. Te miałyby dalsze podtypy. Czy najlepiej przedstawia się to jako drzewo, krata czy ogólny wykres? Zrób to samo dla ruchu koncepcyjnego; za pojęcie zły.

14. Skonstruuj hierarchię typów, w której niektóre typy nie mają wspólnego nadtypu. Dodaj typy, aby stworzyć kratę. Czy tę hierarchię można wyrazić za pomocą dziedziczenia drzewa? Jakie problemy mogą się przy tym pojawić?

15. Każdy z poniższych ciągów znaków jest generowany według jakiejś ogólnej zasady. Opisz reprezentację, której można by użyć do przedstawienia reguł lub relacji wymaganych do kontynuowania każdej sekwencji:

a. 2,4,6,8,. . .

b. 1,2,4,8,16,. . .

c. 1,1,2,3,5,8,. . .

d. 1, a, 2, c, 3, f, 4,. . .

e. o, t, t, f, f, s, s,. . .

16. Przykłady analogicznego rozumowania zostały przedstawione w punkcie 7.3.2. Opisz odpowiednią reprezentację i strategię wyszukiwania, która pozwoliłaby zidentyfikować najlepszą odpowiedź na tego typu problem. Utwórz jeszcze dwie przykładowe analogie, które będą działać z proponowaną reprezentacją. Czy można rozwiązać dwa poniższe przykłady?

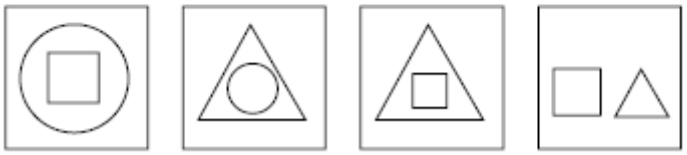
a. gorąco jest do zimna jak wysokie do {ściana, krótkie, mokre, trzymaj}

b. niedźwiedź jest świnią jak krzesło dla {stopa, stół, kawa, truskawka}

17. Opisz reprezentację, której można by użyć w programie do rozwiązania problemów analogii, takich jak to na rysunku. Tej klasy problemów zajął się T. G. Evans (1968). Przedstawienie musi być w stanie uchwycić podstawowe cechy rozmiaru, kształtu i względnego położenia.

Choose one:

A B C D



18. Artykuł Brooksa zawiera ważną dyskusję na temat roli reprezentacji w tradycyjnej sztucznej inteligencji. Przeczytaj ten artykuł i skomentuj ograniczenia jawnych schematów reprezentacji ogólnego przeznaczenia.

19. Na końcu sekcji 7.3.1 znajduje się pięć potencjalnych problemów, którymi musi się zająć architektura subsumpcji firmy Brooks (1991a), aby zaoferować skuteczne, ogólne podejście do rozwiązywania problemów. Wybierz co najmniej jeden z nich i skomentuj go (je).

20. Zidentyfikuj pięć właściwości, które powinien mieć język agenta, aby zapewnić usługę internetową zorientowaną na agenta. Skomentuj rolę języka Java jako języka agentów ogólnego przeznaczenia do tworzenia usług internetowych. Czy widzisz podobną rolę dla CLOS? Dlaczego lub dlaczego nie? Jest

mnóstwo informacji na ten temat w samym internecie.

21. Pod koniec sekcji 7.4 przedstawiono szereg ważnych kwestii związanych z tworzeniem zorientowanych na agentów rozwiązań problemów. Wybierz jedną z nich i omów problem dalej.

22. Załóżmy, że projektujesz system agentów reprezentujący futbol amerykański lub drużynę piłkarską. Aby agenci mogli współpracować podczas manewru obronnego lub punktacji, muszą mieć jakieś pojęcie o swoich planach i możliwych reakcjach na sytuacje. Jak możesz zbudować model celów i planów innego współpracującego agenta?

23 Wybierz jeden z obszarów zastosowań architektur agentów, które podsumowano w sekcji 7.4. Wybierz dokument badawczy lub aplikacyjny w tej dziedzinie. Zaprojektuj organizację agentów, która mogłaby rozwiązać problem. Rozbij problem, aby określić kwestie odpowiedzialności każdego agenta. Wymień odpowiednie procedury współpracy.

PRZECHWYTYWANIE INTELIGENCJI: WYZWANIE AI

Reprezentacja i inteligencja

Pytanie o reprezentację lub jak najlepiej uchwycić krytyczne aspekty inteligentnej aktywności do użycia na komputerze, a nawet do komunikacji z ludźmi, było stałym tematem w ponad sześćdziesięcioletniej historii AI. Zaczynamy Część III od przeglądu trzech dominujących podejść do reprezentacji przyjętych przez społeczność badawczą AI w tym okresie. Pierwszy temat, sformułowany w latach 50. i 60. XX wieku przez Newella i Simona w ich pracy z teoretykiem logiki, znany jest jako rozwiązywanie słabych metod. Drugim tematem, powszechnym w latach 70. i 80. XX wieku, popieranym przez wczesnych ekspertów projektantów systemów, jest silne rozwiązywanie problemów metod (patrz cytat Feigenbauma na początku rozdziału 8). W ostatnich latach, szczególnie w w dziedzinach robotyki i Internetu nacisk kładziony jest na rozproszone i ucieleśnione lub oparte na agentach podejście do inteligencji. Następnie wprowadzamy każdą z tych metod reprezentacji inteligencji. Trzy rozdziały, które składają się na część III, zawierają bardziej szczegółowy opis każdego podejścia. Na przełomie lat 50. i 60. XX wieku Alan Newell i Herbert Simon napisali kilka komputerowych rogramów w celu przetestowania hipotezy, że inteligentne zachowanie jest wynikiem wyszukiwania heurystycznego. Teoretyk logiki, opracowany przez J. C. Shawa, udowodnił twierdzenia w logice elementarnej, używając notacji i aksjomatów Whitehead i Principia Mathematica Russella. Autorzy opisują swoje badania jako mające na celu zrozumienie złożonego procesu (heurystyka), które są skuteczne w rozwiązywaniu problemów. Dlatego nie interesują nas metody gwarantujące rozwiązania, ale wymagające ogromnych obliczeń. Chcemy raczej zrozumieć, na przykład, jak matematyk jest w stanie udowodnić twierdzenie, nawet jeśli nie wie, kiedy zaczyna, w jaki sposób lub czy odniesie sukces.

W późniejszym programie, General Problem Solver lub GPS, Newell i Simon kontynuowali ten wysiłek, aby znaleźć ogólne zasady inteligentnego rozwiązywania problemów. GPS rozwiązał problemy sformułowane jako wyszukiwanie w przestrzeni stanu; prawne kroki rozwiązywania problemów były zestawem operacji służących modyfikacji reprezentacji stanu. GPS szukał sekwencji operacji, które przekształcą stan początkowy w stan docelowy, szukając w taki sam sposób, jak algorytmy omówione wcześniej. Wykorzystano GPS, analizę środków i celów, ogólną heurystykę wyboru spośród alternatywnych operacji transformacji stanu, aby przeprowadzić wyszukiwanie w przestrzeni problemowej. Analiza środków oznacza nie różnic składniowych między stanem bieżącym a stanem celu i wybiera operatora, który może zmniejszyć te różnice. Załóżmy na przykład, że GPS próbuje udowodnić równoważność dwóch wyrażeń logicznych. Jeśli bieżący stan zawiera operator ∧, a cel nie zawiera ∧, wówczas analiza środków i wyników byłaby taka aby wybrać transformację, taką jak prawo de Morgana, aby usunąć ∧ z wyrażeń. Korzystając z heurystyki, która bada tylko składniową formę stanów, mieli nadzieję, że GPS będzie ogólną architekturą inteligentnego rozwiązywania problemów, bez względu na domenę. Programy takie jak GPS, które są ograniczone do strategii opartych na składni i przeznaczone do szerokiej gamy aplikacji, są znane jako słabe metody rozwiązywania problemów. Niestety, wydaje się, że nie istnieje jedna heurystyka, którą można z powodzeniem zastosować we wszystkich domenach problemowych. Ogólnie rzecz biorąc, metody, których używamy do rozwiązywania problemów, wymagają dużej wiedzy na temat sytuacji. Lekarze są w stanie zdiagnozować chorobę, ponieważ oprócz swojej ogólnej umiejętności rozwiązywania problemów mają rozległą wiedzę medyczną. Architekci projektują domy, ponieważ wiedzą o architekturze. Rzeczywiście, heurystyka stosowana w diagnozie medycznej byłaby bezużyteczna przy projektowaniu budynku biurowego. Słabe metody wyraźnie kontrastują z silnymi metodami, które wykorzystują wyraźną wiedzę na temat określonej dziedziny problemowej. Rozważ zasadę analizy diagnostycznej problemów motoryzacyjnych:

if

silnik się nie obraca, i

światła nie zapalają się

else

problemem jest bateria lub kable (.8)

Ta heurystyka, sformułowana jako zasada „jeśli… to…", koncentruje wyszukiwanie na podsystemie akumulatorów / kabli samochodowych, eliminując inne elementy i przycinając przestrzeń poszukiwań. Zauważ, że ta szczególna heurystyka, w przeciwieństwie do analizy środków i celów, wykorzystuje empiryczną wiedzę na temat funkcjonowania samochodów, na przykład znajomość relacji między akumulatorem, światłami i rozrusznikiem. Jest bezużyteczny w każdej problematycznej dziedzinie oprócz naprawy pojazdu. Silne metody, które rozwiązują problemy, nie tylko wykorzystują wiedzę specyficzną dla danej dziedziny, ale na ogół wymagają dużej ilości takiej wiedzy, aby były skuteczne. Na przykład heurystyka złej baterii nie byłaby przydatna w diagnozowaniu złego gaźnika i prawdopodobnie byłaby bezużyteczna w innych domenach. Tak więc głównym wyzwaniem przy projektowaniu programu opartego na wiedzy jest pozyskiwanie i organizacja dużych ilości określonej domeny wiedzy, umiejętności. Reguły domenowe mogą również zawierać miarę, 0,8 w powyższej regule, odzwierciedlającą zaufanie diagnosta do tej wiedzy o domenach. Stosując podejście oparte na silnej metodzie, projektanci programów przyjmują pewne założenia dotyczące natury inteligentnych systemów. Te założenia sformalizował Brian Smith jako hipoteza reprezentacji wiedzy. Ta hipoteza stwierdza, że każdy mechanicznie ucieleśniony inteligentny proces będzie składał się ze składników strukturalnych, które (a) my, zewnętrzni obserwatorzy, naturalnie bierzemy, aby reprezentować

propozycyjne zestawienie wiedzy, którą wykazuje cały proces, oraz (b) niezależnie od takiego zewnętrznego przypisania semantycznego, odgrywać formalną rolę w zachowaniu, które przejawia tę wiedzę.

Ważnymi aspektami tej hipotezy są założenie, że wiedza będzie reprezentowana propozycjonalnie, to znaczy w formie, która wyraźnie reprezentuje daną wiedzę, i która może być postrzegana przez zewnętrznego obserwatora jako „naturalny" opis tej wiedzy. Drugim ważnym założeniem jest to, że zachowanie systemu może być postrzegane jako formalnie spowodowane przez zdania w bazie wiedzy i że to zachowanie powinno być zgodne z naszym postrzeganym znaczeniem tych zdań. Ostatni temat reprezentacji AI jest często opisywany jako oparty na agentach, ucieleśniony lub pojawiający się problem. Kilku badaczy pracujących w takich dziedzinach, jak robotyka, projektowanie gier i Internet, w tym Brooks, Agre i Chapman, Jennings, Wooldridge, Wooldridge , Fatima oraz Vieira zakwestionowało wymóg posiadania scentralizowanej bazy wiedzy lub schematu wnioskowania ogólnego. Rozwiązania problemów są zaprojektowane jako rozproszeni agenci: zlokalizowani, autonomiczni i elastyczni. Z tego punktu widzenia rozwiązywanie problemów jest postrzegane jako rozproszone, przy czym agenci wykonują zadania w różnych podkontekstach swoich domen, na przykład aktywność Internetu przeglądarka lub agent bezpieczeństwa. Zadanie rozwiązywania problemów jest podzielone na kilka części z niewielką lub żadną ogólną koordynacją zadań. Umieszczony agent jest w stanie odbierać dane sensoryczne ze swojego konkretnego środowiska, a także reagować w tym kontekście bez oczekiwania na instrukcje od jakiegoś ogólnego kontrolera. Na przykład w interaktywnej grze agent rozwiązuje konkretny problem lokalny, np. broni się przed konkretnym atakiem lub podnosi konkretny alarm, bez ogólnego widoku całej sytuacji problemowej. Agenci są autonomiczni, ponieważ często muszą działać bez bezpośredniej interwencji ludzi lub ogólnego procesu kontroli. Autonomiczni agenci mają zatem kontrolę nad własnymi działaniami i stanem wewnętrznym. Niektóre systemy agentów mogą nawet uczyć się na podstawie własnych doświadczeń. Wreszcie, agenci są elastyczni, ponieważ reagują na sytuacje w swoim lokalnym środowisku. Są również proaktywni, ponieważ potrafią przewidywać sytuacje. Wreszcie muszą być w stanie współpracować z innymi agentami w dziedzinie problemów, komunikując się o zadaniach, celach i odpowiednich procesach. Kilku badaczy w dziedzinie robotyki zbudowało systemy oparte na agentach. Rodney Brooks w MIT Robotics Laboratory zaprojektował, jak to określa, architekturę subskrypcji, warstwową sekwencję skończonych maszyn stanu, z których każda działa we własnym kontekście, ale także wspiera funkcjonalność na wyższych poziomach. Manuela Veloso i in. pracując w Robotics Lab w Carnegie Mellon, zaprojektowali zespół robotycznych agentów piłki nożnej (piłkarzy), którzy współpracują w nieprzyjacielskim środowisku meczów piłkarskich .

Wreszcie to usytuowane i ucieleśnione podejście do reprezentacji zostało opisane przez kilku filozofów nauki, w tym Dana Dennetta i Andy'ego Clarka jako odpowiednie charakterystyki ludzkiej inteligencji. W dalszej części prezentujemy dalsze schematy reprezentacyjne, w tym podejścia do łączenia się z innymi i genetyczne. Zaczynamy od wczesnego użycia reprezentacji, w tym sieci semantycznych, skryptów, ramek i obiektów. Prezentujemy te schematy z ewolucyjnego punktu widzenia, pokazując, jak powstały nowoczesne narzędzia z pierwszych lat badań nad AI. Następnie przedstawiamy wykresy koncepcyjne Johna Sowy, reprezentację do wykorzystania w zrozumieniu języka naturalnego. Na koniec przedstawiamy oparte na agentach i usytuowane podejście do reprezentacji, w tym architekturę subkonstrukcji Rodneya Brooksa, która podważa potrzebę centralnej bazy jawnej wiedzy wykorzystywanej z kontrolerem ogólnego przeznaczenia. W rozdziale 8 omawiamy systemy wymagające wiedzy i analizujemy problemy związane z pozyskiwaniem, formalizacją i debugowaniem bazy wiedzy. Prezentujemy różne schematy wnioskowania dla systemów reguł, w tym uzasadnianie celów i danych. Oprócz wnioskowania opartego na regułach, prezentujemy systemy oparte na modelach i przypadkach. Pierwszy z nich próbuje wyraźnie przedstawić teoretyczne podstawy i

funkcjonalność domeny, np. obwodu elektronicznego, podczas gdy drugie podejście tworzy kumulatywną bazę danych przeszłych sukcesów i niepowodzeń w dziedzinie problemów, aby pomóc w przyszłym rozwiązywaniu problemów. Zakończymy przeglądem planowania, w którym zorganizowana jest wiedza w celu kontrolowania rozwiązywania problemów w złożonych dziedzinach, takich jak robotyka. Przedstawiamy szereg technik, które rozwiązują problem reprezentacji przy rozumowaniu w sytuacjach niejasności i / lub niepewności. W tych czasach staramy się znaleźć najlepsze wyjaśnienie często niejednoznacznych informacji. Ten rodzaj rozumowania jest często określany jako uprowadzający. Najpierw przedstawiamy logiki niemonotoniczne i podtrzymujące prawdę, w których tradycyjna logika predykatów jest rozszerzona, aby uchwycić sytuacje niepewne. Jednak wiele interesujących i ważnych problemów nie mieści się wygodnie pod parasolem logiki dedukcyjnej. Dla uzasadnienia w tych sytuacjach wprowadzamy szereg innych potężnych narzędzi, w tym techniki bayesowskie, podejście Dempster-Shafer i algebrę współczynnika pewności Stanforda. Mamy także sekcję dotyczącą rozumowania rozmytego. Kończymy Część III stochastyczną metodologią niepewności, przedstawiając bayesowskie sieci wierzeń, modele Markowa, ukryte modele Markowa i podobne podejścia.

REPREZENTACJA WIEDZY

Problemy w reprezentacji wiedzy

Reprezentacja informacji do wykorzystania w inteligentnym rozwiązywaniu problemów oferuje ważne i trudne wyzwania, które leżą u podstaw sztucznej inteligencji. Prezentujemy krótką historyczną retrospektywę wczesnych badań w reprezentacji; Tematy obejmują sieci semantyczne, zależności koncepcyjne, skrypty i ramki. Oferujemy bardziej nowoczesną reprezentację stosowaną w programach języka naturalnego, wykresy koncepcyjne Johna Sowy. Krytykujemy wymóg stworzenia scentralizowanych i wyraźnych schematów reprezentacyjnych. Alternatywą Brooksa jest architektura subskrypcji dla robotów. Przedstawiamy agentom kolejną alternatywę dla scentralizowanej kontroli. Później rozszerzamy naszą dyskusję na temat reprezentacji o stochastyczne, łączące i genetyczne / wschodzące. Dyskusję na temat reprezentacji rozpoczynamy od perspektywy historycznej, w której bazę wiedzy opisuje się jako odwzorowanie między obiektami i relacjami w dziedzinie problemów a obiektami obliczeniowymi i relacjami programu. Zakłada się, że wyniki wnioskowania w bazie wiedzy odpowiadają wynikom działań lub obserwacji na świecie. W obiektach obliczeniowych, relacjach i wnioskach dostępnych dla programistów pośredniczy język reprezentacji wiedzy. Istnieją ogólne zasady organizacji wiedzy, które mają zastosowanie w różnych domenach i mogą być bezpośrednio wspierane przez język reprezentacji. Na przykład hierarchie klas znajdują się zarówno w systemach klasyfikacji naukowej, jak i zdrowej. Jak możemy zapewnić ogólny mechanizm ich reprezentowania? Jak możemy reprezentować definicje? Wyjątki? Kiedy inteligentny system powinien przyjmować domyślne założenia dotyczące brakujących informacji i jak dostosować swoje rozumowanie, jeśli założenia te okażą się błędne? Jak najlepiej reprezentować czas? Przyczynowość? Niepewność? Postęp w budowaniu inteligentnych systemów zależy od odkrywania zasad organizacji wiedzy i wspierania ich za pomocą narzędzi reprezentacyjnych wyższego poziomu. Przydatne jest rozróżnienie między schematem reprezentacyjnym a środkiem jego realizacji. Jest to podobne do rozróżnienia między strukturami danych a językami programowania. Języki programowania są środkiem wdrażania; struktura danych jest schematem. Zasadniczo języki reprezentacji wiedzy są bardziej ograniczone niż rachunek predykatów lub języki programowania. Ograniczenia te mają formę wyraźnych struktur reprezentujących kategorie wiedzy. Ich środkiem implementacji może być Prolog, Lisp lub bardziej popularne języki, takie jak C++ lub Java. Nasza dyskusja na ten temat ilustruje tradycyjny pogląd na schematy reprezentacji AI, który często obejmuje globalną bazę wiedzy o strukturach językowych odzwierciedlającą statyczne i „wstępnie zinterpretowane uprzedzenie" „prawdziwego świata".

Nowsze badania w dziedzinie robotyki, poznania umiejscowionego, rozwiązywania problemów opartych na agentach i filozofii podważyły to tradycyjne podejście. Te domeny problemowe wymagają rozproszonej wiedzy, świata, który sam może być wykorzystany jako częściowa struktura wiedzy, zdolność rozumowania z częściowymi informacjami, a nawet reprezentacji, które mogą ewoluować, gdy doświadczą niezmienników domeny problemowej.

Krótka historia schematów reprezentacyjnych AI

Asocjatywne teorie znaczeń

Logiczne reprezentacje wyrosły z wysiłków filozofów i matematyków, aby scharakteryzować zasady poprawnego rozumowania. Główną troską logiki jest opracowanie formalnych języków reprezentacji z solidnymi i kompletnymi regułami wnioskowania. W rezultacie semantyka rachunku predykatów kładzie nacisk na zachowanie prawdy na dobrze sformułowanych wyrażeniach. Alternatywna linia badań wyrosła z wysiłków psychologów i lingwistów w celu scharakteryzowania natury ludzkiego zrozumienia. Ta praca jest mniej związana z ustanowieniem nauki o prawidłowym rozumowaniu niż z opisem sposobu, w jaki ludzie faktycznie zdobywają, łączą i wykorzystują wiedzę o swoim świecie. Takie podejście okazało się szczególnie przydatne w obszarach sztucznej inteligencji rozumienia języka naturalnego i rozumowania opartego na zdrowym rozsądku. Istnieje wiele problemów, które pojawiają się przy mapowaniu rozumowania zdrowego rozsądku do logiki formalnej. Na przykład często myśli się o operatorach ∨ i → jako odpowiadające angielskiemu „lub" i „jeśli ... to ...". Jednak operatorzy logiczni zajmują się wyłącznie wartościami prawdy i ignorują fakt, że angielski „jeśli ... to ..." sugeruje konkretny związek (często bardziej koorelacyjny niż przyczynowy) między jego przesłankami a ich wnioskiem. Na przykład zdanie „Jeśli ptak jest kardynałem, to jest czerwony" (kojarzenie ptaka kardynała z kolorem czerwonym) można zapisać w rachunku predykatu:

$\forall X (cardinal(X) \rightarrow red(X))$.

Można to zmienić, poprzez szereg operacji zachowania prawdy, w logicznie równoważne wyrażenie

$X (\neg red(X) \rightarrow \neg cardinal(X))$

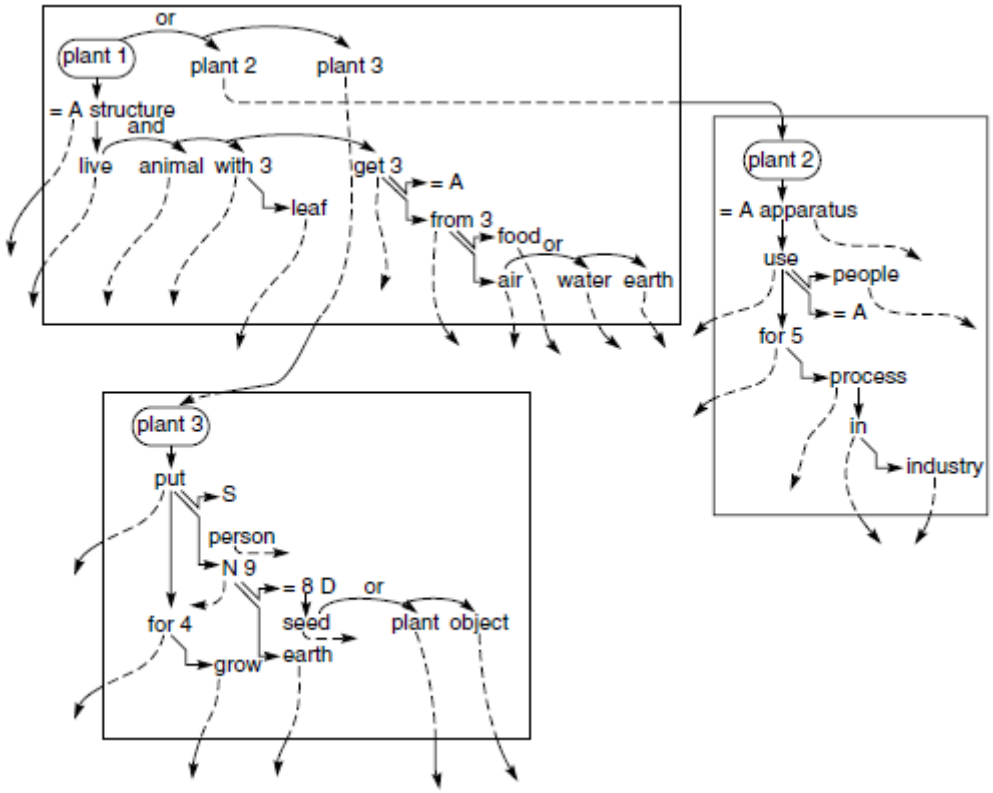
Te dwa wyrażenia mają tę samą wartość prawdy; to znaczy, druga jest prawdziwa wtedy i tylko wtedy, gdy pierwsza jest prawdziwa. Jednak równoważność wartości prawdy jest nieodpowiednia w tej sytuacji. Gdybyśmy mieli szukać fizycznych dowodów prawdziwości tych stwierdzeń, fakt, że ta kartka papieru nie jest czerwona, a także nie jest kardynałem, jest dowodem na prawdziwość drugiego wyrażenia. Ponieważ oba wyrażenia są logicznie równoważne, wynika z tego, że jest to również dowód na prawdziwość pierwszego stwierdzenia. Prowadzi to do wniosku, że biel kartki papieru świadczy o tym, że kardynałowie są czerwoni. Ta linia rozumowania wydaje się nam bezsensowna i raczej głupia. Powodem tej niezgodności jest to, że logiczna implikacja wyraża jedynie związek między wartościami prawdziwości jej operandów, podczas gdy zdanie angielskie sugeruje pozytywną korelację między członkostwem w klasie a posiadaniem własności tej klasy. W rzeczywistości makijaż genetyczny ptaka powoduje, że ma on określony kolor. Ten stan relacji został utracony w drugiej wersji wyrażenia. Chociaż fakt, że papier nie jest czerwony, jest zgodny z prawdą obu zdań, nie ma znaczenia dla przyczynowego charakteru koloru ptaków. Teorie stowarzyszeniowe, zgodnie z tradycją empirystyczną w filozofii, definiują znaczenie obiektu w kategoriach sieci skojarzeń z innymi przedmiotami. Dla stowarzyszenia, gdy ludzie postrzegają przedmiot, postrzeganie to jest najpierw mapowane na pojęcie. Ta koncepcja jest częścią naszej całej wiedzy o świecie i jest powiązana poprzez odpowiednie relacje z innymi pojęciami. Relacje te tworzą zrozumienie właściwości i zachowania obiektów takich jak śnieg. Na przykład poprzez doświadczenie kojarzymy pojęcie śniegu z innymi pojęciami, takimi jak zimno, biały, bałwan, śliski i lód. Nasze rozumienie śniegu i prawdziwość stwierdzeń takich jak „śnieg jest biały"

i „bałwan jest biały” przejawia się w tej sieci skojarzeń. Istnieją dowody psychologiczne, że oprócz zdolności do kojarzenia pojęć, ludzie również organizują swoją wiedzę w sposób hierarchiczny, a informacje przechowywane są na najwyższych odpowiednich poziomach taksonomii. Collins i Quillian modelowali przechowywanie i zarządzanie ludzkimi informacjami za pomocą sieci semantycznej. Struktura tej hierarchii została wyprowadzona z badań laboratoryjnych na ludziach. Badanym zadawano pytania dotyczące różnych właściwości ptaków, takie jak: „Czy kanarek jest ptakiem?” lub „Czy kanarek może śpiewać?” lub „Czy kanarek może latać?”. Jakkolwiek oczywiste mogą się wydawać odpowiedzi na te pytania, badania czasu reakcji wykazały, że uczestnicy potrzebowali więcej czasu na odpowiedź „Czy kanarek może latać?” niż w przypadku odpowiedzi „Czy kanarek może śpiewać?” Collins i Quillian wyjaśniają tę różnicę w czasie odpowiedzi, argumentując, że ludzie przechowują informacje na najbardziej abstrakcyjnym poziomie. Zamiast próbować przypomnieć sobie, że kanarki latają, a latają rudziki, i jaskółki latają, wszystkie przechowywane razem z pojedynczym ptakiem, ludzie pamiętają, że kanarki są ptakami i że ptaki mają (zwykle) właściwość latania. Nawet bardziej ogólne właściwości, takie jak jedzenie, oddychanie i poruszanie się, są przechowywane na poziomie „zwierzęcym”, dlatego próba przypomnienia sobie, czy kanarek może oddychać, powinna trwać dłużej niż przypomnienie, czy kanarek może latać. Jest tak oczywiście dlatego, że człowiek musi podróżować wyżej w hierarchii struktur pamięci, aby uzyskać odpowiedź. Najszybsze przypominanie dotyczyło cech charakterystycznych dla ptaka, powiedzmy, że może śpiewać lub jest żółty. Obsługa wyjątków również wydawała się być wykonywana na najbardziej konkretnym poziomie. Gdy pytano badanych, czy struś może latać, odpowiedź była udzielana szybciej niż wtedy, gdy pytano, czy struś może oddychać. Tak więc wydaje się, że hierarchia struś → ptak → zwierzę nie przechodzi, aby uzyskać informacje o wyjątku: jest on przechowywany bezpośrednio z struś. Ta organizacja wiedzy została sformalizowana w systemach spadkowych. Systemy dziedziczenia pozwalają nam przechowywać informacje na najwyższym poziomie abstrakcji, co zmniejsza rozmiar baz wiedzy i pomaga zapobiegać niespójnościom aktualizacji. Dla na przykład, jeśli budujemy bazę wiedzy o ptakach, możemy zdefiniować cechy wspólne wszystkim ptakom, takie jak latanie lub posiadanie piór, dla ptaka klasy ogólnej i pozwolić, aby określony gatunek ptaka odziedziczył te właściwości. Zmniejsza to rozmiar bazy wiedzy, wymagając od nas, abyśmy zdefiniowali te podstawowe cechy tylko raz, zamiast wymagać ich potwierdzenia dla każdej osoby. Dziedziczenie pomaga nam również zachować spójność bazy wiedzy podczas dodawania nowych klas i osób. Załóżmy, że dodajemy robin gatunkowy do istniejącej bazy wiedzy. Kiedy twierdzimy, że robin jest podklasą śpiewającego ptaka; robin dziedziczy wszystkie wspólne właściwości zarówno ptaków śpiewających, jak i ptaków. Programiści nie muszą pamiętać (lub zapomnieć!) O dodaniu tych informacji. Wykresy, zapewniając środki do wyraźnego przedstawienia relacji za pomocą łuków i węzłów, okazały się idealnym narzędziem do sformalizowania asocjacyjnych teorii wiedzy. Sieć semantyczna reprezentuje wiedzę jako wykres, z węzłami odpowiadającymi faktom lub pojęciom, a łukami relacjami lub powiązaniami między pojęciami. Zarówno węzły, jak i łącza są ogólnie oznaczone. Można użyć tej sieci (z odpowiednimi zasadami wnioskowania), aby odpowiedzieć na szereg pytań na temat śniegu, lodu i bałwanów. Wniosków tych dokonuje się za pomocą linków do powiązanych pojęć. Sieci semantyczne również wdrażają dziedziczenie; na przykład mroźny dziedziczy wszystkie właściwości bałwana. Termin „sieć semantyczna” obejmuje rodzinę reprezentacji graficznych. Różnią się one przede wszystkim nazwami dozwolonymi dla węzłów i łączy oraz wnioskami, które można wykonać. Jednak wspólny zestaw założeń i obaw jest wspólny dla wszystkich języków reprezentacji sieci; ilustruje to dyskusja na temat historii reprezentacji sieci.

Wczesna praca w sieci semantycznej

Reprezentacje sieciowe mają prawie tak długą historię jak logika. Grecki filozof Porfiriusz stworzył oparte na drzewach hierarchie, których korzenie znajdują się u góry, aby opisać kategorie Arystotelesa .Frege opracował notację drzewa dla wyrażeń logicznych. Być może najwcześniejszym dziełem

mającym bezpośredni wpływ na współczesne sieci semantyczne był system grafów egzystencjalnych Charlesa S. Peirce'a, opracowany w XIX wieku. Teoria Peirce'a posiadała całą ekspresyjną moc rachunku predykatów pierwszego rzędu, z aksjomatyczną podstawą i formalnymi regułami wnioskowania. Wykresy od dawna są używane w psychologii do reprezentowania struktur pojęć i skojarzeń. Selz był pionierem tej pracy, wykorzystując wykresy do reprezentowania hierarchii pojęć i dziedziczenia właściwości. Opracował także teorię przewidywania schematów, która wpłynęła na pracę AI w ramkach i schematach. Anderson, Norman, Rumelhart i inni wykorzystali sieci do modelowania ludzkiej pamięci i wydajności intelektualnej. Wiele badań dotyczących reprezentacji sieci przeprowadzono na arenie rozumienia języka naturalnego. W ogólnym przypadku zrozumienie języka wymaga zrozumienia zdrowego rozsądku, sposobu, w jaki zachowują się przedmioty fizyczne, interakcji zachodzących między ludźmi oraz sposobów organizacji instytucji ludzkich. Program języka naturalnego musi rozumieć intencje, przekonania, hipotetyczne rozumowanie, plany i cele. Ze względu na te wymagania zrozumienie języka zawsze było siłą napędową badań w zakresie reprezentacji wiedzy. Pierwsze implementacje komputerowe sieci semantycznych opracowano na początku lat 60. XX wieku do zastosowania w tłumaczeniu maszynowym. Masterman zdefiniował zestaw 100 prymitywnych typów pojęć i użył ich do zdefiniowania słownika 15 000 pojęć. Wilks kontynuował prace Mastermana w semantycznych sieciowych systemach języka naturalnego. Program MIND Shapiro był pierwszą implementacją sieci semantycznej opartej na rachunku zdań. Inni wczesni pracownicy AI badający reprezentacje sieci to Ceccato , Raphael, Reitman i Simmons. Wpływowy program ilustrujący wiele cech wczesnych sieci semantycznych został napisany przez Quilliana pod koniec lat 60. XX wieku . Ten program definiuje angielskie słowa w taki sam sposób, jak robi to słownik: słowo jest definiowane w kategoriach innych słów, a składniki definicji są definiowane w ten sam sposób. Zamiast formalnego definiowania słów w kategoriach podstawowych aksjomatów, każda definicja po prostu prowadzi do innych definicji w sposób nieuporządkowany, a czasem kołowy. Szukając słowa, przemierzamy tę „sieć", dopóki nie upewnimy się, że rozumiemy oryginalne słowo. Każdy węzeł w sieci Quillian odpowiadał pojęciu słowa, z powiązanymi powiązaniami z innymi pojęciami słowa, które stanowiły jego definicję. Baza wiedzy została zorganizowana w płaszczyzny, przy czym każda płaszczyzna była wykresem definiującym pojedyncze słowo. Rysunek



, ilustruje trzy płaszczyzny, które wychwytują trzy różne definicje słowa „roślina:" żywy organizm (roślina 1), miejsce, w którym ludzie pracują (roślina 2), oraz czynność umieszczania nasion w ziemi (roślina 3). Program wykorzystał tę bazę wiedzy do znalezienia związków między parami angielskich słów. Biorąc pod uwagę dwa słowa, przeszukiwałby wykresy na zewnątrz każdego słowa w szerokim zakresie, szukając wspólnej koncepcji lub węzła przecięcia. Ścieżki do tego węzła reprezentowały związek między pojęciami słów. Liczby w odpowiedzi wskazują, że program wybrał spośród różnych znaczeń słów. Quillian zasugerował, że takie podejście do semantyki może zapewnić system rozumienia języka naturalnego z możliwością:

1. Określ znaczenie tekstu w języku angielskim, tworząc zbiory tych węzłów przecięcia.

2. Wybierz spośród wielu znaczeń słów, znajdując znaczenia z najkrótszą ścieżką przecięcia do innych słów w zdaniu. Na przykład może wybrać znaczenie słowa „roślina" w „Tom poszedł do domu, aby podlać swoją nową roślinę", w oparciu o przecięcie słów „woda" i „roślina".

3. Odpowiedz na elastyczny zakres zapytań opartych na powiązaniach między pojęciami słów w zapytaniach i pojęciami w systemie.

Chociaż ta i inne wczesne prace wykazały moc grafów do modelowania znaczenia asocjacyjnego, była ograniczona ekstremalną ogólnością formalizmu. Wiedza była ogólnie ustrukturyzowana w kategoriach konkretnych relacji, takich jak obiekt / właściwość, klasa / podklasa, i agent / czasownik / obiekt.

Standaryzacja relacji sieciowych

Sam wykres notacji zależności ma niewielką przewagę obliczeniową nad logiką; to tylko kolejna notacja dla relacji między obiektami. W rzeczywistości SNePS, Semantic Net Processing System, był sprawdzonym twierdzeniem w rachunku predykatów pierwszego rzędu. Siła jego reprezentacji sieciowych wynika z definicji łączy i powiązanych reguł wnioskowania, takich jak dziedziczenie. Chociaż

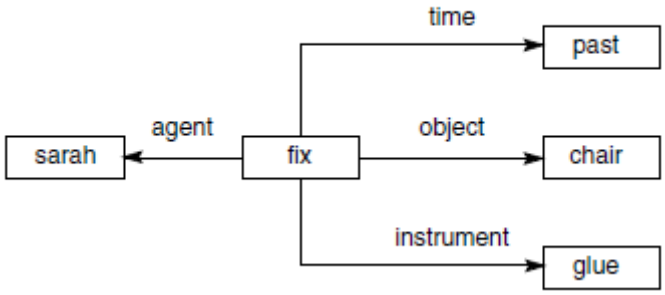
wczesna praca Quillian ustaliła większość istotnych cech formalnego semantycznego formalizmu sieci, takich jak oznaczone łuki i linki, hierarchiczne dziedziczenie i wnioski wzdłuż powiązań asocjacyjnych, okazało się, że jego zdolność do radzenia sobie ze złożonością wielu domen jest ograniczona. Jednym z głównych powodów tej porażki było ubóstwo relacji (powiązań), które uchwyciło głębsze semantyczne aspekty wiedzy. Większość łączy reprezentowała niezwykle ogólne powiązania między węzłami i nie dawała żadnej rzeczywistej podstawy do strukturyzowania relacji semantycznych. Ten sam problem napotyka się przy próbach użycia rachunku predykatu do uchwycenia znaczenia semantycznego. Chociaż formalizm jest bardzo ekspresyjny i może reprezentować prawie każdy rodzaj wiedzy, jest zbyt wolny i nakłada na programistę ciężar konstruowania odpowiednich zestawów faktów i reguł. Znaczna część pracy nad reprezentacjami sieci po Quillian koncentrowała się na zdefiniowaniu bogatszego zestawu etykiet (relacji) linków, które pełniej modelowałyby semantykę języka naturalnego. Wdrażając podstawowe relacje semantyczne języka naturalnego jako część formalizmu, a nie jako część wiedzy domenowej dodanej przez konstruktora systemu, bazy wiedzy wymagają mniej rękodzieła i osiągają większą ogólność i spójność.

Brachman stwierdził:

„Kluczową kwestią jest tutaj izolacja prymitywów dla semantycznych języków sieciowych. Prymitywy języka sieciowego to te rzeczy, które interpreter jest zaprogramowany wcześniej do zrozumienia i które zwykle nie są reprezentowane w samym języku sieci”.

Simmons zaspokoił tę potrzebę standardowych relacji, koncentrując się na strukturze przypadków czasowników angielskich. W tym podejściu zorientowanym na czasowniki, opartym na wcześniejszej pracy Fillmore'a, linki określają role rzeczowników i wyrażeń rzeczownikowych w działaniu zdania. Relacje przypadków obejmują agenta, obiekt, instrument, lokalizację i czas. Zadanie jest reprezentowane jako węzeł czasownika, z różnymi łączami spraw do węzłów reprezentujących innych uczestników akcji. Ta struktura nazywana jest ramką skrzynki. Podczas analizowania zdania program znajduje czasownik i pobiera ramkę sprawy dla tego czasownika ze swojej bazy wiedzy.

Następnie wiąże wartości agenta, obiektu itp. Z odpowiednimi węzłami w ramce sprawy. Przy takim podejściu zdanie „Sarah naprawiła krzesło za pomocą kleju” może być reprezentowane przez sieć na rysunku



Tak więc sam język reprezentacji oddaje dużą część głębokiej struktury języka naturalnego, na przykład związek między czasownikiem a jego podmiotem (relacja agenta) lub ten między czasownikiem a jego przedmiotem. Znajomość struktury przypadków w języku angielskim jest częścią samego formalizmu sieciowego. Po przeanalizowaniu pojedynczego zdania te wbudowane relacje wskazują, że Sarah jest osobą wykonującą naprawę i że klej służy do złożenia krzesła. Zauważ, że te relacje językowe są przechowywane w sposób niezależny od rzeczywistego zdania, a nawet języka, w którym zdanie zostało wyrażone. Podobne podejście zastosowano również w językach sieciowych zaproponowanych

przez Normana i Rumelharta i in. Każdy wysiłek pracował nad stworzeniem pełnego zestawu prymitywów, które mogłyby być użyte do reprezentacji semantycznej struktury wyrażeń języka naturalnego w jednolity sposób. Miały one pomóc w rozumowaniu konstruktów językowych i były niezależne od osobliwości poszczególnych języków lub frazowania. Być może najbardziej ambitną próbą formalnego modelowania głębokiej semantycznej struktury języka naturalnego jest konceptualna teoria zależności Rogera Schanka. Teoria zależności pojęciowej oferuje zestaw czterech prymitywnych konceptualizacji, z których zbudowany jest świat znaczeń. Są równe i niezależne. Oni są:

AKTY działania

Obiekty PP (producenci obrazów)

Modyfikatory działań AA (pomocnicy akcji)

Modyfikatory obiektów PA (pomocnicze obrazy)

Na przykład zakłada się, że wszystkie akcje redukują się do jednego lub więcej pierwotnych AKTÓW. Prymitywy wymienione poniżej są traktowane jako podstawowe składniki działania, a bardziej szczegółowe czasowniki są tworzone przez ich modyfikację i kombinację.

ATRANS przekazuje relację (daje)

PTRANS przekazuje fizyczną lokalizację obiektu (przejdź)

PROPEL przykłada siłę fizyczną do obiektu (push)

MOVE przenieś część ciała według właściciela (kopnięcie)

GRASP złap obiekt przez aktora (chwyć)

POŁKNIĘCIE połknąć przedmiot przez zwierzę (jeść)

EXPEL wydalają z ciała zwierzęcia (płacz)

MTRANS przesyłają informacje mentalne (powiedz)

MBUILD mentalnie tworzy nowe informacje (decyduje)

CONC konceptualizuj lub myśl o pomyśle (pomyśl)

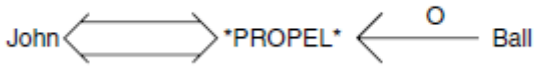
MÓW, wytwarzaj dźwięk (powiedz)

ATTEND focus sens organ (słuchaj)

Te prymitywy są używane do definiowania pojęciowych zależności zależności, które opisują struktury znaczeń, takie jak relacje wielkości liter lub powiązanie obiektów i wartości. Koncepcyjne relacje zależności są koncepcyjnymi regułami składni i stanowią gramatykę znaczących relacji semantycznych. Relacje te można wykorzystać do zbudowania wewnętrznej reprezentacji zdania w języku angielskim. Wykaz podstawowych zależności koncepcyjnych (Schank i Rieger 1974) przedstawiono na rycinie 7.6. Przechwytują one, czują ich twórcy, podstawowe struktury semantyczne języka naturalnego. Na przykład pierwsza zależność pojęciowa na ryc. 7.6 opisuje związek między podmiotem a jego czasownikiem, a trzecia opisuje relację czasownik-obiekt.

PP ⇔ ACT	indicates that an actor acts.
PP ⇔ PA	indicates that an object has a certain attribute.
ACT ←(O) PP	indicates the object of an action.
ACT ← (R) PP / PP	indicates the recipient and the donor of an object within an action.
ACT ← (D) PP / PP	indicates the direction of an object within an action.
ACT ←(1) ⇕	indicates the instrumental conceptualization for an action.
X ⇑ Y	indicates that conceptualization X caused conceptualization Y. When written with a C this form denotes that X COULD cause Y.
PP ⇐ PA2 / PA1	indicates a state change of an object.
PP1 ← PP2	indicates that PP2 is either PART OF or the POSSESSOR OF PP1.

Można je łączyć, aby przedstawić proste zdanie przechodnie, takie jak „John rzuca piłkę”



Na koniec informacje o czasie i trybie mogą zostać dodane do zestawu konceptualizacji. Schank dostarcza listę załączników lub modyfikatorów relacji. Częściowa lista tych obejmuje:

Na koniec informacje o czasie i trybie mogą zostać dodane do zestawu konceptualizacji. Schank dostarcza listę załączników lub modyfikatorów relacji. Częściowa lista tych obejmuje:

p przeszłości

f przyszłość

t przejście

k kontynuuje

Rozpocznij przejście

? badawczy

tf zakończyć przejście

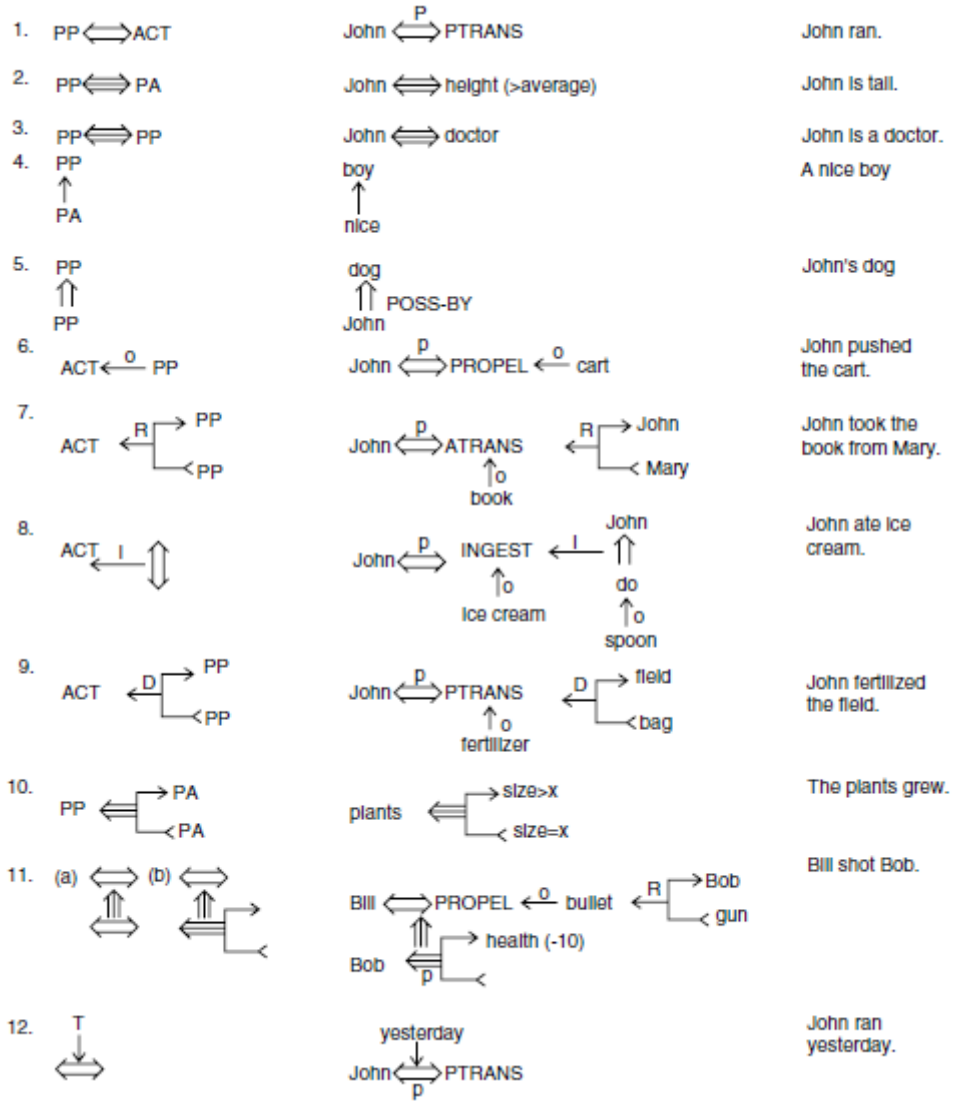
c warunkowe

/ negatywny

zero obecnych

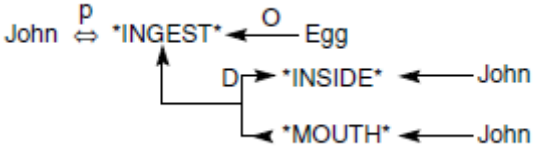
delta? ponadczasowy

Relacje te są konstrukcjami pierwszego poziomu teorii, najprostszymi relacjami semantycznymi, na podstawie których można budować bardziej złożone struktury. Dalsze przykłady, w jaki sposób można zbudować te podstawowe zależności pojęciowe w celu przedstawienia znaczenia prostych zdań w języku angielskim, przedstawiono tu



Na podstawie tych prymitywów angielskie zdanie „John zjadł jajko” jest reprezentowane jako

pokazano na rysunku, gdzie symbole mają następujące znaczenie:



← wskazuje kierunek zależności

⇔ wskazuje relację agent-czasownik

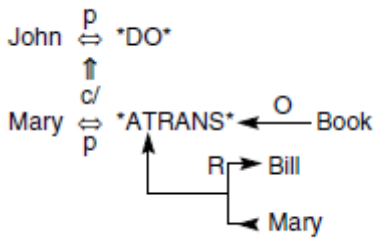
p oznacza czas przeszły

INGEST jest prymitywnym aktem teorii

O relacja obiektu

D wskazuje kierunek obiektu w akcji

Innym przykładem struktur, które można zbudować przy użyciu zależności koncepcyjnych, jest wykres reprezentacyjny dla „Jana uniemożliwił Marii oddanie książki Billowi".



Ten konkretny przykład jest interesujący, ponieważ pokazuje, w jaki sposób można przedstawić przyczynowość. Konceptualna teoria zależności oferuje szereg ważnych korzyści. Dostarczając formalną teorię semantyki języka naturalnego, zmniejsza problemy niejednoznaczności. Po drugie, sama reprezentacja bezpośrednio przechwytuje znaczną część semantyki języka naturalnego, próbując zapewnić kanoniczną formę znaczenia zdań. Oznacza to, że wszystkie zdania o tym samym znaczeniu będą wewnętrznie reprezentowane przez składnie identyczne, a nie tylko semantycznie równoważne wykresy. Ta kanoniczna reprezentacja jest próbą uproszczenia wniosków wymaganych do zrozumienia. Na przykład możemy wykazać, że dwa zdania oznaczają to samo, po prostu dopasowując koncepcyjne wykresy zależności; przedstawienie, które nie zapewniło formy kanonicznej, może wymagać rozległych operacji na grafach o różnej strukturze. Niestety wątpliwe jest, czy program może zostać napisany w celu niezawodnego zredukowania zdań do formy kanonicznej. Jak wykazali Woods (1985) i inni, redukcja do formy kanonicznej jest w sposób oczywisty niemożliwa do obliczenia dla monoidów, rodzaju grupy algebraicznej, która jest znacznie prostsza niż język naturalny. Nie ma również dowodów na to, że ludzie przechowują swoją wiedzę w jakiejkolwiek formie kanonicznej. Inne krytyki tego punktu widzenia, oprócz sprzeciwu wobec ceny obliczeniowej płaconej za zredukowanie wszystkiego do takich prymitywów niskiego poziomu, sugerują, że same prymitywy nie są wystarczające do uchwycenia wielu bardziej subtelnych pojęć, które są ważne w użyciu języka naturalnego. Na przykład reprezentacja „wysoki" w drugim zdaniu nie odnosi się do dwuznaczności tego terminu tak dokładnie, jak dzieje się to w systemach takich jak logika rozmyta . Jednak nikt nie może powiedzieć, że koncepcyjny model zależności nie został gruntownie zbadany i dobrze zrozumiany. Ponad dekada badań prowadzonych przez Schank koncentrowała się na udoskonaleniu i rozszerzeniu modelu. Ważne rozszerzenia pojęciowych zależności obejmują badanie skryptów i pakietów organizacji pamięci lub MOP. Badanie skryptów bada organizację wiedzy w pamięci i rolę, jaką ta organizacja odgrywa w rozumowaniu. MOP stanowiły jeden z elementów wspierających obszary badawcze do projektowania wnioskodawców opartych na analizie przypadków, sekcja 8.3. Konceptualna teoria zależności jest w pełni rozwiniętym modelem semantyki języka naturalnego o spójności celu i szerokim zastosowaniu.

Skrypty

Program rozumienia języka naturalnego musi wykorzystywać dużą ilość wiedzy pomocniczej, aby zrozumieć nawet najprostszą rozmowę (sekcja 15.0). Istnieją dowody na to, że ludzie organizują tę

wiedzę w struktury odpowiadające typowym sytuacjom (Bartlett 1932). Jeśli czytamy artykuł o restauracjach, baseballu lub polityce, rozwiązujemy wszelkie niejasności w tekście w sposób spójny z restauracjami, baseballem lub polityką. Jeśli temat opowieści zmienia się gwałtownie, istnieją dowody na to, że ludzie przerywają czytanie, prawdopodobnie w celu zmiany struktur wiedzy. Trudno jest zrozumieć źle zorganizowaną lub ustrukturyzowaną historię, być może dlatego, że nie możemy łatwo dopasować jej do żadnej z naszych istniejących struktur wiedzy. Mogą również wystąpić błędy w zrozumieniu, gdy temat rozmowy zmienia się nagle, prawdopodobnie dlatego, że nie wiemy, jakiego kontekstu użyć przy rozwiązywaniu odniesień do zaimków i innych niejasności w rozmowie. Skrypt to ustrukturyzowana reprezentacja opisująca stereotypową sekwencję zdarzeń w określonym kontekście. Skrypt został pierwotnie zaprojektowany przez Schanka i jego grupę badawczą jako sposób organizowania pojęciowych struktur zależności w opisy typowych sytuacji. Skrypty są używane w systemach rozumienia języka naturalnego do organizowania bazy wiedzy pod kątem sytuacji, które system ma rozumieć. Większość dorosłych czuje się dość komfortowo (tj. Jako klienci wiedzą, czego się spodziewać i jak się zachować) w restauracji. Spotykają się przy wejściu do restauracji lub widzą znak wskazujący, że powinni wejść dalej i znaleźć stolik. Jeśli menu nie ma przy stole lub nie zostało przedstawione przez kelnera, klient poprosi o jedno. Podobnie, klienci rozumieją procedury zamawiania jedzenia, jedzenia, płacenia i wychodzenia. W rzeczywistości scenariusz restauracji różni się znacznie od innych scenariuszy dotyczących jedzenia, takich jak model „fast food" czy „formalny rodzinny posiłek". W modelu fast food klient wchodzi, ustawia się w kolejce do zamówienia, płaci za posiłek (przed jedzeniem), czeka na tacę z jedzeniem, bierze tacę, próbuje znaleźć czysty stół i tak dalej. Są to dwie różne stereotypowe sekwencje wydarzeń, a każda z nich ma potencjalny scenariusz. Składniki skryptu to: Warunki wejścia lub deskryptory świata, które muszą być prawdziwe, aby skrypt został wywołany. W naszym przykładowym scenariuszu są to otwarta restauracja i głodny klient, który ma trochę pieniędzy. Wyniki lub fakty, które są prawdziwe po zakończeniu działania skryptu; na przykład klient jest pełny i biedniejszy, właściciel restauracji ma więcej pieniędzy. Rekwizyty lub „rzeczy", które wspierają treść scenariusza. Będą to stoły, kelnerzy i menu. Zestaw rekwizytów wspiera rozsądne domyślne założenia dotyczące sytuacji: zakłada się, że restauracja ma stoły i krzesła, chyba że określono inaczej. Role to czynności wykonywane przez poszczególnych uczestników. Kelner przyjmuje zamówienia, dostarcza jedzenie i przedstawia rachunek. Klient zamawia, je i płaci.

Scenariusz. Schank dzieli scenariusz na sekwencję scen, z których każda przedstawia czasowy aspekt scenariusza. W restauracji jest wchodzenie, zamawianie, jedzenie itp. Elementy scenariusza, podstawowe „kawałki" o znaczeniu semantycznym, są reprezentowane za pomocą pojęciowych relacji zależności. Ułożone razem w strukturę przypominającą ramkę, reprezentują sekwencję znaczeń lub sekwencję zdarzeń. Program czyta krótką historię o restauracjach i analizuje ją na wewnętrzną koncepcyjną reprezentację zależności. Ponieważ kluczowe pojęcia w tym wewnętrznym opisie są zgodne z warunkami wejścia skryptu, program wiąże osoby i rzeczy wymienione w historii z rolami i rekwizytami wymienionymi w scenariuszu. Rezultatem jest rozszerzona reprezentacja treści opowieści przy użyciu skryptu do uzupełnienia brakujących informacji i domyślnych założeń. Następnie program odpowiada na pytania dotyczące historii, odwołując się do scenariusza. Skrypt dopuszcza rozsądne domyślne założenia, które są niezbędne do zrozumienia języka naturalnego. Na przykład:

PRZYKŁAD 1

John poszedł wczoraj do restauracji. Zamówił stek. Kiedy zapłacił, zauważył, że kończą mu się pieniądze. Wrócił do domu, odkąd zaczęło padać. Korzystając ze skryptu, system może poprawnie odpowiedzieć na pytania, takie jak: Czy Jan jadł wczoraj obiad (historia tylko to sugerowała)? Czy John użył gotówki czy karty kredytowej? Jak John mógł dostać menu? Co kupił John?

PRZYKŁAD 2

Amy Sue wyszła na lunch. Usiadła przy stole i zadzwoniła do kelnerki, która przyniosła jej menu. Zamówiła kanapkę. Pytania, które można by zadać w tej historii to: Dlaczego kelnerka ma przynieść Amy Sue menu? Czy Amy Sue była w restauracji (przykład tego nie mówi)? Kto zapłacił? Kim była „ona", która zamówiła kanapkę? To ostatnie pytanie jest trudne. Ostatnio wymieniona kobieta to kelnerka, co jest błędnym założeniem dla odniesienia do zaimków. Role skryptów pomagają rozwiązywać takie odniesienia i inne niejasności. Skrypty mogą również służyć do interpretowania nieoczekiwanych wyników lub przerw w działaniu opartym na skryptach. Tak więc na scenie 2 na Rysunku 7.11 jest miejsce wyboru „jedzenie" lub „brak jedzenia" dostarczonego klientowi. Pozwala to na zrozumienie następującego przykładu.

PRZYKŁAD 3

Kate poszła do restauracji. Została zaprowadzona do stolika i zamówiła sushi od kelnerki. Siedziała tam i czekała długo. W końcu się wściekła i wyszła. Pytania, na które można odpowiedzieć w tej historii za pomocą scenariusza restauracji, to: Kim jest „ona", która siedziała i czekała? Dlaczego czekała? Kim była ta „ona", która oszalała i odeszła? Dlaczego się wściekła? Zauważ, że istnieją inne pytania, na które skrypt nie może odpowiedzieć, na przykład dlaczego ludzie się denerwują / wściekają, gdy kelner nie przychodzi szybko? Jak każdy system oparty na wiedzy, skrypty wymagają od inżyniera wiedzy prawidłowego przewidywania wymagana wiedza. Skrypty, takie jak ramki i inne reprezentacje strukturalne, podlegają pewnym problemom, w tym problemowi dopasowania skryptu i problemowi między wierszami. Rozważmy przykład 4, który może wywołać scenariusze restauracji lub koncertów. Wybór jest krytyczny, ponieważ „rachunek" może odnosić się do czeku restauracyjnego lub karty koncertu.

PRZYKŁAD 4

Po drodze na koncert John odwiedził swoją ulubioną restaurację. Był zadowolony z rachunku, ponieważ lubił Mozarta. Ponieważ wybór skryptu zazwyczaj opiera się na dopasowaniu „kluczowych" słów, często trudno jest określić, który z dwóch lub więcej potencjalnych skryptów powinien zostać użyty. Problem dopasowania skryptu jest „głęboki" w tym sensie, że nie istnieje żaden algorytm gwarantujący prawidłowe wybory. Wymaga heurystycznej wiedzy o organizacji świata, a może nawet jakiegoś cofania się; skrypty pomagają tylko w uporządkowaniu tej wiedzy. Problem między wierszami jest równie trudny: nie jest możliwe z góry ustalenie możliwych zdarzeń, które mogą spowodować uszkodzenie skryptu. Na przykład:

PRZYKŁAD 5

Melissa jadła obiad w swojej ulubionej restauracji, kiedy duży kawałek tynku spadł z sufitu i wylądował na randce. Pytania: Czy Melissa jadła sałatkę daktylową? Czy data Melissy była otynkowana? Co ona zrobiła dalej? Jak pokazuje ten przykład, reprezentacje strukturalne mogą być nieelastyczne. Rozumowanie można zamknąć w jednym skrypcie, nawet jeśli nie jest to właściwe. Pakiety organizacji pamięci (MOP) rozwiązują problem nieelastyczności skryptów, przedstawiając wiedzę jako mniejsze komponenty, MOP, wraz z regułami ich dynamicznego łączenia w celu utworzenia schematu, który jest odpowiedni do bieżącej sytuacji. Organizacja wiedzy w pamięci jest szczególnie ważna dla wdrożeń rozumowania opartego na przypadkach, w którym osoba rozwiązująca problem musi skutecznie odzyskać z pamięci odpowiednie wcześniejsze rozwiązanie problemu. Problemy z organizacją i odzyskiwaniem wiedzy są trudne i nieodłączne od modelowania znaczenia semantycznego. Eugene Charniak zilustrował ilość wiedzy potrzebnej do zrozumienia nawet prostych bajek dla dzieci. Rozważmy wypowiedź dotyczącą przyjęcia urodzinowego: Mary dostała dwa latawce na urodziny, więc jeden z nich zabrała z powrotem do sklepu. Musimy wiedzieć o tradycji dawania prezentów na przyjęciu; musimy wiedzieć, czym jest latawiec i dlaczego Mary może nie chcieć dwóch z nich; musimy

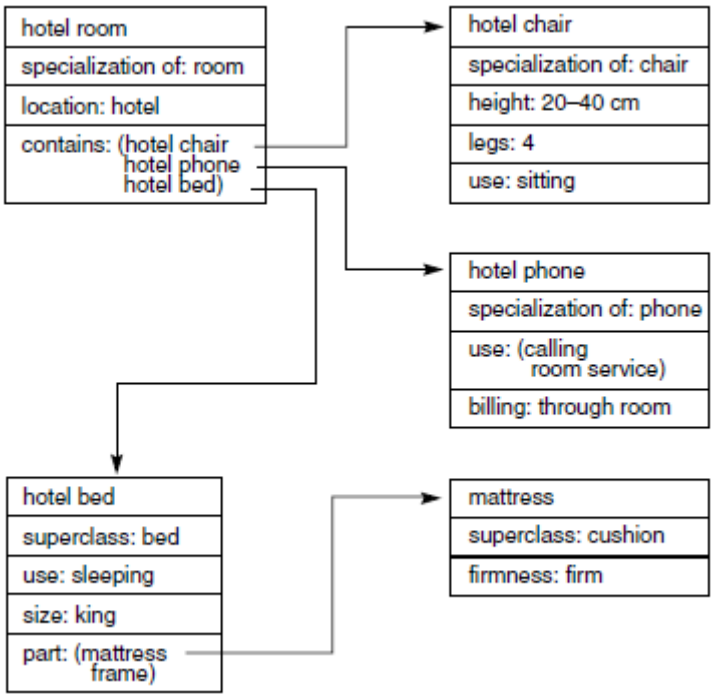
wiedzieć o sklepach i zasadach ich wymiany. Pomimo tych problemów programy wykorzystujące skrypty i inne reprezentacje semantyczne mogą rozumieć język naturalny w ograniczonych dziedzinach. Przykładem takiej pracy jest program, który interpretuje wiadomości przychodzące z serwisów informacyjnych. Korzystając ze skryptów opisujących klęski żywiołowe, zamachy stanu lub inne stereotypowe historie, programy odniosły niezwykły sukces w tej ograniczonej, ale realistycznej dziedzinie

Ramy

Inny schemat reprezentacji, pod wieloma względami podobny do skryptów, który miał na celu uchwycenie w jawnie zorganizowanych strukturach danych niejawnych połączeń informacji w domenie problemowej, został nazwany ramkami. Reprezentacja ta wspiera organizację wiedzy na bardziej złożone jednostki, które odzwierciedlają organizację obiektów w domenie. W artykule z 1975 roku Minsky opisuje ramę:

Oto istota teorii ram: kiedy ktoś napotyka nową sytuację (lub dokonuje istotnej zmiany w swoim spojrzeniu na problem), wybiera z pamięci strukturę zwaną „ramą". Jest to zapamiętana struktura, którą należy dostosować do rzeczywistości, zmieniając szczegóły w razie potrzeby. Według Minsky'ego ramka może być postrzegana jako statyczna struktura danych wykorzystywana do reprezentowania dobrze rozumianych stereotypowych sytuacji. Struktury przypominające ramkę wydają się porządkować naszą własną wiedzę o świecie. Dostosowujemy się do każdej nowej sytuacji, przywołując informacje uporządkowane na podstawie wcześniejszych doświadczeń. Następnie specjalnie dostosowujemy lub poprawiamy szczegóły tych przeszłych doświadczeń, aby przedstawić indywidualne różnice w nowej sytuacji. Każdy, kto przebywał w jednym lub dwóch pokojach hotelowych, nie ma problemu z całkowicie nowymi hotelami i ich pokojami. Można się spodziewać łóżka, łazienki, miejsca do otwierania walizki, telefonu, informacji o cenie i ewakuacji w nagłych wypadkach na tylnej stronie drzwi wejściowych i tak dalej. W razie potrzeby możemy podać szczegóły dotyczące każdego pokoju: kolor zasłon, położenie i użycie włączników światła itp. Są również domyślne informacje dostarczane z hotelem.

rama pokoju: bez prześcieradeł; wezwać sprzątanie; potrzebujesz lodu: spójrz w dół korytarza; i tak dalej. Nie musimy budować zrozumienia dla każdego nowego pokoju hotelowego, do którego wchodzimy. Wszystkie elementy ogólnego pokoju hotelowego są zorganizowane w koncepcyjną strukturę, do której mamy dostęp podczas meldowania się w hotelu; w razie potrzeby dostarczane są dane konkretnego pokoju. Moglibyśmy przedstawić te struktury wyższego poziomu bezpośrednio w sieci semantycznej, organizując ją jako zbiór oddzielnych sieci, z których każda reprezentuje jakąś stereotypową sytuację. Ramki, a także systemy zorientowane obiektowo, zapewniają nam wehikuł dla tej organizacji, przedstawiający jednostki jako obiekty strukturalne z nazwanymi gniazdami i przypisanymi wartościami. W ten sposób ramka lub schemat są postrzegane jako pojedyncza złożona jednostka. Na przykład pokój hotelowy i jego elementy można opisać kilkoma indywidualnymi ramkami. Oprócz łóżka rama może przedstawiać krzesło: oczekiwana wysokość 20 do 40 cm, liczba nóg to 4, wartość domyślna, przeznaczony do siedzenia. Kolejna ramka przedstawia telefon hotelowy: jest to specjalizacja zwykłego telefonu, z tym wyjątkiem, że rozliczenie odbywa się za pośrednictwem pokoju, istnieje specjalny operator hotelowy (domyślnie), a osoba może korzystać z telefonu hotelowego, aby uzyskać posiłki serwowane w pokoju, wykonywania połączeń zewnętrznych i otrzymywania innych usług. Rysunek 7.12 przedstawia ramkę przedstawiającą pokój hotelowy.



Każda pojedyncza ramka może być postrzegana jako struktura danych, pod wieloma względami podobna do tradycyjnego „rekordu", który zawiera informacje istotne dla stereotypowych podmiotów. Gniazda w ramce zawierają informacje takie jak:

1. Informacje identyfikujące ramkę.

2. Relacja tej ramki do innych ramek. „Telefon hotelowy" może być specjalnym przykładem „telefonu", który może być przykładem „urządzenia komunikacyjnego".

3. Deskryptory wymagań dla ramy. Na przykład krzesło ma siedzisko na wysokości od 20 do 40 cm od podłogi, jego oparcie jest wyższe niż 60 cm itd. Te wymagania mogą być wykorzystane do określenia, kiedy nowe przedmioty pasują do stereotypu zdefiniowanego przez ramę.

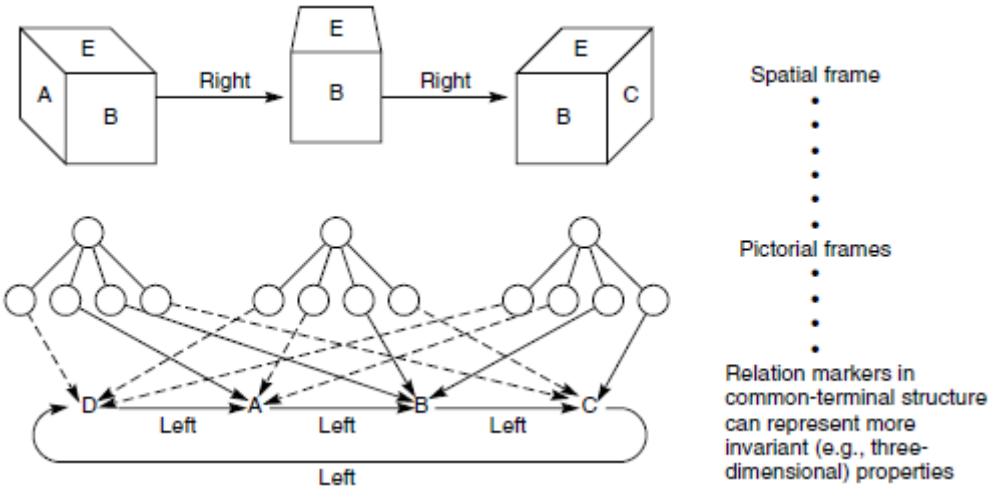
4. Informacje proceduralne dotyczące wykorzystania opisanej konstrukcji. Ważną cechą ramek jest możliwość dołączania instrukcji proceduralnych do slotu.

5. Domyślne informacje ramki. Są to wartości szczelin, które przyjmuje się jako prawdziwe, gdy nie znaleziono dowodów przeciwnych. Na przykład krzesła mają cztery nogi, telefony są przyciskane lub łóżka hotelowe są ścielone przez personel.

6. Informacje o nowej instancji. Wiele szczelin ramek może pozostać nieokreślonych, dopóki nie otrzyma wartości dla konkretnego wystąpienia lub gdy są one potrzebne do jakiegoś aspektu rozwiązywania problemu. Na przykład kolor narzuty może pozostać nieokreślony.

Ramki rozszerzają sieci semantyczne na wiele ważnych sposobów. Chociaż opis ramowy łóżek hotelowych, Rysunek może być odpowiednikiem opisu sieciowego, wersja ramowa znacznie lepiej wyjaśnia, że opisujemy łóżko z jego różnymi atrybutami. W wersji sieciowej jest po prostu zbiór węzłów i jesteśmy zależni więcej na temat naszej interpretacji konstrukcji, aby zobaczyć łóżko hotelowe jako główny opisywany obiekt. Ta umiejętność organizowania naszej wiedzy w takie struktury jest ważnym atrybutem bazy wiedzy.

Ramki ułatwiają hierarchiczne uporządkowanie naszej wiedzy. W sieci każda koncepcja jest reprezentowana przez węzły i łącza na tym samym poziomie specyfikacji. Jednak bardzo często możemy chcieć myśleć o obiekcie jako pojedynczej jednostce do pewnych celów i rozważać tylko szczegóły jego wewnętrznej struktury do innych celów. Na przykład zazwyczaj nie jesteśmy świadomi mechanicznej organizacji samochodu, dopóki coś się nie zepsuje; dopiero wtedy wyciągamy nasz „schemat silnika samochodu" i próbujemy znaleźć problem. Przywiązanie proceduralne jest ważną cechą ramek, ponieważ obsługuje łączenie określonych fragmentów kodu z odpowiednimi elementami w reprezentacji ramek. Na przykład, moglibyśmy chcieć zawrzeć możliwość generowania obrazów graficznych w bazie wiedzy. Język graficzny jest do tego bardziej odpowiedni niż język sieciowy. Używamy przywiązania proceduralnego do tworzenia demonów. Demon to procedura wywoływana jako efekt uboczny innej czynności w bazie wiedzy. Na przykład możemy chcieć, aby system przeprowadzał testy typu lub uruchamiał testy spójności po zmianie określonej wartości gniazda. Systemy ramek obsługują dziedziczenie klas. Szczeliny i domyślne wartości ramki klasy są dziedziczone w całej hierarchii klas / podklas i klas / elementów członkowskich. Na przykład telefon hotelowy może być podklasą zwykłego telefonu, z tym wyjątkiem, że (1) wszystkie wybieranie numeru poza budynkiem przechodzi przez centralę hotelową (do celów rozliczeniowych) i (2) usługi hotelowe mogą być wybierane bezpośrednio. Do wybranych slotów przypisane są wartości domyślne, które mają być używane tylko wtedy, gdy inne informacje nie są dostępne: załóżmy, że pokoje hotelowe są wyposażone w łóżka i dlatego są odpowiednimi miejscami do odwiedzenia, jeśli chcesz spać; jeśli nie wiesz, jak zadzwonić do recepcji hotelu, wpisz „zero"; można założyć, że telefon (brak dowodów przeciwnych) jest przyciskiem. Kiedy tworzona jest instancja klasy, system będzie próbował wypełnić jej szczeliny, wysyłając zapytanie do użytkownika, akceptując wartość domyślną z ramki klasy lub wykonując jakąś procedurę lub demon w celu uzyskania wartości instancji. Podobnie jak w przypadku sieci semantycznych, szczeliny i wartości domyślne są dziedziczone w ramach hierarchii klas / podklas. Oczywiście domyślne informacje mogą spowodować, że opis danych problemu będzie niemonotoniczny, co pozwoli nam przyjąć założenia dotyczące wartości domyślnych, które nie zawsze mogą być poprawne .Praca Minsky'ego na temat widzenia dostarcza przykładu ram i ich zastosowania w domyślnym rozumowaniu: problem rozpoznania, że różne widoki obiektu faktycznie reprezentują ten sam obiekt. Na przykład trzy perspektywy jednej kostki na rysunku wyglądają zupełnie inaczej.



Minsky zaproponował system ramek, który rozpoznaje je jako widoki pojedynczego obiektu, wnioskując o ukrytych stronach jako domyślnych założeniach. Układ ramek na rysunku przedstawia cztery ściany sześcianu. Przerywane linie wskazują, że z tej perspektywy nie widać określonej twarzy. Powiązania między ramkami wskazują relacje między widokami reprezentowanymi przez ramki. Węzły

mogłyby być oczywiście bardziej złożone, gdyby twarze zawierały kolory lub wzory. Rzeczywiście, każda szczelina w jednej ramce może być wskaźnikiem do innej całej ramki. Ponadto, ponieważ podane informacje mogą wypełniać wiele różnych szczelin (powierzchnia E na rysunku), nie ma potrzeby tworzenia nadmiarowości w przechowywanych informacjach.

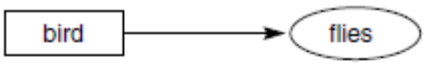
Ramki zwiększają możliwości sieci semantycznych, umożliwiając reprezentowanie złożonych obiektów jako pojedynczej ramki, a nie jako dużej struktury sieciowej. Zapewnia to również bardzo naturalny sposób przedstawiania stereotypowych jednostek, klas, dziedziczenia i wartości domyślnych. Chociaż ramy, takie jak logika i reprezentacje sieciowe, są potężnym narzędziem, wiele problemów związanych z pozyskiwaniem i organizowaniem skomplikowanej bazy wiedzy wciąż musi być rozwiązanych dzięki umiejętnościom i intuicji programisty. Wreszcie, te badania MIT z lat 70. XX wieku, a także podobne prace w Centrum Badawczym Xerox Palo Alto doprowadziły do powstania filozofii projektowania programowania „zorientowanego obiektowo", a także do zbudowania ważnych języków wdrożeniowych, w tym Smalltalk, C ++ i Java.

Grafy koncepcyjne: język sieci

W następstwie wczesnych prac badawczych w dziedzinie sztucznej inteligencji, w ramach których opracowano schematy reprezentacji, stworzono szereg języków sieciowych w celu modelowania semantyki języka naturalnego i innych dziedzin. W tej sekcji szczegółowo przeanalizujemy konkretny formalizm, aby pokazać, jak w tej sytuacji rozwiązano problemy reprezentacji znaczenia. Grafy konceptualne Johna Sowy są przykładem języka reprezentacji sieci. Definiujemy zasady tworzenia i manipulowania grafami pojęciowymi oraz konwencjami do reprezentowania klas, jednostek i związków. W późniejszej sekcji pokazujemy, jak ten formalizm może być używany do reprezentowania znaczenia w rozumieniu języka naturalnego.

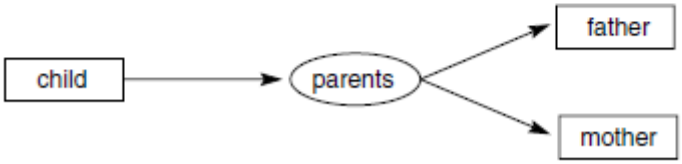
Wprowadzenie do grafów koncepcyjnych

Graf pojęciowy jest grafem skończonym, połączonym, dwudzielnym. Węzły wykresu to albo pojęcia, albo relacje pojęciowe. Grafy koncepcyjne nie używają oznakowanych łuków; zamiast tego węzły relacji pojęciowych reprezentują relacje między pojęciami. Ponieważ grafy konceptualne są dwudzielne, koncepcje mają tylko łuki do relacji i odwrotnie. Na rysunku pies i brąz są węzłami pojęciowymi, a kolor relacją pojęciową. Aby rozróżnić te typy węzłów, przedstawiamy pojęcia jako pudełka, a relacje pojęciowe jako elipsy. Na grafach koncepcyjnych węzły pojęciowe reprezentują konkretne lub abstrakcyjne obiekty w programie świata dyskursu. Konkretne koncepcje, takie jak kot, telefon czy restauracja, charakteryzują się naszą zdolnością do ukształtowania ich obrazu w naszych umysłach. Zwróć uwagę, że konkretne pojęcia obejmują pojęcia ogólne, takie jak kot lub restauracja, a także pojęcia szczegółowe koty i restauracje. Nadal możemy stworzyć obraz ogólnego kota. Pojęcia abstrakcyjne obejmują takie rzeczy, jak miłość, piękno i lojalność, które nie odpowiadają obrazom w naszych umysłach. Węzły relacji pojęciowych wskazują relację obejmującą jedno lub więcej pojęć. Jedną z zalet formułowania grafów pojęciowych jako grafów dwudzielnych, zamiast używania oznaczonych łuków, jest to, że upraszcza to reprezentację relacji dowolnej liczby. Relacja arity n jest reprezentowana przez węzeł relacji pojęciowej mający n łuków, jak pokazano na rysunku



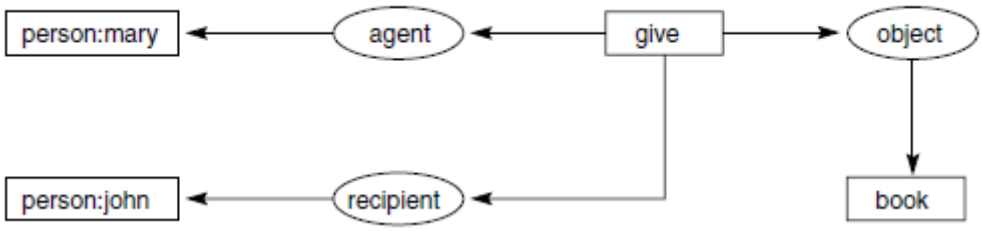
Flies is a 1-ary relation.

Color is a 2-ary relation.



Parents is a 3-ary relation.

Każdy wykres koncepcyjny przedstawia jedną propozycję. Typowa baza wiedzy będzie zawierała szereg takich wykresów. Grafy mogą być dowolnie złożone, ale muszą być skończone. Na przykład jeden wykres na rysunku powyżej przedstawia twierdzenie „Pies ma kolor brązowy". Rysunek poniższy przedstawia nieco bardziej złożony wykres, który przedstawia zdanie „Mary dała Janowi książkę".

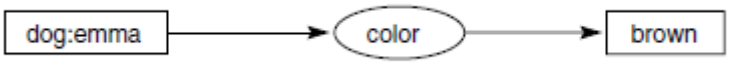


Ten wykres wykorzystuje relacje pojęciowe do reprezentowania przypadków czasownika „dawać" i wskazuje sposób, w jaki grafy pojęciowe są używane do modelowania semantyki języka naturalnego

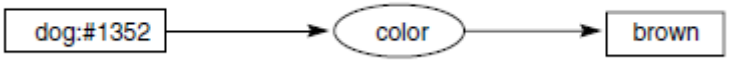
Typy, osoby i imiona

Wielu wczesnych projektantów sieci semantycznych było nieostrożnych przy definiowaniu relacji klasa / element członkowski oraz klasa / podklasa, co powodowało zamieszanie semantyczne. Na przykład relacja między jednostką a jej klasą różni się od relacji między klasą (taką jak pies) a jej nadklasą (mięsożercą). Podobnie, niektóre właściwości należą do jednostek, a inne do samej klasy; przedstawicielstwo powinno stanowić narzędzie do dokonania tego rozróżnienia. Własności futra i lubienia kości należą do poszczególnych psów; klasa „pies" nie ma futra ani niczego nie je. Właściwości, które są odpowiednie dla tej klasy, obejmują jej nazwę i przynależność do taksonomii zoologicznej. Na wykresach pojęciowych każda koncepcja jest unikalną jednostką określonego typu. Każde pudełko koncepcyjne jest oznaczone etykietą typu, która wskazuje klasę lub typ osoby reprezentowane przez ten węzeł. Tak więc pies oznaczony węzłem reprezentuje jakąś osobę tego typu. Typy są zorganizowane w hierarchię. Pies typu jest podtypem drapieżnika, który jest podtypem ssaka itp. Pudełka z tą samą etykietą typu reprezentują koncepcje tego samego typu; jednak te pola mogą, ale nie muszą, reprezentować tę samą indywidualną koncepcję. Każde pudełko koncepcyjne jest oznaczone nazwami

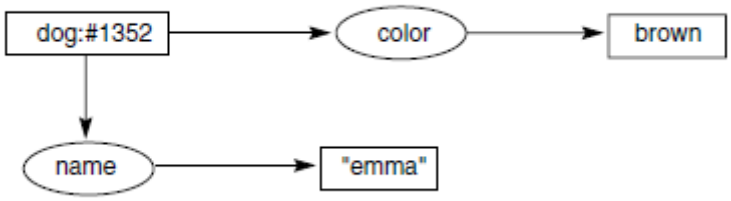
typów i osób. Typ i poszczególne etykiety są oddzielone dwukropkiem „:”. Wykres na rysunku wskazuje, że pies „Emma” jest brązowy.



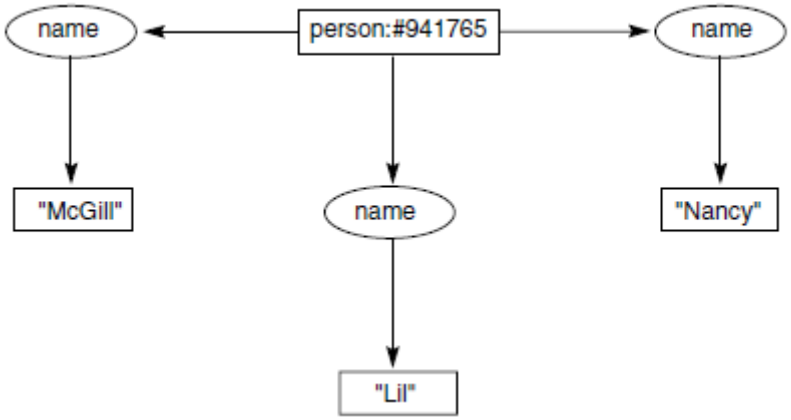
Wykres na rysunku potwierdza, że jakaś nieokreślona jednostka w typie pies ma kolor brązowy.



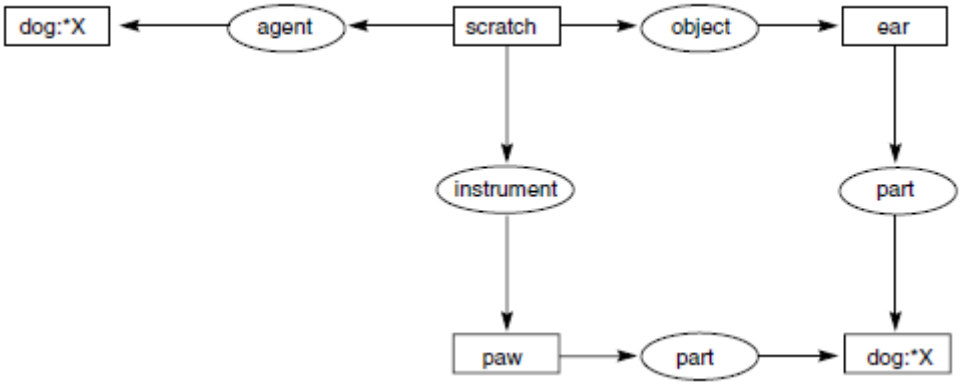
Jeśli nie wskazano osoby, pojęcie to przedstawia nieokreśloną jednostkę tego typu. Grafy koncepcyjne pozwalają nam również wskazać konkretne, ale nienazwane osoby. Unikalny żeton zwany markerem wskazuje każdą osobę w świecie dyskursu. Ten znacznik jest zapisywany jako liczba poprzedzona znakiem #. Znaczniki różnią się od imion tym, że są unikalne: osoby mogą mieć jedno imię, wiele imion lub nie mieć ich wcale, ale mają dokładnie jeden marker. Podobnie, różne osoby mogą mieć to samo imię, ale nie mogą mieć tego samego znacznika. To rozróżnienie daje nam podstawę do radzenia sobie z semantycznymi niejasnościami, które pojawiają się, gdy nadajemy obiektom nazwy. Wykres na rysunku powyżej potwierdza, że konkretny pies, # 1352, jest brązowy. Markery pozwalają nam oddzielić osobę od jej imienia. Jeśli pies nr 1352 nazywa się „Emma”, możemy użyć relacji pojęciowej zwanej imieniem, aby dodać to do wykresu. Wynikiem jest wykres z rysunku 7.18. Nazwa jest ujęta w podwójne cudzysłowy, aby wskazać, że jest to ciąg. Tam, gdzie nie ma niebezpieczeństwa dwuznaczności, możemy uprościć wykres i odnosić się do osoby bezpośrednio po imieniu. Zgodnie z tą konwencją wykres z rysunku jest równoważny wykresowi z rysunku wyżej.



Chociaż często go pomijamy zarówno w swobodnej rozmowie, jak iw formalnych przedstawieniach, to rozróżnienie między jednostką a jej imieniem jest ważne i powinno być poparte językiem reprezentacji. Na przykład, jeśli powiemy, że „Jan” jest imieniem pospolitym wśród mężczyzn, to stwierdzamy, że należy ono raczej do samego imienia niż do osoby o imieniu „Jan”. Dzięki temu możemy przedstawić takie angielskie zdania, jak „Szympans” to nazwa gatunku naczelnych ”. Podobnie możemy chcieć przedstawić fakt, że dana osoba ma kilka różnych imion. Wykres na rysunku przedstawia sytuację opisaną w tekście piosenki: „Nazywała się McGill i nazywała siebie Lil, ale wszyscy znali ją jako Nancy”.



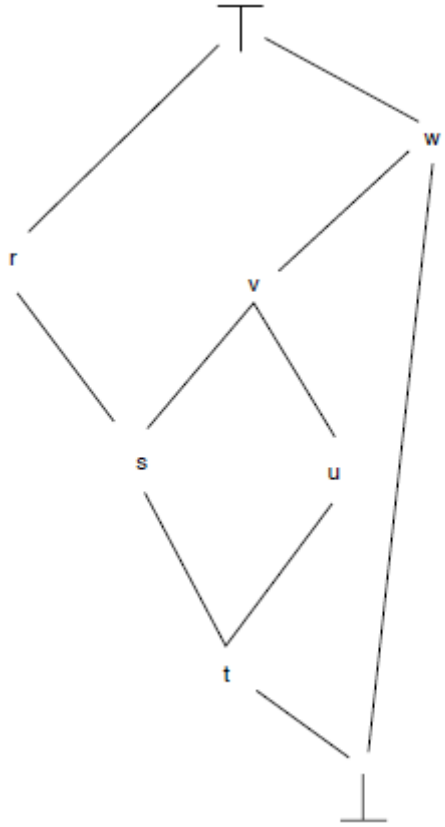
Jako alternatywę dla wskazywania osoby za pomocą jej markera lub nazwiska, możemy również użyć ogólnego markera *, aby wskazać nieokreśloną osobę. Zgodnie z konwencją jest to często pomijane w etykietach pojęć; węzeł, któremu podano tylko etykietę typu, pies, jest odpowiednikiem węzła oznaczonego jako pies: *. Oprócz znacznika ogólnego, wykresy koncepcyjne pozwalają na użycie nazwanych zmiennych. Są one reprezentowane gwiazdką, po której następuje nazwa zmiennej (np. * X lub * foo). Jest to przydatne, jeśli dwa oddzielne węzły mają wskazywać tę samą, ale nieokreśloną osobę. Wykres na rysunku przedstawia stwierdzenie „Pies drapie łapę w ucho”.



Chociaż nie wiemy, który pies drapie swoje ucho, zmienna * X wskazuje, że łapa i ucho należą do tego samego psa, który drapie. Podsumowując, każdy węzeł pojęciowy może wskazywać osobę określonego typu. Ta jednostka jest odniesieniem dla pojęcia. To odniesienie jest wskazane indywidualnie lub ogólnie. Jeśli odniesienie używa indywidualnego markera, pojęcie jest pojęciem indywidualnym; jeśli odniesienie używa ogólnego znacznika, to pojęcie jest ogólne.

Hierarchia typów

Hierarchia typów, jak pokazano poniżej,

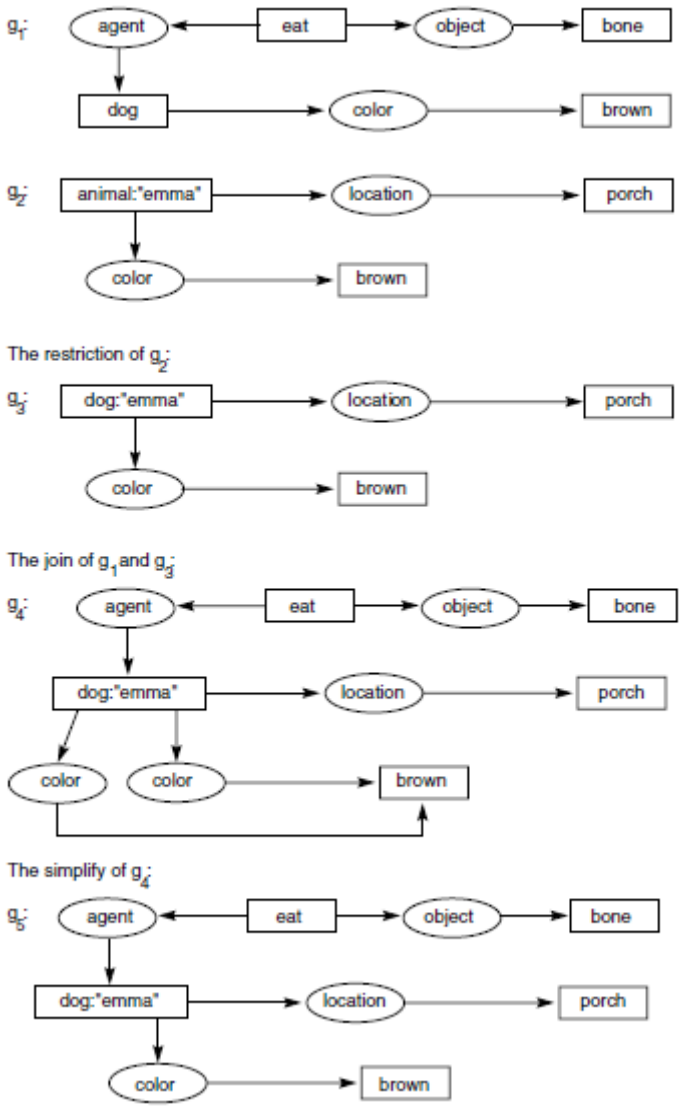


jest częściowym uporządkowaniem zbioru typów, oznaczonych symbolem ≤. Jeśli s i t są typami i $t \leq s$, to mówi się, że t jest podtypem s, a s jest nadtypem t. Ponieważ jest to porządkowanie częściowe, typ może mieć jeden lub więcej nadtypów, a także jeden lub więcej podtypów. Jeśli s, t i u są typami, przy $t \leq s$ i $t \leq u$, to mówi się, że t jest wspólnym podtypem s i u. Podobnie, jeśli $s \leq v$ i $u \leq v$, to v jest wspólnym nadtypem s i u.

Hierarchia typów grafów pojęciowych tworzy kratę, powszechną postać systemu wielokrotnego dziedziczenia. W kracie typy mogą mieć wielu rodziców i dzieci. Jednak każda para typów musi mieć minimalny wspólny nadtyp i maksymalny wspólny podtyp. Dla typów s i u, v jest minimalnym wspólnym nadtypem, jeśli $s \leq v$, $u \leq v$, a dla dowolnego w - wspólnym nadtypem s i u, $v \leq w$. Maksymalny wspólny podtyp ma odpowiednią definicję. Minimalny wspólny typ nadrzędny zbioru typów jest odpowiednim miejscem do definiowania właściwości wspólnych tylko dla tych typów. Ponieważ wiele typów, takich jak emocje i rock, nie ma oczywistych wspólnych nadtypów lub podtypów, konieczne jest dodanie typów, które spełniają te role. Aby hierarchia typów była prawdziwą kratą, grafy koncepcyjne zawierają dwa specjalne typy. Typ uniwersalny, oznaczony literą T, jest nadtypem wszystkich typów. Typ absurdalny, oznaczony ⊥, jest podtypem wszystkich typów.

Uogólnienie i specjalizacja

Teoria grafów pojęciowych obejmuje szereg operacji, które tworzą nowe wykresy z istniejących grafów. Umożliwiają one wygenerowanie nowego wykresu poprzez specjalizację lub uogólnienie istniejącego wykresu, operacje, które są ważne dla reprezentowania semantyki języka naturalnego. Te cztery operacje to kopiowanie, ograniczanie, łączenie i upraszczanie, jak pokazano na rysunku.



Załóżmy, że g_1 i g_2 są grafami pojęciowymi.

Wtedy: Reguła kopiowania pozwala nam utworzyć nowy wykres g, czyli dokładną kopię g_1.

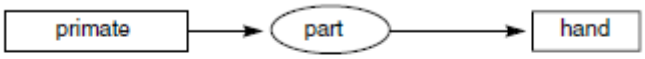
Restrict umożliwia zastąpienie węzłów pojęć na grafie węzłem reprezentującym ich specjalizację. Istnieją dwa przypadki:

1. Jeżeli pojęcie jest oznaczone ogólnym znacznikiem, ogólny znacznik może zostać zastąpiony indywidualnym znacznikiem.

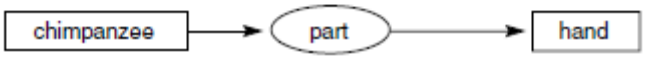
2. Etykietę typu można zastąpić jednym z jej podtypów, jeżeli jest to zgodne z odnośnikiem koncepcji. Na rysunku powyżej możemy zamienić zwierzę na psa.

Reguła łączenia pozwala nam połączyć dwa wykresy w jeden wykres. Jeśli na wykresie s_1 znajduje się węzeł pojęciowy c_1, który jest identyczny z węzłem pojęciowym c_2 w s_2, wówczas możemy utworzyć nowy wykres usuwając c_2 i łącząc wszystkie relacje występujące na c_2 z c_1. Łączenie jest regułą specjalizacji, ponieważ wynikowy wykres jest mniej ogólny niż którykolwiek z jego składników. Jeśli wykres zawiera dwie zduplikowane relacje, to jedną z nich można usunąć wraz ze wszystkimi jej łukami. To jest zasada uproszczenia. Zduplikowane relacje często pojawiają się w wyniku operacji łączenia, jak na wykresie g_4 na powyższym rysunku.

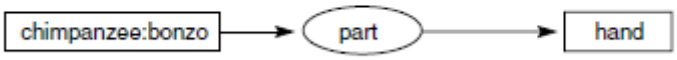
Jednym z zastosowań reguły ograniczającej jest dopasowanie dwóch pojęć, tak aby można było wykonać łączenie. Łącz i ograniczaj razem, zezwalaj na implementację dziedziczenia. Na przykład zastąpienie ogólnego znacznika przez osobę indywidualną implementuje dziedziczenie właściwości typu przez osobę. Zastąpienie etykiety typu etykietą podtypu definiuje dziedziczenie między klasą a nadklasą. Łącząc jeden graf z drugim i ograniczając pewne węzły pojęciowe, możemy zaimplementować dziedziczenie różnych właściwości. Rysunek pokazuje, jak szympansy dziedziczą właściwość posiadania ręki od naczelnych, zastępując tabliczkę znamionową jej podtypem.



A conceptual graph



Inheritance of a property by a subclass



Inheritance of a property by an individual

Pokazuje również, jak osoba, Bonzo, dziedziczy tę właściwość, tworząc wystąpienie ogólnej koncepcji. Podobnie, możemy użyć sprzężeń i ograniczeń, aby wdrożyć wiarygodne założenia, które odgrywają rolę w powszechnym zrozumieniu języka. Jeśli powiedziano nam, że „Mary i Tom poszli razem na pizzę", automatycznie przyjmujemy szereg założeń: jedli okrągły włoski chleb pokryty serem i sosem pomidorowym. Zjedli to w restauracji i musieli mieć za to jakiś sposób zapłaty. To rozumowanie można przeprowadzić za pomocą połączeń i ograniczeń. Tworzymy koncepcyjny wykres zdania, a następnie łączymy go z wykresami pojęciowymi (z naszej bazy wiedzy) dla pizz i restauracji. Wynikowy wykres pozwala założyć, że jedli sos pomidorowy i zapłacili rachunek. Dołącz i ogranicz to zasady specjalizacji. Definiują częściowe uporządkowanie na zbiorze grafów dających się wyprowadzić. Jeśli graf g_1 jest specjalizacją g_2, to możemy powiedzieć, że g_2 jest uogólnieniem g_1. Hierarchie uogólnień są ważne w reprezentacji wiedzy. Oprócz zapewnienia podstawy dla dziedziczenia i innych zdroworozsądkowych schematów rozumowania, hierarchie uogólnień są wykorzystywane w wielu metodach uczenia się, co pozwala nam na przykład skonstruować uogólnione stwierdzenie na podstawie określonej instancji szkoleniowej. Te reguły nie są regułami wnioskowania. Nie gwarantują, że prawdziwe wykresy zostaną wyprowadzone z prawdziwych wykresów. Na przykład w ograniczeniu wykresu na rysunku wcześniejszym wynik może nie być prawdziwy; Emma może być kotem. Podobnie, łączący przykład na rysunku powyżej również nie zachowuje prawdy: pies na ganku i pies, który zjada kości, mogą być różnymi zwierzętami. Te operacje są kanonicznymi regułami tworzenia i chociaż tak jest, nie zachowują prawdy, mają subtelną, ale ważną właściwość zachowania „sensowności". Jest to ważna gwarancja, gdy używamy grafów koncepcyjnych do implementacji rozumienia języka naturalnego. Rozważ trzy zdania:

Albert Einstein sformułował teorię względności.

Albert Einstein gra centrum Los Angeles Lakers.

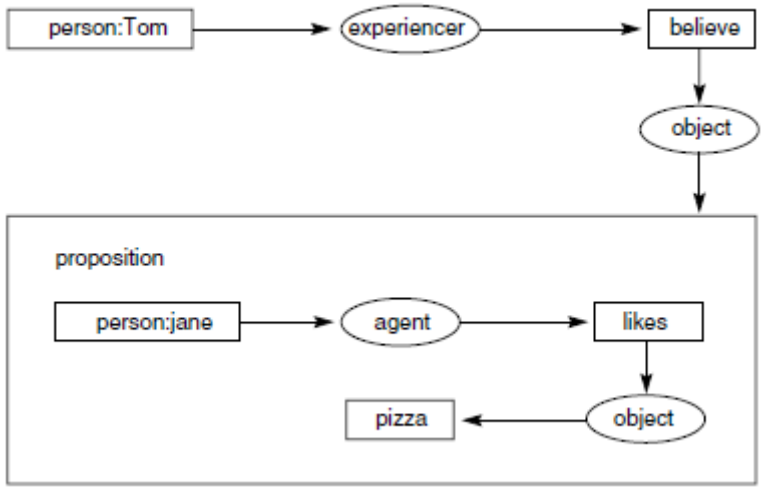
Grafiki koncepcyjne to żółte latające lody popsicles.

Pierwsze z tych zdań jest prawdziwe, a drugie fałszywe. Trzecie zdanie jest jednak bez znaczenia; chociaż gramatycznie poprawne, nie ma sensu. Zdanie drugie, choć fałszywe, ma znaczenie. Mogę

sobie wyobrazić Alberta Einsteina na boisku do koszykówki. Reguły kanoniczne wymuszają ograniczenia znaczenia semantycznego; to znaczy, nie pozwalają nam na tworzenie bezsensownych wykresów ze znaczących wykresów. Chociaż nie są to zdrowe reguły wnioskowania, kanoniczne reguły tworzenia stanowią podstawę dla większości prawdopodobnych rozumowań przeprowadzanych w rozumieniu języka naturalnego i rozumowaniu zdroworozsądkowym.

Węzły propozycji

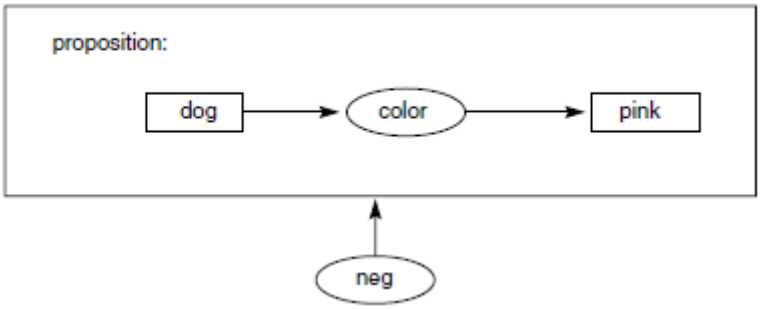
Oprócz używania grafów do definiowania relacji między obiektami na świecie, możemy również chcieć zdefiniować relacje między zdaniami. Rozważ na przykład stwierdzenie „Tomek uważa, że Jane lubi pizzę". „Wierzy" to relacja, której argumentem jest zdanie. Grafy konceptualne obejmują typ pojęcia, propozycję, która przyjmuje zbiór grafów pojęciowych jako odniesienie i pozwala nam zdefiniować relacje obejmujące zdania. Pojęcia zdaniowe zaznaczono jako ramkę zawierającą kolejny graf pojęciowy. Te koncepcje zdań mogą być używane z odpowiednimi relacjami do reprezentowania wiedzy o zdaniach. Rysunek przedstawia schemat koncepcyjny powyższego stwierdzenia dotyczącego Jane, Toma i pizzy.



Relacja doznawania jest luźno analogiczna do relacji sprawcy, ponieważ łączy podmiot i czasownik. Łącze doświadczającego jest używane ze stanami przekonań opartymi na założeniu, że są one czymś, czego się raczej doświadcza niż robi. Rysunek powyższy pokazuje, jak grafy pojęciowe z węzłami zdań mogą być używane do wyrażania modalnych koncepcji wiedzy i przekonań. Logiki modalne zajmują się różnymi sposobami przedstawiania twierdzeń: sądzone, uznawane za możliwe, prawdopodobnie lub koniecznie prawdziwe, zamierzone w wyniku działania lub kontrfaktyczne.

Grafy pojęciowe i logika

Korzystając z grafów pojęciowych, możemy z łatwością przedstawić koncepcje łączące, takie jak „Pies jest duży i głodny", ale nie ustaliliśmy żadnego sposobu przedstawiania negacji lub dysjunkcji. Nie zajęliśmy się również kwestią zmiennej ilościowej. Możemy zaimplementować negację za pomocą pojęć zdaniowych i jednoargumentowej operacji zwanej neg. neg przyjmuje jako argument pojęcie zdania i twierdzi, że to pojęcie jest fałszywe. Na wykresie koncepcyjnym na rysunku zastosowano neg, aby przedstawić stwierdzenie „Nie ma różowych psów"



Używając negacji i koniunkcji, możemy tworzyć wykresy, które reprezentują dysjunktywne twierdzenia zgodnie z regułami logiki. Aby to uprościć, możemy również zdefiniować relację lub, która przyjmuje dwa zdania i reprezentuje ich rozłączność. Na grafach konceptualnych zakłada się, że pojęcia ogólne są kwantyfikowane egzystencjalnie. Na przykład ogólna koncepcja psa na wykresie na rysunku wcześniejszym w rzeczywistości reprezentuje zmienną kwantyfikowaną egzystencjalnie. Ten wykres odpowiada wyrażeniu logicznemu:

$$\exists X \exists Y (dog(X) \wedge color(X,Y) \wedge brown(Y))$$

Używając negacji i kwantyfikacji egzystencjalnej, możemy również przedstawić kwantyfikację uniwersalną. Na przykład wykres na rysunku 7.25 można uznać za reprezentujący twierdzenie logiczne:

$$\forall X \forall Y (\neg (dog(X) \wedge color(X,Y) \wedge pink(Y))).$$

Grafy pojęciowe są równoważne rachunku predykatów w swojej sile wyrazu. Jak sugerują te przykłady, istnieje proste mapowanie z grafów pojęciowych do notacji rachunku predykatów. Algorytm zaczerpnięty z Sowy, służący do zamiany grafu pojęciowego g na wyrażenie rachunku predykatów to:

1. Przypisz unikalną zmienną, x_1, x_2,. . . , x_n, do każdego z n ogólnych pojęć w g.

2. Przypisz unikalną stałą do każdej indywidualnej koncepcji w g. Ta stała może po prostu być nazwą lub znacznikiem używanym do wskazania odnośnika pojęcia.

3. Każdy węzeł pojęciowy należy przedstawić za pomocą jednoargumentowego predykatu o tej samej nazwie, co typ tego węzła i którego argumentem jest zmienna lub stała w danym węźle.

4. Reprezentuj każdą n-arną koncepcyjną relację w g jako n-arny predykat, którego nazwa jest taka sama jak relacja. Niech każdy argument predykatu będzie zmienną lub stałą przypisaną do odpowiedniego węzła pojęciowego powiązanego z tą relacją.

5. Weź koniunkcję wszystkich zdań atomowych utworzonych w 3 i 4. To jest treść wyrażeń rachunku predykatów. Wszystkie zmienne w wyrażeniu są określane ilościowo egzystencjalnie.

Chociaż możemy przeformułować grafy pojęciowe w składnię rachunku predykatów, grafy koncepcyjne obsługują również szereg mechanizmów wnioskowania specjalnego przeznaczenia, takich jak łączenie i ograniczanie, które zwykle nie są częścią rachunku predykatów.

Przedstawiliśmy składnię grafów pojęciowych i zdefiniowaliśmy operację ograniczenia jako sposób implementacji dziedziczenia. Nie zbadaliśmy jeszcze pełnego zakresu operacji i wniosków, które można wykonać na tych wykresach, ani nie zajęliśmy się problemem definiowania pojęć i relacji potrzebnych w takich dziedzinach, jak język naturalny.

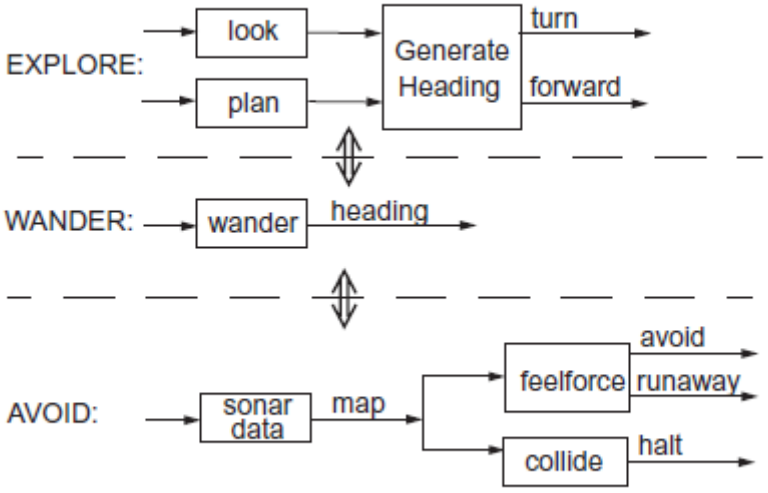
Alternatywne reprezentacje i ontologie

W ostatnich latach badacze sztucznej inteligencji nadal kwestionowali rolę jawnej reprezentacji w inteligencji. Oprócz podejścia koneksjonistycznego i emergentnego, kolejne wyzwanie dla roli tradycyjnej reprezentacji pochodzi z pracy Rodneya Brooksa w MIT .Projektując robota-eksploratora Brooks kwestionuje potrzebę jakiegokolwiek scentralizowanego schematu reprezentacji, a dzięki swojej architekturze subsumpcji próbuje pokazać, jak ogólna inteligencja może wyewoluować z niższych i wspierających form inteligencji. Drugie podejście do problemu reprezentacji jawnych i statycznych pochodzi z prac Melanie Mitchell i Douglasa Hofstadtera z Indiana University. Architektura Copycat to ewoluująca sieć, która dostosowuje się do relacji znaczeniowych, które napotyka poprzez eksperymenty ze światem zewnętrznym. Wreszcie, podczas tworzenia rozwiązań do rozwiązywania problemów w złożonych środowiskach często konieczne jest zastosowanie wielu różnych schematów reprezentacji. Każdy schemat reprezentacji można nazwać ontologią. Konieczne staje się wtedy zbudowanie komunikacji i innych połączeń między tymi różnymi reprezentacjami, aby wspierać zarządzanie wiedzą, komunikację, tworzenie kopii zapasowych i inne aspekty rozwiązywania problemów. Przedstawiamy te kwestie i możliwe rozwiązania pod ogólnym hasłem technologii zarządzania wiedzą. Technologia ta może również wspierać rozwiązywanie problemów oparte na agentach, jak przedstawiono w sekcji 7.4.

Architektura Subsumption Brooksa

Brooks przypuszcza i podaje przykłady poprzez swoje roboty, że inteligentne zachowanie nie pochodzi z bezcielesnych systemów, takich jak dowódcy twierdzeń, ani nawet z tradycyjnych systemów eksperckich .Inteligencja, twierdzi Brooks, jest wynikiem interakcji między odpowiednio zaprojektowanym systemem a jego otoczeniem. Ponadto Brooks jest zwolennikiem poglądu, że inteligentne zachowanie wyłania się z interakcji architektur zorganizowanych prostszych zachowań: jego architektury podporządkowania. Architektura subsumpcji jest warstwową kolekcją programów obsługi zadań. Każde zadanie jest realizowane przez skończoną maszynę stanów, która nieustannie odwzorowuje wejście oparte na percepcji na wyjście zorientowane na działanie. To odwzorowanie jest realizowane za pomocą prostych zestawów warunek → reguł produkcji akcji . Reguły te w dość ślepy sposób, to znaczy bez wiedzy o stanie globalnym, określają, jakie działania są odpowiednie do aktualnego stanu tego podsystemu. Brooks dopuszcza pewne informacje zwrotne do systemów niższego poziomu. Rysunek 7.26, zaadaptowany z Brooks, przedstawia trójwarstwową architekturę subsumpcji.

Każda warstwa składa się z sieci o stałej topologii prostych maszyn o skończonych stanach, z których każda ma kilka stanów, jeden lub dwa rejestry wewnętrzne, jeden lub dwa wewnętrzne zegary oraz dostęp do prostych urządzeń obliczeniowych, na przykład w celu obliczenia sum wektorów. Te maszyny o skończonym stanie działają asynchronicznie, wysyłając i odbierając komunikaty o stałej długości przez przewody. Nie ma centralnego umiejscowienia kontroli. Raczej każda maszyna skończonych stanów jest sterowana danymi przez otrzymywane komunikaty. Nadejście wiadomości lub wygaśnięcie okres czasu powoduje zmianę stanu maszyn. Nie ma dostępu do danych globalnych ani do żadnych dynamicznie tworzonych łączy komunikacyjnych. Oznacza to, że nie ma globalnej kontroli. Rysunek przedstawia podzbiór funkcji architektury trójwarstwowej, która obsługiwała wczesnego robota.



Robot miał wokół siebie pierścień z dwunastu czujników sonarowych. W każdej sekundzie czujniki te dawały dwanaście radialnych pomiarów głębokości. Najniższa warstwa architektury podporządkowania, UNIKAJ, implementuje zachowanie, które zapobiega uderzaniu przez robota w obiekty, niezależnie od tego, czy są one statyczne, czy poruszające się. Dane sonaru oznaczone maszyną emitują natychmiastową mapę, która jest przekazywana w celu zderzenia i siły działania, które z kolei są w stanie wygenerować komunikaty zatrzymania dla maszyny skończonej odpowiedzialnej za prowadzenie robota do przodu. Po aktywowaniu feelforce jest w stanie wygenerować niekontrolowaną ucieczkę lub uniknąć instrukcji dotyczących unikania obiektów i zagrożeń.

Sieć najniższego poziomu maszyn skończonych w architekturze generuje wszystkie zatrzymania i unika instrukcji dla całego systemu. Kolejna warstwa, WANDER, sprawia, że system się porusza, generując losowy kurs dla robota co około dziesięć sekund. Maszyna UNIKAJ (niższego poziomu) przejmuje kierunek z WANDERA i łączy go z siłami obliczonymi przez architekturę UNIKAĆ. WANDER wykorzystuje wynik do stłumienia zachowań niższego poziomu, zmuszając robota do poruszania się w kierunku zbliżonym do tego, co zdecydował wędrowiec, ale jednocześnie omijając przeszkody. Wreszcie, jeśli maszyny turn i forward (najwyższy poziom architektury), będą tłumić wszelkie nowe impulsy wysyłane z WANDERA. Najwyższy poziom EXPLORE sprawia, że robot eksploruje swoje otoczenie, szuka odległych miejsc i stara się do nich dotrzeć, planując ścieżkę. Ta warstwa jest w stanie tłumić instrukcje wędrówki i obserwuje, w jaki sposób dolna warstwa zmienia kierunek robota z powodu przeszkód. Koryguje te rozbieżności i utrzymuje koncentrację robota na swoim celu, hamując zachowanie wędrówki, ale pozwalając na unikanie obiektów najniższego poziomu, aby kontynuować swoją funkcję. Gdy wystąpią odchylenia generowane na niższym poziomie, wywołania poziomu EXPLORE planują utrzymanie systemu zorientowanego na cel. Głównym punktem architektury subsumpcji Brook jest to ,że system nie wymaga scentralizowanego rozumowania symbolicznego i podejmuje wszystkie swoje działania bez przeszukiwania możliwych kolejnych stanów. Chociaż skończona maszyna stanów oparta na zachowaniu generuje sugestie dotyczące działań w oparciu o swój własny bieżący stan, system globalny działa w oparciu o interakcje systemów znajdujących się pod nim. Przedstawiona właśnie trójwarstwowa architektura pochodzi z wczesnego projektu Brooksa - wędrującego robota poszukującego celu. Niedawno jego grupa badawcza zbudowała złożone systemy z kolejnymi warstwami (Brooks 1991a, Brooks i Stein 1994). Jeden system jest w stanie wędrować po laboratorium MIT Robotics w poszukiwaniu pustych aluminiowych puszek po napojach na biurkach ludzi. Wymaga

to warstw do odkrywania biur, szukania biurek i rozpoznawania puszek po napojach. Kolejne warstwy są w stanie prowadzić ramię robota, aby zebrać te puszki do recyklingu. Brooks twierdzi, że zachowanie najwyższego poziomu pojawia się w wyniku projektowania i testowania poszczególnych warstw architektury niższego poziomu. Projekt spójnych zachowań końcowych, wymagający zarówno komunikacji międzywarstwowej, jak i międzywarstwowej, jest odkrywany w drodze eksperymentu. Pomimo tej prostoty projektu, architektura subsumpcji sprawdziła się z powodzeniem w kilku zastosowaniach. Pozostaje wiele ważnych pytań dotyczących architektury subsumpcji i powiązanych podejść do projektowania systemów sterowania:

1. Na każdym poziomie systemu występuje problem wystarczalności informacji. Ponieważ na każdym poziomie czysto reaktywne maszyny stanu podejmują decyzje na podstawie informacji lokalnych, trudno jest zrozumieć, jak takie decyzje mogą uwzględniać informacje nie na tym poziomie. Z definicji będzie krótkowzroczny.

2. Jeśli nie ma absolutnie żadnej „wiedzy" lub „modelu" całego środowiska, w jaki sposób ograniczony wkład w sytuację lokalną może być wystarczający do określenia globalnie akceptowalnych działań? Jak można osiągnąć spójność na najwyższym poziomie?

3. W jaki sposób czysto reaktywny składnik o bardzo ograniczonym stanie może uczyć się od swojego otoczenia? Na pewnym poziomie w systemie musi istnieć stan wystarczający do stworzenia mechanizmów uczenia się, jeśli cały agent ma być nazywany inteligentnym.

4. Występuje problem ze skalą. Chociaż Brooks i jego współpracownicy twierdzą, że zbudowali architektury podporządkowania składające się z sześciu, a nawet dziesięciu warstw, jakie zasady projektowania pozwalają na skalowanie go do dalszych interesujących zachowań, tj. Do dużych, złożonych systemów?

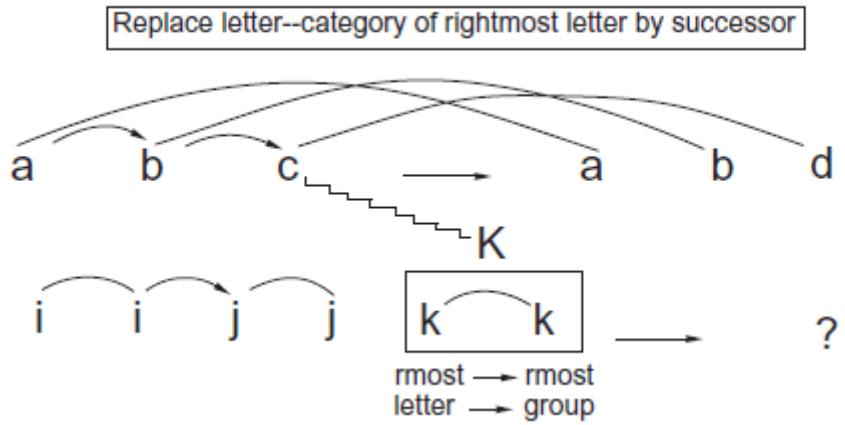
Na koniec musimy zadać pytanie, czym jest „emergencja"? Czy to magia? Na obecnym etapie ewolucji nauki „pojawienie się" wydaje się być słowem opisującym każde zjawisko, dla którego nie możemy jeszcze rozliczyć się w inny sposób. Mówi się nam, żebyśmy budowali systemy i testowali je na świecie, a one wykażą inteligencję. Niestety bez dalszego projektu instrukcji, emergencja staje się po prostu słowem opisującym to, czego jeszcze nie możemy zrozumieć. W rezultacie bardzo trudno jest określić, w jaki sposób możemy wykorzystać tę technologię do budowy coraz bardziej złożonych systemów. W następnej sekcji opiszemy naśladowcę, czyli architekturę hybrydową zdolną poprzez eksplorację do odkrycia i wykorzystania niezmienności znalezionych w dziedzinie problemowej.

Naśladowca

Często słyszaną krytyką tradycyjnych schematów reprezentacji sztucznej inteligencji jest to, że są one statyczne i nie mogą odzwierciedlać dynamicznej natury procesów myślowych i inteligencji. Kiedy na przykład człowiek dostrzega nową sytuację, często uderza go związki z już znanymi lub analogicznymi sytuacjami. W rzeczywistości często zauważa się, że ludzka percepcja jest zarówno oddolna, to znaczy stymulowana przez nowe wzorce w środowisku, jak i odgórna, zapośredniczona przez to, co agent spodziewa się dostrzec. Copycat to architektura rozwiązywania problemów zbudowana przez Melanie Mitchell (1993) jako praca doktorska pod kierunkiem Douglasa Hofstadtera na Uniwersytecie Indiana. Copycat opiera się na wielu technikach reprezentacji, które ją poprzedzały, w tym na tablicach, sieciach semantycznych, sieciach koneksjonistycznych i systemach klasyfikacyjnych. Jest to również zgodne z podejściem Brooksa do rozwiązywania problemów jako aktywnej interwencji w domenie problemowej. Jednak w przeciwieństwie do Brooksa i sieci koneksjonistycznych, naśladowca wymaga globalnego „państwa", aby być częścią rozwiązania problemu. Po drugie, reprezentacja jest ewoluującą cechą tego stanu. Copycat obsługuje semantyczny mechanizm podobny do sieci, który rośnie i zmienia

się wraz z ciągłym doświadczeniem w swoim środowisku. Pierwotnym problemem dla naśladowcy było postrzeganie i budowanie prostych analogii. W tym sensie opiera się na wcześniejszych pracach Evansa i Reitmana. Przykładami tej domeny są uzupełnianie wzorców: gorąco jest do zimna tak wysokie, jak {ściana, niskie, mokre, trzymaj} lub niedźwiedź jest świnią jak krzesło {stopa, stół, kawa, truskawka}. Copycat pracował również nad znalezieniem odpowiednich uzupełnień dla alfabetycznych wzorców ciągów, takich jak: abc is to abd jak ijk? lub znowu, abc jest abd tak, jak iijjkk?, Rysunek 7.27. Copycat składa się z trzech elementów: obszaru roboczego, siatki i pakietu kodowego. W interakcjach tych trzech pośredniczy pomiar temperatury. Temperatura oddaje stopień organizacji percepcji w systemie i kontroluje stopień losowości używanej przy podejmowaniu decyzji. Wyższe temperatury odzwierciedlają fakt, że istnieje niewiele informacji, na podstawie których można by podejmować decyzje, a zatem są one bardziej losowe. Spadek temperatury wskazuje, że system buduje konsensus, a niska temperatura wskazuje, że pojawia się odpowiedź i odzwierciedla „zaufanie" programu do tego rozwiązania. Obszar roboczy jest strukturą globalną, podobną do tablicy w rozdziale 6.3, służącą do tworzenia struktur, które mogą być przeglądane przez inne komponenty systemu. W tym sensie jest również podobny do obszaru wiadomości w systemie klasyfikacyjnym Hollanda (1986). Miejsce pracy jest gdzie

struktury percepcyjne są zbudowane hierarchicznie na górze wejścia (trzy ciągi symboli alfabetycznych na rysunku) i dają możliwe stany dla obszaru roboczego, z wiązaniami (strzałkami) zbudowanymi między powiązanymi składnikami łańcuchów.



Slipnet odzwierciedla sieć pojęć lub potencjalnych skojarzeń dla składników analogii. Jeden widok na slipnet jest dynamicznie odkształcalną siecią semantyczną, której każdy węzeł ma poziom aktywacji. Łącza w sieci mogą być oznaczane przez inne węzły. W zależności od poziomu aktywacji węzłów etykietowania, połączone węzły rosną lub kurczą się. W ten sposób system zmienia stopień powiązania między węzłami w zależności od kontekstu. Rozpowszechnianie aktywacji między węzłami jest zalecane między węzłami, które (w obecnym kontekście) są ściślej powiązane. Koderack jest probabilistyczną kolejką opartą na priorytetach, zawierającą zestawy kodów. Codele to małe fragmenty wykonywalnego kodu zaprojektowane do interakcji z obiektami w obszarze roboczym i próba dalszego rozwoju jakiejś małej części ewoluującego rozwiązania lub, prościej, do zbadania różnych aspektów przestrzeni problemu. Ponownie, kodety są bardzo podobne do indywidualnych klasyfikatorów systemu Hollanda. Węzły w slipnet generują zestawy kodów i wysyłają je do pakietu kodowego, gdzie mają prawdopodobieństwo wykonania. System równolegle utrzymuje fragmenty kodu, które konkurują o możliwość znalezienia i zbudowania struktur w obszarze roboczym. Te fragmenty kodu odpowiadają węzłom, z których pochodzą. Jest to działanie odgórne, polegające na

poszukiwaniu większej liczby przykładów rzeczy, które już wzbudziły zainteresowanie. Codele działają również oddolnie, identyfikując i budując na podstawie relacji już istniejących w obszarze roboczym. W miarę budowania struktur w obszarze roboczym, miara aktywacji jest dodawana do węzłów, które wygenerowały zestawy kodów, które zbudowały struktury. Dzięki temu zachowanie systemu w kontekście obecnego stanu problemu / rozwiązania może wpływać na jego przyszłe zachowanie. Wreszcie temperatura służy jako mechanizm sprzężenia zwrotnego obliczający „spójność" między strukturami zbudowanymi w przestrzeni roboczej. Przy niewielkiej spójności, czyli niewielu strukturach, które dają obietnicę rozwiązania, uprzedzenia probabilistycznych wyborów dokonywanych przez system są mniej ważne: jeden wybór jest tak samo przydatny jak inny. Kiedy spójność jest wysoka, a wewnętrznie spójne rozwiązanie ewoluuje, przy wyborze probabilistycznych bardzo ważne stają się uprzedzenia: dokonane wybory są ściślej powiązane z ewoluującym rozwiązaniem. Istnieje kilka kolejnych projektów w świecie naśladowców, które testują architekturę w bogatszych warunkach. Hofstadter i jego uczniowie (Marshall 1999) kontynuują modelowanie analogicznych zależności. Lewis i Luger (2000) rozszerzyli architekturę naśladowców do wykorzystania jako system sterowania dla robota mobilnego. Domena robotyki zapewnia naśladowcom konkretne środowisko, które wspiera interakcje ukierunkowane na uczenie się. Na podstawie interakcji robota z jego „światem" generowana jest mapa przestrzeni, a ścieżki są planowane i ponownie analizowane.

Wiele reprezentacji, ontologii i usług wiedzy

Wiele złożonych aplikacji AI wymaga zaprojektowania wielu schematów reprezentacji, z których każdy jest skonfigurowany dla określonych beczek, a następnie zintegrowania z platformą oprogramowania zorientowaną na użytkownika. Prostym tego przykładem może być stworzenie wirtualnej przestrzeni spotkań. Różni uczestnicy ze swoich odległych lokalizacji wchodzili w interakcje ze sobą w tej wirtualnej „przestrzeni". Oprócz technologii głosowej wirtualna przestrzeń spotkań wymagałaby przesyłania strumieniowego wideo, udostępniania tekstu i prezentacji, dostępu do nagrań i tak dalej. Jednym z podejść do tego zadania byłoby zaadaptowanie różnych gotowych komponentów, a następnie zintegrowanie ich w ramach usługi wiedzy, która wspiera wymagane interakcje. Opisana właśnie automatyczna usługa wirtualnych spotkań jest przykładem integracji wielu schematów reprezentacji w celu świadczenia usługi. Często używamy terminu ontologia do scharakteryzowania schematu reprezentacji określonego składnika tej usługi. Wynikowa usługa wiedzy wymaga integracji tych komponentów. Ontologia wywodzi się od greckiego słowa, formy czasownika „być", co oznacza byt lub istnienie. Zatem termin odnosi się do komponentów modułu oprogramowania i struktury wspierającej interakcje tych komponentów, to znaczy do ich funkcji lub „bytu składowego" jako część modułu. Gdy moduły są tworzone w celu zbudowania usługi wiedzy, każdy z modułów musi być odpowiednio połączony, aby utworzyć płynną usługę. Mówimy, że ontologie każdego modułu są łączone, aby stworzyć nową, bardziej złożoną ontologię, która obsługuje pełną obsługę. Aby stworzyć nową usługę wiedzy, taką jak oprogramowanie do obsługi wirtualnego spotkania, ontologie każdego z jej komponentów muszą zostać zrozumiane i często ujawnione do integracji. Obecnie istnieje wiele języków ontologicznych zaprojektowanych specjalnie do tworzenia usług wiedzy i innych powiązanych produktów oprogramowania. Należą do nich rodzina języków Owl (Mika i in. 2004), a także open source Java api SOFA, Simple Ontology Framework, (patrz http://sopha.projects.semwebcentral.org/).

W rzeczywistości istnieje wiele sposobów wykorzystania informacji ontologicznej we współczesnych komputerach, niektóre proste, jak zoptymalizowany dostęp do wiedzy przechowywanej w tradycyjnej bazie danych, a inne całkiem nowe. Biorąc pod uwagę znaczenie WWW, wyrafinowane roboty indeksujące i wyszukiwarki internetowe są krytycznymi narzędziami wsparcia. Chociaż większość obecnych pająków internetowych stosuje wyszukiwanie do płaskich, bezpośrednio połączonych plików tekstowych, można sobie wyobrazić bardziej wyrafinowany robot, który może przechodzić przez klasy

i łącza poszczególnych złożonych reprezentacji, a także „interpretować” bardziej przydatne elementy, w tym wykresy i obrazy. Chociaż pełna „interpretacja obrazu” pozostaje daleko poza naszym obecnym stanem możliwości, jej stworzenie pomoże wspierać nasze pragnienia stworzenia bardziej „semantycznej” sieci. Możemy chcieć zeskanować nasze zdjęcia z wakacji, aby znaleźć na przykład wszystko, co ma na sobie określoną osobę lub jakiekolwiek zwierzę. W szerszej sytuacji w sieci WWW możemy chcieć nie tylko śledzić i znajdować artykuły w gazetach, które opisują katastrofy naturalne, ale także lokalizować wszelkie obrazy przedstawiające te katastrofy. Obecnie często jesteśmy ograniczeni do wyszukiwania podpisów ilustracji lub innych (tekstowych) materiałów opisowych, aby uzyskać wskazówkę co do ich treści. Oprócz celu, jakim jest interpretacja danych graficznych i obrazowych, pożądane jest również zlokalizowanie danych tekstowych na określone tematy i sporządzenie podsumowania ich treści. Na przykład, możemy chcieć wyszukać artykuły badawcze online w określonej dziedzinie i przekazać ich zawartość. Chociaż jest to kolejny aktualny temat badawczy w dziedzinie rozumienia języka naturalnego, rozdział 15, jego realizacja pozostaje na granicy naszych umiejętności „semantycznych” opartych na komputerach. Obecnie robimy takie rzeczy, jak wyszukiwanie kilku słów kluczowych, które mogą wskazywać na treść artykułu i być może drukujemy jego streszczenie, ale poza tym nasze umiejętności podsumowywania są ograniczone. Inna usługa internetowa może być potrzebna do wyszukania ogłoszeń „potrzebnych pomocy”. Może zostać uruchomiony w celu wyszukania profesjonalistów zajmujących się oprogramowaniem, na przykład ze znajomością języka Java i C++ oraz chętnych do zlokalizowania się w określonym mieście.

Ten robot sieciowy musiałby wiedzieć coś o tego typu reklamach, w tym o faktach, że zwykle zawiera nazwę firmy i jej lokalizację. Poza umiejętnościami wymaganymi do pracy, te ogłoszenia często opisują potrzebne doświadczenie wraz z możliwymi zarobkami i świadczeniami. W związku z tym serwis internetowy dotyczący zatrudnienia musi mieć wiedzę o strukturach ontologicznych używanych w projektowaniu ogłoszeń o zatrudnieniu i mieć możliwość dostępu do nich. Przykład takiej usługi informacyjnej przedstawiamy w punkcie 15.5. Prawdopodobnie najważniejszym zastosowaniem usług wiedzy opartych na ontologii jest jednak zorientowane na agenta rozwiązywanie problemów. Agenci z definicji mają być autonomiczni, niezależni, elastyczni w realizacji zadań, zlokalizowani w określonych sytuacjach i zdolni do odpowiedniej komunikacji. Rozwiązywanie problemów za pomocą agentów jest również silnie rozproszone w środowiskach, na przykład może być wymagane do zbudowania inteligentnej usługi internetowej. Stworzenie określonych umiejętności agentów może być postrzegane jako kwestia budowania i łączenia ontologii. W następnej sekcji poświęconej rozwiązywaniu problemów za pomocą agentów rozważymy bardziej rozproszony i zorientowany na komponenty widok reprezentacji.

(7.4) Rozproszone i oparte na agentach rozwiązywanie problemów

Społeczność naukowców zajmujących się sztuczną inteligencją w latach 80. miała dwa ważne wnioski i konsekwencje dla przyszłych badań w analizie roli reprezentacji w inteligentnym rozwiązywaniu problemów. Pierwszą była tradycja badawcza społeczności „Distributed Artificial Intelligence” lub „DAI”. Pierwsze warsztaty DAI odbyły się na MIT w 1980 r. W celu omówienia zagadnień związanych z inteligentnym rozwiązywaniem problemów za pomocą systemów składających się z wielu rozwiązań. W tamtym czasie społeczność DAI zdecydowała, że nie interesują jej kwestie równoległości niskiego poziomu, takie jak sposób dystrybucji przetwarzania na różne maszyny lub zrównoleglanie złożonych algorytmów. Ich celem było raczej zrozumienie, w jaki sposób rozproszone osoby rozwiązujące problemy mogą być skutecznie koordynowane w celu inteligentnego rozwiązywania problemów. W rzeczywistości istniała nawet wcześniejsza historia przetwarzania rozproszonego w sztucznej inteligencji, z wykorzystaniem i koordynacją aktorów i demonów oraz projektowaniem systemów tablicowych, jak pokazano w sekcji 6.3. Drugie spostrzeżenie badawcze dotyczące sztucznej inteligencji

z lat 80, dotyczyło Rodneya Brooksa i jego grupy z MIT. Wyzwanie Brooksa dla tradycyjnego poglądu społeczności sztucznej inteligencji na reprezentację i rozumowanie miało wiele ważnych konsekwencji (sekcja 7.3.1). Po pierwsze, przypuszczenie, że inteligentne rozwiązywanie problemów nie wymaga scentralizowanego magazynu wiedzy manipulowanej przez jakiś ogólny schemat wnioskowania, doprowadziło do pojęcia rozproszonych i kooperatywnych modeli inteligencji, w których każdy element rozproszonej reprezentacji był odpowiedzialny za swój własny składnik proces rozwiązywania problemów. Po drugie, fakt, że inteligencja jest umiejscowiona i aktywna w kontekście określonych zadań, pozwala rozwiązującemu problem przenieść aspekty procesu rozwiązywania problemów do samego środowiska. Pozwala to, na przykład, indywidualnemu rozwiązującemu na zajęcie się bieżącym zadaniem, a jednocześnie nie ma żadnej wiedzy o postępach w kierunku rozwiązania w ramach ogólnej domeny problemu. Tak więc, powiedzmy, jeden agent internetowy może sprawdzać informacje o stanie magazynowym, podczas gdy inny agent sprawdza zdolność kredytową klienta, przy czym obaj agenci nie są świadomi podejmowania decyzji na wyższym szczeblu, na przykład, czy zezwolić na zakup, czy nie. Oba te akcenty badawcze z lat osiemdziesiątych zwróciły uwagę na obecne zainteresowanie projektowaniem i wykorzystaniem inteligentnych agentów.

Rozwiązywanie problemów zorientowane na agenta: definicja

Zanim przejdziemy dalej do omówienia badań agentowych, zdefiniujemy, co rozumiemy przez „agent", „system oparty na agentach" i „system wieloagentowy". Istnieją jednak problemy, ponieważ wiele różnych grup w społeczności badawczej agentów ma różne opinie na temat tego, na czym dokładnie polega rozwiązywanie problemów za pomocą agentów. Nasza definicja i dyskusja opiera się na rozszerzeniu pracy Jennings, Sycara i Wooldridge oraz Lewis i Luger. Dla nas system wieloagentowy to program komputerowy z rozwiązaniami problemów umieszczonymi w interaktywnych środowiskach, z których każdy jest zdolny do elastycznych, autonomicznych, ale społecznie zorganizowanych działań, które mogą, ale nie muszą, być skierowane na z góry określone cele lub cele. Dlatego cztery kryteria inteligentnego systemu agentów obejmują rozwiązania problemów z oprogramowaniem, które są zlokalizowane, autonomiczne, elastyczne i społeczne. Umiejscowienie inteligentnego agenta oznacza, że otrzymuje dane wejściowe ze środowiska, w którym jest aktywny, i może również wpływać na zmiany w tym środowisku. Przykłady środowisk dla zlokalizowanych agentów obejmują internet, gry lub robotykę. Konkretnym przykładem może być piłkarz biorący udział w zawodach ROBOCUP, w których agent musi odpowiednio współdziałać z piłką i przeciwnikiem bez pełnej wiedzy na temat lokalizacji, wyzwań i sukcesów innych graczy w walce. To usytuowanie można porównać z bardziej tradycyjnymi rozwiązaniami problemów sztucznej inteligencji, takimi jak planista STRIPS, lub system ekspercki MYCIN, które utrzymują centralnie zlokalizowaną i wyczerpującą wiedzę o domenach aplikacji. System autonomiczny to taki, który może wchodzić w interakcje ze swoim otoczeniem bez bezpośredniej interwencji innych agentów. Aby to zrobić, musi mieć kontrolę nad własnymi działaniami i stanem wewnętrznym. Niektórzy autonomiczni agenci mogą również uczyć się na podstawie swoich doświadczeń, aby z czasem poprawić swoją wydajność (patrz uczenie maszynowe, część IV). Na przykład w Internecie autonomiczny agent mógłby przeprowadzić kontrolę uwierzytelnienia karty kredytowej niezależnie od innych kwestii związanych z transakcją zakupu. W przykładzie ROBOCUP agent mógłby podać piłkę koledze z drużyny lub kopnąć ją do bramki w zależności od indywidualnej sytuacji.

Elastyczny agent reaguje zarówno inteligentnie, jak i proaktywnie, w zależności od bieżącej sytuacji. Reagujący agent odbiera bodźce ze swojego otoczenia i reaguje na nie we właściwy i terminowy sposób. Proaktywny agent nie tylko reaguje na sytuacje w swoim otoczeniu, ale jest również w stanie planować, być oportunistycznym, ukierunkowanym na cel i mieć odpowiednie alternatywy dla różnych sytuacji. Na przykład agent kredytowy mógłby wrócić do użytkownika z niejednoznacznymi wynikami

lub znaleźć inną agencję kredytową, jeśli jedna z możliwości nie jest wystarczająca. Agent piłkarski może zmienić swoją trajektorię w zależności od wzorca wyzwania przeciwnika. Wreszcie agent jest społeczny, który może wchodzić w interakcje, w razie potrzeby, z innym oprogramowaniem lub agentami ludzkimi. W końcu agent jest tylko częścią złożonego procesu rozwiązywania problemów. Interakcje agenta społecznego są zorientowane na cele większego systemu wieloagentowego. Ten społeczny wymiar systemu agentowego musi odnosić się do wielu trudnych sytuacji. Należą do nich: Jak różni agenci mogą składać oferty o podzadanie podczas rozwiązywania problemów? W jaki sposób agenci mogą komunikować się ze sobą, aby ułatwić realizację zadań na wyższym poziomie systemu - w przykładzie ROBOCUP może to być zdobycie gola. W jaki sposób jeden agent może wspierać cele innego agenta, na przykład radzić sobie z kwestiami bezpieczeństwa zadania internetowego? Wszystkie te pytania dotyczące wymiaru społecznego są przedmiotem ciągłych badań. Opisaliśmy podstawy tworzenia systemów wieloagentowych. Systemy wieloagentowe są idealne do reprezentowania problemów, które obejmują wiele metod rozwiązywania problemów, wiele punktów widzenia i wiele podmiotów. W tych dziedzinach systemy wieloagentowe oferują zalety rozproszonego i współbieżnego rozwiązywania problemów, a także zalety wyrafinowanych schematów interakcji. Przykłady interakcji obejmują współpracę w pracy w kierunku wspólnego celu, koordynacja w organizowaniu działań związanych z rozwiązywaniem problemów, aby uniknąć szkodliwych interakcji i wykorzystać korzystne możliwości, oraz negocjowanie ograniczeń podproblemowych, tak aby uzyskać akceptowalną wydajność. To elastyczność tych interakcji społecznych odróżnia systemy wieloagentowe od bardziej tradycyjnego oprogramowania i zapewnia siłę i ekscytację paradygmatowi agenta. W ostatnich latach termin system wieloagentowy odnosi się do wszystkich typów systemów oprogramowania składających się z wielu półautonomicznych komponentów. Rozproszony system agentowy rozważa, w jaki sposób dany problem może zostać rozwiązany przez szereg modułów (agentów), które współpracują, dzieląc i udostępniając wiedzę o problemie i jego ewoluującym rozwiązaniu. Badania w systemach wieloagentowych koncentrują się na zachowaniach zbiorów, niekiedy już istniejących, autonomicznych agentów, mających na celu rozwiązanie danego problemu. System wieloagentowy może być również postrzegany jako luźno powiązana sieć osób rozwiązujących problemy, które wspólnie pracują nad problemami wykraczającymi poza zakres jakiegokolwiek agenta z osobna . Rozwiązania problemów w systemie wieloagentowym, oprócz tego, że są autonomiczne, mogą mieć również heterogeniczną konstrukcję. Na podstawie analizy Jennings, Sycara i Wooldridge'a istnieją cztery ważne cechy wieloagentowego rozwiązywania problemów. Po pierwsze, każdy agent ma niepełne informacje i niewystarczające możliwości rozwiązania całego problemu, przez co może cierpieć z powodu ograniczonego punktu widzenia. Po drugie, nie ma globalnego kontrolera systemu do rozwiązywania całego problemu.

Po trzecie, wiedza i dane wejściowe dotyczące problemu są również zdecentralizowane, a po czwarte, procesy wnioskowania są często asynchroniczne. Co ciekawe, tradycyjni programiści zorientowani obiektowo często nie widzą niczego nowego w systemie opartym na agentach. Biorąc pod uwagę względne właściwości czynników i przedmiotów, może to być zrozumiałe. Obiekty definiuje się jako systemy obliczeniowe ze stanem hermetyzowanym, mają związane z tym stanem metody wspierające interakcje w środowisku i komunikują się poprzez przekazywanie wiadomości. Różnice między obiektami a agentami obejmują fakt, że obiekty rzadko wykazują kontrolę nad własnym zachowaniem. Nie postrzegamy agentów jako wywołujących metody względem siebie, ale raczej jako żądających wykonania czynności serwisowych. Ponadto agenci są projektowani tak, aby zachowywać się elastycznie, tj. Reagować, proaktywnie i zachowywać się społecznie. Wreszcie, agenci wchodzący w interakcje często mają własne, indywidualne wątki kontroli. Wszystkie te różnice nie wskazują, że zorientowane obiektowo języki programowania, takie jak C++, Java czy CLOS, nie oferują odpowiedniego medium do budowania systemów agentowych; wręcz przeciwnie, ich moc i

elastyczność czynią je idealnymi do tego zadania. Aby uściślić idee z poprzedniej sekcji, opiszemy następnie kilka domen aplikacji, w których rozwiązywanie problemów za pomocą agentów jest odpowiednie. Zawieramy również odniesienia do badań w tych obszarach problemowych. Produkcja Dziedzinę produkcji można modelować jako hierarchię obszaru pracy. Mogą istnieć obszary robocze do frezowania, toczenia, malowania, montażu i tak dalej. Te obszary robocze można następnie pogrupować w podsystemy produkcyjne, przy czym każdy podsystem pełni funkcję w ramach większego procesu produkcyjnego. Te podsystemy produkcyjne mogą być następnie zgrupowane w jednej fabryce. Większy podmiot, firma, może wtedy kontrolować aspekty każdej z tych fabryk, na przykład w celu zarządzania zamówieniami, zapasami, poziomami powielania, zyskami i tak dalej. Odniesienia do produkcji opartej na agentach obejmują prace związane z sekwencjonowaniem produkcji , operacje produkcyjne oraz wspólne projektowanie produktów. Sterowanie automatyczne Ponieważ kontrolery procesów są systemami autonomicznymi, reaktywnymi i często rozproszonymi, nie jest zaskakujące, że modele agentów mogą być ważne. Prowadzone są badania w zakresie kontroli systemów transportowych, kontroli statków kosmicznych, akceleratorów wiązek, kontroli ruchu lotniczego i inni. Telekomunikacja Systemy telekomunikacyjne to duże rozproszone sieci współpracujących ze sobą komponentów, które wymagają monitorowania i zarządzania w czasie rzeczywistym. Systemy oparte na agentach są wykorzystywane do sterowania i zarządzania siecią, transmisji i przełączania oraz usługach. Systemy transportowe Systemy ruchu są prawie z definicji rozproszone, usytuowane i autonomiczne. Zastosowania obejmują koordynację osób dojeżdżających do pracy i samochodów do wspólnych przejazdów oraz kooperatywne planowanie transportu. Zarządzanie informacją Bogactwo, różnorodność i złożoność informacji dostępnych dla współczesnego społeczeństwa jest niemal przytłaczająca. Systemy agentowe dają możliwość inteligentnego zarządzania informacjami, szczególnie w Internecie. Wydaje się, że zarówno czynnik ludzki, jak i organizacja informacji sprzeciwiają się wygodnemu dostępowi do informacji. Dwa krytyczne zadania agenta to filtrowanie informacji, tylko niewielka część informacji, do których mamy dostęp, do których faktycznie chcemy, oraz zbieranie informacji, zadanie gromadzenia i nadawania priorytetów fragmentom informacji, których faktycznie potrzebujemy. Zastosowania obejmują WEBMATE, filtrowanie poczty elektronicznej, asystenta przeglądania sieci oraz eksperckiego agenta lokalizującego.

Handel elektroniczny. Wydaje się, że obecnie handel elektroniczny kieruje się działalnością człowieka: decydujemy, kiedy kupować lub sprzedawać, jakie są odpowiednie ilości i cenę, a nawet jakie informacje mogą być odpowiednie w danym momencie. Z pewnością handel jest domeną odpowiednią dla modeli agentów. Chociaż pełny rozwój agentów handlu elektronicznego może nastąpić w przyszłości, kilka systemów jest już dostępnych. Na przykład programy mogą teraz podejmować wiele decyzji dotyczących kupna i sprzedaży na giełdzie na podstawie wielu różnych rozproszonych informacji. Opracowywanych jest kilka systemów agentowych do zarządzania portfelem, pomocy w zakupach oraz interaktywnych katalogów.

Gry interaktywne i teatr Postacie z gier i teatru zapewniają bogate interaktywne symulowane środowisko. Ci agenci mogą rzucić nam wyzwanie w grach wojennych, scenariuszach zarządzania finansami, a nawet w sporcie. Agenci teatralni odgrywają role analogiczne do swoich ludzkich odpowiedników i mogą oferować iluzję życia podczas pracy z sytuacjami emocjonalnymi, symulowania nagłych przypadków medycznych lub szkolenia do różnych zadań. Badania w tej dziedzinie obejmują gry komputerowe, osobowości interaktywne oraz negocjacje. Oczywiście istnieje wiele innych dziedzin, w których podejście oparte na agentach jest odpowiednie. Chociaż technologia agentów oferuje wiele potencjalnych korzyści w zakresie inteligentnego rozwiązywania problemów, nadal istnieje wiele wyzwań do pokonania. Poniższe pytania badawcze są oparte na pomysłach Jenningsa i

inni oraz Bond i Gasser. Jak możemy systematycznie formalizować, dekomponować i przydzielać problemy agentom? Co więcej, w jaki sposób odpowiednio zsyntetyzujemy ich wyniki?

Jak możemy umożliwić agentom komunikację i interakcję? Jakie języki i protokoły komunikacji są dostępne? Co i kiedy jest odpowiednia komunikacja?

W jaki sposób możemy zapewnić spójne działanie agentów przy podejmowaniu działań lub podejmowaniu decyzji?

Jak mogą zająć się efektami nielokalnymi i uniknąć szkodliwych interakcji z czynnikami?

W jaki sposób poszczególni agenci mogą reprezentować i uzasadniać działania, plany i wiedzę innych agentów, aby koordynować z nimi działania? W jaki sposób agenci mogą wnioskować o stanie swoich skoordynowanych procesów?

W jaki sposób można rozpoznać i skoordynować różne punkty widzenia i sprzeczne intencje między agentami?

W jaki sposób można rozpoznać i uniknąć szkodliwego zachowania całego systemu, takiego jak działanie chaotyczne lub oscylacyjne?

W jaki sposób można przydzielić i zarządzać ograniczonymi zasobami, zarówno pojedynczego agenta, jak i całego systemu?

Wreszcie, jakie są najlepsze platformy sprzętowe i technologie oprogramowania do obsługi i rozwoju systemów agentowych?

Epilog

Przeanalizowaliśmy wiele głównych alternatyw reprezentacji wiedzy, w tym użycie logiki, reguł, sieci semantycznych i ramek. Rozważaliśmy również systemy bez scentralizowanej bazy wiedzy lub schematu rozumowania ogólnego przeznaczenia.

Na koniec rozważaliśmy rozproszone rozwiązywanie problemów za pomocą agentów. Wyniki dokładnych badań obejmują lepsze zrozumienie zalet i ograniczeń każdego z tych podejść do reprezentacji. Niemniej jednak trwa debata na temat względnej naturalności, skuteczności i stosowności każdego podejścia. Zamykamy ten rozdział krótkim omówieniem kilku ważnych zagadnień z obszaru reprezentacji wiedzy. Pierwszą z nich jest wybór i ziarnistość symboli atomowych do przedstawiania wiedzy. Dziedziną mapowania są obiekty na świecie; Zasięgiem są obiekty obliczeniowe w bazie wiedzy. Charakter pierwiastków atomowych w języku w dużej mierze determinuje to, co można opisać o świecie. Na przykład, jeśli „samochód” jest najmniejszym atomem reprezentacji, to system nie może wnioskować o silnikach, kołach lub jakiejkolwiek części składowej samochodu. Jednakże, jeśli atomy odpowiadają tym częściom, wówczas może być wymagana większa struktura, aby przedstawić „samochód” jako jedną koncepcję, co może wiązać się z kosztami w zakresie efektywności manipulowania tą większą strukturą. Inny przykład kompromisu w wyborze symboli pochodzi z pracy w zrozumieniu języka naturalnego. Programy, które używają pojedynczych słów jako elementów znaczenia, mogą mieć trudności z przedstawieniem złożonych pojęć, które nie mają denotacji jednowyrazowej. Istnieje również trudność w rozróżnieniu różnych znaczeń tego samego słowa lub różnych słów o tym samym znaczeniu. Jednym z podejść do tego problemu jest użycie semantycznych prymitywów lub niezależnych od języka jednostek pojęciowych, jako podstawy do przedstawienia znaczenia języka. Chociaż ten punkt widzenia unika problemu używania pojedynczych słów jako jednostek znaczeniowych, wiąże się z innymi kompromisami: wiele słów wymaga złożonych struktur dla ich definicji; Ponadto, opierając się na niewielkim zestawie prymitywów, trudno jest

wyrazić wiele subtelnych różnic, takich jak pchnięcie kontra pchnięcie lub wrzask vs. krzyk. Wyczerpalność to właściwość bazy wiedzy, która jest wspomagana przez odpowiednią reprezentację. Odwzorowanie jest wyczerpujące w odniesieniu do właściwości lub klasy obiektów, jeśli wszystkie wystąpienia odpowiadają jawnemu elementowi reprezentacji. Zakłada się, że mapy geograficzne są wyczerpujące do pewnego poziomu szczegółowości; mapa z brakującym miastem lub rzeką nie byłaby dobrze postrzegana jako narzędzie nawigacyjne. Chociaż większość baz wiedzy nie jest wyczerpująca, wyczerpanie informacji w odniesieniu do pewnych właściwości lub obiektów jest pożądanym celem. Na przykład możliwość założenia, że reprezentacja jest wyczerpująca, może pozwolić projektantowi na zignorowanie możliwych skutków problemu z ramką. Kiedy opisujemy problemy jako stan świata, który jest zmieniany przez serię działań lub wydarzeń, te działania lub zdarzenia na ogół zmieniają tylko kilka elementów opisu; program musi być w stanie wywnioskować skutki uboczne i ukryte zmiany w opisie świata. Problem reprezentacji skutków ubocznych działań nazywany jest problemem ramy. Na przykład robot układający ciężkie pudła na ciężarówce może być zmuszony do skompensowania obniżenia platformy ciężarówki ze względu na wagę każdego nowego pudełka. Jeśli przedstawienie jest wyczerpujące, nie będzie żadnych nieokreślonych skutków ubocznych, a problem z ramką skutecznie znika. Trudność problemu z ramami wynika z tego, że niemożliwe jest zbudowanie kompletnej bazy wiedzy dla większości dziedzin. Język reprezentacji powinien pomóc programiście w podjęciu decyzji, jaką wiedzę można bezpiecznie pominąć i pomóc uporać się z konsekwencjami tego pominięcia. (Rozdział 8.4 omawia problem ram w planowaniu). Z wyczerpywalnością związana jest plastyczność lub modyfikowalność reprezentacji: dodawanie wiedzy w odpowiedzi na braki jest podstawowym rozwiązaniem braku wyczerpującego. Ponieważ większość baz wiedzy nie jest wyczerpująca, ich modyfikacja lub aktualizacja powinna być łatwa. Oprócz składniowej łatwości dodawania wiedzy, reprezentacja powinna pomóc zagwarantować spójność bazy wiedzy w miarę dodawania lub usuwania informacji. Dziedziczenie, poprzez zezwolenie na dziedziczenie właściwości klasy przez nowe instancje, jest przykładem tego, jak schemat reprezentacji może pomóc w zapewnieniu spójności. Kilka systemów, w tym Copycat i robot Brooksa, rozwiązało problem plastyczności, projektując struktury sieciowe, które zmieniają się i ewoluują w miarę spełniania ograniczeń świata przyrody. W tych systemach reprezentacja jest wynikiem oddolnego pozyskiwania nowych danych, ograniczonych oczekiwaniami systemu postrzegającego. Zastosowaniem tego typu systemu jest rozumowanie analogiczne.

Inna użyteczna właściwość reprezentacji dotyczy zakresu, w jakim mapowanie między światem a bazą wiedzy jest homomorficzne. Tutaj homomorficzność implikuje zgodność jeden do jednego między obiektami i działaniami na świecie a obiektami obliczeniowymi i operacjami języka. W mapowaniu homomorficznym baza wiedzy odzwierciedla postrzeganą organizację domeny i może być zorganizowana w bardziej naturalny i intuicyjny sposób. Oprócz naturalności, bezpośredniości i łatwości użycia schematy reprezentacyjne można również oceniać na podstawie ich wydajności obliczeniowej. Levesque i Brachman omawiają kompromis między ekspresją a skutecznością. Logika, gdy jest używana jako schemat reprezentacji, jest wysoce ekspresyjna ze względu na swoją kompletność; jednakże systemy oparte na logice nieograniczonej wiążą się z wysoką wydajnością, patrz Rozdział 14. Większość przedstawionych właśnie kwestii reprezentacji dotyczy wszelkich informacji, które mają być przechwycone i wykorzystane przez komputer. Istnieją dalsze kwestie, które muszą rozwiązać projektanci systemów rozproszonych i agentowych. Wiele z tych kwestii dotyczy podejmowania decyzji na podstawie częściowych (lokalnych) informacji, podziału odpowiedzialności za wykonywanie zadań, języków komunikacji agentów oraz opracowywania algorytmów współpracy i wymiany informacji między agentami. Wreszcie, jeśli filozoficzne podejście do rozproszonej inteligencji środowiskowej zaproponowane przez Clarka ,Haugelanda i Dennetta ma zostać zrealizowane, w którym tak zwany „przeciekający” system wykorzystuje oba Środowisko i inne czynniki jako nośniki

krytyczne dla przechowywania i wykorzystywania wiedzy, być może zupełnie nowe języki reprezentacji czekają na wynalazek. Gdzie są narzędzia reprezentacyjne i wsparcie, jak proponuje Brooks, „używać świata jako własnego modelu”? Teorie asocjacjonistyczne były badane jako modele pamięci i rozumowania komputera i człowieka . SNePS, jedna z najwcześniejszych reprezentacji opartych na asocjacjach / logice, została ostatnio rozszerzona jako narzędzie metapoznania, gdzie predykaty mogą przyjmować inne predykaty jako argumenty, wspierając w ten sposób umiejętność wnioskowania o strukturach bazy wiedzy.

Ważna praca w strukturalnych językach reprezentacji wiedzy obejmuje język reprezentacji Bobrow i Winograd KRL oraz język reprezentacji Brachmana (1979) KL-ONE, który zwraca szczególną uwagę na semantyczne podstawy strukturalnych reprezentacji. Nasz przegląd grafów pojęciowych zawdzięcza znaczną część książce Johna Sowy , Struktury konceptualne (1984). Czytelnika odsyła się do tej książki, aby poznać szczegóły, które pominęliśmy. W swoim pełnym ujęciu grafy pojęciowe łączą ekspresyjną moc rachunku predykatów, a także logiki modalnej i logiki wyższego rzędu z bogatym zestawem wbudowanych pojęć i relacji wywodzących się z epistemologii, psychologii i lingwistyki.

Istnieje wiele innych podejść do problemu reprezentacji. Na przykład Brachman, Fikes i Levesque zaproponowali przedstawienie, które podkreśla specyfikacje funkcjonalne; to znaczy o jakie informacje można poprosić lub przekazać do bazy wiedzy. Wiele książek może pomóc w zaawansowanym badaniu tych zagadnień. Readings in Knowledge Representation autorstwa Brachmana i Levesque'a to zbiór ważnych, ale datowanych artykułów na ten temat. Obecnie istnieje znaczna liczba artykułów zgodnych ze wskazówkami zaproponowanymi przez Brooksa w jego architekturze subsumpcji. Istnieją również kontrastujące stanowiska.

Badania oparte na agentach są szeroko rozpowszechnione we współczesnej sztucznej inteligencji. Istnieją teraz całe sekcje corocznych konferencji AI (IAAI i IJCAI) poświęconych badaniom agentów. Zalecamy przeczytanie ostatnich materiałów z tych konferencji, aby zapoznać się z bardziej aktualnymi dyskusjami na temat aktualnych zagadnień badawczych. Polecamy również materiały konferencyjne i lektury z zakresu sztucznej inteligencji rozproszonej, czyli DAI

Sztuczna inteligencja : Silna metoda rozwiązywania problemów

(VIII / XVI)

ĆWICZENIA

1. W podrozdziale 8.2 przedstawiliśmy zestaw zasad diagnozowania problemów samochodowych. Zidentyfikuj potencjalnych inżynierów wiedzy, ekspertów dziedzinowych i potencjalnych użytkowników końcowych dla takiej aplikacji. Omów oczekiwania, zdolności i potrzeby każdej z tych grup.

2. Wykonaj ćwiczenie 1 powyżej. Utwórz w języku angielskim lub pseudokodowym 15 reguł „jeśli-to" (innych niż te opisane w sekcji 8.2), aby opisać relacje w tej dziedzinie. Utwórz wykres przedstawiający relacje między tymi 15 regułami.

3. Rozważ wykres z ćwiczenia 2. Czy polecasz wyszukiwanie oparte na danych czy na celu? przeszukiwanie wszerz czy w głąb? W jaki sposób heurystyka może pomóc w wyszukiwaniu? Uzasadnij swoje odpowiedzi na te pytania.

4. Wybierz inny obszar zainteresowania do projektowania systemu ekspertowego. Odpowiedź Ćwiczenia 1–3 dla tej aplikacji.

5. Wdrożenie systemu eksperckiego przy użyciu komercyjnego programu powłoki. Są one szeroko dostępne zarówno dla komputerów osobistych, jak i większych maszyn. Szczególnie polecamy CLIPS z NASA lub JESS z Sandia National Laboratories.

6. Skrytykuj muszlę, której użyłeś w ćwiczeniu 5. Jakie są jej mocne i słabe strony? Co byś zrobił, żeby to poprawić? Czy było to odpowiednie dla twojego problemu? Jakie problemy najlepiej pasuje do tego narzędzia?

7. Stworzyć modelowy system rozumowania dla prostego urządzenia elektronicznego. Połącz kilka małych urządzeń, aby stworzyć większy system. Możesz użyć reguł if ... then ..., aby scharakteryzować funkcjonalność systemu.

8. Przeczytaj i skomentuj artykuł Diagnoza na podstawie opisu struktury i funkcji.

9. Przeczytaj jeden z wczesnych artykułów wykorzystujących rozumowanie oparte na modelach do uczenia dzieci arytmetyki (Brown i Burton 1978) lub umiejętności elektronicznych. Skomentuj to podejście.

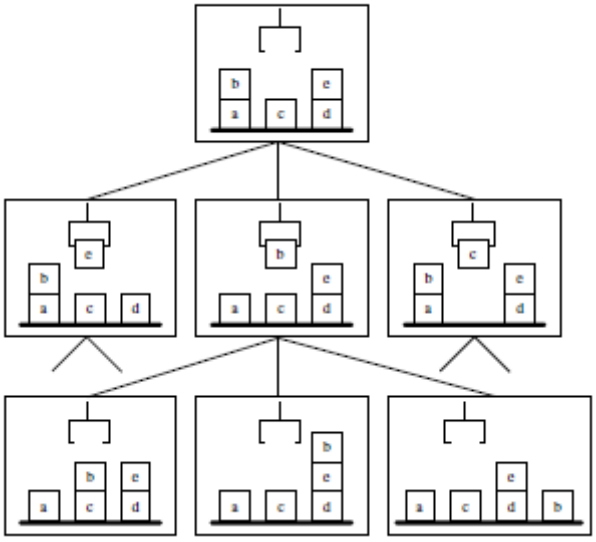
10. Zbuduj argumentację opartą na przypadkach dla wybranej aplikacji. Jednym z obszarów może być wybór kursów informatycznych i inżynierskich w celu ukończenia studiów licencjackich lub magisterskich.

11. Skorzystaj z komercyjnego oprogramowania (sprawdź w sieci WWW) do zbudowania systemu wnioskowania opartego na przypadkach w Ćwiczeniu 10. Jeśli żadne oprogramowanie nie jest dostępne, rozważ zbudowanie takiego systemu w Prologu, Lispie lub Javie.

12. Przeczytaj i skomentuj ankietę Janet Kolodner Poprawa podejmowania decyzji przez człowieka poprzez wspomaganie decyzji na podstawie przypadków.

13. Utwórz pozostałe aksjomaty ramek niezbędne do podnoszenia, odkładania, układania i zdejmowania ze stosu przez czterech operatorów, opisanych w zasadach od 4 do 7 w Sekcji 8.4.

14. Użyj operatorów i aksjomatów ramek z poprzedniego pytania, aby wygenerować przestrzeń poszukiwań z rysunku.



15. Pokaż, jak listy dodawania i usuwania mogą być używane do zastępowania aksjomatów ramek w generowaniu STANU 2 ze STANU 1 w sekcji 8.4.

16. Użyj list dodaj i usuń, aby wygenerować przestrzeń wyszukiwania przedstawioną na rysunku powyżej.

17. Zaprojektuj automatyczny kontroler, który mógłby używać list dodawania i usuwania do generowania wyszukiwania graficznego podobnego do tego na rysunku powyżej.

18. Pokaż jeszcze dwa niekompatybilne (warunkowe) cele cząstkowe w blokach operatorów światowych na rysunku powyżej.

19. Przeczytaj badanie ABSTRIPS i pokaż, jak rozwiązuje ono problem liniowości (lub niekompatybilnych celów podrzędnych) w planowaniu.

20. Przedstawiliśmy planer stworzony przez Nilssona i jego uczniów ze Stanford (Benson i Nilsson 1995). Planowanie teleoreaktywne umożliwia działania opisane jako trwałe, tj. Takie, które muszą być prawdziwe w różnych okresach czasu. Dlaczego planowanie teleo-reaktywne ma być bardziej preferowanym niż planer typu STRIPS? Zbuduj planner reaktywny tele w Prologu, Lispie lub Javie.

21. Przeczytaj Williams i Nayak , aby zapoznać się z dokładniejszym omówieniem ich systemu planowania opartego na modelach.

22. Rozwiń schemat reprezentacji rachunku zdań wprowadzony przez Williamsa i Nayaka) w celu opisania przejść między stanami ich układu napędowego.

SILNA METODA ROZWIĄZYWANIA PROBLEMÓW

Pierwsza zasada inżynierii wiedzy głosi, że zdolność rozwiązywania problemów, jaką wykazuje działanie inteligentnego agenta, jest przede wszystkim konsekwencją jego bazy wiedzy, a dopiero po drugie konsekwencją zastosowanej metody wnioskowania. Systemy eksperckie muszą być bogate w wiedzę, nawet jeśli są ubogie w metody. To ważny wynik, który dopiero niedawno został dobrze zrozumiany w sztucznej inteligencji. Przez długi czas sztuczna inteligencja skupiała się prawie wyłącznie na rozwoju sprytnych metod wnioskowania; prawie każda metoda wnioskowania wystarczy. Siła tkwi w wiedzy.

—EDWARD FEIGENBAUM, Uniwersytet Stanforda

Wprowadzenie

Kontynuujemy badanie zagadnień reprezentacji i inteligencji, rozważając ważny składnik sztucznej inteligencji: wymagające wiedzy lub mocne metody rozwiązywania problemów. Eksperci są w stanie osiągać sukcesy, ponieważ wiedzą dużo o swoich obszarach specjalizacji. Ta prosta obserwacja jest podstawą do zaprojektowania solidnych metod lub opartych na wiedzy rozwiązań do rozwiązywania problemów . Na przykład system ekspercki wykorzystuje wiedzę specyficzną dla domeny problemowej, aby zapewnić „jakość ekspercką" w tym obszarze zastosowań. Ogólnie rzecz biorąc, projektanci systemów eksperckich zdobywają tę wiedzę z pomocą ekspertów z dziedziny ludzkiej, a system naśladuje metodologię i wydajność ekspertów. Podobnie jak w przypadku wykwalifikowanych ludzi, systemy ekspertowe są zwykle specjalistami, koncentrującymi się na wąskim zestawie problemów. Podobnie jak ludzie, ich wiedza jest zarówno teoretyczna, jak i praktyczna: ludzcy eksperci, którzy dostarczają wiedzę systemową, ogólnie poszerzyli swoje własne teoretyczne rozumienie domeny problemowej za pomocą sztuczek, skrótów i heurystyk dotyczących wykorzystania wiedzy zdobytej podczas rozwiązywania problemów doświadczenie. Ze względu na swoją heurystyczną, opartą na wiedzy naturę, systemy ekspertowe generalnie:

1. Wspieraj kontrolę ich procesów rozumowania, zarówno podczas przedstawiania etapów pośrednich, jak i odpowiadania na pytania dotyczące procesu rozwiązywania.

2. Umożliwić łatwe modyfikowanie dodawania i usuwania umiejętności z bazy wiedzy.

3. Rozumuj heurystycznie, używając (często niedoskonałej) wiedzy, aby uzyskać przydatne rozwiązania.

Rozumowanie systemu ekspertowego powinno być otwarte dla wglądu, dostarczając informacji o stanie rozwiązania problemu oraz wyjaśniając wybory i decyzje, które podejmuje program. Wyjaśnienia są ważne dla eksperta będącego człowiekiem, na przykład lekarza lub inżyniera, jeśli ma on przyjąć zalecenia z komputera. Rzeczywiście, niewielu ludzkich ekspertów przyjmie rady od innego człowieka, nie mówiąc już o maszynie, bez zrozumienia ich uzasadnienia. Eksploracyjny charakter sztucznej inteligencji i programowania systemów eksperckich wymaga łatwego prototypowania, testowania i modyfikowania programów. Języki i środowiska programowania AI są zaprojektowane tak, aby wspierać tę iteracyjną metodologię programowania. Na przykład w czystym systemie produkcyjnym modyfikacja pojedynczej reguły nie ma żadnych globalnych składniowych skutków ubocznych. Reguły można dodawać lub usuwać bez konieczności wprowadzania dalszych zmian w większym programie. Projektanci systemów eksperckich często komentują, że łatwość modyfikacji bazy wiedzy jest głównym czynnikiem w tworzeniu udanego programu.

Kolejną cechą systemów ekspertowych jest stosowanie przez nie heurystycznych metod rozwiązywania problemów. Jak odkryli projektanci systemów eksperckich, nieformalne „sztuczki handlowe" i „praktyczne zasady" stanowią istotne uzupełnienie standardowej teorii przedstawionej w podręcznikach i na zajęciach. Czasami zasady te w zrozumiały sposób poszerzają wiedzę teoretyczną;

często są to po prostu skróty, które, jak wykazano empirycznie, działają. Systemy eksperckie są tworzone w celu rozwiązywania szerokiego zakresu problemów w takich dziedzinach, jak medycyna, matematyka, inżynieria, chemia, geologia, informatyka, biznes, prawo, obronność i edukacja. Programy te rozwiązują różnorodne problemy; Poniższa lista, z Watermana (1986), jest użytecznym podsumowaniem ogólnych kategorii problemów systemu ekspertowego. Interpretacja - formułowanie wniosków wysokiego poziomu na podstawie zbiorów surowych danych.

Predykcja - przewidywanie prawdopodobnych konsekwencji danych sytuacji.

Diagnoza - ustalenie przyczyny nieprawidłowości w złożonych sytuacjach na podstawie możliwych do zaobserwowania objawów.

Projektowanie - znajdowanie konfiguracji komponentów systemu, która spełnia cele wydajnościowe, jednocześnie spełniając zestaw ograniczeń projektowych.

Planowanie - opracowanie sekwencji działań, które pozwolą osiągnąć zestaw celów przy określonych warunkach początkowych i ograniczeniach czasu wykonywania.

Monitorowanie - porównanie obserwowanego zachowania systemu z jego oczekiwanym zachowaniem.

Instruktaż - pomoc w procesie edukacji w dziedzinach technicznych.

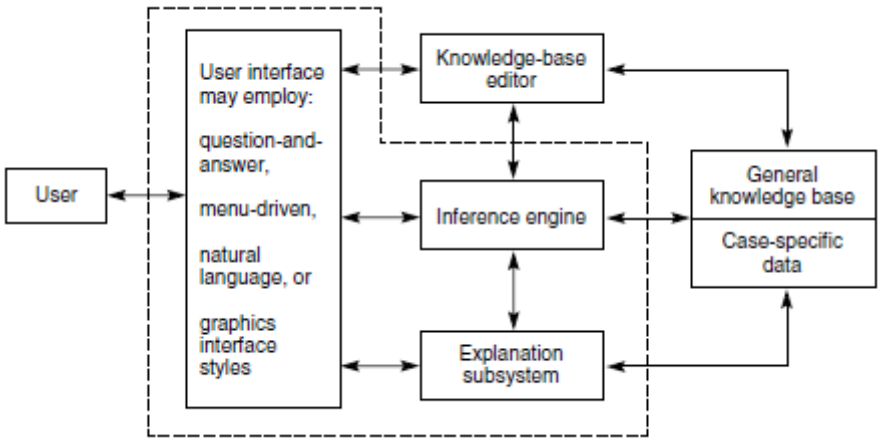
Kontrola — rządząca zachowaniem złożonego środowiska.

Najpierw przyjrzymy się technologii, która umożliwia rozwiązywanie problemów w oparciu o wiedzę. Skuteczna inżynieria wiedzy musi rozwiązać szereg problemów, od wyboru odpowiedniej domeny aplikacji po jej nabycie i sformalizowanie wiedzy na temat rozwiązywania problemów. Później przedstawiamy systemy oparte na regułach i przedstawiamy system produkcyjny jako architekturę oprogramowania dla procesów rozwiązywania i wyjaśniania. Przeanalizujemy techniki rozumowania opartego na modelu i na podstawie przypadków. Kolejna sekcja przedstawia planowanie, proces organizowania fragmentów wiedzy w spójną sekwencję działań, które pozwolą osiągnąć cel.

Przegląd technologii systemu eksperckiego

Projektowanie systemów ekspertowych opartych na regułach

Rysunek przedstawia moduły składające się na typowy system ekspertowy.



Użytkownik współdziała z systemem za pośrednictwem interfejsu użytkownika, który upraszcza komunikację i ukrywa większość złożoności, na przykład wewnętrzną strukturę bazy reguł. Interfejsy systemu eksperckiego wykorzystują różne style użytkownika, w tym interfejsy pytań i odpowiedzi, interfejsy menu lub interfejsy graficzne. Ostateczna decyzja co do typu interfejsu jest kompromisem pomiędzy potrzebami użytkownika a wymaganiami bazy wiedzy i systemu wnioskowania. Sercem systemu ekspertowego jest baza wiedzy, która zawiera wiedzę z określonej dziedziny aplikacji. W systemie eksperckim opartym na regułach wiedza ta jest najczęściej reprezentowana w postaci reguł jeśli ... to ..., jak w naszych przykładach w sekcji 8.2. Baza wiedzy zawiera zarówno wiedzę ogólną, jak i informacje dotyczące konkretnych przypadków. Mechanizm wnioskowania stosuje wiedzę do rozwiązywania rzeczywistych problemów. Zasadniczo jest to tłumacz bazy wiedzy. W systemie produkcyjnym silnik wnioskowania wykonuje cykl sterowania rozpoznawaj-działaj. Procedury, które wdrażają cykl kontrolny, są oddzielone od samych reguł produkcji. Utrzymanie tego oddzielenia bazy wiedzy od silnika wnioskowania jest ważne z kilku powodów:

1. Ten rozdział umożliwia przedstawienie wiedzy w bardziej naturalny sposób. Jeśli… wtedy… reguły, na przykład, są bliższe sposobowi, w jaki ludzie opisują swoje umiejętności rozwiązywania problemów, niż kod komputerowy niższego poziomu.

2. Ponieważ baza wiedzy jest oddzielona od struktur sterowania niższego poziomu programu, konstruktorzy systemów eksperckich mogą skupić się na gromadzeniu i organizowaniu wiedzy dotyczącej rozwiązywania problemów, a nie na szczegółach jej implementacji komputerowej.

3. W idealnym przypadku rozdzielenie wiedzy i kontroli pozwala na dokonywanie zmian w jednej części bazy wiedzy bez wywoływania skutków ubocznych w innych.

4. Rozdzielenie wiedzy i elementów sterujących programu pozwala na użycie tego samego oprogramowania sterującego i interfejsu w różnych systemach. Powłoka systemu ekspertowego zawiera wszystkie komponenty z rysunku 8.1, z wyjątkiem tego, że baza wiedzy i dane dotyczące przypadku są puste i można je dodać do nowej aplikacji. Linie przerywane na rysunku 8.1 wskazują moduły powłoki.

System ekspercki musi śledzić dane dotyczące konkretnego przypadku: fakty, wnioski i inne informacje istotne dla rozpatrywanej sprawy. Obejmuje to dane podane w przypadku problemu, częściowe wnioski, miary zaufania wniosków i martwe ,kończy się procesem wyszukiwania. Informacje te są oddzielone od ogólnej bazy wiedzy. Podsystem wyjaśnień umożliwia programowi wyjaśnienie użytkownikowi swojego uzasadnienia. Te wyjaśnienia zawierają uzasadnienia wniosków systemu, w odpowiedzi na to, jak zapytania , wyjaśnienia, dlaczego system potrzebuje określonych danych, dlaczego zapytania oraz, jeśli jest to przydatne, wyjaśnienia samouczków lub głębsze teoretyczne uzasadnienia działań programu. Wiele systemów zawiera również edytor bazy wiedzy. Edytory bazy wiedzy pomagają programiście zlokalizować i poprawić błędy w działaniu programu, często uzyskując dostęp do informacji dostarczanych przez podsystem wyjaśniający. Mogą również pomagać w dodawaniu nowej wiedzy, utrzymywać prawidłową składnię reguł i przeprowadzać kontrole spójności każdej zaktualizowanej bazy wiedzy. Ważną przyczyną skrócenia czasu projektowania i wdrażania obecnych systemów ekspertowych jest dostępność powłok systemów ekspertowych. NASA stworzyła CLIPS, JESS jest dostępny w Sandia National Laboratories, a w naszych materiałach uzupełniających oferujemy muszle w Lisp i Prolog. Niestety programy powłoki nie rozwiązują wszystkich problemów związanych z budową systemów ekspertowych. Chociaż rozdzielenie wiedzy i kontroli, modułowość architektury systemu produkcyjnego oraz użycie odpowiedniego języka reprezentacji wiedzy, wszystko to pomaga w budowaniu systemu eksperckiego, nabycie i sformalizowanie wiedzy dziedzinowej nadal pozostaje trudnym zadaniem

Wybór problemu i procesu inżynierii wiedzy

Systemy eksperckie wymagają znacznych nakładów finansowych i ludzkiego wysiłku. Próby rozwiązania problemu, który jest zbyt złożony, zbyt słabo zrozumiany lub w inny sposób niedostosowany do dostępnej technologii, mogą prowadzić do kosztownych i krępujących niepowodzeń. Badacze opracowali wytyczne, aby określić, czy problem jest odpowiedni do rozwiązania systemu eksperckiego:

1. Potrzeba rozwiązania uzasadnia koszt i wysiłek budowy systemu ekspertowego. Wiele systemów eksperckich zostało zbudowanych w takich dziedzinach, jak poszukiwanie złóż, biznes, obrona i medycyna, w których istnieje duży potencjał oszczędności w zakresie pieniędzy, czasu i życia ludzkiego.

2. Ludzka wiedza nie jest dostępna we wszystkich sytuacjach, w których jest potrzebna. Na przykład w geologii potrzebna jest wiedza specjalistyczna w odległych miejscach wydobycia i wierceń. Często geolodzy i inni inżynierowie podróżują na duże odległości, aby odwiedzić miejsca, co wiąże się z wydatkami i stratą czasu. Umieszczając systemy eksperckie w odległych lokalizacjach, wiele problemów można rozwiązać bez konieczności wizyty.

3. Problem można rozwiązać za pomocą rozumowania symbolicznego. Rozwiązania problemów nie powinny wymagać sprawności fizycznej ani zdolności percepcyjnych. Obecnie robotom i systemom wizyjnym brakuje wyrafinowania i elastyczności jak ludzie.

4. Dziedzina problemowa jest dobrze skonstruowana i nie wymaga zdrowego rozsądku. Obszary wysoce techniczne mają tę zaletę, że są dobrze zbadane i sformalizowane: terminy są dobrze zdefiniowane, a domeny mają jasne i konkretne modele pojęciowe. Natomiast rozsądne rozumowanie jest trudne do zautomatyzowania.

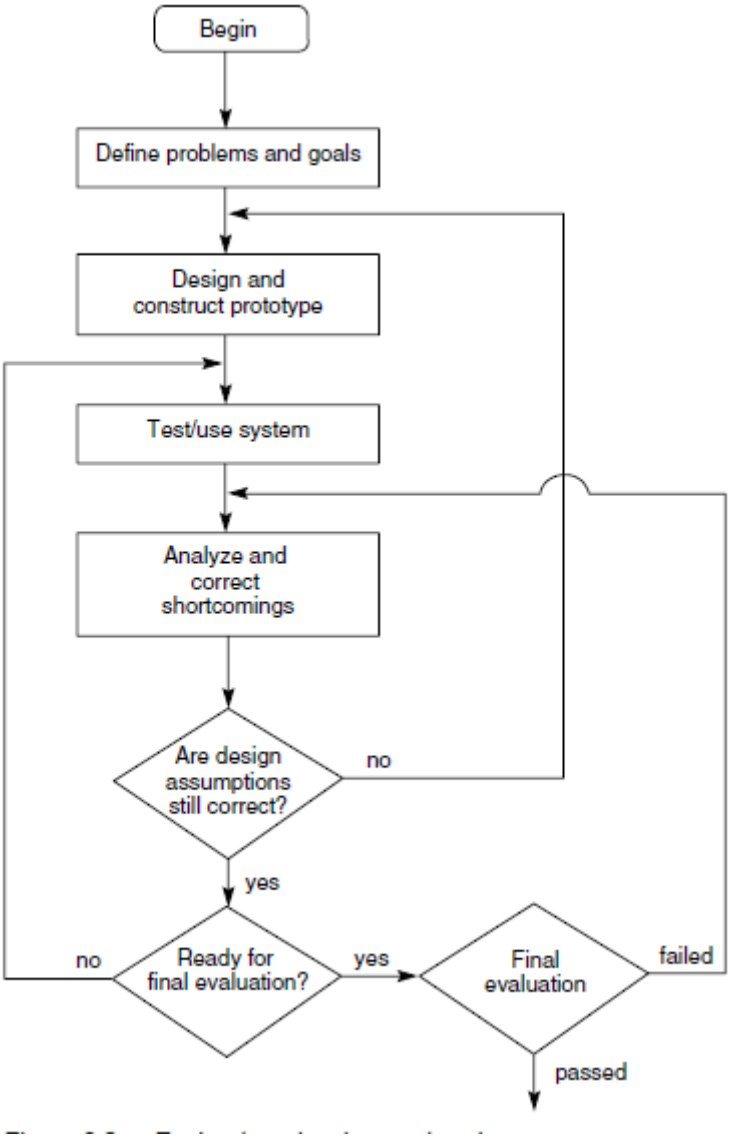
5. Problemu nie da się rozwiązać tradycyjnymi metodami obliczeniowymi. Nie należy stosować technologii systemu eksperckiego, jeśli nie jest to konieczne. Jeśli problem można rozwiązać w sposób zadowalający przy użyciu bardziej tradycyjnych technik, to nie jest on kandydatem.

6. Istnieją współpracujący i wyartykułowani eksperci. Wiedza wykorzystywana przez systemy ekspertowe pochodzi z doświadczenia i oceny ludzi pracujących w tej dziedzinie. Ważne jest, aby ci eksperci byli zarówno chętni, jak i zdolni do dzielenia się wiedzą.

7. Problem ma odpowiednią wielkość i zakres. Na przykład program, który próbowałby uchwycić całą wiedzę lekarza, byłby niewykonalny; bardziej odpowiedni byłby program, w którym lekarze zalecaliby stosowanie określonego sprzętu diagnostycznego lub określonego zestawu diagnoz.

Głównymi osobami zaangażowanymi w tworzenie systemu ekspertowego są inżynier wiedzy, ekspert domeny i użytkownik końcowy. Inżynier wiedzy jest ekspertem w dziedzinie języka sztucznej inteligencji i reprezentacji. Jego głównym zadaniem jest dobór oprogramowania i narzędzi sprzętowych projektu, pomóż ekspertowi dziedzinowemu w wyartykułowaniu niezbędnej wiedzy i wdrożeniu tej wiedzy w prawidłowej i wydajnej bazie wiedzy. Często inżynier wiedzy początkowo nie zna domeny aplikacji. Ekspert dziedzinowy zapewnia wiedzę o obszarze problemowym. Ekspert domeny to na ogół osoba, która pracowała w danej dziedzinie i rozumie jej techniki rozwiązywania problemów, takie jak skróty, posługiwanie się nieprecyzyjnymi danymi, ocenianie częściowych rozwiązań i wszystkie inne umiejętności, które wyróżniają osobę jako eksperta w rozwiązywaniu problemów. Ekspert domeny jest przede wszystkim odpowiedzialny za przedstawienie tych umiejętności inżynierowi wiedzy. Podobnie jak w większości aplikacji, użytkownik końcowy określa główne ograniczenia projektowe. O ile użytkownik nie jest zadowolony, wysiłek związany z rozwojem jest w zasadzie zmarnowany. Umiejętności i potrzeby użytkownika muszą być brane pod uwagę przez cały cykl projektowania: czy program ułatwi, szybszą i wygodniejszą pracę użytkownika? Jakiego

poziomu wyjaśnienia potrzebuje użytkownik? Czy użytkownik może podać poprawne informacje do systemu? Czy interfejs jest odpowiedni? Czy środowisko pracy użytkownika nakłada ograniczenia na korzystanie z programu? Na przykład interfejs wymagający pisania na klawiaturze nie byłby odpowiedni do użycia w kokpicie myśliwca. Podobnie jak w przypadku większości programów AI, tworzenie systemów ekspertowych wymaga nietradycyjnego cyklu rozwoju opartego na wczesnym prototypowaniu i stopniowej weryfikacji kodu. Generalnie prace nad systemem rozpoczynają się od próby zapoznania się inżyniera wiedzy z dziedziną problemową. Pomaga to w komunikacji z ekspertem domeny. Odbywa się to podczas wstępnych rozmów z ekspertem oraz poprzez obserwację ekspertów podczas wykonywania ich pracy. Następnie inżynier wiedzy i ekspert rozpoczynają proces wydobywania wiedzy eksperta dotyczącej rozwiązywania problemów. Często polega to na przedstawieniu ekspertowi dziedzinowemu serii przykładowych problemów i poproszeniu go o wyjaśnienie technik zastosowanych w ich rozwiązaniu. Taśmy wideo i / lub audio są często niezbędne do przechwycenia tego procesu. Dla inżyniera wiedzy często przydatne jest bycie nowicjuszem w dziedzinie problemów. Ludzcy eksperci są notorycznie niewiarygodni w wyjaśnianiu dokładnie tego, co dzieje się podczas rozwiązywania złożonego problemu. Często zapominają o krokach, które stały się dla nich oczywiste lub wręcz automatyczne po latach pracy w ich dziedzinie. Inżynierowie wiedzy, dzięki swojej względnej naiwności w tej dziedzinie, mogą dostrzec te koncepcyjne przeskoki i poprosić o pomoc. Po uzyskaniu przez inżyniera wiedzy ogólnego przeglądu domeny problemu i przeszedł kilka sesji rozwiązywania problemów z ekspertem, jest on gotowy do rozpoczęcia właściwego projektowania systemu: wybór sposobu reprezentacji wiedzy, na przykład reguł lub ram, określenie strategii wyszukiwania, do przodu, do tyłu, po pierwsze, najpierw najlepsze itp. oraz zaprojektowanie interfejsu użytkownika. Po podjęciu tych zobowiązań projektowych inżynier wiedzy buduje prototyp. Prototyp ten powinien być w stanie rozwiązywać problemy na niewielkim obszarze domeny i zapewniać stanowisko testowe dla wstępnych założeń projektowych. Po wdrożeniu prototypu inżynier wiedzy i ekspert dziedzinowy testują i udoskonalają swoją wiedzę, dając mu problemy do rozwiązania i korygowania jego niedociągnięć. Jeśli założenia przyjęte podczas projektowania prototypu okażą się słuszne, prototyp można stopniowo rozszerzać, aż stanie się systemem ostatecznym. Systemy eksperckie są budowane poprzez progresywne przybliżenia, a błędy programu prowadzą do poprawek lub uzupełnień w bazie wiedzy. W pewnym sensie baza wiedzy jest raczej „rozwijana” niż konstruowana. Rysunek przedstawia schemat blokowy opisujący cykl rozwoju programowania eksploracyjnego.

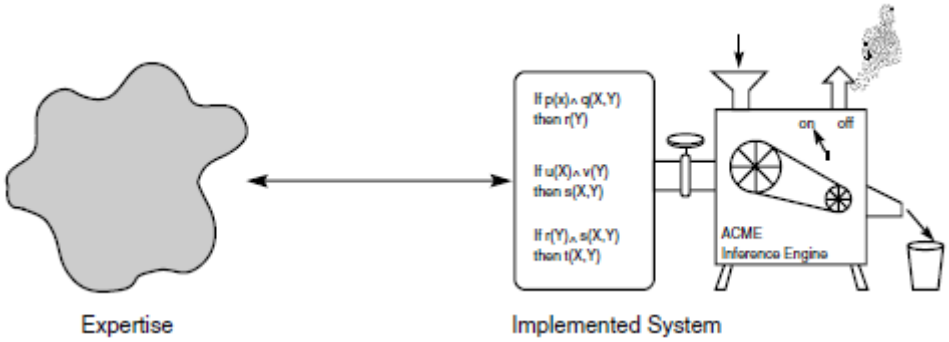


To podejście do programowania zostało zbadane przez Seymoura Paperta w jego języku LOGO (Papert 1980), a także przez pracę Alana Kaya z Smalltalk w Xerox PARC. Filozofia LOGO twierdzi, że obserwowanie, jak komputer reaguje na niewłaściwie sformułowane idee reprezentowane przez kod, prowadzi do nich korekta (debugowanie) i wyjaśnienie za pomocą bardziej precyzyjnego kodu. Ten proces próbowania i poprawiania projektów kandydatów jest wspólny dla rozwoju systemów ekspertowych i kontrastuje z tak starannie zhierarchizowanymi procesami, jak projektowanie odgórne.

Rozumie się również, że prototyp może zostać wyrzucony, jeśli stanie się zbyt uciążliwy lub jeśli projektanci zdecydują się zmienić swoje podstawowe podejście do problemu. Prototyp pozwala konstruktorom programów zbadać problem i jego ważne relacje poprzez faktyczne skonstruowanie programu do jego rozwiązania. Po ukończeniu tego stopniowego wyjaśniania mogą oni często napisać bardziej przejrzystą wersję, zwykle z mniejszą liczbą reguł. Drugą ważną cechą programowania systemów eksperckich jest to, że program nigdy nie musi być uważany za „zakończony”. Duża baza wiedzy heurystycznej zawsze będzie miała ograniczenia. Modułowość modelu systemu produkcyjnego sprawia, że w dowolnym momencie naturalne jest dodawanie nowych reguł lub nadrabianie braków w obecnej bazie reguł.

Modele koncepcyjne i ich rola w zdobywaniu wiedzy

Rysunek przedstawia uproszczony model procesu zdobywania wiedzy, który posłuży jako użyteczne „pierwsze przybliżenie” do zrozumienia problemów związanych z pozyskiwaniem i formalizowaniem działań ekspertów.



Ekspert ludzki, działający w obszarze zastosowań, działa w dziedzinie wiedzy, umiejętności i praktyki. Wiedza ta jest często niejasna, nieprecyzyjna i tylko częściowo zwerbalizowana. Inżynier wiedzy musi przetłumaczyć tę nieformalną wiedzę fachową na język formalny dostosowany do systemu obliczeniowego. Szereg ważnych kwestii pojawia się w procesie formalizowania ludzkich umiejętności:

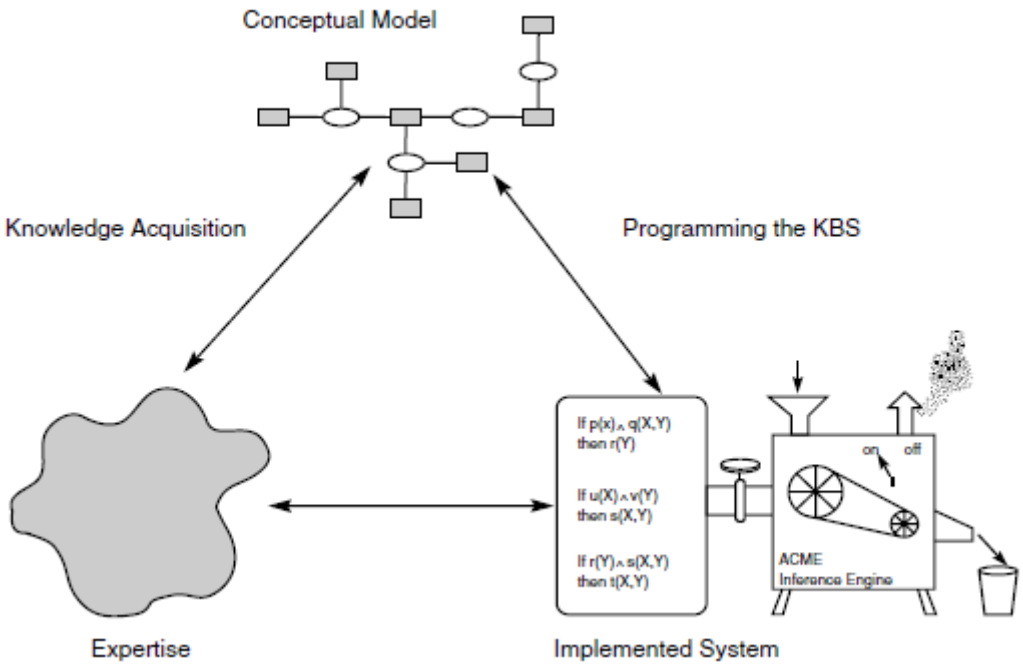
1. Umiejętności ludzkie są często niedostępne dla świadomego umysłu. Jak wskazuje Arystoteles w swojej etyce, „czego musimy się nauczyć, uczymy się przez działanie”. Umiejętności, takich jak te posiadane przez lekarzy, uczy się tak samo przez lata stażu i rezydencji, z ich stałym skupieniem na pacjentach, jak na wykładach z fizjologii, gdzie nacisk kładzie się na eksperyment i teorię. Świadczenie opieki medycznej odbywa się w dużym stopniu opartym na praktyce. Po latach występów umiejętności te stają się silnie zintegrowane i funkcjonują na zasadniczo nieświadomym poziomie. Ekspertom może być trudno dokładnie opisać, co robią w rozwiązywaniu problemów.

2. Ekspertyza ludzka często przybiera formę wiedzy, jak radzić sobie w danej sytuacji, a nie wiedzy, jaka może być racjonalna charakterystyka sytuacji, rozwijania wykwalifikowanych mechanizmów działania, a nie podstawowego zrozumienia, czym są te mechanizmy. Oczywistym tego przykładem jest jazda na monocyklu: odnoszący sukcesy jeździec nie rozwiązuje w czasie rzeczywistym świadomie wielu układów równań różniczkowych, aby zachować równowagę; raczej używa intuicyjnej kombinacji uczuć „grawitacji”, „pędu” i „bezwładności”, aby stworzyć użyteczną procedurę kontrolną.

3. Często myślimy o zdobywaniu wiedzy jako zdobywaniu faktycznej wiedzy o obiektywnej rzeczywistości, tzw. „Realnym świecie”. Jak pokazały zarówno teoria, jak i praktyka, ludzka wiedza reprezentuje model świata jednostki lub społeczności. Na takie modele wpływają zarówno konwencje, procesy społeczne i ukryte programy, jak i metodologie empiryczne.

4. Zmiany eksperckie. Nie tylko ludzcy eksperci zdobywają nową wiedzę, ale także istniejąca wiedza może podlegać radykalnym przeformułowaniu, o czym świadczą trwające kontrowersje zarówno na polu naukowym, jak i społecznym.

W związku z tym inżynieria wiedzy jest trudna i powinna być postrzegana jako obejmująca cykl życia każdego systemu ekspertowego. Aby uprościć to zadanie, warto rozważyć, jak na Rysunku 8.4, model koncepcyjny, który leży pomiędzy ludzką wiedzą a wdrożonym programem.



Przez model konceptualny rozumiemy ewoluującą koncepcję wiedzy dziedzinowej inżyniera wiedzy. Chociaż niewątpliwie różni się to od ekspertyzy dziedzinowej, to właśnie ten model determinuje konstrukcję formalnej bazy wiedzy. Ze względu na złożoność najbardziej interesujących problemów nie powinniśmy traktować tego pośredniego etapu za pewnik. Inżynierowie wiedzy powinni dokumentować i upubliczniać swoje założenia dotyczące domeny za pomocą wspólnych metodologii inżynierii oprogramowania. System ekspercki powinien zawierać dokument wymagań; jednakże ze względu na ograniczenia programowania eksploracyjnego wymagania systemu eksperckiego należy traktować jako współewoluujące z prototypem. Słowniki danych, graficzne reprezentacje przestrzeni stanów i komentarze w samym kodzie są częścią tego modelu. Publikując te decyzje projektowe, zmniejszamy liczbę błędów zarówno we wdrażaniu, jak i utrzymywaniu programu. Inżynierowie wiedzy powinni zapisywać nagrania rozmów z ekspertami dziedzinowymi. Często wraz ze wzrostem zrozumienia domeny przez inżynierów wiedzy formułują oni nowe interpretacje lub odkrywają nowe informacje o domenie. Nagrania, wraz z dokumentacją nadanej interpretacji, odgrywają cenną rolę w ocenie decyzji projektowych i testowaniu prototypów. Wreszcie model ten pełni rolę pośrednią w formalizowaniu wiedzy. Wybór języka reprezentacji wywiera silny wpływ na model domeny inżyniera wiedzy. Model koncepcyjny nie jest formalny ani nie można go bezpośrednio wykonać na komputerze. Jest to konstrukcja pośrednia, szablon, który ma zacząć ograniczać i kodyfikować ludzkie umiejętności. Jeśli inżynier wiedzy użyje modelu rachunku predykatów, może zacząć się jako kilka prostych sieci reprezentujących stany rozumowania poprzez typowe sytuacje rozwiązywania problemów. Dopiero po dalszym udoskonaleniu ta sieć staje się wyraźna, jeśli… wtedy… zasady.

Pytania często zadawane w kontekście modelu konceptualnego obejmują: Czy rozwiązanie problemu jest deterministyczne czy oparte na wyszukiwaniu? Czy rozumowanie jest oparte na danych, być może ze smakiem generowania i testowania, czy też jest ukierunkowane na cel, oparte na małym zestawie prawdopodobnych hipotez dotyczących sytuacji? Czy istnieją etapy rozumowania? Czy dziedzina jest dobrze poznana i jest w stanie zapewnić głębokie modele predykcyjne, czy też cała wiedza dotycząca rozwiązywania problemów jest zasadniczo heurystyczna? Czy możemy wykorzystać przykłady przeszłych problemów i ich rozwiązań, aby bezpośrednio rozwiązać przyszłe problemy, czy też musimy

najpierw przekształcić te przykłady w ogólne zasady? Czy wiedza jest dokładna, czy też „rozmyta" i przybliżona, umożliwiająca liczbowe oceny pewności (rozdział 9)? Czy strategie rozumowania pozwolą nam wnioskować o stabilnych faktach na temat domeny, czy też zmiana i niepewność w systemie będą wymagały niemonotonicznego rozumowania, to znaczy zdolności do formułowania twierdzeń na temat domeny, które można później zmodyfikować lub wycofać (sekcja 9.1)? Wreszcie, czy struktura wiedzy dziedzinowej wymaga od nas rezygnacji z wnioskowania opartego na regułach na rzecz alternatywnych schematów, takich jak sieci neuronowe lub algorytmy genetyczne (część IV)? Ewentualne potrzeby użytkowników należy również uwzględnić w kontekście modelu konceptualnego: Jakie są ich oczekiwania co do ostatecznego programu? Gdzie jest ich poziom wiedzy: nowicjusz, średniozaawansowany czy ekspert? Jakie poziomy wyjaśnienia są odpowiednie? Jaki interfejs najlepiej spełnia ich potrzeby? W oparciu o odpowiedzi na te i powiązane pytania, wiedzę uzyskaną od ekspertów dziedzinowych oraz wynikający z nich model koncepcyjny, przystępujemy do tworzenia systemu ekspertowego. Ponieważ system produkcyjny, przedstawiony po raz pierwszy w Rozdziale 6, oferuje szereg nieodłącznych mocnych stron w organizowaniu i stosowaniu wiedzy, często jest używany jako podstawa reprezentacji wiedzy w systemach ekspertowych opartych na regułach.

(8.2) Systemy eksperckie oparte na regułach

Oparte na regułach systemy eksperckie reprezentują wiedzę dotyczącą rozwiązywania problemów, tak jakby… wtedy… reguły. Podejście to nadaje się do architektury przedstawionej na pierwszym rysunku i jest jedną z najstarszych technik przedstawiania wiedzy dziedzinowej w systemie eksperckim. Jest również jednym z najbardziej naturalnych i pozostaje szeroko stosowany w praktycznych i eksperymentalnych systemach ekspertowych. Architekturę systemów ekspertowych opartych na regułach można najlepiej zrozumieć w kontekście modelu systemu produkcyjnego do rozwiązywania problemów przedstawionego w części II. Podobieństwo między nimi jest czymś więcej niż analogią: system produkcyjny był intelektualnym prekursorem nowoczesnych architektur systemów ekspertowych, w których zastosowanie reguł produkcji prowadzi do udoskonalenia zrozumienia konkretnej sytuacji problemowej. Kiedy Newell i Simon opracowali system produkcyjny, ich celem było modelowanie ludzkiej wydajności w rozwiązywaniu problemów.

Jeśli weźmiemy pod uwagę architekturę systemu ekspertowego jako system produkcyjny, to baza wiedzy specyficznej dla domeny jest zbiorem reguł produkcyjnych. W systemie opartym na regułach te pary akcji warunku są reprezentowane tak, jakby ... wtedy ... reguły, z przesłankami reguł, częścią jeśli odpowiadającą warunkowi i wnioskiem, częścią wtedy, odpowiadającą warunkowi działanie: gdy warunek jest spełniony, system ekspertowy podejmuje działanie potwierdzające wniosek jako prawdziwy. Dane dotyczące konkretnego przypadku mogą być przechowywane w pamięci roboczej. Mechanizm wnioskowania implementuje cykl rozpoznawania-aktu systemu produkcyjnego; kontrola ta może być oparta na danych lub na celu. Wydaje się, że wiele problematycznych domen w bardziej naturalny sposób nadaje się do wyszukiwania do przodu. Na przykład w przypadku problemu interpretacyjnego większość danych dotyczących problemu jest podawana początkowo i często trudno jest sformułować hipotezę lub cel. Sugeruje to proces wnioskowania z wyprzedzeniem, w którym fakty są umieszczane w pamięci roboczej, a system szuka interpretacji. W systemie eksperckim opartym na celach wyrażenie celu jest początkowo umieszczane w pamięci roboczej. System dopasowuje wnioski reguły do celu, wybierając jedną regułę i umieszczając jej założenia w pamięci roboczej. Odpowiada to rozkładowi celu problemu na prostsze cele podrzędne. Proces jest kontynuowany w następnej iteracji systemu produkcyjnego, przy czym te przesłanki stają się nowymi celami, które należy dopasować do wniosków reguł. W ten sposób system działa wstecz od pierwotnego celu, aż wszystkie cele cząstkowe w pamięci roboczej są prawdziwe, co wskazuje, że hipoteza została zweryfikowana. Zatem przeszukiwanie wstecz w systemie eksperckim odpowiada z grubsza procesowi testowania hipotez w

rozwiązywaniu ludzkich problemów, jak również po raz pierwszy przedstawiono w sekcji 3.2. W systemie eksperckim cele cząstkowe można rozwiązać, prosząc użytkownika o informacje. Niektóre systemy eksperckie pozwalają projektantowi systemu określić, które cele cząstkowe można rozwiązać, pytając użytkownika. Inni po prostu pytają użytkownika o cele cząstkowe, które nie pasują do reguł w bazie wiedzy; tj. jeśli program nie może wywnioskować prawdziwości celu podrzędnego, pyta użytkownika. Jako przykład rozwiązywania problemów zorientowanego na cel za pomocą zapytań użytkowników, oferujemy następnie mały system ekspercki do analizy problemów motoryzacyjnych. Nie jest to pełny system diagnostyczny, ponieważ zawiera tylko cztery bardzo proste zasady. Ma to służyć jako przykład do zademonstrowania łańcuch reguł zorientowanych na cel, integracja nowych danych i korzystanie z ułatwień wyjaśniających:

Zasada 1: jeśli

w silniku robi się gaz i

silnik się przewróci,

następnie

problemem są świece zapłonowe.

Zasada 2: jeśli

silnik się nie obraca i

światła się nie włączają

następnie

problem to akumulator lub kable.

Zasada 3: jeśli

silnik się nie obraca i

włączają się światła

następnie

problemem jest rozrusznik.

Zasada 4: jeśli

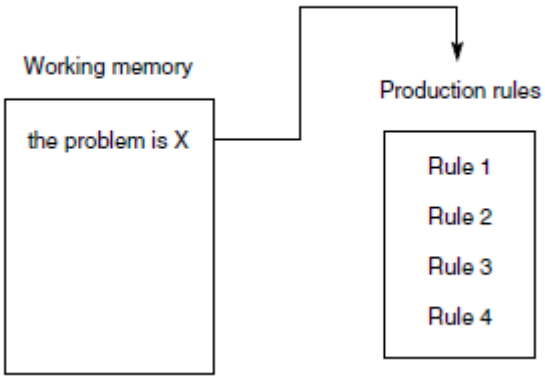
w zbiorniku paliwa jest gaz i

w gaźniku jest gaz

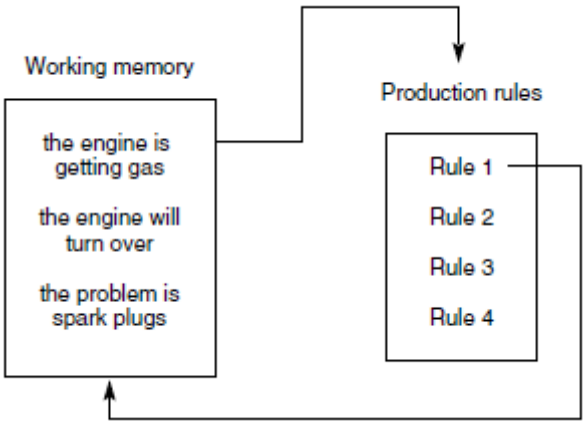
następnie

silnik się pali.

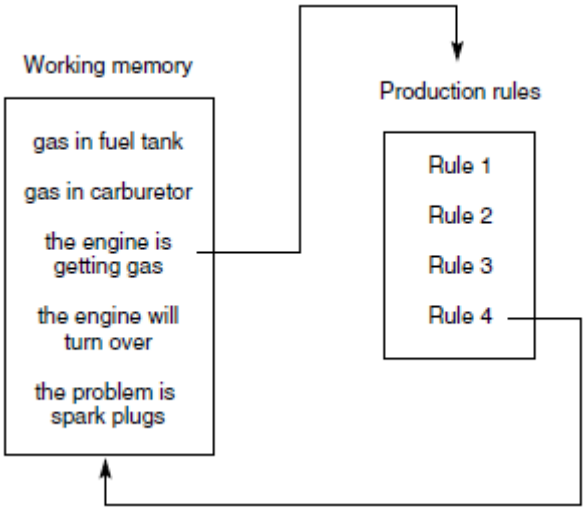
Aby uruchomić tę bazę wiedzy w ramach systemu kontroli ukierunkowanej na cel, umieść cel najwyższego poziomu, problemem jest X, w pamięci roboczej, jak pokazano na rysunku .



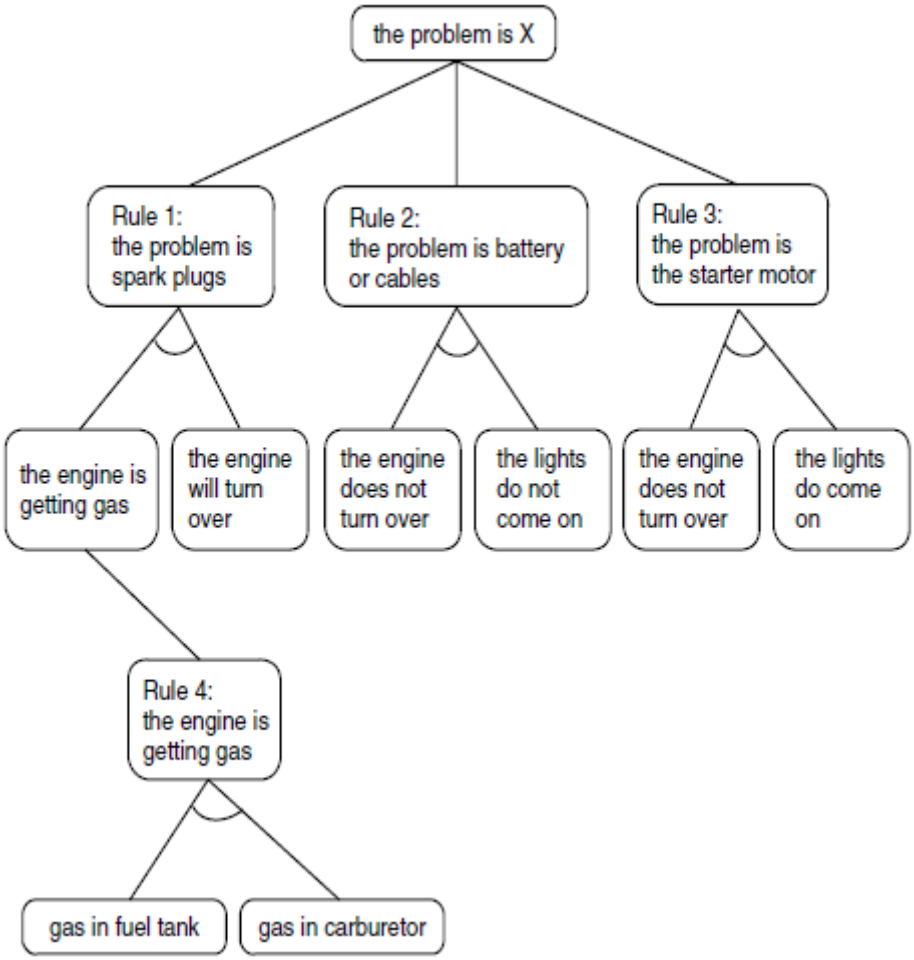
X to zmienna, która może pasować do dowolnej frazy, na przykład problem dotyczy baterii lub kabli; będzie związany z rozwiązaniem, gdy problem zostanie rozwiązany. Trzy reguły pasują do tego wyrażenia w pamięci roboczej: reguła 1, reguła 2 i reguła 3. Jeśli rozwiążemy konflikty na korzyść reguły o najniższym numerze, wówczas zostanie uruchomiona reguła 1. To powoduje, że X wiąże się z wartością świec zapłonowych, a przesłanki reguły 1 są umieszczane w pamięci roboczej, jak na rysunku.



System zdecydował się zatem zbadać możliwą hipotezę, że świece zapłonowe są złe. Innym sposobem spojrzenia na to jest to, że system wybrał gałąź lub gałąź na wykresie i / lub (rozdział 3). Należy zauważyć, że istnieją dwie przesłanki reguły 1, z których obie muszą zostać spełnione, aby wniosek był prawdziwy. Są to i gałęzie wykresu wyszukiwania przedstawiające rozkład problemu (stwierdzenie, czy problemem są świece zapłonowe) na dwa podproblemy (sprawdzenie, czy silnik dostaje gaz i czy silnik się obróci). Możemy wtedy odpalić regułę 4, której konkluzja jest zgodna z otrzymaniem gazu przez silnik, powodując umieszczenie jego przesłanek w pamięci roboczej, jak na rysunku.



W tym momencie w pamięci roboczej znajdują się trzy wpisy, które nie pasują do żadnych wniosków reguł. W takiej sytuacji nasz system ekspercki zapyta użytkownika bezpośrednio o te cele cząstkowe. Jeśli użytkownik potwierdzi, że wszystkie trzy z nich są prawdziwe, system ekspercki z powodzeniem ustali, że samochód nie uruchomi się, ponieważ świece zapłonowe są złe. Znajdując to rozwiązanie, system przeszukał skrajną lewą gałąź wykresu i / lub wykresu przedstawionego na rysunku.



To oczywiście bardzo prosty przykład. Nie tylko jego wiedza motoryzacyjna jest w najlepszym przypadku ograniczona, ale także pomija szereg ważnych aspektów rzeczywistych wdrożeń. Zasady są sformułowane w języku angielskim, a nie w języku formalnym. Po znalezieniu rozwiązania prawdziwy system ekspertowy podpowie użytkownikowi swoją diagnozę (nasz model po prostu się zatrzymuje). Powinniśmy również zachować wystarczająco dużo śladu rozumowania, aby w razie potrzeby umożliwić cofanie się. W naszym przykładzie, gdybyśmy nie stwierdzili, że świece zapłonowe są złe, musielibyśmy cofnąć się do najwyższego poziomu i zamiast tego wypróbować zasadę 2. Zauważ, że ta informacja jest niejawna w kolejności celów podrzędnych w pamięci roboczej na rysunku i na wykresie. Jednak pomimo swojej prostoty, ten przykład podkreśla znaczenie wyszukiwania opartego na systemie produkcyjnym i jego reprezentacji za pomocą i / lub wykresu jako podstawy dla systemów eksperckich opartych na regułach. Wcześniej podkreślaliśmy, że system ekspercki musi być otwarty na wgląd, łatwy do modyfikacji i heurystyczny. Architektura systemu produkcyjnego jest ważnym czynnikiem w każdym z tych wymagań. Na przykład łatwość modyfikacji jest wspierana przez syntaktyczną niezależność reguł produkcji: każda reguła jest kawałkiem wiedzy, który można niezależnie modyfikować. Istnieją jednak ograniczenia semantyczne, ponieważ znaczenia poszczególnych reguł są powiązane. Należy je uzgodnić w każdym procesie edycji / modyfikacji. Następnie omówimy generowanie wyjaśnień i monitorowanie wnioskowania.

Wyjaśnienie i przejrzystość w rozumowaniu ukierunkowanym na cel

System produkcyjny implementuje przeszukiwanie grafów. Procedury wyjaśniające z pierwszego rysunku śledzą przeszukiwanie wykresów i wykorzystują ślad wyszukiwania, aby odpowiadać na zapytania użytkownika. Same reguły produkcji dokumentują każdy etap procesu rozumowania. Ogólnie rzecz biorąc, dwa pytania, na które odpowiadają systemy eksperckie oparte na regułach, to dlaczego? i jak? Pytanie „dlaczego” pojawia się, gdy program prosi użytkownika o informacje, a użytkownik odpowiada „dlaczego”, co oznacza „dlaczego poprosiłeś o te informacje?” Odpowiedzią jest obecna reguła, że system produkcyjny próbuje odpalić. Odpowiedź na pytanie „Jak masz uzyskać taki wynik? ” to sekwencja reguł, które zostały użyte do osiągnięcia celu. Załóżmy, że nasz samochodowy system ekspercki ma funkcję wyjaśniającą i załóżmy, że konsultacja osiągnęła etap reprezentowany przez konfigurację pamięci roboczej na rysunku. Poniższy dialog rozpoczyna się, gdy komputer pyta użytkownika o cele obecne w pamięci roboczej.

gaz w zbiorniku paliwa?

tak

gaz w gaźniku?

tak

silnik się przewróci?

czemu

Tutaj użytkownik wpisuje pytanie „dlaczego”, czyli dlaczego chcesz wiedzieć, czy silnik się obróci? System odpowiada dosłowną prezentacją danej reguły:

Ustalono, że:

1. silnik robi gaz,

dlatego jeśli

2. silnik się przewróci,

wtedy problemem są świece zapłonowe.

Ponieważ użytkownik odpowiedział twierdząco na dwa pierwsze pytania, reguła 4 była w stanie odpalić, weryfikując pierwszą przesłankę reguły 1. Kiedy zadawano pytanie, system próbował rozwiązać drugą przesłankę reguły 1. Zauważ, że wyjaśnienie ponieważ zachowanie systemu jest tylko powtórzeniem aktualnie rozważanej zasady. Daje to jednak odpowiednie wyjaśnienie, dlaczego program chce wiedzieć, czy silnik się obróci. W odpowiedzi na pytanie „Why" system stwierdza, że stwierdził, że silnik pobiera gaz. Załóżmy, że użytkownik zapyta następnie:

jak silnik odpala

System zareaguje na to, jak stwierdził, że silnik pobiera gaz. Odpowiedź jest śladem rozumowania, które doprowadziło do tego wniosku, cofając się od celu, wzdłuż reguł, które go wspierają, do odpowiedzi użytkowników:

Wynika to z zasady 4:

jeśli

gaz w zbiorniku paliwa oraz

gaz w gaźniku

następnie

silnik się pali.

gaz w zbiorniku paliwa podawał użytkownik

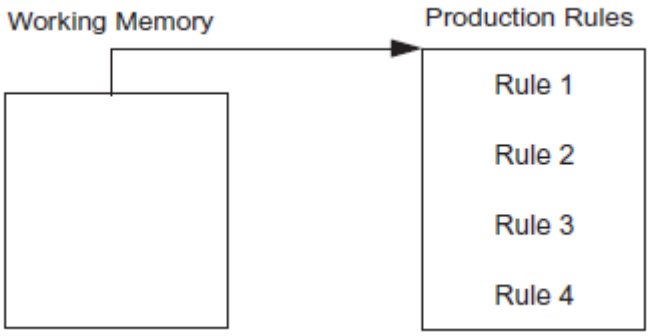
gaz w gaźniku podawał użytkownik

Po raz kolejny architektura systemu produkcyjnego stanowi istotną podstawę dla tych wyjaśnień. Każdy cykl pętli sterowania wybiera i uruchamia inną regułę. Program można zatrzymać po każdym cyklu i sprawdzić. Ponieważ każda reguła reprezentuje kompletny fragment wiedzy na temat rozwiązywania problemów, bieżąca reguła zapewnia kontekst dla wyjaśnienia. Porównaj to podejście do systemu produkcyjnego z bardziej tradycyjnymi architekturami programów: jeśli zatrzymamy program w języku C, C ++ lub Java w trakcie wykonywania, wątpliwe jest, aby bieżąca instrukcja miała duże znaczenie.

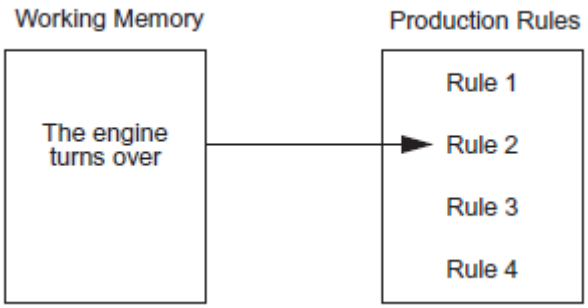
Podsumowując, system oparty na wiedzy odpowiada, dlaczego pyta, pokazując bieżącą regułę, którą próbuje uruchomić; odpowiada w jaki sposób pyta, podając ślad rozumowania, które doprowadziło do celu lub celu cząstkowego. Chociaż mechanizmy są koncepcyjnie proste, mogą wykazywać niezwykłą moc wyjaśniającą, jeśli baza wiedzy jest zorganizowana w sposób logiczny. Dodatkowe materiały w językach Java, Lisp i Prolog demonstrują użycie stosów reguł i drzew próbnych do implementacji tych wyjaśnień. Jeśli wyjaśnienia mają zachowywać się logicznie, ważne jest nie tylko, aby baza wiedzy otrzymała poprawną odpowiedź, ale także aby każda reguła odpowiadała pojedynczemu logicznemu krokowi w procesie rozwiązywania problemów. Jeśli baza wiedzy łączy kilka kroków w jedną regułę lub jeśli łamie reguły w sposób arbitralny, może uzyskać prawidłowe odpowiedzi, ale wydaje się niejasne, arbitralne lub nielogiczne w odpowiadaniu na pytania, jak i dlaczego. Może to nie tylko podważyć wiarę użytkownika w system, ale może również znacznie utrudnić jego konstruktorom zrozumienie i modyfikację

Korzystanie z systemu produkcyjnego do wnioskowania opartego na danych

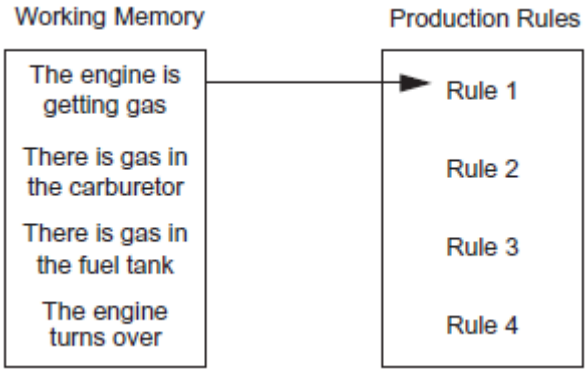
Demonstracja diagnostyki samochodowej zilustrowała użycie systemu produkcyjnego do wdrożenia wyszukiwania ukierunkowanego na cel. Wyszukiwanie było również w pierwszej kolejności w głąb, ponieważ przeszukiwało wszystkie cele cząstkowe znalezione w bazie reguł przed przejściem do celów dla rodzeństwa. Jednak, jak widzieliśmy wcześniej, system produkcyjny jest również idealną architekturą do rozumowania opartego na danych. Przykład zademonstrował ten proces za pomocą 8-puzzle, a dwa kolejne przykłady z trasą skoczka. W przypadku każdego z tych problemów rozwiązywaliśmy konflikty, przyjmując pierwszą regułę znalezioną w bazie wiedzy, a następnie postępowaliśmy zgodnie z wynikami tej reguły. To nadało poszukiwaniom posmak w pierwszej kolejności, chociaż nie było mechanizmu, takiego jak cofanie się, aby poradzić sobie z problemem „ślepych zaułków" w przestrzeni wyszukiwania. Wyszukiwanie wszerz jest jeszcze bardziej powszechne w rozumowaniu opartym na danych. Algorytm jest prosty: porównaj zawartość pamięci roboczej z warunkami każdej reguły w bazie reguł zgodnie z kolejnością reguł w bazie reguł. Jeśli dane w pamięci roboczej obsługują odpalenie reguły, wynik jest umieszczany w pamięci roboczej, a następnie sterowanie przechodzi do następnej reguły. Po uwzględnieniu wszystkich reguł wyszukiwanie rozpoczyna się ponownie od początku zestawu reguł. Rozważmy na przykład problem diagnostyki samochodowej i zasady z poprzedniej sekcji. Jeśli informacja, która stanowi (część) przesłanki reguły, nie jest konkluzją innej reguły, fakt ten zostanie uznany za „możliwy do zaakceptowania", gdy kontrola dojdzie do sytuacji (reguły), w której ta informacja jest potrzebna. Na przykład, silnik pobiera gaz, nie jest możliwy w założeniu reguły 1, ponieważ ten fakt jest konkluzją innej reguły, mianowicie reguły 4. Rozpoczynający się od szerokości, oparty na danych przykład, nie zawiera żadnych informacji. w pamięci roboczej, jak na rysunku.



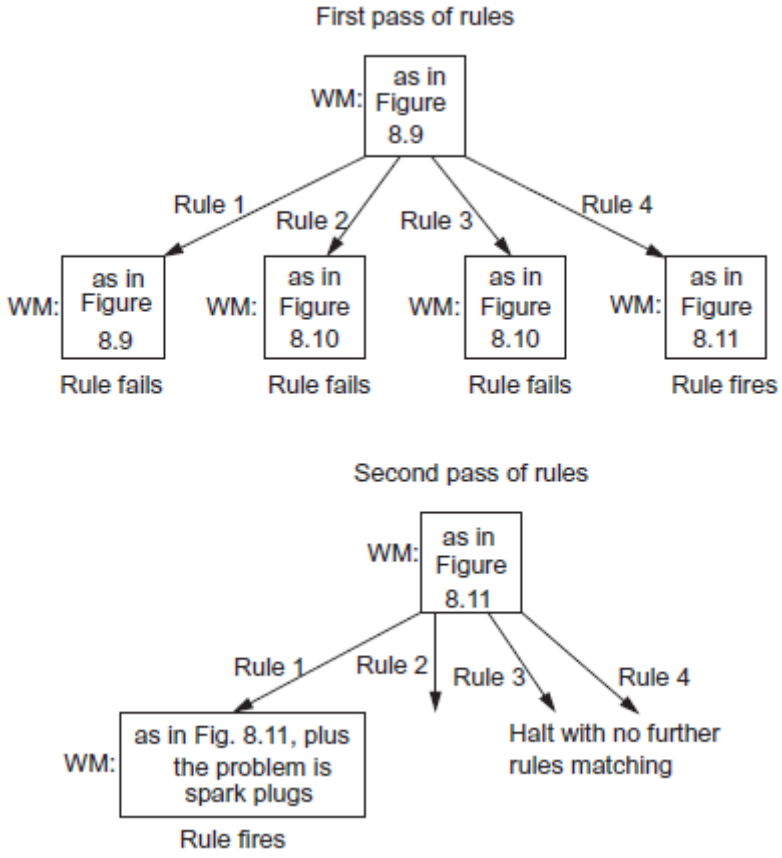
Najpierw przeanalizujemy przesłanki czterech reguł, aby zobaczyć, jakie informacje są „możliwe do ustalenia". Założenie, że silnik jest na gazie, nie jest wykonalne, więc przepis 1 zawodzi i kontroluje ruch do przepisu 2. Silnik nie włącza się i można go ustawić. Załóżmy, że odpowiedź na to zapytanie jest fałszywa, więc silnik się odwróci jest umieszczony w pamięci roboczej, jak na rysunku.



Ale reguła 2 zawodzi, ponieważ pierwsza z dwóch przesłanek jest fałszywa, a rozważenie przenosi się do reguły 3, gdzie ponownie zawodzi pierwsza przesłanka. Zgodnie z zasadą 4 obie przesłanki są dopuszczalne. Załóżmy, że odpowiedź na oba pytania jest prawdziwa. Następnie w zbiorniku paliwa jest gaz i jest gaz w gaźniku są umieszczone w pamięci roboczej, jak wynika z reguły, silnik dostaje gaz. W tym momencie wszystkie reguły zostały uwzględnione, więc funkcja wyszukiwania powraca teraz, z nową zawartością pamięci roboczej, w celu ponownego rozpatrzenia reguł po raz drugi. Jak widać na rysunku, gdy pamięć robocza jest dopasowana do reguły 1, jej konkluzja, czyli problem świec zapłonowych, jest umieszczana w pamięci roboczej.



W tym przykładzie żadne więcej reguł nie będzie pasować ani uruchamiać, a sesja rozwiązywania problemów jest zakończona. Wykres procesu wyszukiwania z zawartością informacyjną pamięci roboczej (WM) jako węzłami wykresu przedstawiono na rysunku.



Ważnym udoskonaleniem strategii wyszukiwania wszerz, zastosowanej w poprzednim przykładzie, jest tak zwane wyszukiwanie oportunistyczne. Ta strategia wyszukiwania jest prosta: za każdym razem, gdy

reguła jest uruchamiana w celu zawarcia nowej informacji, następuje ruch kontrolny w celu rozważenia tych reguł, które mają te nowe informacje jako przesłankę. To sprawia, że każda nowo zawarta informacja (wyszukiwanie nie zmienia się w wyniku „możliwych do przyjęcia" przesłanek) staje się siłą kontrolną dla znalezienia kolejnych reguł strzelania. Nazywa się to oportunistycznym, ponieważ każdy wniosek dotyczący nowych informacji napędza poszukiwania. Przez przypadek uporządkowania reguł, właśnie przedstawiony bardzo prosty przykład był również oportunistyczny. Kończymy tę sekcję dotyczącą rozumowania opartego na danych kilkoma komentarzami na temat wyjaśnień i przejrzystości w systemach łączenia przyszłości. Po pierwsze, w porównaniu z systemami zorientowanymi na cel, poszukiwania oparte na danych są znacznie mniej „skoncentrowane". Przyczyna tego jest oczywista: w systemie zorientowanym na cel rozumowanie to dążenie do określonego celu; ten cel jest podzielony na cele pośrednie, które wspierają cel najwyższego poziomu, a te cele podrzędne mogą być jeszcze bardziej podzielone. W rezultacie wyszukiwanie jest zawsze kierowane przez ten cel i hierarchię celów podrzędnych. W systemach opartych na danych taka orientacja na cel nie istnieje. Raczej wyszukiwanie porusza się po drzewie w zależności tylko od kolejności reguł i odkrycia nowych informacji. W rezultacie postęp poszukiwań może często wydawać się bardzo rozproszony i nieostry.

Po drugie, i jako bezpośredni wynik pierwszego punktu, wyjaśnienie dostępne dla użytkownika w dowolnym momencie wyszukiwania jest dość ograniczone. Istnieje odpowiedzialność na poziomie reguły polegająca na tym, że gdy użytkownik pyta, dlaczego niektóre informacje są wymagane. Wyjaśnienie nie może jednak pójść dużo dalej, chyba że do systemu zostanie dodane wyraźne śledzenie reguł, na przykład wyszukiwanie oportunistyczne. Rozproszony charakter wyszukiwania opartego na danych sprawia, że jest to trudne. Wreszcie, gdy cel zostaje osiągnięty, uzyskanie pełnego wyjaśnienia tego celu jest również trudne. Jedyne, co można wykorzystać jako częściowe i bardzo ograniczone wyjaśnienie, to prezentacja zawartości pamięci roboczej lub lista odpalonych reguł. Ale znowu, nie zapewnią one spójnej, skoncentrowanej odpowiedzialności, jaką widzieliśmy, z rozumowaniem zorientowanym na cel.

Heurystyka i kontrola w systemach ekspertowych

Ze względu na oddzielenie bazy wiedzy i silnika wnioskowania oraz ustalonych reżimów sterowania zapewnianych przez silnik wnioskowania, ważną metodą sterowania wyszukiwaniem przez programistę jest strukturyzacja i uporządkowanie reguł w wiedzy.

Baza. To mikro-zarządzanie zestawem reguł stwarza ważną szansę, zwłaszcza że strategie kontrolne wymagane do rozwiązywania problemów na poziomie eksperckim są zwykle specyficzne dla domeny i wymagają dużej wiedzy. Chociaż reguła postaci, jeśli p, q i r, to s przypomina a wyrażenie logiczne, może być również interpretowane jako seria procedur lub kroków do rozwiązania problemu: zrobić s, najpierw p, potem q, potem r. Rola porządku reguł i przesłanek została ukryta w przykładach przedstawionych właśnie w sekcji 8.2. Ta proceduralna metoda interpretacji reguł jest istotnym składnikiem praktycznego wykorzystania wiedzy i często odzwierciedla strategię rozwiązań ekspertów. Na przykład, możemy tak uporządkować założenia reguły, aby najpierw wypróbować to, co jest najbardziej prawdopodobne, że zawiedzie lub jest najłatwiejsze do potwierdzenia. Daje to możliwość jak najszybszego wyeliminowania reguły (a tym samym części przestrzeni poszukiwań). Reguła 1 w przykładzie motoryzacyjnym próbuje określić, czy silnik pobiera gaz, zanim zapyta, czy silnik się obraca. Jest to nieefektywne, ponieważ próba ustalenia, czy silnik pobiera gaz, wywołuje inną regułę i ostatecznie zadaje użytkownikowi dwa pytania. Odwracając kolejność lokali, negatywna odpowiedź na zapytanie „silnik się obróci?" eliminuje tę regułę z rozważań przed zbadaniem bardziej zaangażowanego stanu, dzięki czemu system jest bardziej wydajny. Bardziej sensowne jest również ustalenie, czy silnik się obraca przed sprawdzeniem, czy dostaje gaz; jeśli silnik się nie uruchamia, nie

ma znaczenia, czy pali się, czy nie! W regule 4 użytkownik jest proszony o sprawdzenie zbiornika paliwa przed sprawdzeniem gaźnika pod kątem gazu. W tej sytuacji najpierw przeprowadza łatwiejszą kontrolę. Jest tu ważna kwestia, jeśli cały system ma być bardziej wydajny, należy wziąć pod uwagę wszystkie aspekty: kolejność reguł, organizację lokali, koszt testów, ilość poszukiwań wyeliminowaną dzięki odpowiedziom na testy, sposób, w jaki występuje większość przypadków , i tak dalej. Zatem planowanie kolejności reguł, organizacja założeń reguły oraz koszty różnych testów mają zasadniczo charakter heurystyczny. W większości systemów eksperckich opartych na regułach te heurystyczne wybory odzwierciedlają podejście przyjęte przez ludzkiego eksperta i rzeczywiście mogą mieć błędne wyniki. W naszym przykładzie, jeśli silnik nabiera gazu i się kręci, problemem może być raczej zły dystrybutor, a nie złe świece zapłonowe. Rozumowanie oparte na danych zapewnia dodatkowe problemy i możliwości kontrolowania rozumowania. Niektóre z nich obejmują wysokopoziomowe heurystyki, takie jak refrakcja, niedawność (wyszukiwanie oportunistyczne) i szczegółowość. Bardziej specyficzne dla domeny podejście grupuje zestawy reguł według etapów procesu rozwiązania. Na przykład podczas diagnozowania problemów z samochodem możemy stworzyć cztery odrębne etapy: 1) uporządkowania sytuacji, 2) zebrania danych, 3) przeprowadzenia analizy (może być więcej niż jeden problem z samochodem) i wreszcie 4) raportu użytkownikowi wnioski i zalecane poprawki. To etapowe rozwiązywanie problemów można osiągnąć, tworząc deskryptory dla każdego etapu rozwiązania i umieszczając ten opis jako pierwszą przesłankę we wszystkich regułach należących do tego etapu. Na przykład, możemy zacząć od umieszczenia w pamięci roboczej sytuacji zorganizowanej asercji. Jeśli w pamięci roboczej nie pojawią się żadne inne opisy etapów, to uruchomione zostaną tylko reguły, które mają zorganizowaną sytuację w swoim zbiorze pomieszczeń. Oczywiście powinno to być pierwszą przesłanką każdej z tych reguł. Przechodzilibyśmy do następnego etapu, mając ostatnią regułę do odpalenia na etapie organizacyjnym, usuwając (wycofując) fakt, że etap jest organizacją rozwiązania i potwierdzając nowy fakt, że etapem jest zbieranie danych. Wszystkie zasady na etapie gromadzenia danych miałyby wtedy swoje pierwsze założenie, JEŚLI etap jest gromadzeniem danych i. . .. Po zakończeniu etapu zbierania danych ostatnia reguła wycofałaby ten fakt i wprowadziła fakt analizy danych do pamięci roboczej, aby dopasować tylko te reguły, których przesłanki zaczynają się od faktu, że etap JEŻELI to analiza danych. . .. W naszej dotychczasowej dyskusji opisaliśmy zachowanie systemu produkcyjnego w kategoriach wyczerpujących rozważań na temat podstawy reguł. Chociaż jest to kosztowne, oddaje zamierzoną semantykę systemu produkcyjnego. Istnieje jednak kilka algorytmów takie jak RETE, których można użyć do optymalizacji wyszukiwania dla wszystkich potencjalnie użytecznych reguł. Zasadniczo algorytm RETE kompiluje reguły w strukturę sieciową, która umożliwia systemowi dopasowanie reguł do danych przez bezpośrednie podążanie za wskaźnikiem do reguły. Algorytm ten znacznie przyspiesza wykonanie, szczególnie w przypadku większych zestawów reguł, zachowując zachowanie semantyczne, które opisaliśmy w tej sekcji.

Podsumowując, reguły są najstarszym podejściem do reprezentacji wiedzy w systemach ekspertowych i pozostają ważną techniką budowania rozwiązań problemów wymagających dużej wiedzy. Reguły systemu ekspertowego wychwytują ludzką wiedzę ekspercką stosowaną w praktyce; w konsekwencji są one często połączeniem wiedzy teoretycznej, heurystyki wywodzącej się z doświadczenia i specjalnych reguł postępowania w przypadku dziwnych przypadków i innych wyjątków od normalnej praktyki. W wielu sytuacjach to podejście okazało się skuteczne. Niemniej jednak mocno systemy heurystyczne mogą zawieść, albo napotykając problem, który nie pasuje do żadnych dostępnych reguł, albo przez niewłaściwe zastosowanie reguły heurystycznej do niewłaściwej sytuacji. Ludzcy eksperci nie cierpią z powodu tych problemów, ponieważ mają głębsze, teoretyczne zrozumienie dziedziny problemowej, które pozwala im inteligentnie stosować reguły heurystyczne lub uciekać się do wnioskowania z „pierwszych zasad” w nowych sytuacjach. Podejścia oparte na modelach, opisane dalej, są próbą nadania systemowi eksperckiemu takiej mocy i elastyczności. Umiejętność uczenia się

na przykładach to kolejna ludzka zdolność, którą naśladują osoby zajmujące się rozwiązywaniem problemów intensywnie wykorzystujące wiedzę. Rozumowania oparte na przypadkach, prowadzą bazę wiedzy zawierającą przykładowe rozwiązania problemów lub przypadki. W obliczu nowego problemu, osoba rozumująca wybiera z tego przechowywanego zestawu przypadek przypominający obecny problem, a następnie próbuje zastosować formę strategii rozwiązania tego problemu. W rozumowaniu prawniczym argumentacja poprzez precedens jest typowym przykładem rozumowania opartego na przypadkach.

Systemy oparte na modelach, oparte na przypadkach i hybrydowe

Wprowadzenie do rozumowania opartego na modelu

Ekspertyza ludzka jest niezwykle złożonym połączeniem wiedzy teoretycznej, heurystyk rozwiązywania problemów opartych na doświadczeniu, przykładów przeszłych problemów i ich rozwiązań, umiejętności percepcyjnych i interpretacyjnych oraz innych zdolności, które są tak słabo rozumiane, że możemy je opisać jedynie jako intuicyjne. Dzięki wieloletniemu doświadczeniu ludzcy eksperci opracowują bardzo potężne zasady postępowania w często spotykanych sytuacjach. Reguły te są często w dużym stopniu „kompilowane", przybierając postać bezpośrednich skojarzeń między obserwowalnymi objawami a ostatecznymi diagnozami i ukrywając ich głębiej wyjaśniające podstawy. Na przykład system ekspercki MYCIN zaproponowałby diagnozę na podstawie takich obserwowalnych objawów, jak „bóle głowy", „nudności" czy „wysoka gorączka". Chociaż parametry te mogą wskazywać na chorobę, zasady, które łączą je bezpośrednio z diagnozą, nie odzwierciedlają głębszego, przyczynowego zrozumienia fizjologii człowieka. Zasady MYCIN wskazują na skutki infekcji, ale nie wyjaśniają jej przyczyn. Głębsze podejście wyjaśniające pozwoliłoby wykryć obecność czynników infekcyjnych, odnotować wynikający z tego stan zapalny wyściółki komórkowej, obecność ciśnień międzyczaszkowych i wywnioskować związek przyczynowy z obserwowanymi objawami bólu głowy, podwyższonej temperatury i nudności. W opartym na regułach systemie eksperckim, na przykład do analizy uszkodzeń półprzewodników, podejście opisowe może oprzeć diagnozę awarii obwodu na takich objawach, jak odbarwienie komponentów (prawdopodobnie wskazujące na wypalony komponent), historia usterek w podobnych urządzeniach lub nawet obserwacje wnętrz elementów za pomocą mikroskopu elektronowego.

Jednak podejścia wykorzystujące reguły do łączenia obserwacji i diagnoz nie zapewniają korzyści w postaci głębszej analizy struktury i funkcji urządzenia. Bardziej solidne, dogłębnie wyjaśniające podejście rozpoczynałoby się od szczegółowego modelu fizycznej struktury obwodu i równań opisujących oczekiwane zachowanie każdego elementu i ich interakcje. Oparłby swoją diagnostykę na odczytach sygnałów z różnych miejsc w urządzeniu, wykorzystując te dane i model obwodu do określenia dokładnych punktów awarii.

Ponieważ systemy eksperckie pierwszej generacji opierały się na regułach heurystycznych uzyskanych z opisu technik rozwiązywania problemów dokonanego przez ekspertów, wykazywały one szereg fundamentalnych ograniczeń (Clancy 1985). Gdyby wystąpienie problemu nie odpowiadało ich heurystyce, po prostu zawodziło, nawet jeśli bardziej teoretyczna analiza przyniosłaby rozwiązanie. Często systemy eksperckie stosowały heurystykę w nieodpowiednich sytuacjach, na przykład gdy głębsze zrozumienie problemu wskazywało inny kierunek. Są to ograniczenia, które próbują rozwiązać podejścia oparte na modelach. Rozumowanie oparte na wiedzy, którego analiza opiera się bezpośrednio na specyfikacji i funkcjonalności systemu fizycznego, nazywa się systemem opartym na modelach. W swoim projekcie i zastosowaniu rozumownik oparty na modelu tworzy symulację oprogramowania, często określaną jako „jakościowa", funkcji tego, co ma być zrozumiane lub naprawione. (Istnieją oczywiście inne typy systemów opartych na modelach, w szczególności oparte

na logice i stochastyczne, które przedstawiamy w kolejnej Części). Najwcześniejsze rozumowania oparte na modelach pojawiły się w połowie lat 70, dojrzewa w latach 80-tych. Warto zauważyć, że niektóre z najwcześniejszych prac miały na celu stworzenie modeli oprogramowania różnych urządzeń fizycznych, takich jak obwody elektroniczne, do celów instruktażowych. W tych wczesnych sytuacjach inteligentnego nauczania specyfikacje urządzenia lub obwodu były odzwierciedlane w zestawach reguł, np. prawach Kirchoffa i Ohma. System korepetycji został przetestowany przez wiedzę ucznia na temat urządzenia lub obwodu, jak również przekazana uczniowi wiedza, której może brakować. Reguły były zarówno reprezentacją funkcjonalności sprzętu, jak i środkiem przekazywania tej wiedzy uczniowi. Z tych wczesnych systemów nauczania, w których zadaniem było zarówno modelowanie, jak i nauczanie funkcjonalności systemu, osoby rozumujące oparte na modelach jakościowych przeszły do systemów rozwiązywania problemów. Podczas rozwiązywania problemów w systemie fizycznym model prowadzi do zestawów przewidywanych zachowań. Błąd znajduje odzwierciedlenie w rozbieżności między przewidywanym a obserwowanym zachowaniem. System oparty na modelach informuje użytkownika, czego ma się spodziewać, a gdy obserwacje różnią się od tych oczekiwań, w jaki sposób te rozbieżności prowadzą do identyfikacji błędów. Jakościowe rozumowanie oparte na modelu obejmuje:

1. Opis każdego komponentu w urządzeniu. Te opisy mogą symulować zachowanie komponentu.

2. Opis wewnętrznej struktury urządzenia. Zwykle jest to reprezentacja jego komponentów i ich wzajemnych połączeń, wraz z możliwością symulacji interakcji komponentów. Zakres wymaganej wiedzy o strukturze wewnętrznej zależy od zastosowanych poziomów abstrakcji i pożądanej diagnozy.

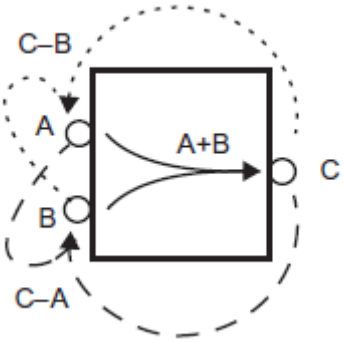
3. Diagnoza konkretnego problemu wymaga obserwacji rzeczywistego działania urządzenia, zazwyczaj pomiarów jego wejść i wyjść. Pomiary we / wy są najłatwiejsze do uzyskania, ale w rzeczywistości można zastosować dowolną miarę.

Zadaniem jest następnie określenie, który z komponentów mógł zawieść w sposób uwzględniający obserwowane zachowania. Wymaga to dodatkowych reguł, które opisują znane tryby awarii dla różnych komponentów i ich połączeń. Rozumujący musi znaleźć najbardziej prawdopodobne awarie, które mogą wyjaśnić obserwowane zachowanie systemu. Szereg struktur danych może służyć do reprezentowania informacji przyczynowych i strukturalnych w modelach. Wielu projektantów programów opartych na modelach używa reguł, aby odzwierciedlić przyczynowość i funkcjonalność urządzenia. Reguł można również używać do przechwytywania relacji między komponentami. System zorientowany obiektowo oferuje również doskonałe narzędzie reprezentacyjne do odzwierciedlania struktury urządzenia i komponentów w modelu, ze szczelinami obiektu reprezentującymi stan urządzenia lub komponentu, a metody obiektu lub klasy definiują jego funkcjonalność. Aby być bardziej konkretnym w projektowaniu i ocenie modelu, rozważymy teraz kilka przykładów analizy urządzeń i obwodów z Davis i Hamscher (1992). Zachowanie urządzenia jest reprezentowane przez zestaw wyrażeń, które przechwytują relacje między wartościami na terminalach urządzenia. W przypadku sumatora będą trzy wyrażenia:

Jeśli znamy wartości w A i B, wartość C to A + B (linia ciągła).

Jeśli znamy C i A, wartość w B to C - A (linia przerywana).

Jeśli znamy C i B, wartość w A to C - B (linia przerywana).

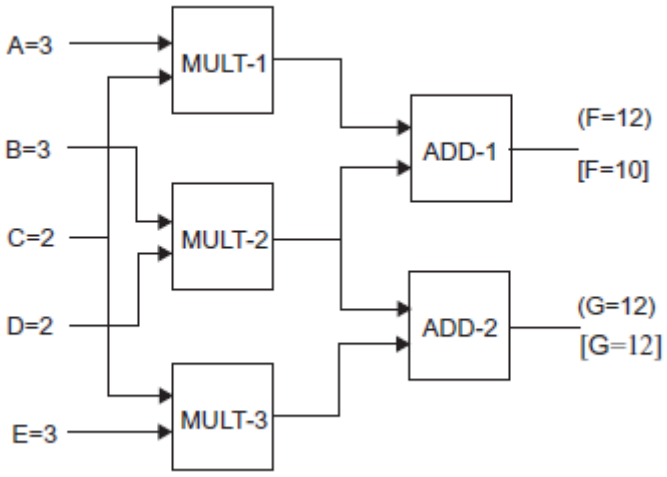


Nie musieliśmy używać formy algebraicznej do przedstawienia tych relacji. Równie dobrze mogliśmy użyć krotek relacyjnych lub przedstawić ograniczenia za pomocą funkcji Lispa. Celem rozumowania opartego na modelu jest przedstawienie wiedzy, która oddaje funkcjonalność sumatora. W drugim przykładzie rozważ obwód trzech mnożników i dwóch sumatorów połączonych jak na rysunku ponizej. W tym przykładzie wartości wejściowe są podane od A do E, a wartości wyjściowe w F i G. Oczekiwane wartości wyjściowe są podane w (), a rzeczywiste wartości wyjściowe w []. Zadanie polega na ustaleniu, gdzie leży usterka, która wyjaśni tę rozbieżność. W F mamy konflikt, oczekując 12 i otrzymując 10. Sprawdzamy zależności w tym miejscu i ustalamy, że wartość w F jest funkcją ADD-1, która z kolei zależy od wyników MULT-1 i MULT- 2. Jedno z tych trzech urządzeń musi mieć usterkę, więc mamy trzy hipotezy do rozważenia: albo zachowanie sumatora jest złe, albo jedno z jego dwóch wejść było nieprawidłowe, a problem leży dalej w obwodzie. Rozumując na podstawie wyniku (10) w F i zakładając prawidłowe zachowanie ADD-1 i jednego z jego danych wejściowych X (6), dane wejściowe Y do ADD-1 muszą być 4. Ale to jest sprzeczne z oczekiwaniem 6, które jest prawidłowe zachowanie MULT-2 i wejść B i D. Zaobserwowaliśmy te wejścia i wiemy, że są poprawne, więc MULT-2 musi być uszkodzony. W równoległym argumencie nasza druga hipoteza jest taka, że ADD-1 jest poprawny, a MULT-1 jest błędny. Kontynuując to rozumowanie, jeśli pierwsze wejście X do ADD-1 jest poprawne, a sam ADD-1 jest poprawne, to drugie wejście Y musi być 4. Gdyby było 4, G byłoby 10 zamiast 12, więc wyjście MULT-2 musi być liczbą 6 i poprawną. Pozostajemy z hipotezami, że wina leży w MULT-1 lub ADD-1 i możemy nadal ograniczać te urządzenia do dalszych testów. W naszym rozumowaniu sytuacji przedstawionej na rysunku poniżej mieliśmy trzy zadania:

1. Generowanie hipotezy, w której biorąc pod uwagę rozbieżność, postawiliśmy hipotezę, które elementy urządzenia mogły ją wywołać.

2. Testowanie hipotez, w ramach którego, mając zbiór potencjalnych wadliwych komponentów, określiliśmy, które z nich mogły wyjaśnić obserwowane zachowanie.

3. Rozróżnianie hipotez, w którym gdy więcej niż jedna hipoteza przetrwa fazę testowania, jak to miało miejsce w przypadku rysunku, musimy określić, jakie dodatkowe informacje można zebrać, aby kontynuować poszukiwanie usterki.



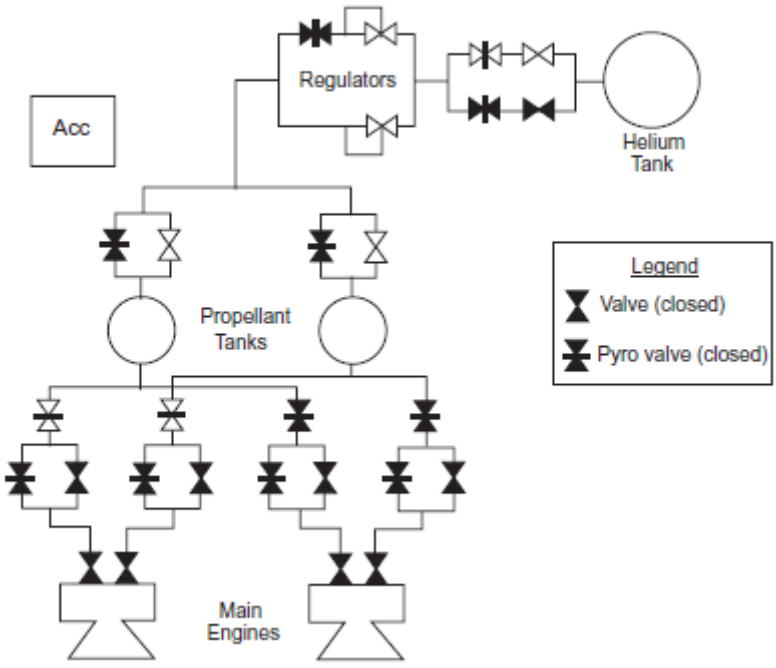
Na koniec należy zauważyć, że w przykładzie z rysunku założono, że istnieje jedno wadliwe urządzenie. Świat nie zawsze jest tak prosty, chociaż założenie o pojedynczym błędzie jest użyteczną i często poprawną heurystyką. Ponieważ są one oparte na teoretycznym zrozumieniu omawianych urządzeń, jakościowe techniki oparte na modelach eliminują wiele ograniczeń bardziej heurystycznych podejść. Zamiast wnioskować bezpośrednio od obserwowanych zjawisk do wyjaśnień przyczynowych, podejścia oparte na modelach próbują przedstawić urządzenia i konfiguracje urządzeń na poziomie przyczynowym lub funkcjonalnym. Kod programu odzwierciedla zarówno funkcję urządzeń, jak i zależności w systemie urządzeń. Takie modele są często bardziej niezawodne niż podejścia heurystyczne. Jednak wadą tego wyraźnego modelowania funkcji jest to, że etap zdobywania wiedzy może być dość wymagający, a wynikający z niego program duży, uciążliwy i powolny. Ponieważ podejścia heurystyczne „kompilują" typowe przypadki w jedną regułę, są one często bardziej wydajne, o ile odpowiednie są inne ograniczenia systemowe.

Z takim podejściem wiążą się jednak głębsze problemy. Podobnie jak w przypadku rozumowania opartego na regułach, model systemu jest po prostu modelem. Z konieczności będzie to abstrakcja systemu, a na pewnym poziomie szczegółowości będzie niepoprawna. Na przykład rozważ przewody wejściowe z rysunku. W naszej dyskusji uznaliśmy te wartości za podane i prawidłowe. Nie zbadaliśmy stanu samego drutu, aw szczególności drugiego końca, w którym łączył się z mnożnikami. Co by się stało, gdyby przewód został zerwany lub miał wadliwe połączenie z mnożnikiem? Jeśli użytkownik nie wykrył tego wadliwego połączenia, model nie pasowałby do rzeczywistego urządzenia.

Każdy model próbuje opisać idealną sytuację, co system ma robić, a niekoniecznie to, co robi system. Błąd „mostkujący" to punkt styku w systemie, w którym dwa przewody lub urządzenia są nieumyślnie połączone, jak w przypadku złego połączenia lutowanego mostki między dwoma przewodami, które nie powinny się stykać. Większość rozumowań opartych na modelach ma trudności z postawieniem hipotezy błędu pomostowego ze względu na założenia a priori leżące u podstaw modelu i metody wyszukiwania służące do określania anomalii. Błędy mostkowania to po prostu „nowe" przewody, które nie są częścią oryginalnego projektu. Istnieje domniemane „założenie o zamkniętym świecie" (sekcja 9.1), że zakłada się, że opis struktury modelu jest kompletny, a wszystko, czego nie ma w modelu, po prostu nie istnieje. Pomimo tych niedociągnięć rozumowanie oparte na modelach jest ważnym dodatkiem do zestawu narzędzi inżyniera wiedzy. Naukowcy nadal poszerzają naszą wiedzę na temat diagnozy, zarówno tego, w jaki sposób eksperci od ludzi robią to tak skutecznie, jak również w jaki sposób lepsze algorytmy mogą być wdrażane na maszynach.

Rozumowanie oparte na modelach: przykład NASA (Williams i Nayak)

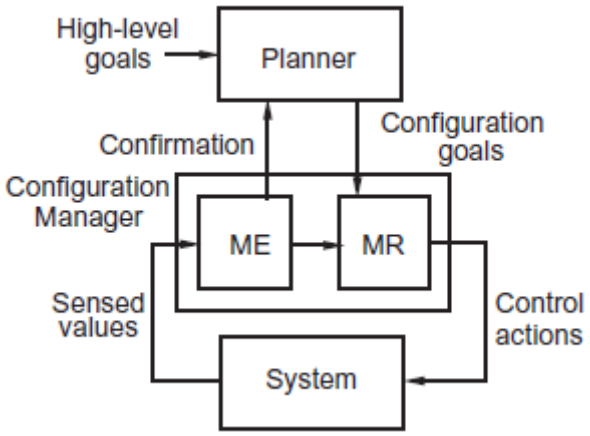
NASA wsparła swoją obecność w kosmosie, rozwijając flotę inteligentnych sond kosmicznych, które w sposób autonomiczny badają Układ Słoneczny. Wysiłki te rozpoczęto od oprogramowania dla pierwszej sondy w 1997 r. I wystrzelenia Deep Space 1 w 1998 r. Aby osiągnąć sukces przez lata w trudnych warunkach podróży kosmicznych, statek musi być w stanie radykalnie zmienić swój system kontroli w odpowiedzi na awarie, a następnie zaplanuj ich obejście podczas pozostałego lotu. Aby osiągnąć akceptowalny koszt i szybką rekonfigurację, trzeba będzie szybko złożyć jedyne w swoim rodzaju moduły, aby automatycznie generować oprogramowanie lotnicze. Wreszcie, NASA spodziewa się, że zestaw potencjalnych scenariuszy awarii i możliwych odpowiedzi będzie o wiele za duży, aby używać oprogramowania obsługującego wyliczanie wszystkich zdarzeń losowych przed inspekcją. Zamiast tego statek kosmiczny będzie musiał reaktywnie przemyśleć wszystkie konsekwencje swoich opcji rekonfiguracji. Livingstone to zaimplementowane jądro dla systemu opartego na modelu reaktywnego samokonfigurującego się systemu autonomicznego. Językiem reprezentacji rozumowania opartym na modelach dla Livingstone'a jest rachunek zdań, odejście od rachunku predykatów pierwszego rzędu, tradycyjnego języka reprezentacji dla diagnozy opartej na modelach. Williams i Nayak, opierając się na swoich wcześniejszych badaniach nad budowaniem szybkich algorytmów diagnostycznych opartych na konfliktach logiki propozycji, uznali, że możliwy jest szybki system reaktywny, dokonujący znacznych dedukcji w pętli sens / odpowiedź. Długotrwała wizja rozumowania opartego na modelach polegała na wykorzystaniu jednego scentralizowanego modelu do obsługi różnych zadań inżynieryjnych. W przypadku systemów autonomicznych opartych na modelach oznacza to używanie jednego modelu do obsługi różnorodnych zadań wykonawczych. Obejmują one śledzenie opracowywania planów, potwierdzanie trybów sprzętu, rekonfigurację sprzętu, wykrywanie anomalii, diagnostykę i usuwanie usterek. Livingstone automatyzuje wszystkie te zadania za pomocą jednego modelu i jednego podstawowego algorytmu, czyniąc w ten sposób znaczący postęp w kierunku osiągnięcia wizji rozwiązywania problemów opartych na modelach. Rysunek przedstawia wyidealizowany schemat głównego podzespołu silnika Cassini, najbardziej złożonego statku kosmicznego zbudowanego do tej pory.



Składa się ze zbiornika helu, zbiornika paliwa, zbiornika utleniacza, pary głównych silników, regulatorów, zaworów zatrzaskowych, pirozaworów i rur. Zbiornik helu zwiększa ciśnienie w dwóch

zbiornikach paliwa pędnego, a regulatory działają w celu zmniejszenia wysokiego ciśnienia helu do niższego ciśnienia roboczego. Gdy ścieżki paliwa do głównego silnika są otwarte (ikona zaworu jest niewypełniona), zbiornik ciśnieniowy wtłacza paliwo i utleniacz do głównego silnika, gdzie łączą się, spontanicznie zapalają i wytwarzają ciąg. Piro-zawory można odpalić dokładnie raz, to znaczy mogą zmienić stan tylko raz, przechodząc z otwartego na zamknięty lub odwrotnie. Ich zadaniem jest izolowanie części głównego podsystemu silnika, dopóki nie będą potrzebne, lub trwałe odizolowanie uszkodzonych komponentów. Zatrzaski zaworów są sterowane za pomocą sterowników zaworów (nie pokazano na rysunku), a Acc (akcelerometr) wykrywa ciąg generowany przez silniki główne. Zaczynając od konfiguracji pokazanej na rysunku, nadrzędny cel, jakim jest wytworzenie ciągu, można osiągnąć za pomocą wielu różnych konfiguracji: ciąg może być zapewniony przez dowolny z głównych silników i istnieje wiele sposobów otwierania ścieżek paliwa pędnego do jednego z nich. główny silnik. Na przykład, ciąg można uzyskać przez otwarcie zaworów zatrzaskowych prowadzących do silnika po lewej stronie lub przez odpalenie pary pyro i otwarcie zestawu zaworów zatrzaskowych prowadzących do silnika po prawej stronie. Inne konfiguracje odpowiadają różnym kombinacjom wypalania pirotechnicznego. Różne konfiguracje mają różne cechy ponieważ odpalanie pirozaworów jest działaniem nieodwracalnym, a odpalanie pirozaworów wymaga znacznie większej mocy niż otwieranie lub zamykanie zaworów zatrzaskowych. Załóżmy, że główny podsystem silnika został skonfigurowany tak, aby zapewnić ciąg z lewego silnika głównego, otwierając prowadzące do niego zawory zatrzaskowe. Załóżmy, że ten silnik ulegnie awarii, na przykład przez przegrzanie, tak że nie zapewnia wymaganego ciągu. Aby zapewnić pożądany ciąg nawet w tej sytuacji, statek kosmiczny musi zostać przestawiony do nowej konfiguracji, w której ciąg jest teraz zapewniany przez silnik główny po prawej stronie. W idealnym przypadku osiąga się to poprzez odpalenie dwóch pirozaworów prowadzących na prawą stronę i otwarcie pozostałych zaworów zatrzaskowych, zamiast odpalania dodatkowych zaworów piroelektrycznych. Menedżer konfiguracji nieustannie próbuje przenieść statek kosmiczny na konfiguracje o najniższych kosztach, które osiągają zestaw dynamicznie zmieniających się celów wysokiego poziomu. Kiedy statek kosmiczny zboczy z wybranej konfiguracji z powodu awarii, kierownik analizuje dane z czujników, aby zidentyfikować bieżącą konfigurację statku kosmicznego, a następnie przenosi statek kosmiczny do nowej konfiguracji, która ponownie osiąga pożądane cele konfiguracyjne. Menedżer konfiguracji to dyskretny system sterowania, który zapewnia, że konfiguracja statku kosmicznego zawsze osiąga wartość zadaną zdefiniowaną przez cele konfiguracji. Rozumowanie na temat konfiguracji (i autonomicznych rekonfiguracji) systemu wymaga koncepcji trybów pracy i awarii, napraw możliwych do naprawy awarii i zmian konfiguracji. NASA wyraża te koncepcje na diagramie przestrzeni stanów: naprawialne awarie to przejścia od stanu awarii do stanu nominalnego; zmiany konfiguracji są między stanami nominalnymi; a awarie są przejściami od stanu nominalnego do stanu awarii. Williams i Nayak postrzegają system autonomiczny jako połączenie reaktywnego menedżera konfiguracji i planisty wysokiego poziomu. Planer, , generuje sekwencję celów konfiguracji sprzętowej. Menedżer konfiguracji rozwija system przejściowy domeny aplikacji, w tym przypadku system napędowy statku kosmicznego, wzdłuż żądanej trajektorii. W ten sposób zarządzanie konfiguracją osiąga się poprzez wykrywanie i kontrolowanie stanu systemu przejściowego. Menedżer konfiguracji oparty na modelu to menedżer konfiguracji, który używa specyfikacji systemu przejścia do obliczenia żądanej sekwencji wartości sterujących. Na przykładzie z rysunku powyżej każdy komponent sprzętowy jest modelowany przez system przejścia komponentów. Komunikacja między komponentami, oznaczona na rysunku przewodami (łączami), jest modelowana za pomocą wspólnych terminów zdaniowych między odpowiednimi systemami przejść składowych. Menedżer konfiguracji szeroko wykorzystuje model, aby wywnioskować bieżący stan systemu i wybrać optymalne działania kontrolne, aby osiągnąć cele konfiguracji. Jest to niezbędne w sytuacjach, w których błędy mogą prowadzić do katastrofy, wykluczając proste metody prób i błędów. Menedżer konfiguracji oparty na modelu używa

modelu do określenia żądanej sekwencji sterowania w dwóch etapach: oszacowanie trybu i rekonfiguracja trybu (ME i MR na rysunku).



ME w sposób przyrostowy generuje zestaw wszystkich trajektorii systemu, zgodnie z modelem przejścia zakładu oraz sekwencją sterowania systemem i wykrytymi wartościami. MR wykorzystuje model przejścia zakładu i częściowe trajektorie generowane przez ME aż do stanu bieżącego w celu określenia zestawu wartości kontrolnych, tak aby wszystkie przewidywane trajektorie osiągały cel konfiguracji następnego stanu. Zarówno ME, jak i MR są reaktywne. ME wnioskuje o aktualnym stanie na podstawie wiedzy o poprzednim stanie i obserwacji w obecnym stanie. MR bierze pod uwagę tylko działania, które osiągają cel konfiguracji w następnym stanie. W następnej sekcji rozważymy rozumowanie oparte na przypadkach, technikę wymagającą dużej wiedzy, która wspiera ponowne wykorzystanie wcześniejszych doświadczeń w domenie problemowej do rozwiązywania nowych sytuacji.

Wprowadzenie do rozumowania opartego na przypadkach

Reguły heurystyczne i modele teoretyczne to dwa rodzaje informacji, których eksperci używają do rozwiązywania problemów. Innym potężnym strategiem używanym przez ekspertów jest rozumowanie na podstawie przypadków, przykładów problemów z przeszłości i ich rozwiązań. Rozumowanie oparte na przypadkach (CBR) wykorzystuje jawną bazę danych rozwiązań problemów w celu uwzględnienia nowych sytuacji wymagających rozwiązania problemu. Rozwiązania te mogą być zbierane od ekspertów w procesie inżynierii wiedzy lub mogą odzwierciedlać wyniki poprzednich sukcesów lub niepowodzeń związanych z wyszukiwaniem. Na przykład medyczna edukacja nie opiera się wyłącznie na teoretycznych modelach anatomii, fizjologii i chorób; zależy to również w dużym stopniu od historii przypadków i doświadczenia stażysty z innymi pacjentami i ich leczenia. CASEY (Koton 1988a, b) i PROTOS (Bareiss i wsp. 1988) są przykładami rozumowania opartego na przypadkach stosowanego w medycynie. Prawnicy wybierają przeszłe sprawy prawne, które są podobne do tych ich klientów i sugerują korzystne decyzje i spróbuj przekonać sąd, że te podobieństwa wymagają podobnych ustaleń. Chociaż ogólne prawa są tworzone przez procesy demokratyczne, ich interpretacja jest zwykle oparta na precedensach prawnych. To, jak prawo zostało zinterpretowane w jakiejś wcześniejszej sytuacji, ma kluczowe znaczenie dla jego obecnej interpretacji. Zatem ważnym elementem rozumowania prawnego jest identyfikacja z precedensów orzecznictwa dotyczących decyzji w konkretnej sprawie.

Rissland oraz Rissland i Ashley opracowali argumenty oparte na przypadkach, aby wspierać argumenty prawne. Programiści komputerowi często ponownie wykorzystują swój kod, dostosowując stary program do nowej sytuacji o podobnej strukturze. Architekci wykorzystują swoją wiedzę o

estetycznych i użytecznych budynkach z przeszłości, aby projektować nowe budynki, które ludzie uważają za przyjemne i wygodne. Historycy wykorzystują historie z przeszłości, aby pomóc mężom stanu, biurokratom i obywatelom zrozumieć przeszłe wydarzenia i zaplanować przyszłość. Umiejętność wnioskowania na podstawie przypadków ma fundamentalne znaczenie dla ludzkiej inteligencji. Inne oczywiste obszary wnioskowania na podstawie przypadków obejmują projektowanie, w którym aspekty pomyślnie wykonanego artefaktu mogą być odpowiednie dla nowej sytuacji, oraz diagnozę, w której często powtarzają się niepowodzenia z przeszłości. Diagnoza sprzętu jest tego dobrym przykładem. Ekspert w tej dziedzinie, oprócz wykorzystania rozległej wiedzy teoretycznej na temat układów elektronicznych i mechanicznych, wnosi udane i nieudane doświadczenia diagnostyczne z przeszłości do aktualnego problemu. CBR był ważnym komponentem wielu sprzętowych systemów diagnostycznych, w tym prac nad konserwacją źródeł sygnału i baterii w satelitach okrążających ziemię (Skinner i Luger 1992) oraz analizą uszkodzeń półprzewodników składowych dyskretnych (Stern i Luger 1997). CBR oferuje szereg korzyści przy budowie systemów ekspertowych. Zdobywanie wiedzy można uprościć, jeśli zapiszemy rozwiązania wielu problemów, które zostały rozwiązane przez eksperta, a osoba odpowiedzialna za rozważanie w oparciu o przypadek wybierze i wnioskuje z odpowiedniego przypadku. Oszczędziłoby to inżynierowi wiedzy trudu tworzenia ogólnych reguł na podstawie przykładów eksperta; zamiast tego rozumujący uogólniłby zasady automatycznie, poprzez proces stosowania ich do nowych sytuacji. Podejścia oparte na przypadkach mogą również umożliwić systemowi ekspertowemu wyciąganie wniosków z własnego doświadczenia. Po znalezieniu rozwiązania problemu opartego na wyszukiwaniu system może zapisać to rozwiązanie, dzięki czemu następnym razem, gdy wystąpi podobna sytuacja, wyszukiwanie nie będzie konieczne. Ważne może być również przechowywanie w bazie przypadków informacji o powodzeniu lub niepowodzeniu poprzednich prób rozwiązania; w ten sposób CBR oferuje potężny model uczenia się. Wczesnym tego przykładem jest program gry w warcaby Samuel, w którym pozycje na szachownicy, które podczas poszukiwań lub doświadczenia uznano za ważne, są zachowywane na wypadek, gdyby ta pozycja powtórzyła się w późniejszej grze. Rozumowani na podstawie przypadków mają wspólną strukturę. W przypadku każdego nowego problemu:

1. Pobierz odpowiednie przypadki z pamięci. Przypadek jest odpowiedni, jeśli jego rozwiązanie można z powodzeniem zastosować w nowej sytuacji. Ponieważ osoby rozumujące nie mogą wiedzieć tego z góry, zazwyczaj używają heurystyki wybierania przypadków podobnych do wystąpienia problemu. Zarówno ludzie, jak i sztuczni myśliciele określają podobieństwo na podstawie wspólnych cech: na przykład, jeśli dwóch pacjentów ma wiele wspólnych cech w objawach i historiach medycznych, istnieje duże prawdopodobieństwo, że mają tę samą chorobę i zareagują na to samo. leczenie. Wydajne pobieranie spraw wymaga również zorganizowania pamięci sprawy, aby ułatwić takie pobieranie. Zazwyczaj sprawy są indeksowane według ich istotnych cech, co umożliwia wydajne wyszukiwanie spraw, które mają najwięcej cech wspólnych z bieżącym problemem. Identyfikacja najistotniejszych cech jest wysoce zależna od sytuacji.

2. Zmodyfikuj pobrany przypadek, aby miał zastosowanie do bieżącej sytuacji. Zazwyczaj przypadek zaleca sekwencję operacji, które przekształcają stan początkowy w stan docelowy. Rozumujący musi przekształcić przechowywane rozwiązanie w operacje odpowiednie dla bieżącego problemu. Przydatne mogą być metody analityczne, takie jak krzywa dopasowywania parametrów wspólnych dla zapisanych przypadków i nowych sytuacji; na przykład w celu określenia odpowiednich temperatur lub materiałów do spawania. Gdy relacje analityczne między przypadkami nie są dostępne, odpowiednie mogą być bardziej heurystyczne metody; na przykład w punktach pomocy do diagnostyki sprzętu.

3. Zastosuj przekształcony przypadek. Krok 2 modyfikuje przechowywaną sprawę, która po zastosowaniu może nie gwarantować zadowalającego rozwiązania problemu. Może to wymagać modyfikacji w przypadku rozwiązania, z dalszymi iteracjami tych pierwszych trzech kroków.

4. Zachowaj rozwiązanie wraz z zapisem sukcesu lub porażki do wykorzystania w przyszłości. Przechowywanie nowej sprawy wymaga aktualizacji struktury indeksu. Istnieją metody, które można wykorzystać do utrzymania wskaźników, w tym algorytmy grupowania (Fisher 1987) i inne techniki z uczenia maszynowego

Struktury danych dla rozumowania opartego na przypadkach mogą być bardzo zróżnicowane. W najprostszej sytuacji przypadki są rejestrowane jako krotki relacyjne, w których podzbiór argumentów rejestruje cechy do dopasowania, a inne argumenty wskazują kroki rozwiązania. Przypadki można również przedstawić jako bardziej złożone struktury, takie jak drzewa dowodowe. Dość powszechnym mechanizmem przechowywania spraw jest przedstawianie spraw jako zbioru obszernych reguł sytuacja - działanie. Fakty opisujące sytuację reguły są najistotniejszymi cechami zarejestrowanego przypadku, a operatory składające się na działanie reguły są przekształceniami, które należy zastosować w nowej sytuacji. Kiedy ten typ reprezentacji reguł jest używany, algorytmy takie jak RETE (Forgy 1982) mogą być używane do organizowania i optymalizacji wyszukiwania odpowiednich przypadków. Najtrudniejszą kwestią w rozwiązywaniu problemów opartych na przypadkach, niezależnie od wybranej struktury danych do reprezentacji przypadku, jest wybór najistotniejszych cech do indeksowania i wyszukiwania przypadków. Kolodner (1993) i inni aktywnie zaangażowani w rozwiązywanie problemów opartych na przypadkach, przyjęli jako podstawową zasadę, że sprawy powinny być organizowane według celów i potrzeb problemu. Oznacza to, że należy przeprowadzić dokładną analizę deskryptorów przypadków w kontekście tego, jak te przypadki zostaną wykorzystane w procesie rozwiązywania.

Na przykład załóżmy, że problem ze słabym sygnałem komunikacyjnym występuje na satelicie o godzinie 10:24:35 GMT. Dokonuje się analizy, a także określa, że system elektroenergetyczny jest niski. Niska moc może wystąpić, ponieważ panele słoneczne nie są odpowiednio zorientowane w kierunku słońca. Kontrolery naziemne dostosowują orientację satelity, moc poprawia się, a sygnał komunikacyjny jest znowu silny. Istnieje wiele istotnych cech, które można wykorzystać do zarejestrowania tego przypadku, z których najbardziej oczywistym jest słaby sygnał komunikacyjny lub niski poziom zasilania. Kolejna cecha opisująca to przypadek jest taki, że czas wystąpienia problemu był 10:24:35 GMT. Cele i potrzeby osoby rozwiązującej problem w tym przypadku sugerują, że istotnymi cechami są słaby sygnał komunikacyjny i / lub niskie zasilanie; czas, w którym to wszystko się dzieje, może być nieistotny, chyba że usterka nastąpi tuż po tym, jak słońce zniknie za horyzontem. Innym istotnym problemem do rozwiązania w CBR jest reprezentacja takich pojęć, jak słaby sygnał lub mała moc. Ponieważ dokładna sytuacja prawdopodobnie nigdy więcej nie zostanie dopasowana, np. Jakaś dokładna liczba rzeczywista opisująca siłę sygnału, rozumujący prawdopodobnie przedstawi wartości jako zakres liczb rzeczywistych wskazujących na przykład dobre, poziomy graniczne-dobre, słabe i ostrzegające o niebezpieczeństwie. Kolodner oferuje zestaw możliwych heurystyk preferencji, aby pomóc w organizacji przechowywania i wyszukiwania przypadków. Obejmują one:

1. Preferencja ukierunkowana na cel. Uporządkuj sprawy, przynajmniej częściowo, według opisów celów. Pobierz sprawy, które mają ten sam cel, co obecna sytuacja.

2. Preferencja istotnych cech. Preferuj przypadki, które pasują do najważniejszych cech lub tych, które pasują do największej liczby ważnych funkcji.

3. Określ preferencje. Przed rozważeniem bardziej ogólnych dopasowań szukaj możliwie najdokładniejszych dopasowań funkcji.

4. Preferencja częstotliwości. Sprawdź najpierw najczęściej pasujące przypadki.

5. Preferencje dotyczące aktualności. Preferuj przypadki używane ostatnio.

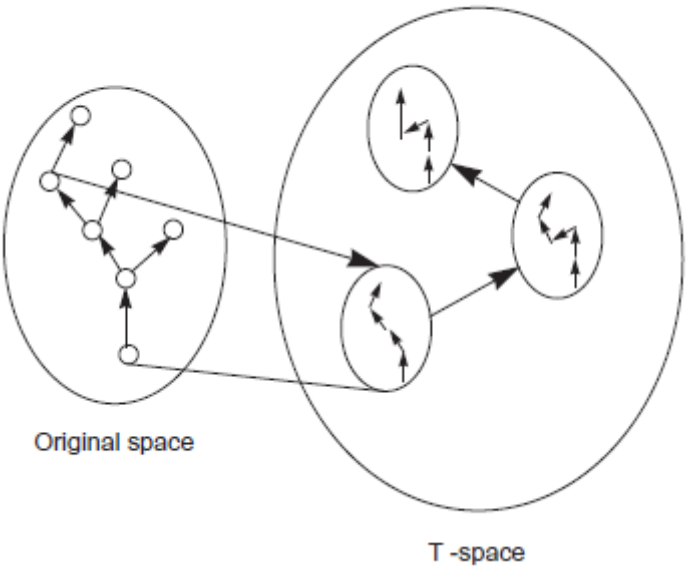
6. Łatwość preferencji adaptacyjnych. Korzystaj z pierwszych przypadków, które najłatwiej dostosować do aktualnej sytuacji.

Rozumowanie oparte na przypadkach ma wiele zalet przy projektowaniu systemów ekspertowych. Gdy inżynierowie wiedzy uzyskają właściwą reprezentację przypadku, dalsze zdobywanie wiedzy jest proste: wystarczy zebrać i przechowywać więcej przypadków. Często zbieranie przypadków można przeprowadzić na podstawie zapisów historycznych lub poprzez monitorowanie bieżących operacji, minimalizując zapotrzebowanie na czas eksperta.

Ponadto CBR stawia szereg ważnych pytań teoretycznych związanych z uczeniem się i rozumowaniem człowieka. Jedną z najbardziej subtelnych i krytycznych kwestii podniesionych przez CBR jest kwestia zdefiniowania podobieństwa. Chociaż pogląd, że podobieństwo jest funkcją liczby cech wspólnych dwóch przypadków, jest całkiem rozsądny, maskuje szereg głębokich subtelności. Na przykład większość obiektów i sytuacji ma dużą liczbę potencjalnych właściwości opisowych; Rozumowani na podstawie przypadków zazwyczaj wybierają przypadki na podstawie niewielkiego słownika wyszukiwania. Zazwyczaj osoby rozumujące oparte na przypadkach wymagają od inżyniera wiedzy zdefiniowania odpowiedniego słownictwa o wysoce istotnych cechach. Chociaż trwają prace nad umożliwieniem rozumu określenia odpowiednich cech na podstawie własnego doświadczenia określenie trafności pozostaje trudnym problemem. Kolejny ważny problem w rozumowaniu opartym na przypadkach dotyczy kompromisów między sklepami i obliczeniami. Gdy osoba rozumująca na podstawie przypadków zdobywa więcej przypadków, staje się bardziej inteligentna i lepiej potrafi rozwiązywać różnorodne docelowe problemy. Rzeczywiście, gdy dodamy przypadki do rozumowania, jego wydajność poprawi się - do pewnego momentu.

Problem polega na tym, że wraz ze wzrostem liczby przypadków rośnie również czas potrzebny do wyszukania i przetworzenia odpowiedniej sprawy. Spadek wydajności w przypadku dużych baz przypadków może również wynikać z nakładających się koncepcji, szumu i rozmieszczenia typów problemów. Jednym z rozwiązań tego problemu jest zapisanie tylko „najlepszych" lub „prototypów" przypadków, usuwając te, które są zbędne lub używane rzadko; tj. zapominając o przypadkach, które nie okazały się przydatne. Zobacz algorytm retencji Samuela do zapisywania pozycji szachownicy, aby uzyskać ważny wczesny przykład tego podejścia. Ogólnie jednak nie jest jasne, w jaki sposób możemy zautomatyzować takie decyzje; pozostaje to aktywnym obszarem badań. Automatyczne wyjaśnienie, dlaczego takie rozwiązanie jest zalecane, jest również trudne dla osoby rozważającej przypadki. Na pytanie, dlaczego wybrano rozwiązanie mające zaradzić bieżącej sytuacji, jedynym wyjaśnieniem, jakie system może dostarczyć, jest stwierdzenie, że ta konkretna poprawka działała w poprzednim okresie. Mogą również istnieć słabe wyjaśnienia oparte na podobieństwach opisów celów najwyższego poziomu między bieżącą sytuacją a zbiorem przechowywanych przypadków. Na przykładzie problemu komunikacji satelitarnej odpowiedni przypadek został wybrany na podstawie słabego sygnału komunikacyjnego. Ten sposób rozumowania nie ma głębszego wyjaśnienia niż ten, który działał wcześniej w podobnej sytuacji. Ale jak wspomniano wcześniej, może to być wystarczającym wyjaśnieniem wielu sytuacji. Wielu badaczy uważa jednak, że proste powtórzenie celów najwyższego poziomu i przypadek, który ma być zastosowany, nie dają wystarczającego wyjaśnienia, zwłaszcza gdy potrzebujemy wyjaśnienia, dlaczego niektóre poprawki nie działają. Przyjrzyjmy się ponownie sytuacji satelitarnej. Załóżmy, że reorientacja panelu słonecznego działa, ale trzy godziny później sygnał ponownie staje się słaby. Korzystając zarówno z heurystyki częstotliwości, jak i czasu trwania, ponownie zmieniamy orientację panelu słonecznego. Za trzy godziny powraca słaby sygnał. I znowu

trzy godziny później; zawsze stosujemy tę samą poprawkę. Ten przykład jest oparty na rzeczywistej sytuacji satelitarnej, w której stwierdzono, że istnieje bardziej złożony problem, a mianowicie przegrzanie żyroskopu i zniekształcenie odczytu, co zdezorientowało satelitę. System, który ostatecznie rozwiązał problem, wykorzystał rozumowanie oparte na modelu do symulacji zachowania satelity w celu określenia pierwotnych przyczyn słabego sygnału komunikacyjnego. Rozumowanie oparte na przypadku wiąże się z problemem uczenia się przez analogię. Aby ponownie wykorzystać przeszłe doświadczenia, musimy zarówno rozpoznać najistotniejsze cechy przeszłości, jak i stworzyć mapę tego, jak można je wykorzystać w obecnej sytuacji. Analogia transformacyjna jest przykładem opartego na przypadkach podejścia do rozwiązywania problemów. Rozwiązuje nowe problemy, modyfikując istniejące rozwiązania, dopóki nie będą mogły zostać zastosowane w nowej instancji. Operatorzy, którzy modyfikują kompletne rozwiązania problemów, definiują wyższy poziom abstrakcji lub przestrzeń T, w której stany są rozwiązaniami problemu, a operatorzy przekształcają te rozwiązania, jak na rysunku.



Celem jest przekształcenie rozwiązania źródłowego w możliwe rozwiązanie docelowego problemu. Operatorzy modyfikują rozwiązania w taki sposób, jak wstawianie lub usuwanie kroków w ścieżce rozwiązania, zmiana kolejności kroków w rozwiązaniu, łączenie nowych rozwiązań z częścią starego rozwiązania lub zmiana powiązań parametrów w obecnym rozwiązaniu. Analogia transformacyjna charakteryzuje podejście stosowane przy rozwiązywaniu problemów opartych na przypadkach. Późniejsze prace dopracowały to podejście, uwzględniając takie kwestie, jak reprezentacja przypadków, strategie organizowania pamięci wcześniejszych spraw, wyszukiwanie odpowiednich wcześniejszych spraw i wykorzystanie przypadków w rozwiązywaniu nowych problemów. Więcej informacji na temat rozumowania opartego na przypadku można znaleźć w Hammond (1989) i Kolodner. Rozumowanie poprzez analogie jest omówione dalej w kontekście uczenia maszynowego opartego na symbolach

Projektowanie hybrydowe: mocne / słabe strony mocnych systemów metod

Sukcesy w budowaniu systemów eksperckich, które rozwiązują trudne, praktyczne problemy, pokazały prawdę o centralnej idei stojącej za systemami opartymi na wiedzy: siła rozumującego tkwi w jego domenie, a nie w wyrafinowaniu jego metod rozumowania. Ta obserwacja podnosi jednak jedną z centralnych kwestii sztucznej inteligencji: reprezentację wiedzy. Na poziomie praktycznym każdy

inżynier wiedzy musi dokonywać wyborów dotyczących sposobu reprezentowania wiedzy dziedzinowej w sposób najbardziej odpowiedni dla danej dziedziny. Reprezentacja wiedzy podnosi również szereg teoretycznie ważnych, intelektualnie trudnych zagadnień, takich jak postępowanie z brakującymi i niepewnymi informacjami, pomiar ekspresji reprezentacji, związek między językiem reprezentacji a takimi kwestiami, jak uczenie się, zdobywanie wiedzy i skuteczność rozumowania.

Rozważaliśmy kilka podstawowych podejść do reprezentacji wiedzy: oparte na regułach systemy eksperckie, wnioskujący na podstawie modeli i wnioskujący na podstawie przypadków. Aby pomóc w praktycznej inżynierii wiedzy, podsumowujemy mocne i słabe strony każdego opartego na wiedzy podejścia do rozwiązywania problemów.

Rozumowanie oparte na regułach

Zalety podejścia opartego na regułach obejmują:

1. Umiejętność korzystania w bardzo bezpośredni sposób z empirycznej wiedzy zdobytej od ludzkich ekspertów. Jest to szczególnie ważne w domenach, które w dużym stopniu opierają się na heurystyce do zarządzania złożonością i / lub brakującymi informacjami.

2. Reguły odwzorowują przeszukiwanie przestrzeni stanów. Funkcje wyjaśniające obsługują debugowanie.

3. Oddzielenie wiedzy od sterowania upraszcza tworzenie systemów ekspertowych, umożliwiając iteracyjny proces rozwoju, w ramach którego inżynier wiedzy pozyskuje, wdraża i testuje poszczególne reguły.

4. Dobra wydajność jest możliwa w niektórych domenach. Z powodu dużej ilości wiedzy wymaganej do inteligentnego rozwiązywania problemów systemy ekspertowe są ograniczone do wąskich dziedzin. Jest jednak wiele dziedzin, w których zaprojektowanie odpowiedniego systemu okazało się niezwykle przydatne.

5. Dobre możliwości wyjaśniania. Chociaż podstawowe ramy oparte na regułach obsługują elastyczne, specyficzne dla problemów wyjaśnienia, należy wspomnieć, że ostateczna jakość tych wyjaśnień zależy od struktury i treści reguł. Możliwości wyjaśniania różnią się znacznie między systemami opartymi na danych i celach.

Wady rozumowania opartego na regułach obejmują:

1. Często reguły uzyskane od ludzkich ekspertów mają wysoce heurystyczny charakter i nie obejmują funkcjonalnej lub opartej na modelach wiedzy o tej dziedzinie.

2. Reguły heurystyczne są zwykle „kruche” i mogą mieć trudności z obsługą brakujących informacji lub nieoczekiwanych wartości danych.

3. Innym aspektem kruchości reguł jest tendencja do szybkiej degradacji w pobliżu „krawędzi” wiedzy dziedzinowej. W przeciwieństwie do ludzi systemy oparte na regułach zwykle nie są w stanie oprzeć się na pierwszych zasadach rozumowania, gdy stają w obliczu nowych problemów.

4. Wyjaśnienia funkcjonują tylko na poziomie opisowym, pomijając wyjaśnienia teoretyczne. Wynika to z faktu, że reguły heurystyczne zyskują znaczną część swojej mocy poprzez bezpośrednie kojarzenie symptomów problemu z rozwiązaniami, bez konieczności (lub wspierania) głębszego rozumowania.

5. Wiedza jest bardzo zależna od zadań. Sformalizowana wiedza dziedzinowa ma zwykle bardzo specyficzne zastosowanie. Obecnie języki reprezentacji wiedzy nie zbliżają się do elastyczności ludzkiego rozumowania

Uzasadnienie w oparciu o przypadki

Zalety rozumowania opartego na przypadkach obejmują:

1. Umiejętność bezpośredniego kodowania wiedzy historycznej. W wielu dziedzinach przypadki można uzyskać z istniejących historii przypadków, dzienników napraw lub innych źródeł, co eliminuje potrzebę intensywnego nabywania wiedzy z ekspertem.

2. Pozwala na skróty w rozumowaniu. Jeśli uda się znaleźć odpowiedni przypadek, nowe problemy często można rozwiązać w znacznie krótszym czasie niż wygenerowanie rozwiązania na podstawie reguł lub modeli i wyszukiwanie.

3. Pozwala systemowi uniknąć błędów z przeszłości i wykorzystać wcześniejsze sukcesy. CBR zapewnia model uczenia się, który jest zarówno teoretycznie interesujący, jak i wystarczająco praktyczny, aby można go było zastosować do złożonych problemów.

4. Obszerna analiza wiedzy dziedzinowej nie jest wymagana. W przeciwieństwie do systemu opartego na regułach, w którym inżynier wiedzy musi przewidywać interakcje reguł, CBR pozwala na prosty model addytywny do pozyskiwania wiedzy. Wymaga to odpowiedniej reprezentacji przypadków, użytecznego indeksu wyszukiwania i strategii dostosowania przypadku.

5. Odpowiednie strategie indeksowania zwiększają możliwości wglądu i rozwiązywania problemów. Umiejętność rozróżnienia różnic w docelowych problemach i wybrania odpowiedniego przypadku jest ważnym źródłem mocy rozumującego na podstawie przypadków; często algorytmy indeksowania mogą zapewniać tę funkcję automatycznie.

Wady rozumowania opartego na przypadkach obejmują:

1. Przypadki często nie obejmują głębszej znajomości domeny. Ułatwia to wyjaśnienie utrudnień, aw wielu sytuacjach stwarza możliwość niewłaściwego zastosowania spraw, co prowadzi do złej jakości lub złej porady.

2. W przypadku dużej bazy przypadków mogą wystąpić problemy związane z kompromisami przechowywania / obliczania.

3. Trudno jest określić dobre kryteria indeksowania i dopasowywania przypadków. Obecnie słowniki wyszukiwania i algorytmy dopasowywania podobieństw muszą być starannie opracowywane ręcznie; może to zrównoważyć wiele zalet CBR w zdobywaniu wiedzy.

Rozumowanie oparte na modelu

Zalety rozumowania opartego na modelach obejmują:

1. Umiejętność wykorzystania funkcjonalnej / strukturalnej wiedzy z dziedziny w rozwiązywaniu problemów. Zwiększa to zdolność rozumującego do radzenia sobie z różnymi problemami, w tym takimi, których projektanci systemu mogli nie przewidzieć.

2. Rozumowania oparte na modelach są zazwyczaj bardzo solidne. Z tych samych powodów, dla których ludzie często wycofują się do podstawowych zasad w obliczu nowego problemu, osoby rozumujące oparte na modelach mają tendencję do dokładnego i elastycznego rozwiązywania problemów.

3. Część wiedzy można przenosić między zadaniami. Rozumowania oparte na modelach są często budowane przy użyciu wiedzy naukowej i teoretycznej. Ponieważ nauka dąży do ogólnie stosowanych teorii, ta ogólność często rozciąga się na osoby rozumujące oparte na modelach.

4. Często osoby rozumujące oparte na modelach mogą dostarczyć wyjaśnień przyczynowych. Mogą one przekazać głębsze zrozumienie błędu użytkownikom, a także odgrywać ważną rolę w samouczku.

Wady rozumowania opartego na modelu obejmują:

1. Brak empirycznej (opisowej) wiedzy w dziedzinie. Metody heurystyczne stosowane w podejściach opartych na regułach odzwierciedlają cenną klasę wiedzy.

2. Wymaga jawnego modelu domeny. Wiele dziedzin, takich jak diagnostyka uszkodzeń obwodów elektronicznych, ma solidne podstawy naukowe, które wspierają podejścia oparte na modelach. Jednak w wielu dziedzinach, takich jak niektóre specjalizacje medyczne, większość problemów projektowych lub wiele zastosowań finansowych, brakuje dobrze zdefiniowanej teorii naukowej. W takich przypadkach nie można stosować podejść opartych na modelach.

3. Wysoka złożoność. Rozumowanie oparte na modelu na ogół działa na poziomie szczegółowości, który prowadzi do znacznej złożoności; jest to w końcu jeden z głównych powodów, dla których eksperci od ludzi rozwinęli heurystykę na pierwszym miejscu.

4. Wyjątkowe sytuacje. Nietypowe okoliczności, na przykład błędy mostkowania lub interakcja wielu awarii w komponentach elektronicznych, mogą zmienić funkcjonalność systemu w sposób trudny do przewidzenia przy użyciu modelu a priori.

Projekt hybrydowy

Ważnym obszarem badań i zastosowań jest połączenie różnych modeli rozumowania. W przypadku architektury hybrydowej zintegrowane są dwa lub więcej paradygmatów, aby uzyskać efekt współpracy, w którym mocne strony jednego systemu mogą kompensować słabość innego. W połączeniu możemy zająć się wadami wskazanymi w poprzedniej dyskusji. Na przykład połączenie systemów opartych na regułach i przypadkach może:

1. Zaproponuj naturalne, pierwsze sprawdzenie znanych przypadków, zanim podejmie się rozumowanie oparte na regułach i związane z tym koszty wyszukiwania.

2. Przedstawienie przykładów i wyjątków od rozwiązań poprzez przechowywanie ich w bazie przypadków.

3. Zapisuj wyniki wyszukiwania jako przypadki do wykorzystania w przyszłości. Zapisując odpowiednie przypadki, osoba rozumująca może uniknąć powielania kosztownych poszukiwań.

Połączenie systemów opartych na regułach i modelach może:

1. Uzupełnij wyjaśnienia o wiedzę funkcjonalną. Może to być szczególnie przydatne w samouczkach.

2. Poprawić niezawodność, gdy zasady zawodzą. Jeśli nie ma reguł heurystycznych, które mają zastosowanie do danego wystąpienia problemu, osoba wnioskująca może uciec się do rozumowania od pierwszych zasad.

3. Dodaj wyszukiwanie heurystyczne do wyszukiwania opartego na modelach. Może to pomóc w zarządzaniu złożonością rozumowania opartego na modelach i pozwolić rozumującemu na inteligentny wybór między możliwymi alternatywami.

Połączenie systemów opartych na modelach i przypadkach może:

1. Przedstaw bardziej dojrzałe wyjaśnienia sytuacji odnotowanych w sprawach.

2. Zaproponuj naturalne, pierwsze sprawdzenie przechowywanych przypadków przed rozpoczęciem bardziej rozległych poszukiwań wymaganych przez rozumowanie oparte na modelu.

3. Zapewnij zapis przykładów i wyjątków w bazie przypadków, która może być wykorzystana do prowadzenia wnioskowania na podstawie modelu.

4. Zapisz wyniki wnioskowania opartego na modelu do wykorzystania w przyszłości.

Metody hybrydowe zasługują na uwagę zarówno badaczy, jak i twórców aplikacji. Budowa takich systemów nie jest jednak sprawą prostą, wymagającą rozwiązania takich problemów, jak ustalenie, jaką metodę rozumowania zastosować w danej sytuacji,

kiedy zmienić metody rozumowania, rozwiązać różnice między metodami rozumowania i zaprojektować reprezentacje, które pozwalają na dzielenie się wiedzą. Następnie rozważymy planowanie lub organizację procedur w potencjalne rozwiązania.

(8.4) Planowanie

Zadaniem planisty jest znalezienie sekwencji działań, które pozwolą rozwiązującemu problem, na przykład systemowi sterowania, wykonać określone zadanie. Tradycyjne planowanie wymaga bardzo dużej wiedzy, ponieważ tworzenie planów wymaga uporządkowania fragmentów wiedzy i częściowe plany w procedurę rozwiązania. Planowanie odgrywa rolę w systemach eksperckich w rozumowaniu zdarzeń zachodzących w czasie. Planowanie ma wiele zastosowań w produkcji, np. Sterowanie procesami. Jest to również ważne przy projektowaniu sterownika robota.

Wprowadzenie do planowania: robotyka

Na początek rozważymy przykłady z tradycyjnej robotyki. (Wiele współczesnych robotów wykorzystuje sterowanie reaktywne, a nie planowanie przemyślane) Kroki tradycyjnego planu robota składają się z atomowych działań robota. W naszym przykładzie nie opisujemy działań atomowych na poziomie mikro: „obróć szósty silnik krokowy o jeden obrót”. Zamiast tego określamy działania na wyższym poziomie, na przykład poprzez ich wpływ na świat. Na przykład robot na świecie bloków może obejmować takie czynności, jak „podnieś przedmiot a” lub „idź do lokalizacji x”. Mikrokontrola jest wbudowana w te działania wyższego poziomu. Zatem sekwencja działań „przejdź do bloku a z pokoju b” może wyglądać następująco:

1. odłóż to, co jest teraz trzymane

2. przejdź do pokoju b

3. przejdź do bloku a

4. podnieś blok a

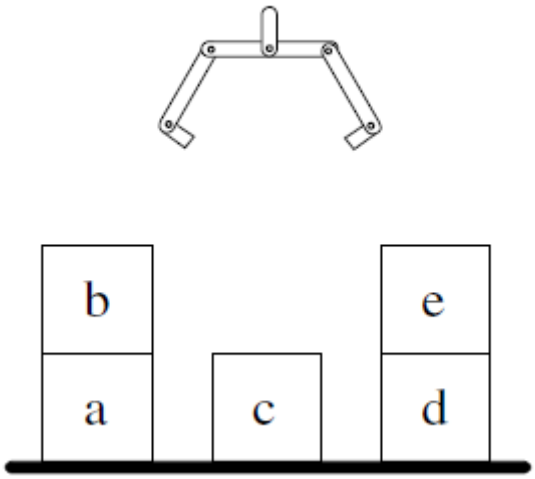
5. opuścić pokój b

6. powrócić do pierwotnej lokalizacji

Plany tworzy się przeszukując przestrzeń możliwych działań, aż do odkrycia sekwencji niezbędnej do wykonania zadania. Ta przestrzeń przedstawia stany świata, które ulegają zmianie w wyniku zastosowania każdego z działań. Przeszukiwanie kończy się z chwilą utworzenia stanu celu (odpowiedniego opisu świata). W związku z tym wiele zagadnień związanych z wyszukiwaniem

heurystycznym, w tym znajdowanie algorytmów A *, jest również odpowiednich w planowaniu. Czynność planowania nie zależy jednak od istnienia rzeczywistego robota do wykonania planów. We wczesnych latach planowania komputerowego (lata 60. XX wieku) całe plany formułowano przed wykonaniem przez robota pierwszej czynności. Tak więc plany zostały wymyślone bez obecności robota! Niedawno, wraz z wdrożeniem wyrafinowanych czujników i urządzeń reaktywnych, badania skupiły się na bardziej zintegrowanym planowaniu / sekwencjonowaniu działań. Tradycyjne planowanie opiera się na technikach wyszukiwania i rodzi szereg interesujących problemów. Po pierwsze, opis stanów świata może być znacznie bardziej złożony niż w poprzednich przykładach wyszukiwania. Rozważ liczbę predykatów, które mogą być konieczne do opisania pomieszczeń, korytarzy i obiektów w środowisku robota. Nie tylko musimy reprezentować świat robotów; musimy także przedstawić wpływ działań atomowych na ten świat. Pełny opis każdego stanu przestrzeni problemowej może być obszerny.

Kolejną różnicą w planowaniu jest potrzeba scharakteryzowania tego, co nie jest zmieniane przez określone działanie. Podniesienie przedmiotu zmienia (a) położenie przedmiotu i (b) fakt, że ręka robota teraz chwyta przedmiot. Nie zmienia to (a) lokalizacji drzwi i pomieszczeń lub (b) lokalizacji innych obiektów. Określenie tego, co jest prawdą w jednym stanie świata i dokładnie tego, co zmienia się w wyniku wykonania jakiejś czynności na świecie, stało się znane jako problem ramy (McCarthy 1980, McCarthy i Hayes 1969). Wraz ze wzrostem złożoności przestrzeni problemowej, kwestia śledzenia zmian zachodzących z każdym działaniem oraz cech opisu stanu, które pozostają niezmienione, staje się coraz ważniejsza. Przedstawiamy dwa sposoby radzenia sobie z problemem ramy, ale jak zobaczymy, żadne z nich nie jest w pełni zadowalające. Inne ważne kwestie obejmują generowanie planów, zapisywanie i uogólnianie dobrych planów, odzyskiwanie po nieoczekiwanych niepowodzeniach planów (część świata może nie być taka, jak oczekiwano, być może przez przypadkowe przeniesienie z przewidywanej lokalizacji) oraz utrzymanie spójności między światem a wewnętrzną model świata. W przykładach w tej sekcji ograniczamy świat naszego robota do zestawu bloków na blat i działania robota do ramienia, które może układać, rozkładać i w inny sposób przesuwać bloki wokół stołu. Na rysunku 8.18 mamy pięć bloków, oznaczonych a, b, c, d, e, leżących na blacie stołu.



Bloki to wszystkie kostki tego samego rozmiaru i stosy bloków, jak na rysunku , mają bloki bezpośrednio jeden na drugim. Ramię robota ma chwytak, który może chwycić dowolny przezroczysty blok (taki, na którym nie ma żadnego bloku) i przenieść go w dowolne miejsce na blacie lub umieścić na dowolnym

innym bloku (bez żadnego bloku na nim). Ramię robota może wykonywać następujące zadania (U, V, W, X, Y i Z to zmienne):

goto (X, Y, Z): Idź do lokalizacji opisanej współrzędnymi X, Y i Z. Ta lokalizacja może być niejawna w pobraniu polecenia (W), gdzie blok W ma położenie X, Y, Z.

pickup (W): Podnieś blok W z jego aktualnej lokalizacji. Zakłada się, że blok W jest czysty na górze, chwytak jest pusty w tym czasie, a komputer zna lokalizację bloku W.

putdown (W): Umieść klocek W w jakimś miejscu na stole i zapisz nowe położenie dla W. W musi być trzymany przez chwytak w tym czasie.

stos (U, V): Umieść klocek U na wierzchu klocka V. Chwytak musi trzymać U, a góra V musi być wolna od innych klocków.

unstack (U, V): Usuń blok U z góry V. U musi być wolna od innych bloków, V musi mieć blok U na wierzchu, a ręka musi być pusta przed wykonaniem tego polecenia.

Stan świata opisuje zestaw predykatów i relacji predykatów:

lokalizacja (W, X, Y, Z): Blok W znajduje się na współrzędnych X, Y, Z.

on (X, Y): Blok X znajduje się bezpośrednio nad blokiem Y.

clear (X): Blok X nie ma nic na wierzchu.

chwytanie (X): Ramię robota trzyma blok X.

chwytanie (): chwytak robota jest pusty.

ontable (W): Blok W jest na stole.

ontable (W) to skrócona forma dla lokalizacji predykatu (W, X, Y, Z), gdzie Z to poziom tabeli. Podobnie, on (X, Y) wskazuje, że blok X znajduje się tak, że jego dolna część pokrywa się z górną częścią bloku Y. Możemy znacznie uprościć opisy świata poprzez zapisanie przez komputer aktualnej lokalizacji (X, Y, Z) każdego bloku i śledź jego ruchy do nowych lokalizacji. Przy takim założeniu lokalizacji polecenie goto staje się niepotrzebne; polecenie takie jak pickup (X) lub stos (X) zawiera niejawnie położenie X. Świat bloków z rysunku 8.18 może być teraz reprezentowany przez następujący zestaw predykatów. Nazywamy ten zbiór predykatów STANEM 1 dla naszego dalszego przykładu.

Ponieważ wszystkie predykaty opisujące stan świata z rysunku 8.18 są jednocześnie prawdziwe, pełny opis stanu jest koniunkcją (∧) wszystkich tych predykatów.

STAN 1

w tabeli (a). na (b, a). jasne (b).

dostępny (c). na (e, d). jasne (c).

ontable (d). chwytanie (). jasne (e).

Następnie tworzy się szereg relacji prawdy (w sensie deklaratywnym) lub reguł wykonania (w sensie proceduralnym) dla clear (X), ontable (X) i gripping():

1. $(\forall X)\ (\text{clear}(X) \leftarrow \neg\ (\exists Y)\ (\text{on}(Y,X)))$
2. $(\forall Y)\ (\forall X)\ \neg\ (\text{on}(Y,X) \leftarrow \text{ontable}(Y))$
3. $(\forall Y)\ \text{gripping}(\ ) \leftrightarrow \neg\ (\text{gripping}(Y))$

Pierwsze stwierdzenie mówi, że jeśli blok X jest czysty, nie ma żadnego bloku Y takiego, że Y znajduje się na szczycie X. Interpretacja proceduralna mówi, że „aby wyczyścić blok X, idź i usuń każdy blok Y, który może znajdować się na szczycie X . ” Teraz projektujemy reguły, które mają działać na stanach i tworzyć nowe stany. Robiąc to, ponownie przypisujemy semantykę proceduralną reprezentacji podobnej do logiki predykatu. Operatorzy (odbiór, odłożenie, ułożenie, rozpakowanie) to:

4. $(\forall X)\ (\text{pickup}(X) \rightarrow (\text{gripping}(X) \leftarrow (\text{gripping}(\ ) \wedge \text{clear}(X) \wedge \text{ontable}(X))))$.
5. $(\forall X)\ (\text{putdown}(X) \rightarrow ((\text{gripping}(\ ) \wedge \text{ontable}(X) \wedge \text{clear}(X)) \leftarrow \text{gripping}(X)))$.
6. $(\forall X)\ (\forall Y)\ (\text{stack}(X,Y) \rightarrow ((\text{on}(X,Y) \wedge \text{gripping}(\ ) \wedge \text{clear}(X)) \leftarrow (\text{clear}(Y) \wedge \text{gripping}(X))))$.
7. $(\forall X)(\forall Y)\ (\text{unstack}(X,Y) \rightarrow ((\text{clear}(Y) \wedge \text{gripping}(X)) \leftarrow (\text{on}(X,Y) \wedge \text{clear}(X) \wedge \text{gripping}(\ )))$.

Rozważ czwartą zasadę: dla wszystkich bloków X podniesienie (X) oznacza chwycenie (X), jeśli ręka jest pusta, a X jest czysty. Zwróć uwagę na formę tych reguł: A → (B ← C). To mówi, że operator A tworzy nowy predykat (y) B, gdy warunek (a) C jest (są) prawdziwe. Używamy tych reguł do generowania nowych stanów w przestrzeni. Oznacza to, że jeśli predykaty C są prawdziwe w stanie, to B jest prawdziwe w stanie potomnym. Zatem operator A może być użyty do stworzenia nowego stanu opisanego przez predykaty B, gdy predykaty C są prawdziwe. Alternatywne podejścia do tworzenia tych operatorów obejmują STRIPS oraz Rich and Knight’s (1991). Najpierw musimy rozwiązać problem ramki, zanim będziemy mogli użyć tych relacji reguł do wygenerowania nowych stanów świata bloków. Aksjomaty ram to reguły mówiące, które predykaty opisujące stan nie są zmieniane przez zastosowanie reguł, a zatem są przenoszone w stanie nienaruszonym, aby pomóc opisać nowy stan świata. Na przykład, jeśli zastosujemy blok pobierania operatora b na rysunku 8.18, wówczas wszystkie predykaty związane z pozostałymi blokami pozostaną prawdziwe w stanie potomnym. Dla naszego świata bloków możemy określić kilka takich reguł ramek:

8. $(\forall X)\ (\forall Y)\ (\forall Z)\ (\text{unstack}(Y,Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X)))$.
9. $(\forall X)\ (\forall Y)\ (\forall Z)\ (\text{stack}(Y,Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X)))$.

Te dwie reguły mówią, że operatory stack i unstack nie mają wpływu na ontable. Dzieje się tak nawet wtedy, gdy X i Z są identyczne; jeśli Y = Z 6 lub 7 powyżej nie będzie prawdziwe. Inne aksjomaty ramek mówią, że operatory on i clear mają wpływ na operatory stack i unstack tylko wtedy, gdy ta konkretna relacja jest usunięta ze stosu lub gdy jest ustawiona w stosie czysta relacja. Zatem w naszym przykładzie na (b, a) nie ma wpływu unstack (c, d). Podobnie aksjomaty ram mówią, że na relacje jasne (X) nie ma wpływu chwytanie (Y), nawet jeśli X = Y lub chwytanie () jest prawdą. Więcej aksjomatów mówi, że chwytanie nie wpływa na relacje (X, Y), ale wpływa tylko na relację ontable (X), w której chwytany jest X. W naszym przykładzie należy określić szereg dalszych aksjomatów ramek. Razem te operatory i aksjomaty ramek definiują przestrzeń stanów, co ilustruje operator unstack. unstack (X, Y) wymaga jednoczesnego spełnienia trzech warunków, mianowicie na (X, Y), chwytanie () i jasne (X). Gdy te warunki są spełnione, nowe predykaty chwytanie (X) i usuwanie (Y) są tworzone przez zastosowanie

operatora unstack. Szereg innych predykatów, również prawdziwe dla STANU 1, pozostanie prawdziwe w STANIE 2. Te stany są zachowane dla STANU 2 przez aksjomaty ramki. Teraz tworzymy dziewięć predykatów opisujących STAN 2, stosując operator unstack i aksjomaty ramki do dziewięciu predykatów STANU 1, gdzie wynikiem netto jest zdejmowanie bloku e ze stosu:

STATE 2

ontable(a). on(b,a). clear(b).

ontable(c). clear(c). clear(d).

ontable(d). gripping(e). clear(e).

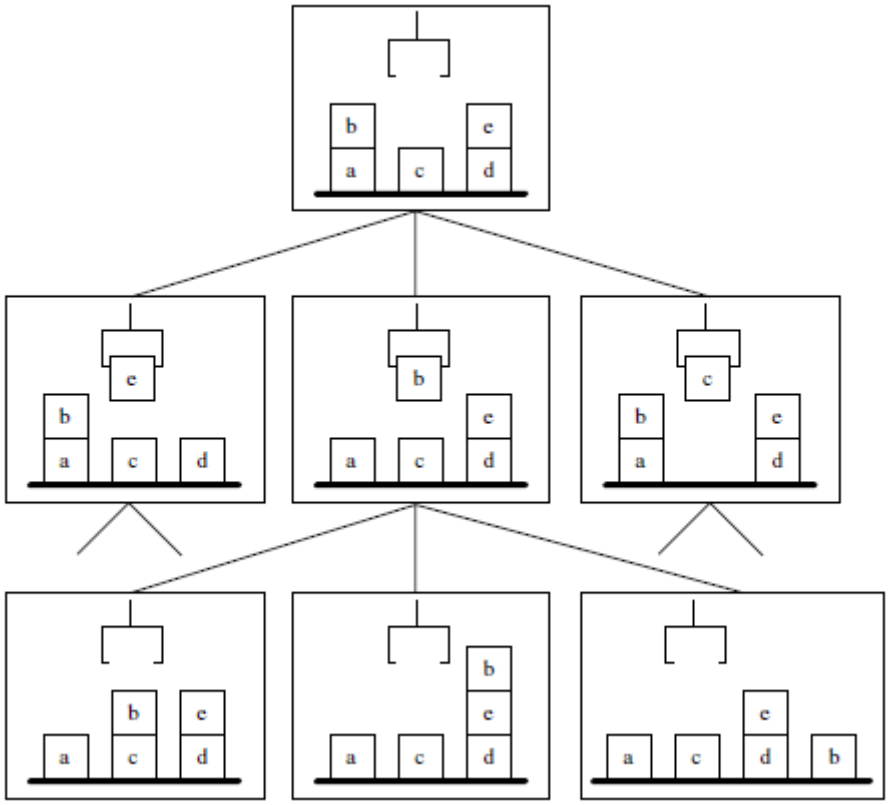
Podsumowując:

1. Planowanie można postrzegać jako przeszukiwanie przestrzeni stanów.

2. Nowe stany są tworzone przez ogólne operatory, takie jak stack i unstack oraz reguły ramek.

3. Techniki przeszukiwania grafów można zastosować do znalezienia ścieżki od stanu początkowego do stanu docelowego. Działania na tej ścieżce stanowią plan.

Rysunek przedstawia przykład przestrzeni stanów przeszukiwanej za pomocą operatorów opisanych powyżej.



Jeśli do tego procesu rozwiązywania problemów dodany zostanie opis celu, wówczas plan można postrzegać jako zbiór operatorów tworzących ścieżkę prowadzącą od obecnego stanu ten wykres do celu. Ta charakterystyka problemu planowania definiuje jego teoretyczne korzenie w wyszukiwaniu w przestrzeni stanów oraz reprezentacji rachunku predykatów i wnioskowaniu. Należy jednak zauważyć, jak skomplikowany może być ten sposób rozwiązania. W szczególności użycie reguł ramek do obliczenia

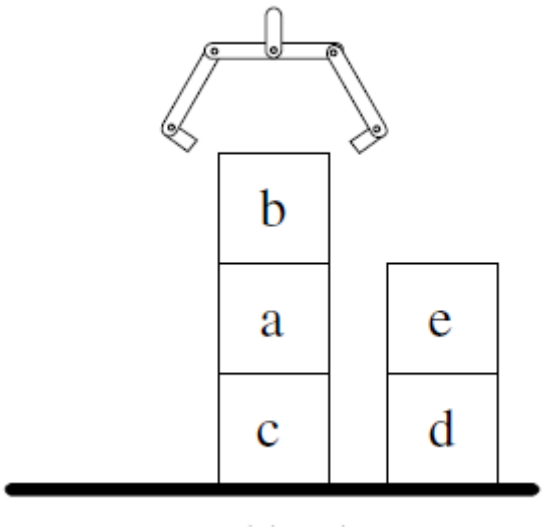
tego, co pozostaje niezmienione między stanami, może zwiększyć wykładniczo do wyszukiwania, jak widać na podstawie złożoności naszego przykładu bardzo prostych bloków. W rzeczywistości, kiedy wprowadzany jest nowy deskryptor predykatu dla koloru, kształtu lub rozmiaru, należy zdefiniować nowe reguły ramek, aby powiązać go ze wszystkimi innymi odpowiednimi działaniami! W tej dyskusji założono również, że podproblemy składające się na zadanie są niezależne, a zatem można je rozwiązać w dowolnej kolejności. Bardzo rzadko ma to miejsce w interesujących i / lub złożonych dziedzinach problemowych, gdzie warunki wstępne i działania wymagane do osiągnięcia jednego celu podrzędnego mogą często kolidować z warunkami wstępnymi i działaniami wymaganymi do osiągnięcia innego. Następnie zilustrujemy te problemy i omówimy podejście do planowania, które bardzo pomaga w radzeniu sobie z tą złożonością.

Używanie makr planowania: STRIPS

STRIPS, opracowany w obecnym SRI International, oznacza STanford Research Institute Planning System (Fikes i Nilsson 1971, Fikes i in. 1972). Ten kontroler był używany do sterowania robotem SHAKEY z początku lat 70. STRIPS rozwiązał problem efektywnie reprezentowania i wdrażania działań planisty. Zajęło się problemem sprzecznych celów cząstkowych i dostarczyło wczesnego modelu uczenia się; udane plany zostały zapisane i uogólnione jako makrooperatory, których można by użyć w podobnych przyszłych sytuacjach. W pozostałej części tej sekcji przedstawiamy wersję planowania w stylu STRIPS i tabel trójkątnych, strukturę danych używaną do organizowania i przechowywania operacji makr. Korzystając z naszego przykładu bloków, czterech operatorów podnoszących, odkładających, układających i odkładających w stosy są reprezentowane jako trzy opisy. Pierwszym elementem trójki jest zestaw warunków wstępnych (P), czyli warunków, które świat musi spełnić, aby można było zastosować operator. Drugim elementem trójki jest lista add (A), czyli dodatki do opisu stanu, które są wynikiem zastosowania operatora. Wreszcie istnieje lista usuwania (D), czyli elementy, które są usuwane z opisu stanu w celu utworzenia nowego stanu po zastosowaniu operatora. Listy te mają na celu wyeliminowanie potrzeby oddzielnych aksjomatów ramek. Możemy przedstawić czterech operatorów w ten sposób:

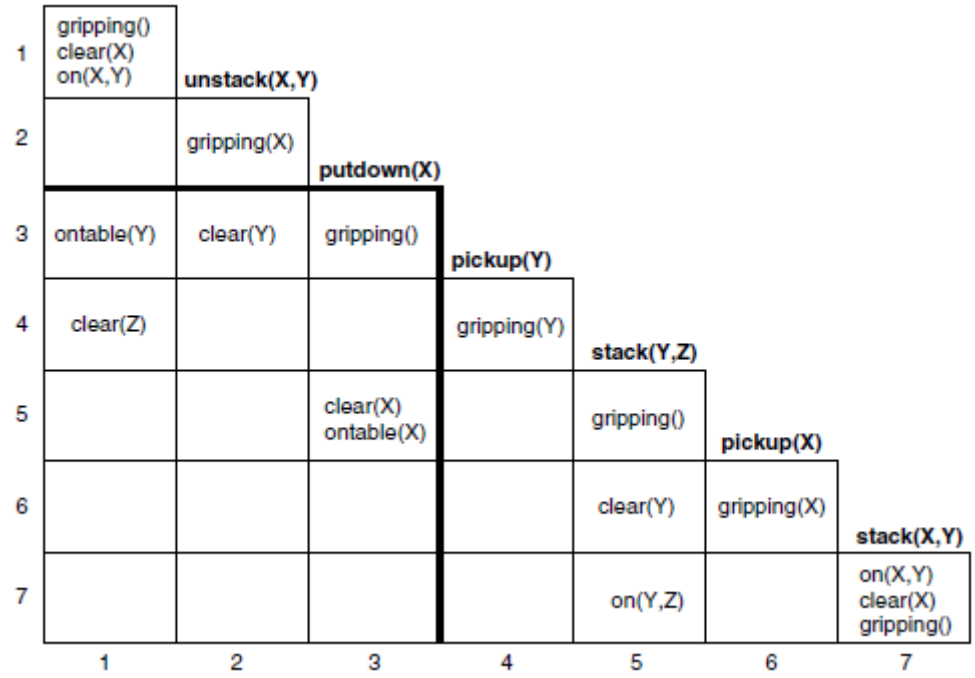
pickup(X)	P: gripping( ) ∧ clear(X) ∧ ontable(X) A: gripping(X) D: ontable(X) ∧ gripping( )
putdown(X)	P: gripping(X) A: ontable(X) ∧ gripping( ) ∧ clear(X) D: gripping(X)
stack(X,Y)	P: clear(Y) ∧ gripping(X) A: on(X,Y) ∧ gripping( ) ∧ clear(X) D: clear(Y) ∧ gripping(X)
unstack(X,Y)	P: clear(X) ∧ gripping( ) ∧ on(X,Y) A: gripping(X) ∧ clear(Y) D: gripping( ) ∧ on(X,Y)

Ważną rzeczą dotyczącą list dodawania i usuwania jest to, że określają one wszystko, co jest konieczne, aby spełnić aksjomaty ramek! W podejściu do listy dodawania i usuwania istnieje pewna nadmiarowość. Na przykład przy rozpakowywaniu dodawanie chwytania (X) może oznaczać usunięcie chwytania (). Ale zaletą tej nadmiarowości jest to, że każdy deskryptor stanu, który nie jest wymieniony w dodawaniu lub usuwaniu, pozostaje taki sam w nowym opisie stanu. Powiązaną słabością podejścia do listy warunek wstępny-dodaj-usuń jest to, że nie używamy już procesu dowodzenia twierdzeń do tworzenia (przez wnioskowanie) nowych stanów. Nie jest to jednak poważny problem, ponieważ dowody równoważności tych dwóch podejść mogą zagwarantować poprawność metody warunek-dodaj-usuń. Podejście z listą warunek wstępny-adddelete może być użyte do uzyskania takich samych wyników, jakie uzyskaliśmy w wyniku wnioskowania zasady i aksjomaty ram w naszym wcześniejszym przykładzie. Przeszukiwanie w przestrzeni stanów, jak na rysunku powyżej, byłoby identyczne dla obu podejść. Szereg innych problemów nieodłącznie związanych z planowaniem nie zostało rozwiązanych przez żadne z dwóch przedstawionych dotychczas podejść. Podczas rozwiązywania celu często dzielimy go na podproblemy, na przykład unstack (e, d) i unstack (b, a). Próba samodzielnego rozwiązania tych celów cząstkowych może powodować problemy, jeśli działania potrzebne do osiągnięcia jednego celu faktycznie cofają drugi. Niezgodne cele cząstkowe mogą wynikać z błędnego założenia o liniowości (niezależności) celów cząstkowych. Nieliniowość obszaru planu / działania może niepotrzebnie utrudniać lub nawet uniemożliwiać wyszukiwanie rozwiązań. Teraz pokazujemy bardzo prosty przykład niezgodnego celu cząstkowego, wykorzystując stan początkowy STAN 1 na rysunku 8.18. Załóżmy, że celem planu jest STAN G, jak na Rysunku



, z on (b, a) ∧ on (a, c) i bloki d i e pozostające jak w STANIE 1. Można zauważyć, że jedna z części łącznego celu na (b, a) ∧ on (a, c) jest prawdą w STANIE 1, a mianowicie w (b, a). Ta już spełniona część celu musi zostać cofnięta, zanim będzie można osiągnąć drugi cel podrzędny (a, c). Przedstawienie tabeli trójkątów ma na celu złagodzenie niektórych z tych anomalii. Tabela trójkątów to struktura danych służąca do organizowania sekwencji działań, w tym potencjalnie niekompatybilnych celów podrzędnych w ramach planu. To rozwiązuje problem sprzecznych celów podrzędnych w ramach działań makr, przedstawiając globalną interakcję sekwencji operacji. Tabela trójkątów wiąże warunki wstępne jednej akcji z warunkami końcowymi, połączonymi listami dodawania i usuwania działań poprzedzających ją. Tabele trójkątów służą do określania, kiedy ten operator makra może zostać użyty podczas tworzenia planu. Zapisując te makrooperatory i wykorzystując je ponownie, STRIPS zwiększa efektywność wyszukiwania planistycznego. Rzeczywiście, możemy uogólnić operator makra, używając nazw zmiennych do zastąpienia nazw bloków w konkretnym przykładzie. Następnie możemy wywołać nowe uogólnione makro do wyszukiwania przycinania. W rozdziale 10, w naszej prezentacji uczenia się

w środowiskach opartych na symbolach, omawiamy techniki uogólniania operacji makr. Ponowne wykorzystanie operatorów makr pomaga również rozwiązać problem sprzecznych celów podrzędnych. Jak ilustruje poniższy przykład, po opracowaniu przez planistę planu celów stosu formularzy (X, Y) ∧ stosu (Y, Z), może on przechowywać i ponownie wykorzystywać ten plan. Eliminuje to potrzebę dzielenia celu na cele cząstkowe i pozwala uniknąć komplikacji, które mogą wystąpić. Rysunek przedstawia tabelę trójkątów dla stosu akcji makr (X, Y) ∧ stosu (Y, Z).



Tę akcję makra można zastosować do stanów, w których on (X, Y) ∧ wyczyść (X) ∧ wyczyść (Z) jest prawdziwe. Ta tabela trójkątów jest odpowiednia do rozpoczęcia STANU 1 z X = b, Y = a i Z = c. Atomowe działania planu są rejestrowane wzdłuż przekątnej. Są to cztery akcje, podniesienie, odłożenie, ułożenie i odłożenie, omówione wcześniej w tej sekcji. Zestaw warunków wstępnych każdej z tych akcji znajduje się w wierszu poprzedzającym tę akcję, a warunki końcowe każdej akcji znajdują się w kolumnie poniżej. Na przykład wiersz 5 zawiera warunki wstępne dla odbioru (X), a kolumna 6 zawiera warunki końcowe (listy dodawania i usuwania) odbioru (X). Te warunki końcowe są umieszczane w wierszu akcji, która używa ich jako warunków wstępnych, organizując je w sposób odpowiedni dla dalszych działań. Celem tabeli trójkątów jest prawidłowe przeplatanie warunków wstępnych i końcowych każdego z mniejszych działań, które składają się na większy cel. W związku z tym tabele trójkątów odnoszą się do kwestii nieliniowości w planowaniu na poziomie makrooperatorów; Planiści częściowego zamówienia (Russell i Norvig 1995) i inne podejścia dalej rozwiązywały te problemy.

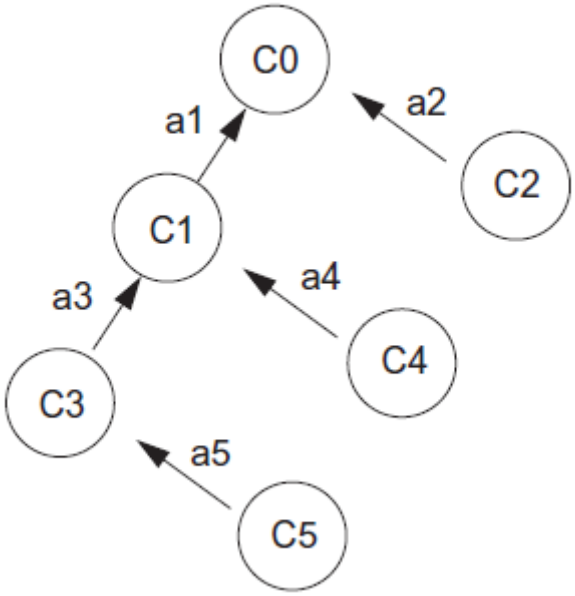
Jedną z zalet tabel trójkątnych jest pomoc, jaką mogą zaoferować w próbach powrotu do zdrowia po nieoczekiwanych zdarzeniach, takich jak nieznaczne przesunięcie bloku lub wypadki, takie jak upuszczenie bloku. Często wypadek może wymagać wykonania kopii zapasowej kilku kroków, zanim będzie można wznowić plan. Gdy coś pójdzie nie tak z rozwiązaniem, planista może wrócić do wierszy i kolumn tabeli trójkątów, aby sprawdzić, co jest prawdą. Gdy planista zorientował się, co nadal jest prawdą w wierszach i kolumnach, wie, jaki powinien być następny krok, jeśli większe rozwiązanie ma zostać ponownie uruchomione. Jest to sformalizowane pojęciem jądra. N-te jądro to część wspólna wszystkich wierszy poniżej, w tym n-ty wiersz i wszystkie kolumny na lewo od n-tej kolumny włącznie. Na rysunku wytłuszczono trzecie jądro. Wykonując plan przedstawiony w tabeli trójkątów, i-ta operacja (czyli operacja w wierszu i) może być wykonana tylko wtedy, gdy wszystkie predykaty zawarte

w i-tym jądra są prawdziwe. Daje to prosty sposób sprawdzenia, czy krok można podjąć, a także wspiera systematyczne usuwanie skutków zakłóceń w planie. Mając tablicę trójkątów, znajdujemy i wykonujemy akcję o najwyższym numerze, której jądro jest włączone. To nie tylko pozwala nam tworzyć kopie zapasowe w planie, ale także dopuszcza możliwość, że nieoczekiwane zdarzenie pozwoli nam przeskoczyć do przodu w planie. Warunki w skrajnej lewej kolumnie są warunkami wstępnymi dla akcji makra jako cały. Warunki w dolnym wierszu to warunki dodane do świata przez operator makra. W ten sposób tabelę trójkątów można zapisać jako operator makra z własnym zestawem warunków przed i po, lub dodawać i usuwać listy. Oczywiście podejście oparte na tabeli trójkątów traci część semantyki poprzednich modeli planowania. Zwróćmy na przykład uwagę, że zachowane są tylko te warunki końcowe aktu, które są również warunkami wstępnymi późniejszych aktów. Tak więc, jeśli gwarantowana poprawność jest pożądanym wynikiem, pożądana może być dalsza weryfikacja tabel trójkątów, być może z dodatkowymi informacjami, które mogą pozwolić na tworzenie sekwencji tabel trójkątów. Inne problemy pojawiają się przy stosowaniu makroperatorów w planowaniu. Wraz ze wzrostem liczby operatorów makr, planista ma do wykorzystania potężniejsze operacje, zmniejszając rozmiar przestrzeni stanów, którą należy przeszukiwać. Niestety na każdym etapie wyszukiwania należy sprawdzić wszystkie te operatory. Dopasowanie wzorców potrzebne do określenia, czy można zastosować operator, może zwiększyć obciążenie procesu wyszukiwania, często przeciwdziałając korzyściom osiągniętym przez zapisywanie operacji makr. Problematyka określenia, kiedy należy zapisać operację makra i jak określić następny operator, pozostaje przedmiotem wielu badań. W następnej sekcji opiszemy algorytm, w ramach którego można jednocześnie realizować wiele celów cząstkowych, planowanie teleoreaktywne.

Planowanie teleoreaktywne

Od wczesnych prac nad planowaniem opisanych w poprzedniej sekcji nastąpił szereg znaczących postępów. Wiele z tych prac poświęcono planowaniu niezależnemu od domeny, ale ostatnio ważne stało się planowanie bardziej specyficzne dla domeny, w którym stosowane są rozproszone mechanizmy wyczuwania / reagowania. W następnych sekcjach opisujemy system niezależny od domeny, planowanie teleo-reaktywne, a także bardziej zależny od domeny planner, Burton, z NASA. Jako podsumowanie pierwszych dwóch dekad planowania badań polecamy. Planowanie teleoreaktywne (TR) zostało po raz pierwszy zaproponowane przez Nilssona i Bensona na Uniwersytecie Stanforda. Planowanie TR oferuje architekturę ogólnego przeznaczenia, z zastosowaniem w wielu dziedzinach, w których kontrola złożonych podsystemów musi być skoordynowana, aby osiągnąć wyższy poziom. W ten sposób łączy podejście odgórne hierarchicznej architektury sterowania z „opartą na agentach" lub oddolną pomocą. Rezultatem jest system, który umożliwia rozwiązywanie złożonych problemów poprzez koordynację prostych agentów specyficznych dla zadania. Uzasadnieniem tego podejścia jest to, że zwykli agenci mają tę zaletę, że pracują w mniejszych i bardziej ograniczonych przestrzeniach problemowych. Z drugiej strony, kontroler wyższego poziomu może podejmować bardziej globalne decyzje dotyczące całego systemu, na przykład w jaki sposób bieżący wynik decyzji czysto lokalnej może wpływać na wynik ogólnego zadania rozwiązywania problemów. Sterowanie teleoreaktywne łączy aspekty sterowania opartego na sprzężeniu zwrotnym i dyskretnego planowania działań. Programy teleoreaktywne sekwencjonują wykonywanie działań, które zostały zebrane w plan zorientowany na cel. W przeciwieństwie do bardziej tradycyjnych środowisk planowania AI nie przyjmuje się żadnych założeń, że działania są dyskretne i nieprzerywalne, a skutki każdego działania są całkowicie przewidywalne. Wręcz przeciwnie, zderzenia mogą być podtrzymywane przez dłuższy czas, to znaczy, zdalne działania są wykonywane, o ile spełnione są warunki wstępne działań, a powiązany z nimi cel nie został jeszcze osiągnięty. Nilsson (1994) określa ten typ działania jako duratywny. Działania trwające mogą zostać przerwane, gdy aktywowana jest inna akcja bliżej celu najwyższego poziomu. Krótki cykl zmysł-reakcja zapewnia, że

gdy zmienia się środowisko, działania kontrolne również szybko się zmieniają, odzwierciedlając nowy stan rozwiązania problemu. Sekwencje działań teloreaktywnych są reprezentowane przez strukturę danych zwaną drzewem TR, jak na rysunku.



Drzewo TR jest opisane przez zestaw par warunek → akcja (lub reguły produkcji, rozdział 6.3). Na przykład:

$$C_0 \rightarrow A_0$$
$$C_1 \rightarrow A_1$$
$$C_2 \rightarrow A_2$$
$$\ldots$$
$$C_n \rightarrow A_n$$

gdzie Ci to warunki, a Ai to powiązane działania. Odnosimy się do C0 jako celu najwyższego poziomu drzewa, a A0 jako działania zerowego, co wskazuje, że po osiągnięciu celu najwyższego poziomu nie trzeba już nic robić. Na każdym cyklu realizacji teleoreaktywności systemu, każdy Ci jest oceniany od góry reguł do dołu (C0, C1, ... Cn), aż do znalezienia pierwszego prawdziwego warunku. Następnie wykonywana jest akcja powiązana z tym prawdziwym warunkiem. Cykl oceny jest następnie powtarzany z częstotliwością zbliżoną do reaktywności sterowania opartego na obwodzie. Produkcje Ci → Ai są zorganizowane w taki sposób, że każda akcja Ai, jeśli jest wykonywana w sposób ciągły w normalnych warunkach, ostatecznie sprawi, że jakiś warunek będzie wyższy w drzewie reguł TR. Wykonanie drzewa TR może być postrzegane jako adaptacyjne, ponieważ jeśli jakieś nieoczekiwane zdarzenie w środowisku kontrolnym odwróci efekt poprzednich działań, wykonanie TR powróci do warunku reguły niższego poziomu, który odzwierciedla ten warunek. Od tego momentu wznowi swoją pracę w kierunku spełnienia wszystkich celów wyższego poziomu. Podobnie, jeśli nieumyślnie wydarzy się coś dobrego, wykonanie TR jest oportunistyczne, ponieważ kontrola automatycznie przejdzie na działanie tego prawdziwego warunku. Drzewa TR można konstruować za pomocą algorytmu planowania, który wykorzystuje popularne metody redukcji celów AI. Rozpoczynając od celu najwyższego poziomu, planista wyszukuje działania, których efektem jest osiągnięcie tego celu. Warunki wstępne tych działań generują nowy zestaw celów podrzędnych i ta procedura się powtarza. Zakończenie jest osiągane, gdy warunki wstępne węzłów liści są spełnione przez aktualny stan

środowiska. W ten sposób algorytm planowania cofa się od celu najwyższego poziomu, poprzez redukcję celu, do stanu bieżącego. Działania oczywiście często mają skutki uboczne i planista musi uważać, aby sprawdzić, czy działanie na jakimkolwiek poziomie nie zmienia warunków, które są wymagane jako warunki wstępne działań na wyższym poziomie w drzewie TR. Redukcja celu jest zatem połączona ze spełnieniem ograniczeń, gdzie stosuje się różne strategie zmiany kolejności działań, aby wyeliminować możliwe naruszenia ograniczeń. Dlatego algorytmy planowania TR są wykorzystywane do budowania planów, których węzły-liścia są spełnione w aktualnym stanie środowiska problemowego. Zwykle nie budują planów pełnych, czyli planów, które można rozpocząć w dowolnym państwie świata, ponieważ takie plany są generalnie zbyt duże, aby je przechowywać lub skutecznie wykonywać. Ten ostatni punkt jest ważny, ponieważ czasami nieoczekiwane zdarzenie środowiskowe może doprowadzić świat do stanu, w którym żadne warunki wstępne działania w drzewie TR nie są spełnione i konieczna jest jakaś forma ponownego planowania. Odbywa się to zwykle poprzez ponowne aktywowanie planera TR. Benson i Nilsson wykorzystali planowanie teleoreaktywne w wielu dziedzinach aplikacji, w tym w sterowaniu rozproszonymi agentami robotów i budowaniu symulatora lotu. Klein i inni użyli planera reaktywnego w trybie tele do zbudowania i przetestowania przenośnej architektury sterującej przyspieszeniem wiązki cząstek naładowanych. Ci ostatni badacze podali szereg powodów, dla których warto zastosować kontroler teloreaktywny w domenie kontrolerów wiązek cząstek:

1. Belki akceleratora i związane z nimi diagnostyki są zwykle dynamiczne i hałaśliwe.

2. Na osiągnięcie celów strojenia akceleratora często wpływają procesy stochastyczne, przebicie RF lub oscylacje w źródle wiązki.

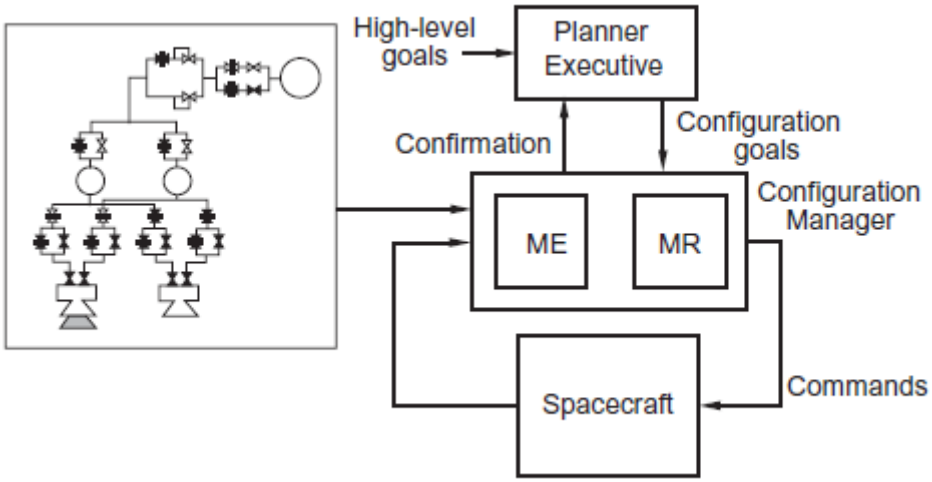
3. Wiele czynności wymaganych do strojenia ma charakter trwały. Dotyczy to zwłaszcza operacji poprawiania i optymalizacji, które należy kontynuować do momentu spełnienia określonych kryteriów.

4. Drzewa TR oferują intuicyjną strukturę do kodowania planów strojenia uzyskanych od fizyków zajmujących się akceleratorami. W rzeczywistości z bardzo niewielką pomocą fizycy byli w stanie opracować własne drzewa TR.

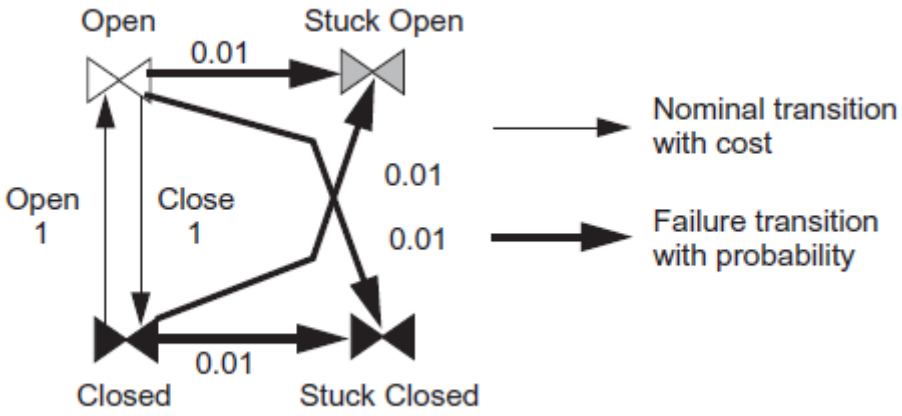
Dalsze szczegóły dotyczące tych aplikacji można znaleźć w źródłach. Następnie wrócimy do przykładu rozumowania opartego na modelu NASA z sekcji 8.3, aby opisać algorytmy sterowania / planowania dla układu napędowego pojazdów kosmicznych.

Planowanie: przykład NASA

W tej sekcji opisujemy, jak można zaimplementować planera w kontekście rozumowania opartego na modelu. Kontynuujemy przykład Williamsa i Nayaka NASA przedstawiony wcześniej. Livingstone to reaktywny menedżer konfiguracji, który wykorzystuje kompozycyjny, oparty na komponentach model układu napędowego statku kosmicznego do określania działań konfiguracyjnych, jak pokazano na rysunku.



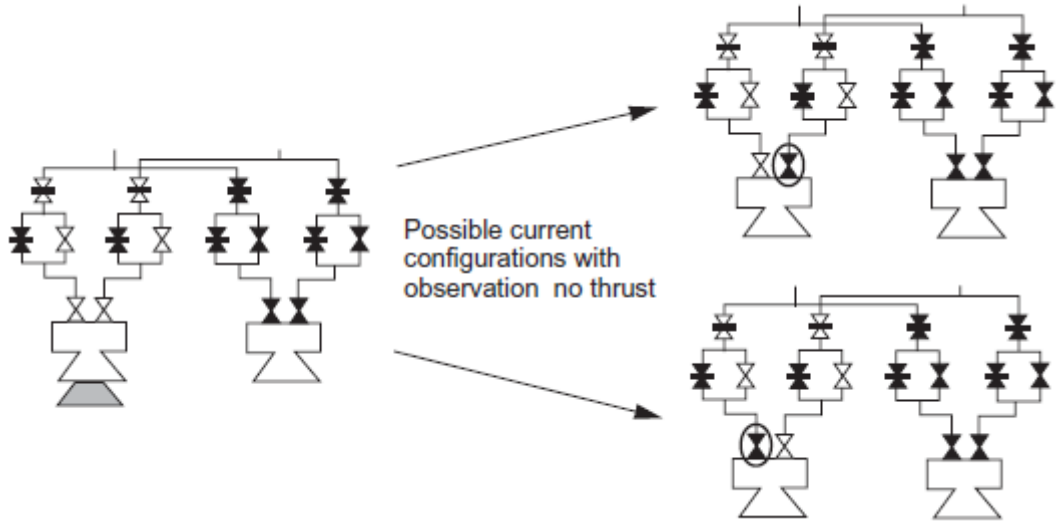
Każdy komponent napędu jest modelowany jako system przejściowy, który określa zachowanie trybu pracy i trybu awaryjnego elementu, przejścia nominalne i przejścia między trybami oraz koszty i prawdopodobieństwo przejść, jak na rysunku.



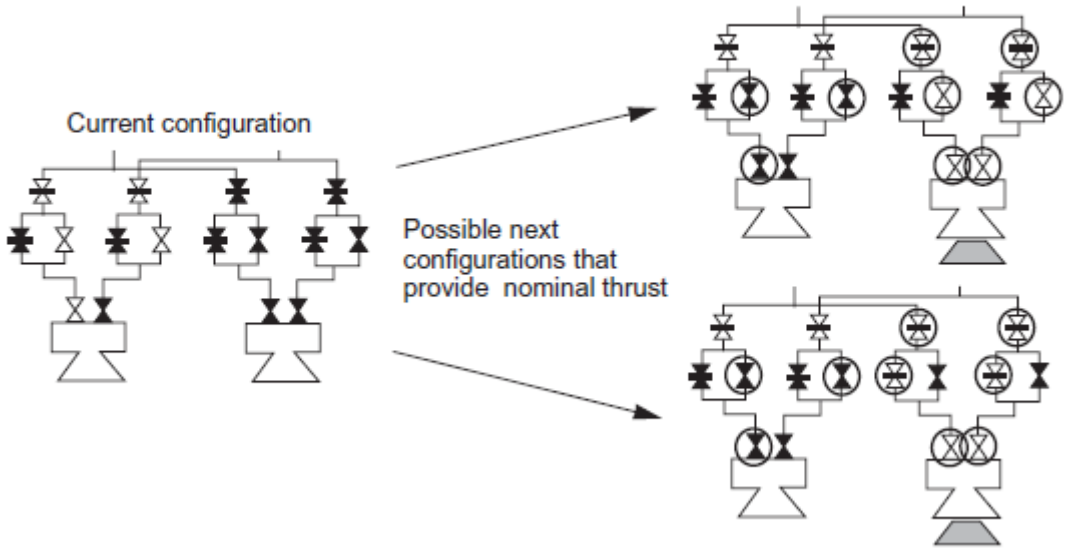
Na rysunku powyżej, otwarte i zamknięte to normalne tryby pracy, ale zablokowanie otwarte i zablokowanie zamknięte to tryby awarii. Polecenie otwórz ma koszt jednostkowy i powoduje przejście z trybu zamkniętego do otwartego, podobnie jak polecenie zamknięcia. Przejścia awaryjne przenoszą zawór z normalnego trybu pracy do jednego z trybów awaryjnych z prawdopodobieństwem 0,01. Zachowania modów są określane za pomocą wzorów w logice zdań, ale przejścia między trybami są określane za pomocą formuł w ograniczonej czasowej logice zdań. Ograniczona logika czasowa, propozycyjna jest odpowiednia do modelowania sprzętu cyfrowego, sprzętu analogowego z wykorzystaniem abstrakcji jakościowych oraz oprogramowania czasu rzeczywistego wykorzystującego modele współbieżnych systemów reaktywnych. Model systemu przejść statku kosmicznego jest kompozycją jego składowych układów przejściowych, w których zbiór konfiguracji statku kosmicznego jest iloczynem krzyżowym zbiorów modów składowych. Zakładamy, że układy przejść składowych działają synchronicznie; to znaczy, dla każdego przejścia statku kosmicznego każdy element wykonuje przejście.

Oparty na modelu menedżer konfiguracji wykorzystuje swój model systemu przejścia do identyfikacji bieżącej konfiguracji statku kosmicznego, zwanego estymacją trybu ME, oraz do przeniesienia statku kosmicznego do nowej konfiguracji, która osiąga pożądane cele konfiguracyjne, zwane rekonfiguracją

trybu, MR. ME przyrostowo generuje wszystkie przejścia statków kosmicznych z poprzedniej konfiguracji, tak że modele otrzymanych konfiguracji są zgodne z bieżącymi obserwacjami. Zatem na rysunku pokazano sytuację, w której lewy silnik pracuje normalnie w poprzednim stanie, ale w obecnym stanie nie obserwuje się ciągu.



ME muszą zidentyfikować konfiguracje, do których przeszedł statek kosmiczny, które są odpowiedzialne za tę obserwację. Rysunek przedstawia dwa możliwe przejścia, odpowiadające awarii jednego z głównych zaworów silnika. Zepsute lub zablokowane zamknięte zawory są oznaczone kółkiem. Wiele innych przejść, w tym mało prawdopodobne podwójne błędy, może również odpowiadać za obserwacje. MR określa polecenia, które mają być wysłane do statku kosmicznego w taki sposób, że wynikające z tego przejścia wprowadzają statek kosmiczny w konfigurację, która osiąga cel konfiguracji w następnym stanie, jak na rysunku



Ten rysunek przedstawia sytuację, w której identyfikacja trybu zidentyfikowała awarię zaworu głównego silnika prowadzącego do lewego silnika. MR uzasadnia, że normalny ciąg można przywrócić w następnym stanie, jeśli zostanie otwarty odpowiedni zestaw zaworów prowadzących do właściwego

silnika. Rysunek przedstawia dwie z wielu konfiguracji, które mogą osiągnąć zamierzony cel, kiedy zawory w okręgu otrzymują polecenie zmiany stanu. Przejście do konfiguracji na górze jest tańsze, ponieważ uruchamiane są tylko niezbędne pirozawory. Zawory prowadzące do lewego silnika są wyłączane, aby spełnić ograniczenie, zgodnie z którym w danym momencie może odpalać co najwyżej jeden silnik. Zastosowanie modelu statku kosmicznego zarówno w ME, jak i MR zapewnia prawidłowe osiąganie celów konfiguracyjnych. Zarówno ME, jak i MR są reaktywne (patrz sekcja 6.4). ME wnioskuje o aktualnej konfiguracji ze znajomości poprzedniej konfiguracji i bieżących obserwacji. MR bierze pod uwagę tylko polecenia, które osiągają cel konfiguracji w następnym stanie. Biorąc pod uwagę te zobowiązania, kluczowa jest decyzja o zamodelowaniu przejść komponentów jako synchronicznych. Alternatywą jest modelowanie przejść wielu elementów poprzez przeplatanie. Jednak przeplatanie może ustawić dowolną odległość między bieżącą konfiguracją a konfiguracją celu, pokonując pragnienie ograniczenia wnioskowania do małej ustalonej liczby stanów. Stąd przejścia komponentów modelu są synchroniczne. Jeśli przejścia komponentów w bazowym oprogramowaniu sprzętowym nie są synchroniczne, Livingstone zakłada w modelowaniu, że wszystkie przeploty przejść są prawidłowe i umożliwiają osiągnięcie pożądanej konfiguracji. To założenie zostało usunięte w Burton, kontynuacji Livingstone'a, którego planista określa sekwencję działań kontrolnych, które powodują wszystkie pożądane przejścia (Williams i Nayak 1997). NASA używa architektury Burton w misji o nazwie Tech Sat 21. W przypadku Livingstone, ME i MR nie muszą generować odpowiednio wszystkich przejść i poleceń sterujących. Wymagane są raczej tylko najbardziej prawdopodobne przejścia i optymalne polecenie sterujące. Są one skutecznie regenerowane przez przekształcenie ME i MR jako kombinatorycznych problemów optymalizacji. W tym przeformułowaniu ME stopniowo śledzi prawdopodobne trajektorie statku kosmicznego, zawsze rozszerzając trajektorie prowadzące do obecnych konfiguracji o najbardziej prawdopodobne przejścia. MR następnie identyfikuje polecenie o najniższym oczekiwanym koszcie, które przechodzi z prawdopodobnych bieżących konfiguracji do konfiguracji, która osiąga pożądany cel. Te kombinatoryczne problemy optymalizacji są skutecznie rozwiązywane za pomocą algorytmu wyszukiwania w pierwszej kolejności ukierunkowanego na konflikt. Zobacz Williams i Nayak w celu uzyskania bardziej formalnej charakterystyki ME i MR, a także bardziej szczegółowego opisu algorytmów wyszukiwania i planowania.

Epilog

Architektura systemu eksperckiego opartego na regułach to system produkcyjny. Niezależnie od tego, czy produkt końcowy jest oparty na danych, czy na celu, model oprogramowania to wyszukiwanie wykresów generowane przez system produkcyjny. Wdrażamy system produkcyjny oparty na eksperckich powłok systemowych w językach Java, Prolog i Lisp. Reguły tych systemów mogą obejmować miary pewności w ograniczonej formie do projektowania wyszukiwania heurystycznego.

Sztuczna inteligencja : Rozumowanie w niepewnych sytuacjach

(IX / XVI)

ĆWICZENIA

1. Zidentyfikuj trzy dziedziny zastosowań, w których konieczne jest rozumowanie w warunkach niepewności. Wybierz jeden z tych obszarów i zaprojektuj sześć reguł wnioskowania odzwierciedlających rozumowanie w tej dziedzinie.

2. Biorąc pod uwagę następujące zasady w aplikacji systemu eksperckiego typu „back-chain":

A ∧ nie (B) ⇒ C (.9)

C v D ⇒ E (0,75)

F⇒A (.6)

G⇒ D (.8)

System może podsumować następujące fakty (z zwierzeniami):

F (0,9)

B (-. 8)

G (.7)

Użyj algebry współczynników pewności Stanforda, aby określić E i jej pewność.

3. Rozważmy prostą regułę podobną do MYCIN: jeśli A ∧ (B v C) ⇒ D (.9) ∧ E (.75). Omów kwestie, które pojawiają się przy uchwyceniu tych niepewności w kontekście bayesowskim. Jak można potraktować tę regułę w rozumowaniu Dempstera-Shafera?

4. Utwórz nowy przykład rozumowania diagnostycznego i użyj rozszerzenia Równania Dempstera – Shafera z sekcji 9.2.3 w celu uzyskania rozkładów przekonań, jak w tabeli 1 i 2.

5. Skorzystaj z aksjomatów schematu przedstawionych przez McCarthy'ego, aby utworzyć wyniki opisowe przedstawione w sekcji 9.1.3.

6. Utwórz kolejną sieć rozumowania podobną do tej z rysunku 4 i pokaż siatkę zależności dla jej obiektów, tak jak pokazano na rysunku 5.

7. Rozumowanie przy założeniu minimalnego modelu jest ważne w życiu codziennym człowieka. Opracuj jeszcze dwa przykłady, które zakładają minimalne modele.

8. Kontynuuj przykład odwróconego wahadła z sekcji 9.2.2 z dwoma kolejnymi iteracjami kontrolera, gdzie wyjście jednej iteracji dostarcza wartości wejściowe dla następnej iteracji.

9. Napisz program, który implementuje kontroler rozmyty z punktu 9.2.2.

10. Przejdź do literatury, na przykład Rossa, i opisz dwa inne obszary, w których kontrola rozmyta może być odpowiednia. Skonstruuj zestaw rozmytych reguł dla tych domen.

11. Tom zainwestował część swoich oszczędności w portfel pięciu akcji, H = {Bank of China, Citibank, Intel, Nokia, Legend}. Akcje te można podzielić na akcje banków (Bank of China i Citibank), akcje Hi-tech (Intel, Nokia i Legend) oraz akcje Chin (Bank of China i Legend). Zaproponowałeś zbudowanie systemu eksperckiego, który może udzielić mu porady, jak powinien zarządzać swoim portfelem akcji. W tym celu przeprowadziłeś wywiady z wieloma konsultantami finansowymi i opracowałeś zestaw reguł dla systemu ekspertowego. Te zasady mówią Tomowi, jak powinien podzielić swoje pieniądze między różne akcje w swoim portfelu. Zalecane proporcje są wprost proporcjonalne do prawdopodobieństwa wzrostu ceny akcji lub ich rodzaju:

R1: JEŚLI stopa procentowa wzrośnie, WTEDY kup akcje banków, akcje Hi-tech i Nokię w proporcjach odpowiednio 0,8, 0,15 i 0,05.

R2: JEŚLI stopa zatrudnienia wzrośnie, WTEDY kup akcje Bank of China, Intel i Hi-Tech w proporcjach odpowiednio 0,5, 0,2 i 0,3.

R3: JEŚLI stopa inflacji spada, WTEDY kup akcje Intela, Citibanku i Hi-Tech w proporcji odpowiednio 0,1, 0,4 i 0,5

David zauważył, że stopa procentowa właśnie wzrosła, korzystając z teorii dowodów Dempstera-Shafera, obliczyć przedziały wierzeń dla trzech typów akcji, tj. Akcji Banku, akcji Chin i akcji Hi-Tech oraz dla Nokii . Czy jest jakaś różnica w wierzeniach i przedziałach wierzeń między akcjami Nokii i Hi-Tech? Dlaczego lub dlaczego nie? (Ten problem został zaproponowany przez Jamesa Liu Nga Kwoka z Uniwersytetu Politechnicznego w Hongkongu, rozwiązanie w Przewodniku instruktora jest autorstwa studenta Qi Xianming).

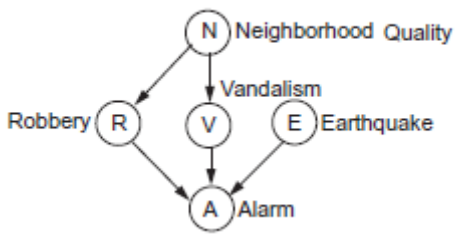
12. Umieść kolejne łącze na rysunku 16, powiedzmy, że sezon łączący jest bezpośrednio połączony z gładkim chodnikiem, a następnie utwórz drzewo klikowe, które będzie reprezentowało tę sytuację. Porównaj kwestie złożoności z problemami z drzewa klikowego na rysunku 17.

13. Uzupełnij symboliczne oceny wymagane do zakończenia tabeli 4.

14. Utwórz algorytm propagacji przekonań bayesowskich i zastosuj go do domeny śliskiego chodnika w sekcji 9.3.2. Możesz użyć metody przekazywania komunikatów Pearl'a (1988) lub metody triangulacji klikowej zaproponowanej przez Lauritzena i Spiegelhaltera (1988).

15. Utwórz diagram przekonań bayesowskich dla innej aplikacji, na przykład diagnozy medycznej, odkrycia geologicznego lub analizy usterek samochodowych. Wskaż przykłady separacji d i utwórz drzewo klikowe dla tej sieci.

16. Utwórz kliknięcia i drzewo węzłów dla następującej sytuacji



Napad, wandalizm i trzęsienie ziemi mogą wywołać alarm w domu. Istnieje również miara potencjalnych złości w sąsiedztwie domu.

17. Weźmy pod uwagę sytuację rozumowania diagnostycznego przedstawioną w tabeli 1 i 2 modelu Dempstera-Shafera z sekcji 9.2.3 i przekształć ją w sieć przekonań bayesowskich. Porównaj i porównaj te dwa podejścia do diagnozy.

18. Zakładając, że chciałeś zaprojektować model Markowa drugiego rzędu, tj. W którym każdy obserwowalny stan byłby zależny od dwóch poprzednich obserwowalnych stanów. Jak byś to zrobił? Jak wyglądałaby macierz prawdopodobieństwa przejścia?

19. Biorąc pod uwagę możliwy do zaobserwowania model pogody Markowa z sekcji 9.3.4:

za. Określ prawdopodobieństwo, że (dokładnie) następne pięć dni będzie słońce.

b. Jakie jest prawdopodobieństwo dokładnie trzech dni słońca, potem jednego dnia opadów, a potem dokładnie jednego dnia słonecznego?

ROZUMOWANIE W NIEPEWNYCH SYTUACJACH

9.0 Wprowadzenie

W większości części I, II i III nasze procedury wnioskowania były zgodne z modelem rozumowania stosowanym w rachunku predykatów: z prawidłowych przesłanek, prawidłowe reguły wnioskowania dają nowe, gwarantowane prawidłowe wnioski. Jednak, jak widzieliśmy w częściach 5 i 7, istnieje wiele sytuacji, które nie pasują do tego podejścia; to znaczy, musimy wyciągać użyteczne wnioski ze źle sformułowanych i niepewnych dowodów, stosując błędne reguły wnioskowania. Wyciąganie pożytecznych wniosków z niepełnych i nieprecyzyjnych danych przy błędnym uzasadnieniu nie jest zadaniem niemożliwym; robimy to z powodzeniem w prawie każdym aspekcie naszego codziennego życia. Stawiamy prawidłowe diagnozy lekarskie i zalecamy leczenie niejednoznacznych objawów; analizujemy problemy z naszymi samochodami lub stereo; rozumiemy wypowiedzi językowe, które są często niejednoznaczne lub niepełne; rozpoznajemy przyjaciół po ich głosach lub gestach; i tak dalej. Aby zademonstrować problem rozumowania w niejednoznacznych sytuacjach, rozważ Regułę 2 z samochodowego systemu eksperckiego:

gdyby

silnik się nie obraca i

światła się nie włączają

następnie

problem to akumulator lub kable.

Na pozór ta reguła wygląda jak normalna relacja predykatów do stosowania we wnioskach dźwiękowych (modus ponens). Tak jednak nie jest; ma charakter heurystyczny. Możliwe, choć bardzo mało prawdopodobne, że akumulator i kable są w porządku, ale samochód ma po prostu zły rozrusznik i przepalone reflektory. Awaria silnika i zapalenie się lampek niekoniecznie oznacza, że akumulator i kable są złe. Ciekawe, że prawdą jest odwrotność reguły:

gdyby

problem to akumulator lub kable

następnie

silnik się nie obraca i

światła się nie włączają.

Poza tym, co nadprzyrodzone, z rozładowaną baterią, ani światła, ani rozrusznik nie będą działać! Nasz system ekspercki oferuje przykład abdukcyjnego rozumowania. Formalnie uprowadzenie stwierdza, że z P → Q i Q można wywnioskować P. Porwanie jest błędną regułą wnioskowania, co oznacza, że wniosek niekoniecznie jest prawdziwy dla każdej interpretacji, w której przesłanki są prawdziwe (sekcja 2.3). Chociaż uprowadzenie jest niewłaściwe, często jest niezbędne do rozwiązania problemów. „Logicznie poprawna" wersja reguły baterii nie jest zbyt przydatna w diagnozowaniu awarii samochodu, ponieważ jej przesłanka, czyli zły akumulator, jest naszym celem, a wnioski z niej to obserwowalne objawy, z którymi musimy pracować. Reguła może być jednak stosowana w sposób abdukcyjny z zasady w wielu diagnostycznych systemach ekspertowych. Usterki lub choroby powodują (implikują) objawy, a nie odwrotnie; ale diagnoza musi działać od objawów aż po ich przyczyny. W systemach opartych na wiedzy często dodajemy do reguły czynnik pewności, aby zmierzyć zaufanie do jej konkluzji. Na przykład reguła P → Q (.9) wyraża przekonanie „Jeśli uważasz, że P jest prawdziwe, to wierzysz, że Q wydarzy się w 90% przypadków". W ten sposób reguły heurystyczne mogą wyrażać wyraźną politykę dotyczącą przekonań.

Inną kwestią związaną z rozumowaniem systemu eksperckiego jest sposób wyciągania przydatnych wyników z danych z brakującymi, niepełnymi lub niepoprawnymi informacjami. Możemy użyć środków zapewniających pewność, aby odzwierciedlić naszą wiarę w jakość danych, na przykład stwierdzając, że światła mają pełną moc (.2), mogą wskazywać, że reflektory się włączają, ale są słabe i ledwo widoczne. Przekonania i niedoskonałe dane można propagować za pomocą reguł ograniczających wnioski. W tym rozdziale omówimy kilka sposobów radzenia sobie z wnioskowaniem abdukcyjnym i niepewnością, zwłaszcza, że jest to wymagane do rozwiązywania problemów wymagających dużej wiedzy. W sekcji 9.1 pokazujemy, jak można rozszerzyć formalizm oparty na logice, aby zająć się zadaniem abdukcyjnym, w tym wykorzystanie systemów niemonotonicznych wspieranych przez algorytmy utrzymywania prawdy. W 9.2 rozważymy kilka alternatyw dla logiki, w tym algebrę czynników pewności ze Stanforda, rozumowanie „rozmyte" oraz teorię dowodów Dempstera-Shafera. Te proste obliczenia są często wykorzystywane do rozwiązywania niektórych problemów związanych ze złożonością stosowania pełnego podejścia bayesowskiego do budowania systemów ekspertowych. W podrozdziale 9.3 wprowadzamy stochastyczne podejście do niepewnego rozumowania. Techniki te opierają się na twierdzeniu Bayesa dotyczącym rozumowania częstotliwości zdarzeń w oparciu o wcześniejsze informacje o tych zdarzeniach. Kończymy 9.3, wprowadzając modele graficzne, w tym sieci przekonań bayesowskich oraz obserwowalne i ukryte modele Markowa. W Części 13 przedstawiamy stochastyczne podejście do uczenia się.

9.1 Abdukcyjne wnioskowanie oparte na logice

Najpierw przedstawiamy podejście do uprowadzenia oparte na logice. Dzięki logice fragmenty wiedzy są wyraźnie używane w rozumowaniu i mogą być częścią wyjaśnień wynikających z wniosków. Ale tradycyjna logika ma również swoje ograniczenia, zwłaszcza w obszarach, w których brakuje informacji, są one zmieniane lub niepewne; w takich sytuacjach tradycyjne procedury wnioskowania mogą się nie nadawać. Przedstawiamy kilka rozszerzeń tradycyjnej logiki, które pozwalają jej wspierać wnioskowanie abdukcyjne. W 9.1.1 rozszerzamy logikę, aby opisała świat zmieniających się informacji i przekonań. Tradycyjna logika matematyczna jest monotoniczna: zaczyna się od zestawu aksjomatów uznawanych za prawdziwe i wyciąga wnioski z ich konsekwencji. Jeśli dodamy nowe informacje do tego systemu, może to spowodować zwiększenie zestawu prawdziwych stwierdzeń. Dodanie wiedzy nigdy

nie zmniejszy zestawu prawdziwych stwierdzeń. Ta monotoniczna właściwość prowadzi do problemów, gdy próbujemy modelować rozumowanie w oparciu o przekonania i założenia. Rozumując z pewnością, ludzie wyciągają wnioski na podstawie ich aktualnego zestawu przekonań; jednakże, w przeciwieństwie do aksjomatów matematycznych, przekonania te, wraz z ich konsekwencjami, mogą się zmieniać w czasie, gdy dostępnych będzie więcej informacji. Rozumowanie niemonotoniczne rozwiązuje problem zmiany przekonań. Niemonotoniczny system rozumowania radzi sobie z niepewnością, przyjmując najbardziej rozsądne założenia w świetle niepewnych informacji. Następnie kontynuuje rozumowanie tak, jakby te założenia były prawdziwe. W późniejszym czasie przekonanie może się zmienić, co wymaga ponownego przeanalizowania wszelkich wniosków wynikających z tego przekonania. Następnie można zastosować algorytmy utrzymywania prawdy, sekcja 9.1.2, w celu utrzymania spójności bazy wiedzy. Inne porywające rozszerzenia logiki obejmują „modele minimalne", sekcja 9.1.3, oraz podejście „ustalonego pokrycia", sekcja 9.1.4.

9.1.1 Logiki dla wnioskowania niemonotonicznego

Nonmonotoniczność jest ważną cechą rozwiązywania problemów człowieka i zdrowego rozsądku. W większości planów, na przykład podczas jazdy do pracy, przyjmujemy liczne założenia dotyczące dróg i ruchu. Jeśli stwierdzimy, że jedno z tych założeń jest naruszone, być może przez budowę lub wypadek na naszej zwykłej trasie, zmieniamy plany i znajdujemy alternatywną trasę. Konwencjonalne rozumowanie przy użyciu logiki predykatów opiera się na trzech ważnych założeniach. Po pierwsze, opisy predykatów muszą być wystarczające w odniesieniu do naszej domeny aplikacji. Oznacza to, że należy przedstawić wszystkie informacje niezbędne do rozwiązania problemu. Po drugie, baza informacji musi być spójna; to znaczy, fragmenty wiedzy nie mogą być ze sobą sprzeczne. Wreszcie, dzięki zastosowaniu reguł wnioskowania, znane informacje rodną monotonnie. Jeśli którekolwiek z tych trzech założeń nie zostanie spełnione, konwencjonalne podejście oparte na logice nie zadziała. Systemy niemonotoniczne rozwiązują każdy z tych trzech problemów. Po pierwsze, systemy rozumowania często borykają się z brakiem wiedzy na temat domeny. Jest tu ważna kwestia: przypuśćmy, że nie mamy wiedzy o predykacie p; czy to oznacza brak wiedzy nie jesteśmy pewni, czy p jest prawdziwe, czy też jesteśmy pewni, że nie p jest prawdziwe? Na to pytanie można odpowiedzieć na wiele sposobów. Prolog wykorzystuje założenie o zamkniętym świecie, aby określić jako fałszywe wszystko, czego jego system rozumowania nie może okazać się prawdziwy. Jako ludzie często przyjmujemy alternatywne podejście, zakładając, że coś jest prawdą, chyba że można wyraźnie wykazać, że jest fałszywe.

Innym podejściem do problemu braku wiedzy jest formułowanie wyraźnych założeń dotyczących prawdy. W ludzkim rozumowaniu zakładamy niewinność ludzi niezwiązanych bezpośrednio z przestępstwem. Prawdopodobnie posunęlibyśmy się nawet dalej i założylibyśmy niewinność tych, którzy nie mogli skorzystać na zbrodni. Efektem tych założeń jest skuteczne uzupełnienie brakujących szczegółów wiedzy i rozszerzenie naszego rozumowania o nowe wnioski oparte na tych założeniach. Omówimy założenie zamkniętego świata i jego alternatywy w sekcji 9.1.3. Ludzie rozumują na podstawie tego, jak zwykle działa świat. Większość ptaków lata. Rodzice zwykle kochają i wspierają swoje dzieci. Wyciągamy wnioski na podstawie zgodności rozumowania z naszymi założeniami dotyczącymi świata. W tej sekcji omówimy dodawanie operatorów modalnych, takich jak jest spójne z i chyba, aby przeprowadzić wnioskowanie oparte na założeniach. Drugie założenie wymagane od tradycyjnych systemów opartych na logice jest takie, że wiedza wspierająca rozumowanie musi być spójna. Dla ludzkich rozumów byłoby to bardzo ograniczające założenie. Diagnozując problem, często mamy wiele możliwych wyjaśnień sytuacji, zakładając, że coś jest prawdą, dopóki alternatywne założenie nie okaże się bardziej owocne. Na przykład, analizując wypadek lotniczy, ekspert ds. Katastrof rozważy kilka alternatywnych przyczyn, eliminując (wyjaśniając) tylko niektóre w miarę odkrywania

nowych informacji. My, ludzie, wykorzystujemy wiedzę o świecie, jak to zwykle bywa, próbując kierować rozumowaniem poprzez alternatywne scenariusze. Chcielibyśmy, aby systemy logiczne były w stanie przyjmować alternatywne hipotezy. Wreszcie, jeśli chcemy używać logiki, musimy zająć się problemem aktualizacji bazy wiedzy. Są tutaj dwie kwestie: po pierwsze, w jaki sposób możemy dodać wiedzę opartą wyłącznie na założeniach, a po drugie, co możemy zrobić, gdy jedno z naszych założeń okaże się później błędne. Aby rozwiązać pierwszą kwestię, możemy pozwolić na dodanie nowej wiedzy na podstawie założeń. Zakłada się, że ta nowa wiedza jest poprawna i może z kolei zostać wykorzystana do wnioskowania o większej ilości nowej wiedzy. Koszt tej praktyki polega na tym, że musimy śledzić wszystkie rozumowania i dowody oparte na założeniach: musimy być przygotowani do ponownego rozważenia wszelkiej wiedzy opartej na tych założeniach. Rozumowanie niemonotoniczne, ponieważ wnioski trzeba czasem ponownie przemyśleć, jest takie nazywany wykonalnym; to znaczy nowe informacje mogą czasami unieważnić poprzednie wyniki. Reprezentacje i procedury wyszukiwania, które śledzą etapy rozumowania systemu logicznego, nazywane są systemami utrzymania prawdy lub TMS. W rozsądnym rozumowaniu TMS zachowuje spójność bazy wiedzy, śledząc wnioski, które mogą później wymagać zakwestionowania. W Sekcji 9.1.2 rozważymy kilka podejść do utrzymania prawdy. Najpierw rozważymy operatory, które mogą sprawić, że tradycyjne systemy rozumowania oparte na logice będą możliwe do pokonania. Wdrażając rozumowanie niemonotoniczne, możemy rozszerzyć naszą logikę o operator, chyba że. chyba że popiera wnioski oparte na przekonaniu, że jego argument jest nieprawdziwy. Załóżmy, że mamy następujący zestaw zdań logiki predykatów:

p (X) chyba, że q (X) → r (X)

p (Z)

r (W) → s (W)

Pierwsza reguła oznacza, że możemy wywnioskować r(X), jeśli p(X) jest prawdziwe i nie wierzymy, że q(X) jest prawdziwe. Kiedy te warunki są spełnione, wnioskujemy r(X), a używając r(X), możemy wnioskować o s(X). Następnie, jeśli zmienimy nasze przekonanie lub stwierdzimy, że q(X) jest prawdziwe, r(X), a także s(X) muszą zostać wycofane. Zauważ, że o ile nie dotyczy raczej kwestii wiary niż prawdy. W konsekwencji zmiana wartości argumentu z „nieznane lub uważane za fałszywe" na „uważane lub uznawane za prawdziwe" może spowodować, że wycofamy wszystkie wnioski, które zależą od tych przekonań. Rozszerzając naszą logikę o rozumowanie z przekonaniami, które można później wycofać, wprowadzamy do systemu niemonotoniczność. Opisany schemat rozumowania może być również użyty do zakodowania reguł domyślnych. Jeśli zamienimy p (X), chyba że q (X) → r (X) z p (X), chyba że ab p (X) → r (X), gdzie ab p (X) reprezentuje nieprawidłowe p (X), stwierdzamy, że jeśli nie mamy nienormalnego wystąpienia p, na przykład ptaka ze złamanym skrzydłem, możemy wyciągnąć wniosek, że jeśli X jest ptakiem, to X może latać. Drugi operator modalny do rozszerzania systemów logicznych jest sugerowany przez McDermotta i Doyle'a. Rozszerzają one logikę predykatów pierwszego rzędu o operator modalny M, który jest umieszczony przed odczytaniem predykatu zgodnie z. Na przykład:

∀ X good_student (X) ∧ M study_hard (X) → absolwenci (X)

Ten punkt można przeczytać: Dla wszystkich X, gdzie X jest dobrym uczniem i jeśli fakt, że X uczy się ciężko, jest zgodny z wszystkim innym, co wiemy, to X ukończy szkołę. Oczywiście najtrudniejszą częścią jest tutaj dokładne zdefiniowanie, co jest zgodne ze wszystkim, co wiemy, co może w rzeczywistości oznaczać. Po pierwsze, zauważamy, że jest to zgodne ze wszystkim, co wiemy, może nie być rozstrzygalne. Powodem jest to, że operator modalny tworzy nadzbiór już nierozstrzygalnego systemu, a zatem będzie nierozstrzygalny. Istnieją dwa sposoby rozwiązania problemu nierozstrzygalności. Po pierwsze, możemy użyć negacji jako dowodu niepowodzenia w wykazaniu, z którym jest zgodny. W

naszym przykładzie spróbowalibyśmy udowodnić, że nie (study_hard (X)) i jeśli nie moglibyśmy udowodnić, że X nie prowadzi badań, to zakładamy, że X studiuje. Często używamy tego podejścia w aproksymacji logiki predykatów podobnej do Prologa. Niestety, negacja jako porażka może nadmiernie ograniczyć naszą domenę interpretacji. Drugie podejście do problemu jest zgodne z problemem polega na przeprowadzeniu opartego na heurystyce i ograniczonego (ograniczonego czasowo lub pamięcią) poszukiwania prawdy predykatu, w naszym przykładzie study_hard(X), a następnie, jeśli nie ma przeciwnych dowodów, załóżmy, że to prawda, ze zrozumieniem, że być może będziemy musieli później wycofać wnioski absolwentów i wszystkie dalsze wnioski na ich podstawie. Możemy również wygenerować potencjalnie sprzeczne wyniki, używając jest zgodne z operatorem. Załóżmy, że ktoś, Piotr, jest dobrym uczniem, ale też bardzo lubi imprezy. Wtedy możemy mieć następujący zestaw predykatów opisujących sytuację:

$$\forall \, \mathrm{X} \; \mathrm{good\_student(X)} \wedge \mathrm{M} \; \mathrm{study\_hard(X)} \rightarrow \mathrm{graduates(X)}$$

$$\forall \, \mathrm{Y} \; \mathrm{party\_person(Y)} \wedge \mathrm{M} \; \mathrm{not(study\_hard(Y))} \rightarrow \mathrm{not(graduates(Y))}$$

$$\mathrm{good\_student(peter)}$$

$$\mathrm{party\_person(peter)}$$

Dzięki temu zestawowi klauzul, w których nie mamy dalszych informacji na temat zwyczajów Piotra związanych ze studiowaniem, niezależnie od tego, czy uczy się ciężko, czy nie, możemy wywnioskować, że Piotr ukończy szkołę i nie będzie jej! Jedną z metod rozumowania, która chroni przed takimi sprzecznymi wynikami, jest śledzenie powiązań zmiennych używanych z operatorem modalnym, z którymi jest zgodny. Tak więc, gdy Piotr był związany albo z predykatem study_hard lub not (study_hard), system zapobiegał wiązaniu Piotra z innym orzeczeniem. Inne niemonotoniczne układy są jeszcze bardziej konserwatywne i uniemożliwiają wyciąganie jakichkolwiek wniosków z takich potencjalnie sprzecznych zestawów klauzul. Możemy stworzyć kolejną anomalię:

$$\forall \, \mathrm{Y} \; \mathrm{very\_smart(Y)} \wedge \mathrm{M} \; \mathrm{not(study\_hard(Y))} \rightarrow \mathrm{not(study\_hard(Y))}$$

$$\forall \, \mathrm{X} \; \mathrm{not(very\_smart(X))} \wedge \mathrm{M} \; \mathrm{not(study\_hard(X))} \rightarrow \mathrm{not(study\_hard(X))}$$

Z tych klauzul możemy wywnioskować nową klauzulę:

$$\forall \, \mathrm{Z} \; \mathrm{M} \; \mathrm{nie} \; (\mathrm{study\_hard} \; (\mathrm{Z})) \rightarrow \mathrm{not} \; (\mathrm{study\_hard} \; (\mathrm{Z}))$$

Dalszy rozwój semantyki jest zgodny z operatorem adresowania takiego anomalnego rozumowania. Kolejnym rozszerzeniem jest logika autoepistemiczna. Innym niemonotonicznym systemem logicznym jest logika domyślna, stworzona przez Reitera. Logika domyślna wykorzystuje nowy zestaw reguł wnioskowania w postaci:

$$\mathrm{A} \; (\mathrm{Z}) \wedge \mathrm{:} \; \mathrm{B} \; (\mathrm{Z}) \rightarrow \mathrm{C} \; (\mathrm{Z})$$

co brzmi: Jeśli A (Z) jest możliwe do udowodnienia i jest zgodne z tym, co wiemy, zakładając B (Z), możemy wywnioskować C (Z). W tym momencie logika domyślna brzmi podobnie jak opisana właśnie niemonotoniczna logika McDermotta i Doyle'a. Istotną różnicą między nimi jest metoda rozumowania. W logice domyślnej te specjalne reguły wnioskowania są używane do wnioskowania o zestawach prawdopodobnych rozszerzeń oryginalnego zestawu aksjomatów / twierdzeń. Każde rozszerzenie jest tworzone przy użyciu jednej z domyślnych reguł wnioskowania logicznego na wiedzy reprezentowanej przez oryginalny zbiór aksjomatów / twierdzeń. W związku z tym naturalne byłoby posiadanie wielu wiarygodnych rozszerzeń oryginalnej bazy wiedzy. Można to zobaczyć w klauzulach dotyczących absolwentów:

$\forall$ X good_student(X) $\wedge$: study_hard(X) $\rightarrow$ graduates(X)

$\forall$ Y party(Y) $\wedge$: not(study_hard(Y)) $\rightarrow$ not(graduates(Y))

Każda klauzula może zostać użyta do stworzenia unikalnego, wiarygodnego rozszerzenia opartego na oryginalnym zestawie wiedzy. Logika domyślna pozwala następnie na przyjęcie dowolnego twierdzenia wywnioskowanego w prawdopodobnym rozszerzeniu jako aksjomat do dalszego rozumowania. Musi istnieć jakiś przewodnik podejmowania decyzji, aby ostatecznie określić, które rozszerzenie ma zostać użyte do dalszego rozwiązywania problemów. Logika domyślna nie mówi nic o tym, jak wybrać spośród możliwych prawdopodobnych rozszerzeń bazy wiedzy. Reiter, Criscuolo oraz Touretzky rozwijają te kwestie. Wreszcie, istnieje również niemonotoniczna sytuacja rozumowania stworzona przez wyszukiwanie dziedziczenia po reprezentacjach, w których obiekty mogą dziedziczyć od więcej niż jednego rodzica. Piotr, wspomniany wcześniej dobry student kochający imprezy, mógł odziedziczyć jeden zestaw cech z bycia dobrym uczniem, tj. Taki, który najprawdopodobniej ukończyłby. Piotr mógł również odziedziczyć inne, w tym przypadku częściowo sprzeczne, właściwości z bycia partyjnym, to znaczy, że nie ukończyłby studiów. Ważnym problemem, przed którym stają niemonotoniczne systemy rozumowania, jest zadanie skutecznej rewizji zestawu wniosków w świetle zmieniających się przekonań. Jeśli na przykład użyjemy predykatu r do wnioskowania s, to usunięcie r usuwa również wsparcie dla s, a także wszystkie inne wnioski, które wykorzystały s. O ile nie istnieje niezależny zestaw wniosków wspierających s, należy go wycofać. Wdrożenie tego procesu wycofania wymaga, w najgorszym przypadku, przeliczenia wszystkich wniosków za każdym razem, gdy zmienia się przekonanie. Przedstawione poniżej systemy utrzymania prawdy oferują mechanizmy utrzymania spójności plików bazy wiedzy.

9.1.2 Systemy utrzymania prawdy

Do ochrony logicznej spójności wniosków z systemu wnioskowania można zastosować system utrzymania prawdy (TMS). Jak wskazano w poprzedniej sekcji, konieczne jest przeliczenie poparcia dla pozycji w bazie wiedzy zawsze, gdy przekonania wyrażone przez zmienione klauzule bazy wiedzy. Systemy utrzymania prawdy rozwiązują tę kwestię, przechowując uzasadnienia dla każdego wniosku, a następnie ponownie rozważając poparcie dla wniosków w świetle nowych przekonań. Jednym ze sposobów spojrzenia na ten problem jest przejrzenie algorytmu cofania. Wycofywanie się jest systematyczną metodą badania wszystkich alternatyw dla punktów decyzyjnych w rozwiązywaniu problemów w oparciu o wyszukiwanie. Ważną wadą algorytmu cofania jest jednak sposób, w jaki systematycznie (i na ślepo) wycofuje się ze ślepych zaułków przestrzeni i szuka alternatyw dla swoich ostatnich wyborów. Takie podejście jest czasami nazywane cofaniem chronologicznym. Zapewniamy, że chronologiczne cofanie będzie systematycznie sprawdzać wszystkie alternatywy w przestrzeni; jednak sposób, w jaki to przebiega, jest czasochłonny, nieefektywny i na bardzo dużej przestrzeni bezużyteczny. To, czego naprawdę chcemy w wyszukiwaniu opartym na logice, to możliwość cofnięcia się bezpośrednio do punktu w przestrzeni, w którym występuje problem, i dostosowania rozwiązania w ten stan. To podejście jest nazywane wycofywaniem ukierunkowanym na zależności. Rozważmy przykład z rozumowania niemonotonicznego. Musimy dowiedzieć się o p, czego nie możemy bezpośrednio wywnioskować. Istnieje jednak prawdopodobne założenie q, które, jeśli jest prawdziwe, potwierdzi p. Więc zakładamy q i wyprowadzamy p. Nasze rozumowanie jest kontynuowane i na podstawie p kończymy r i s. Kontynuujemy nasze rozumowanie i kończymy bez poparcia p, r lub s wyniki t i u. Wreszcie udowadniamy, że nasze wcześniejsze założenie q jest fałszywe. Co mamy robić? Chronologiczne cofanie się wróciłoby do naszych kroków rozumowania w odwrotnej kolejności w której zostały wykonane. Cofanie się w kierunku zależności od razu wróciłoby do źródła sprzecznych informacji, a mianowicie do pierwszego założenia q. Wtedy będzie się poruszał, chowając p, r i s. Możemy w tym momencie sprawdzić, czy ri s można wyprowadzić niezależnie od p i q. To, że zostały

pierwotnie wyprodukowane z błędnym założeniem, nie oznacza, że nie są obsługiwane w inny sposób. Wreszcie, ponieważ t i u zostały wyprowadzone bez p, r lub s, nie musielibyśmy ich ponownie rozważać. Aby zastosować w systemie wnioskowania wycofywanie się w kierunku zależności, musimy:

1. Powiązać z przedstawieniem każdego wniosku jego uzasadnienie. To uzasadnienie wskazuje na proces wyprowadzania tego wniosku. Uzasadnienie musi zawierać wszystkie fakty, zasady i założenia użyte do sformułowania wniosku.

2. Zapewnić mechanizm, który po przedstawieniu sprzeczności wraz z jej uzasadnieniem znajdzie w uzasadnieniu zbiór fałszywych założeń, które doprowadziły do sprzeczności.

3. Wycofać fałszywe założenia.

4. Stworzyć mechanizm, który podąża za wycofanymi założeniami i wycofuje wszelkie wnioski, które wykorzystują w swoich uzasadnieniach wycofane fałszywe założenie.

Oczywiście wszystkie wycofane wnioski niekoniecznie są fałszywe, więc należy je ponownie sprawdzić, aby sprawdzić, czy można je uzasadnić niezależnie od wycofanych klauzul. Następnie przedstawimy dwie metody budowania systemów wstecznego śledzenia ukierunkowanego na zależności. Jon Doyle stworzył jeden z najwcześniejszych systemów utrzymania prawdy, zwany systemem utrzymania prawdy opartym na uzasadnieniu lub JTMS. Doyle był pierwszym badaczem, który wyraźnie oddzielił system utrzymywania prawdy, sieć zdań i ich uzasadnień, od systemu rozumowania działającego w jakiejś dziedzinie. Rezultatem tego podziału jest to, że JTMS komunikuje się z osobą rozwiązującą problem, być może automatyczną weryfikacją twierdzeń, otrzymując informacje o nowych zdaniach i uzasadnieniach, a z kolei dostarczając rozwiązującemu problem informacje o tym, w które zdania należy wierzyć na podstawie aktualnie istniejących uzasadnień. Istnieją trzy główne operacje, które są wykonywane przez JTMS. Po pierwsze, JTMS bada sieć uzasadnień. Ta inspekcja może być wywołana przez zapytania od osoby rozwiązującej problem, takie jak: Czy mam wierzyć w propozycję p? Dlaczego mam wierzyć twierdzeniu p? Jakie założenia leżą u podstaw zdania p? Drugą operacją JTMS jest modyfikacja sieci zależności, w której modyfikacje są napędzane informacjami dostarczanymi przez rozwiązującego problem. Modyfikacje obejmują dodawanie nowych propozycji, dodawanie lub usuwanie przesłanek, dodawanie sprzeczności i uzasadnianie wiary w zdanie. Ostatnim działaniem JTMS jest aktualizacja sieci. Ta operacja jest wykonywana za każdym razem, gdy wprowadzana jest zmiana w sieci zależności. Operacja aktualizacji przelicza etykiety wszystkich propozycji w sposób zgodny z istniejącymi uzasadnieniami. Aby zademonstrować JTMS, tworzymy prostą sieć zależności. Rozważmy operator modalny M przedstawiony w sekcji 9.1.1, który znajduje się przed odczytaniem predykatu jako zgodny z. Na przykład:

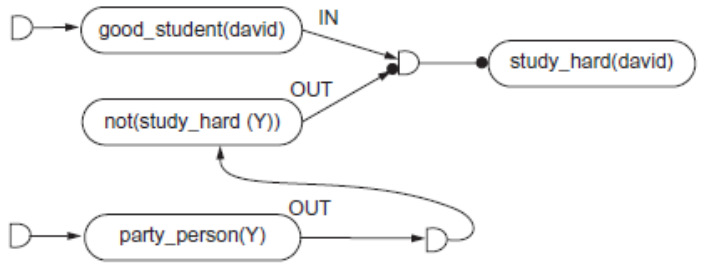
$$\forall \, \mathrm{X} \; \mathrm{good\_student(X)} \wedge \mathrm{M} \; \mathrm{study\_hard(X)} \rightarrow \mathrm{study\_hard(X)}$$

$$\forall \, \mathrm{Y} \; \mathrm{party\_person(Y)} \rightarrow \mathrm{not} \; (\mathrm{study\_hard(Y)})$$

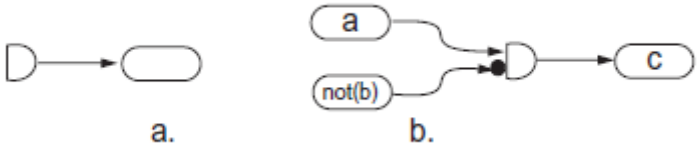
$$\mathrm{good\_student(david)}$$

Teraz tworzymy z tego zestawu twierdzeń sieć uzasadnienia.

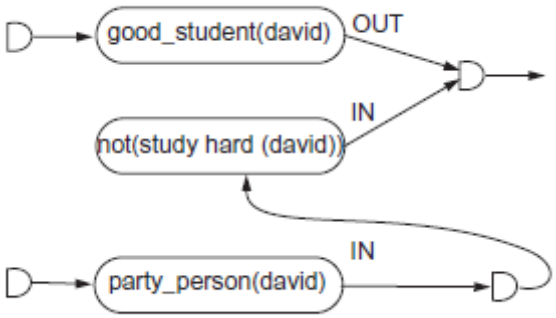
W JTMS każdy predykat reprezentujący przekonanie jest powiązany z dwoma innymi zestawami przekonań. Pierwszy zbiór, oznaczony jako IN na rysunku 1, jest zbiorem zdań, w które należy wierzyć, aby zdanie się zachowało.



Drugie, oznaczone jako OUT, są propozycjami, w które nie należy wierzyć, aby były one aktualne. Rysunek 1 przedstawia uzasadnienie, które wspiera study_hard (david) wyprowadzone z wcześniej wymienionych predykatów. Notacje z rysunku 1 wyjaśniono na rysunku 2.

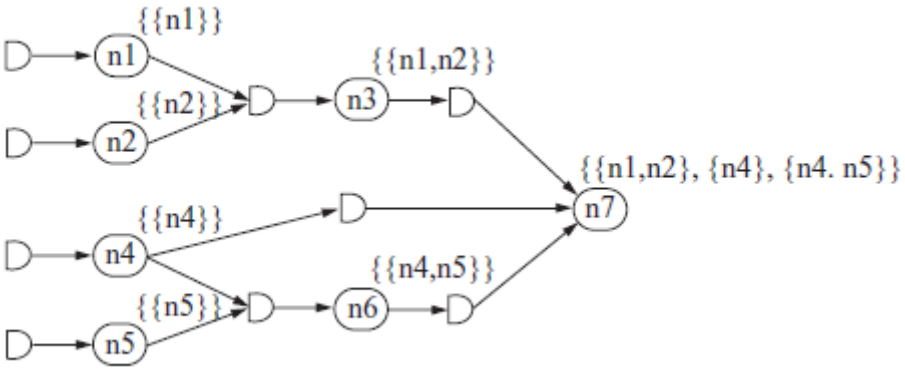


Przesłanki uzasadnień są oznaczone jak na rysunku 2(a), a kombinacje zdań, które wspierają wniosek, są oznaczone jak na rysunku 2(b). Dzięki informacjom z sieci przedstawionym na rysunku 1 osoba rozwiązująca problem może wnioskować, że metoda study_hard (david) jest obsługiwana, ponieważ założenie good_student (david) jest uważane za prawdziwe i jest zgodne z faktem, że dobrzy uczniowie ciężko się uczą. W tym przykładzie nie ma również żadnych dowodów ani innych wskazówek, że David nie uczy się ciężko. Załóżmy, że dodamy założenie party_person (david). To dodanie umożliwia wyprowadzenie nie (study_hard (david)), a przekonanie study_hard (david) nie jest już obsługiwane. Uzasadnienia takiej sytuacji przedstawiono na rysunku 3;

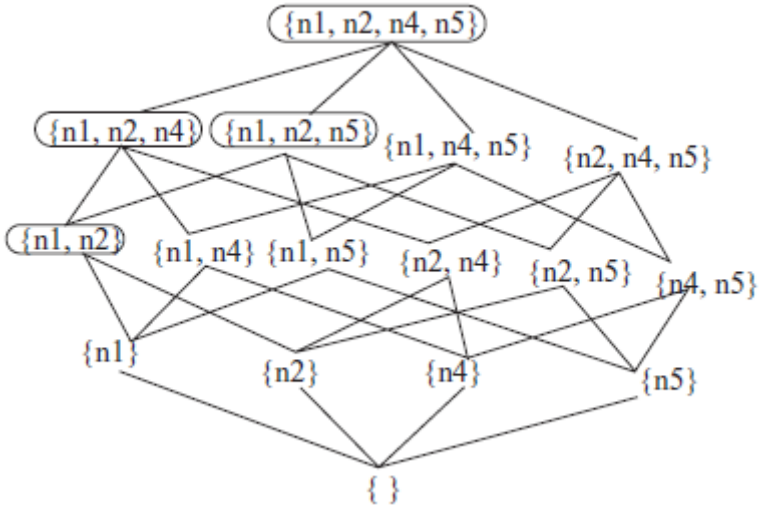


zwróć uwagę na zmianę etykiety IN i OUT. Jak pokazują rysunki 1 i 3, JTMS nie reprezentuje bezpośrednio relacji predykatów wyrażonych w pierwotnym zbiorze zdań. JTMS jest raczej prostą siecią, która bierze pod uwagę tylko relacje między zdaniami atomowymi i ich negacją i organizuje je w związki wspierające przekonania. Pełny zestaw łączników predykatów i schematów wnioskowania (∀X, ∧, ∨, → itp.) Jest używany w samym rozwiązaniu problemu. Systemy McAllestera oraz Martinsa i Shapiro połączyły TMS i narzędzie do rozwiązywania problemów w jedną reprezentację. JTMS zajmuje się tylko zależnościami między przekonaniami i nie zajmuje się treścią tych przekonań. Dlatego przekonania możemy zastąpić identyfikatorami, często w postaci n_1, n_2, ..., które są skojarzone z obiektami w sieci zwanymi węzłami. Następnie algebra IN i OUT, którą widzieliśmy zaimplementowaną w przykładzie study_hard, umożliwia JTMS wnioskowanie o poparciu dla przekonań. Podsumowując,

JTMS działa z zestawami węzłów i uzasadnień. Węzły oznaczają przekonania, a uzasadnienia wspierają przekonania w węzłach. Z węzłami powiązane są etykiety IN i OUT, które wskazują stan przekonań skojarzonego węzła. Możemy wnioskować o wsparciu dla dowolnego węzła, odnosząc go do IN i OUT innych węzłów, które stanowią jego uzasadnienie (a). Podstawowymi operacjami algebry JTMS jest przeprowadzanie inspekcji, modyfikacji i aktualizacji operatorów wymienionych powyżej. Wreszcie, ponieważ sprawdzanie uzasadnienia jest wymuszane przez tworzenie kopii zapasowych na linkach samej sieci justowania, mamy przykład cofania opartego na zależnościach. Więcej informacji na temat tego podejścia do JTMS można znaleźć w Doyle lub Reinfranka . Drugim typem systemu utrzymania prawdy jest system utrzymania prawdy oparty na założeniach (ATMS). Termin oparty na założeniach został po raz pierwszy wprowadzony przez deKleera, chociaż podobne pomysły można znaleźć u Martinsa i Shapiro (1983). W tych systemach etykiety węzłów w sieci nie są już IN i OUT, ale raczej zestawami przesłanek (założeń) leżących u podstaw ich wyprowadzenia. deKleer dokonuje również rozróżnienia między węzłami przesłanek, które są uniwersalne, a węzłami, które mogą być założeniami poczynionymi przez osobę rozwiązującą problem, a które później mogą zostać wycofane. Przewaga ATMS nad JTMS wynika z dodatkowej elastyczności, jaką zapewnia ATMS w radzeniu sobie z wieloma możliwymi stanami przekonań. Oznaczając przekonania zbiorami przesłanek, w których się trzymają, nie ma już jednego stanu przekonania (w JTMS wszystkie węzły oznaczone jako IN), ale raczej szereg możliwych stanów, zbiór wszystkich podzbiorów przesłanek wspierających. Stworzenie różnych zestawów przekonań lub możliwych światów pozwala na porównanie wyników różnych wyborów przesłanek, istnienie różnych rozwiązań problemu oraz wykrycie i wyleczenie ze sprzeczności. Wady ATMS obejmują brak możliwości reprezentowania zestawów założeń, które same w sobie są niemonotoniczne oraz kontrolę nad rozwiązaniem problemu. Komunikacja między ATMS a osobą rozwiązującą problemy jest podobna do komunikacji między JTMS a jego osobą rozwiązującą problemy z operatorami w celu kontroli, modyfikacji i aktualizacji. Jedyna różnica polega na tym, że w przypadku ATMS nie ma już jednego stanu wiary, ale raczej podzbiory potencjalnych przesłanek wspierających. Celem obliczeń w ATMS jest znalezienie minimalnych zestawów przesłanek wystarczających do obsługi każdego węzła. Obliczenia te są wykonywane przez propagowanie i łączenie etykiet, zaczynając od etykiet dla pomieszczeń. Następnie przedstawiamy szczegółowy przykład zaadaptowany z Martins. Załóżmy, że mamy sieć ATMS z rysunku 4.



W tej sieci n1, n2, n4 i n5 są przesłankami i przyjmuje się, że są prawdziwe. Sieć zależności odzwierciedla również relacje, które z przesłanek n1 i n2 obsługujemy n3, z n3 obsługujemy n7, z n4 obsługujemy n7, z n4 i n5 obsługujemy n6, a na koniec z n6 obsługujemy n7. Rysunek 5 przedstawia siatkę podzbioru / superzbioru dla zależności przesłanek znalezionych na rysunku 4.



Ta krata podzbiorów lokali oferuje przydatny sposób wizualizacji przestrzeni kombinacji pomieszczeń. Tak więc, jeśli okaże się, że jakaś przesłanka jest podejrzana, ATMS będzie w stanie określić, w jaki sposób ta przesłanka odnosi się do innych podzbiorów obsługi pomieszczeń. Na przykład węzeł n3 na rysunku 4 będzie obsługiwany przez wszystkie zestawy przesłanek, które znajdują się powyżej {n1, n2} w kracie na rysunku 5. Rozumujący ATMS usuwa sprzeczności, usuwając z węzłów te zestawy przesłanek, które okazały się niespójne. Załóżmy na przykład, że zrewidujemy poparcie dla rozumowania przedstawionego na rysunku 4, aby uczynić n3 węzłem sprzeczności. Ponieważ etykietą dla n3 jest {n1, n2}, ten zbiór przesłanek jest określony jako niespójny. Kiedy ta niespójność zostanie wykryta, wszystkie zbiory przesłanek, które są w relacji nadzbiór do {n1, n2} na rysunku 5, są oznaczane jako niespójne (zakreśliliśmy je) i usuwane z sieci zależności. W tej sytuacji jedna z możliwych etykiet obsługujących n7 również będzie musiała zostać usunięta. Pełny opis algorytmu usuwania sprzeczności można znaleźć w deKleer. Istnieje kilka innych ważnych przyczynków do rozumowania TMS. TMS oparty na logice jest oparty na pracy McAllestera . W LTMS relacje między zdaniami są reprezentowane przez klauzule, które można wykorzystać do wyprowadzenia wartości prawdziwości dowolnego z opisywanych przez nie zdań. Inne podejście, wielokrotny rozumowanie przekonań (MBR) jest podobne do rozumowania ATMS, z tym wyjątkiem, że rozwiązanie problemu i system utrzymania prawdy są połączone w jeden system. MBR jest oparty na języku logicznym zwanym SWM , który opisuje stany wiedzy. Każdy stan wiedzy składa się z pary deskryptorów, z których pierwszy odzwierciedla bazę wiedzy, a drugi zbiór zestawów znanych niespójnych przesłanek w bazie wiedzy. Algorytmy sprawdzania niespójności podczas rozumowania można znaleźć w Martins (1991). Dalszą dyskusję na temat systemów utrzymania prawdy można znaleźć w The Encyclopedia of Artificial Intelligence

9.1.3 Logika oparta na modelach minimalnych

W poprzednich sekcjach rozszerzyliśmy logikę o kilka różnych operatorów modalnych, które zostały specjalnie zaprojektowane, aby rozumieć świat taki, jaki jest zwykle, łagodząc wymóg, aby nasza wiedza o świecie była w jakiś sposób kompletna. Operatorzy ci zrodzili się z konieczności stworzenia bardziej elastycznego i zmiennego spojrzenia na świat. W tej sekcji przedstawiamy logikę zaprojektowaną specjalnie dla dwóch sytuacji: po pierwsze, aby uzasadnić, w którym zbiór twierdzeń określa tylko te rzeczy, które są prawdziwe, a po drugie, aby uzasadnić, w przypadku których ze względu na naturę zadania rozwiązywania problemów zbiory przypuszczeń są zwykle prawdziwe. W pierwszej sytuacji posługujemy się założeniem świata zamkniętego, w drugiej okręgach. Oba te podejścia do logiki są często nazywane rozumowaniem na podstawie modeli minimalnych. Widzieliśmy, że model jest interpretacją spełniającą S, zbiór wyrażeń predykatów, dla wszystkich

przypisań zmiennych. Istnieje wiele sposobów zdefiniowania, co oznacza model minimum. Definiujemy model minimum jako model tak, że nie ma mniejszych modeli, które mogą spełnić zestaw wyrażeń S dla wszystkich przypisań zmiennych. Idea, która sprawia, że modele minimalne są ważne dla rozumowania, jest taka: istnieje (potencjalnie) nieskończona liczba predykatów, których można użyć do opisania sytuacji na świecie. Rozważmy na przykład nieograniczone predykaty, które można wykorzystać do opisania sytuacji problemu farmer-wilk-koza-kapusta: łódź nie tonie powoli, brzegi rzeki są na tyle blisko, że wiosłowanie będzie przepłynąć łodzią, wiatr i prąd nie są istotnymi czynnikami i tak dalej. Kiedy opisujemy problem, jesteśmy zwykle dość oszczędni w naszych opisach. Tworzymy tylko te predykaty, które są zarówno istotne, jak i potrzebne do rozwiązania problemu. Założenie o zamkniętym świecie opiera się na tym minimalnym modelu świata. Tworzone są dokładnie te predykaty, które są niezbędne do rozwiązania. Założenie o zamkniętym świecie wpływa na semantykę negacji w rozumowaniu. Na przykład, gdybyśmy chcieli ustalić, czy uczeń jest zarejestrowanym członkiem klasy, moglibyśmy przejść do bazy danych rejestracji, a jeśli uczeń nie jest wyraźnie wymieniony w tej bazie danych (model minimalny), nie byłby zarejestrowany. Podobnie, gdybyśmy chcieli wiedzieć, czy dwa miasta są bezpośrednio połączone samolotem, przeszlibyśmy do listy wszystkich połączeń lotniczych. Gdybyśmy nie wyszczególnili lotu bezpośredniego (model minimalny), to wywnioskowalibyśmy, że nie istnieje. Założenie o zamkniętym świecie to stwierdzenie, że jeśli nasz system obliczeniowy nie może stwierdzić, że p (X) jest prawdziwe, to nie (p (X)) musi być prawdziwe. Jak zobaczymy, założenie o zamkniętym świecie wspiera wnioskowanie w Prologu. Zobaczymy trzy założenia (aksjomaty) implikowane w stosowaniu modeli minimalnych. Te aksjomaty są unikalną nazwą, tj. Wszystkie atomy o różnych nazwach są różne; zamknięty świat, tj. jedynymi przykładami relacji są te, które są implikowane przez obecne klauzule; i zamknięcie domeny. tj. atomy domeny są dokładnie tymi z modelu. Gdy te trzy elementy są spełnione, model minimalny staje się specyfikacją opartą w pełni na logice. Jeśli aksjomaty nie są spełnione, wymagana jest jakaś forma algorytmu utrzymania prawdy. Jeśli świat zamknięty wymaga, aby wszystkie predykaty składające się na model zostały określone, opis okrążenia wymaga podania tylko tych predykatów, które są istotne dla rozwiązania problemu. W opisie aksjomaty są dodawane do systemu, który wymusza minimalną interpretację predykatów bazy wiedzy. Te metapredykaty (predykaty dotyczące predykatów stwierdzenia problemu) opisują sposób, w jaki mają być interpretowane poszczególne predykaty. Oznacza to, że ograniczają lub zawężają możliwe interpretacje orzeczników. McCarthy przedstawił ideę okręgów w eksperymencie myślowym dotyczącym problemu misjonarzy i kanibali. Opis problemu prosi osobę rozwiązującą o wymyślenie serii ruchów, w których sześć postaci, pod pewnymi warunkami, może użyć łodzi do przepłynięcia rzeki. McCarthy przywołuje wiele absurdalnych sytuacji, które całkiem słusznie można zapytać o stwierdzenie problemu. Wiele z nich, takich jak wolno tonąca łódź lub czynnik wiatru, zostało przedstawionych wcześniej w tym rozdziale. Chociaż ludzie uważają takie sytuacje za absurdalne, rozumowanie, którego używamy, nie jest oczywiste. Aksjomaty opisujące problem, które McCarthy dodałby do specyfikacji problemu, precyzyjnie ograniczyłyby predykaty opisujące problem. Jako inny przykład okólnika rozważ wyrażenie predykatu ze specyfikacji rozumowania zorientowanego obiektowo, opartego na zdrowym rozsądku;

$$\forall \, \mathrm{X} \; \mathrm{bird(X)} \wedge \mathrm{not} \; (\mathrm{abnormal(X)}) \rightarrow \mathrm{flies(X)}$$

To wyrażenie może pojawić się w rozumowaniu, w którym jedną z właściwości ptaków są muchy. Ale co mogłoby ograniczyć definicję predykatu anormalnego? Że ptak nie jest pingwinem, że nie ma złamanego skrzydła, że nie jest martwy? Specyfikacja predykatu anormalnego jest potencjalnie nierozstrzygalna. Circumscription wykorzystuje schemat aksjomatów lub zestaw reguł meta w ramach rachunku predykatów pierwszego rzędu do generowania predykatów dla dziedziny problemowej. Reguły schematu powodują, że niektóre formuły mają najmniejsze możliwe rozszerzenia. Na przykład, jeśli B jest systemem przekonań obejmującym wiedzę o świecie K i wiedzę dziedzinową A(p) o

predykacie p, to możemy uznać, że p jest zminimalizowane, ponieważ jak najmniej atomów a_i spełnia $p(a_i)$ zgodnie z wymogami z A(p) i K. Wiedza o świecie K wraz z A (p) i schematem opisowym są używane do wyciągania wniosków w standardowym rachunku predykatów pierwszego rzędu. Te wnioski są następnie dodawane do B, systemu przekonań. Jako przykład załóżmy, że w świecie bloków , mamy wyrażenie:

isblock (a) ∧ isblock (b) ∧ isblock (c)

twierdząc, że a, b i c są blokami. Opisanie predykatu isblock daje:

∀X (isblock (X) ← ((X = a) v (X = b) v (X = c)))

To wyrażenie stwierdza, że tylko bloki a, b i c, tj. Tylko te obiekty, których wymaga predykat isblock, są blokami w tej domenie. Podobnie predykat:

isblock (A) v isblock (B)

można ograniczyć do:

∀X (isblock (X) ← ((X = a) v (X = b)))

Aby uzyskać szczegółowe informacje, w tym aksjomaty schematu użyte do wyprowadzenia tych wyników. Circumscription, gdy jest używany z operatorami takimi jak anormal, jest bardzo podobny do założenia zamkniętego świata, ponieważ wytwarza dokładnie te zmienne wiązania, które może obsługiwać anormalność. Jednak algebra opisowa pozwala nam rozszerzyć to rozumowanie na reprezentacje predykatów, ponieważ, jak właśnie zauważyliśmy, jeśli mamy predykat p (X) v q (X), możemy opisać albo orzeczenie p lub q, albo jedno i drugie. Tak więc, w przeciwieństwie do założenia o zamkniętym świecie, opis obwodu pozwala nam opisać instancje możliwe w zbiorach opisy predykatów. Lifschitz wniósł ważny wkład, proponując punktowy opis, w którym model minimum może być przeprowadzony dla poszczególnych predykatów i ich możliwych instancji, a nie dla całej dziedziny. Innym ważnym wkładem jest Perlis (1988), gdzie rozumowanie może dotyczyć braku wiedzy określonego agenta.

9.1.4 Ustawianie osłony i odwodzenie oparte na logice

Jak zauważono na początku, w rozumowaniu abdukcyjnym mamy reguły postaci p → q, wraz z rozsądną wiarą w q. Chcemy więc przedstawić argumenty za prawdziwością orzeczenia p. Rozumowanie abdukcyjne nie jest rozsądne, ale to, co często nazywa się rozumowaniem, stanowi najlepsze wyjaśnienie obecności danych q. W tej sekcji przyjrzymy się dokładniej generowaniu wyjaśnień w domenach wnioskowania abdukcyjnego. Oprócz przedstawionych już opisów rozumowania abdukcyjnego, naukowcy zajmujący się sztuczną inteligencją wykorzystali również zestaw pokrycia i analizy wspomagane logiką. Zestaw obejmuje podejście do uprowadzenia , próbuje wyjaśnić akt przyjęcia odwołalnej wiary w jakąś hipotezę wyjaśniającą na tej podstawie, że wyjaśnia ona niewytłumaczalny zestaw faktów. Oparte na logice podejście do wzięcia opisuje reguły wnioskowania dla wzięcia wraz z definicją ich legalnych formularzy do użytku. Podejście typu set cover definiuje abdukcyjne wyjaśnienie jako pokrycie predykatów opisujących obserwacje predykatami opisującymi hipotezy. Reggia i inni opisują pokrycie oparte na binarnej relacji przyczynowej R, gdzie R jest podzbiorem {Hypotheses X Observations}. Zatem abdukcyjne wyjaśnienie zbioru obserwacji S2 jest kolejnym zestawem hipotez S1 wystarczającym do wywołania S2. Optymalnym wyjaśnieniem zgodnie z podejściem zestawu pokrycia jest minimalne pokrycie zestawu S2. Wadą tego podejścia jest to, że sprowadza ono wyjaśnienie do prostej listy hipotez przyczynowych (z S1). W sytuacjach, w których istnieją wzajemnie powiązane lub oddziałujące przyczyny lub gdy wymagane jest zrozumienie struktury

lub sekwencjonowania interakcji przyczynowych, ustalony model pokrycia jest nieodpowiedni. Z drugiej strony, podejście do uprowadzenia oparte na logice opiera się na bardziej wyrafinowanej koncepcji wyjaśnienia. Levesque (1989) definiuje abdukcyjne wyjaśnienie jakiegoś wcześniej niewyjaśnionego zbioru obserwacji O jako minimalny zestaw hipotez H zgodny z podstawową wiedzą agenta K. Hipotezy H wraz z wiedzą podstawową K muszą pociągać za sobą O. Bardziej formalnie:

abduce (K, O) = H, wtedy i tylko wtedy, gdy

1. K nie pociąga za sobą O

2. H ∪ K pociąga za sobą O

3. H ∪ K jest zgodne, i

4. Żaden podzbiór H nie ma właściwości 1, 2 i 3.

Należy zauważyć, że ogólnie może istnieć wiele zestawów hipotez; to znaczy, może istnieć wiele potencjalnych abdukcyjnych zestawów wyjaśnień dla danego zbioru obserwacji O. Oparta na logice definicja abdukcyjnego wyjaśnienia sugeruje odpowiedni mechanizm odkrywania wyjaśnień w kontekście systemu opartego na wiedzy. Jeśli hipotezy wyjaśniające muszą pociągać za sobą obserwacje O, to sposób na skonstruowanie całości wyjaśnieniem jest wnioskowanie wstecz od O. Możemy zacząć od łącznych składników O i rozumować z powrotem od następników do ich poprzedników. To podejście z łańcuchem wstecznym wydaje się również naturalne, ponieważ warunki warunkowe, które ją wspierają, można łatwo uznać za prawa przyczynowe, co pozwala uchwycić kluczową rolę, jaką wiedza przyczynowa odgrywa w konstruowaniu wyjaśnień. Model jest również wygodny, ponieważ ładnie pasuje do czegoś, z czym już społeczność AI ma doświadczenie: backchaining i modele obliczeniowe do dedukcji. Istnieją również sprytne sposoby na znalezienie pełnego zestawu uprowadzających wyjaśnień. Systemy podtrzymywania prawdy oparte na założeniach ATMS zawierają algorytm obliczania minimalnych zbiorów podpór, czyli zbioru (nieaksjomatów) zdań, które logicznie pociągają za sobą dane zdanie w teorii. Aby znaleźć wszystko, co możliwe dla abdukcyjnych wyjaśnień dla zbioru obserwacji, po prostu bierzemy iloczyn kartezjański przez zbiory podpór. Tak proste, precyzyjne i wygodne, jak oparte na logice ujęcie wzięcia, istnieją dwa powiązane wady: duża złożoność obliczeniowa i słabość semantyczna. Selman i Levesque stwierdzili, że złożoność zadań związanych z porwaniami jest podobna do złożoności obliczeniowych zestawów pomocniczych dla ATMS. Standardowy dowód na to, że problem ATMS jest NP-trudny, zależy od istnienia instancji problemowych z wykładniczą liczbą rozwiązań. Selman i Levesque unikają liczby potencjalnych problemów ze złożonością rozwiązań, pytając, czy znalezienie mniejszego zestawu rozwiązań jest również NP-trudne. Mając bazę wiedzy dotyczącą klauzuli Horna, Selman i Levesque tworzą algorytm, który znajduje jedno wyjaśnienie w kolejności O (k * n), gdzie k wskazuje liczbę zmiennych zdaniowych, a n liczbę wystąpień literałów. Jednak gdy nakłada się ograniczenia na rodzaje poszukiwanych wyjaśnień, problem ponownie staje się NP-trudny, nawet w przypadku klauzul Horna. Jednym z interesujących wyników analizy Selmana i Levesque'a jest fakt, że dodanie pewnych rodzajów celów lub ograniczeń do zadania uprowadzenia w rzeczywistości znacznie utrudnia obliczenia. Z naiwnego punktu widzenia człowieka rozwiązującego problemy ta dodatkowa złożoność jest zaskakująca: człowiek zakłada, że dodanie dalszych ograniczeń w poszukiwaniu odpowiednich wyjaśnień ułatwia zadanie. Powodem, dla którego zadanie porwania jest trudniejsze w modelu opartym na logice, jest to, że wnosi tylko dodatkowe klauzule do problemu, a nie dodatkową strukturę przydatną do wyprowadzenia rozwiązania. Odnajdywanie wyjaśnień w modelu opartym na logice jest scharakteryzowane jako zadanie znalezienia zestawu hipotez o określonych właściwościach logicznych. Te właściwości, w tym zgodność z podstawową wiedzą i wynikanie z tego, co ma być wyjaśnione, mają na celu uchwycenie niezbędnych warunków wyjaśnień: minimalnych warunków, które musi spełnić

zestaw hipotez wyjaśniających, aby można je było uznać za wyjaśnienie uprowadzające. Zwolennicy tego podejścia uważają, że dodając dodatkowe ograniczenia, podejście można rozszerzyć, aby zapewnić scharakteryzowanie dobrych lub rozsądnych wyjaśnień. Jedną z prostych strategii tworzenia jakościowych wyjaśnień jest zdefiniowanie zestawu klauzul faktów, które są dające się porwać, to znaczy z których należy wybierać potencjalne hipotezy. Ten zestaw klauzul pozwala zawczasu ograniczyć wyszukiwanie do tych czynników, które mogą potencjalnie odgrywać rolę przyczynową w wybranej dziedzinie. Inną strategią jest dodanie kryteriów wyboru do oceny i wyboru między wyjaśnieniami. Zaproponowano różne kryteria selekcji, w tym minimalność zbioru, która preferuje jedną hipotezę nad inną, gdzie obie są spójne i pociągają za sobą to, co należy wyjaśnić, jeśli pierwsza jest zawarta w drugiej. Kryterium prostoty daje pierwszeństwo oszczędnym zestawom hipotez, zawierających mniej niezweryfikowanych założeń . Zarówno minimalność, jak i prostota mogą być postrzegane jako zastosowania maszynki do golenia Ockhama. Niestety, minimalność zbioru ma ograniczoną moc jako narzędzie do przycinania wyszukiwania; eliminuje jedynie ostateczne wyjaśnienia, które są nadzbiorem istniejących wyjaśnień. Sama prostota jest również wątpliwa jako kryterium wyboru wyszukiwania. Nie jest trudno skonstruować przykłady, w których wyjaśnienie wymagające szerszego zestawu hipotez jest lepsze od prostszego, ale płytszego zestawu hipotez. Rzeczywiście, złożone mechanizmy przyczynowe będą zwykle wymagały większych zestawów hipotez; jednakże uprowadzenie takich mechanizmów przyczynowych może być dobrze uzasadnione, zwłaszcza gdy obecność pewnych kluczowych elementów tego mechanizmu została już zweryfikowana przez obserwację. Ciekawe są również dwa inne mechanizmy doboru wyjaśnień, ponieważ uwzględniają zarówno właściwości zestawu hipotez, jak i właściwości procedury dowodowej. Po pierwsze, uprowadzenie oparte na kosztach nakłada koszt na potencjalne hipotezy, a także na koszt reguł. Całkowity koszt wyjaśnienia oblicza się na podstawie całkowitego kosztu hipotez powiększonego o koszt reguł zastosowanych do zniesienia hipotez. Następnie porównuje się konkurencyjne zestawy hipotez pod względem kosztów. Jedną naturalną heurystyką, którą można dołączyć do tego schematu, jest heurystyka probabilistyczna. Wyższe koszty hipotez reprezentują mniej prawdopodobne zdarzenia; wyższe koszty reguł stanowią mniej prawdopodobne mechanizmy przyczynowe. Metryki oparte na kosztach można łączyć z algorytmami wyszukiwania o najniższych kosztach, takimi jak wyszukiwanie best-first, , znacznie zmniejszając złożoność obliczeniową zadania. Drugi mechanizm, selekcja oparta na koherencji, jest szczególnie interesująca, gdy to, co ma być wyjaśnione, nie jest prostym zdaniem, ale raczej zbiorem zdań. Ng i Mooney argumentowali, że miara spójności jest lepsza niż miara prostoty przy wyborze wyjaśnień w analizie tekstu w języku naturalnym. Definiują spójność jako właściwość wykresu dowodowego, w którym wyjaśnienia z większą liczbą powiązań między dowolną parą obserwacji i mniejszą liczbą rozłącznych podziałów są bardziej spójne. Kryterium spójności opiera się na heurystycznym założeniu, że to, o co nas proszono, to pojedyncze zdarzenie lub działanie o wielu aspektach. Uzasadnienie dla metryki koherencji w rozumieniu języka naturalnego opiera się na Griceańskich warunkach szczęścia, to znaczy na zobowiązaniu mówcy do bycia spójnym i trafnym. Nie jest trudno rozszerzyć ich argumentację na wiele innych sytuacji. Na przykład podczas diagnozy obserwacje, które obejmują początkowy zestaw rzeczy do wyjaśnienia, są łączone, ponieważ uważa się, że są one związane z tym samym podstawowym mechanizmem błędu lub niepowodzenia. W sekcji 9.1 rozważaliśmy rozszerzenia tradycyjnej logiki, które wspierały rozumowanie przy użyciu niepewnych lub brakujących danych. Następnie opiszemy nielogiczne alternatywy dla rozumowania w sytuacjach niepewności, w tym algebrę czynników pewności Stanforda, wnioskowanie ze zbiorami rozmytymi oraz teorię dowodów Dempstera-Shafera.

9.2 Abdukcja: alternatywa dla logiki

Podejścia oparte na logice opisane w sekcji 9.1 są uciążliwe i trudne do wykonania obliczeniowo w przypadku wielu zastosowań, zwłaszcza systemów eksperckich. Alternatywnie, kilka wczesnych

projektów systemów eksperckich, np. PROSPECTOR, próbowało zaadaptować techniki bayesowskie, do wnioskowania abdukcyjnego. Założenia dotyczące niezależności, ciągłe aktualizacje danych statystycznych oraz obliczenia wymagane do obsługi algorytmów wnioskowania stochastycznego ograniczają to podejście. Alternatywny mechanizm wnioskowania, zaprojektowany w celu złagodzenia tych ograniczeń, został użyty w Stanford do opracowania wczesnych systemów eksperckich, w tym MYCIN. Rozumując z wykorzystaniem wiedzy heurystycznej, ludzcy eksperci są w stanie podać adekwatne i użyteczne szacunki ich zaufania do relacji reguł. Ludzie ważą wnioski takimi terminami, jak wysoce prawdopodobne, mało prawdopodobne, prawie na pewno lub możliwe. Te wagi oczywiście nie są oparte na dokładnej analizie prawdopodobieństw. Zamiast tego są one heurystykami wywodzącymi się z doświadczenia w rozumowaniu w dziedzinie problemowej. W sekcji 9.2 wprowadzimy trzy metodologie wnioskowania abdukcyjnego: teorię pewności Stanforda, rozumowanie rozmyte i teorię dowodów Dempstera-Shafera. W sekcji 9.3 przedstawiamy stochastyczne podejście do niepewności.

9.2.1 Algebra współczynnika pewności Stanforda

Teoria pewności Stanforda opiera się na szeregu obserwacji. Po pierwsze, w tradycyjnej teorii prawdopodobieństwa suma zaufania dla związku i ufności względem tego samego związku musi się sumować. Jednak często zdarza się, że ekspert-człowiek może mieć pewność 0,7 (z 1,0), że jakiś związek jest prawdziwy i nie ma w ogóle poczucia, że jest nieprawdziwy. Kolejnym założeniem leżącym u podstaw teorii pewności jest to, że wiedza zawarta w regułach jest znacznie ważniejsza niż algebra przy obliczaniu ufności. Miary zaufania odpowiadają nieformalnym ocenom, które ludzcy eksperci dołączają do swoich wniosków, takich jak „to prawdopodobnie prawda", „to prawie na pewno prawda" lub „jest wysoce nieprawdopodobne". Teoria pewności ze Stanforda przyjmuje pewne proste założenia do tworzenia miar zaufania i ma kilka równie prostych reguł łączenia tych ufności, gdy program zmierza w kierunku zakończenia. Pierwszym założeniem jest rozdzielenie „zaufania dla" od „zaufania przeciwko" relacji:

Nazwij MB (H|E) miarą wiary w hipotezę H danym dowodem E.

Nazwij MD (H|E) miarą niedowierzania hipotezy H podanym dowodem E.

Teraz albo:

1> MB (H|E)> 0, podczas gdy MD (H|E) = 0 lub

1> MD (H|E)> 0, podczas gdy MB (H|E) = 0.

Te dwie miary ograniczają się wzajemnie, ponieważ dany dowód jest albo za, albo przeciw określonej hipotezie, co jest ważną różnicą między teorią pewności a teorią prawdopodobieństwa. Kiedy już zostanie ustalony związek między miarami wiary i niewiary, można je ponownie powiązać poprzez:

CF (H|E) = MB (H|E) - MD (H|E).

Gdy współczynnik pewności (CF) zbliża się do 1, dowody są silniejsze dla hipotezy; gdy CF zbliża się do -1, pewność wobec hipotezy staje się silniejsza; a CF około 0 wskazuje, że istnieje niewiele dowodów przemawiających za lub przeciw hipotezie lub że dowody przemawiające za i przeciw hipotezie są zrównoważone. Kiedy eksperci tworzą podstawę reguł, muszą uzgodnić CF, który będzie pasował do każdej reguły. Ten CF odzwierciedla ich zaufanie do wiarygodności reguły. Pewne miary można dostosować, aby dostroić działanie systemu, chociaż wykazano, że niewielkie zmiany miary ufności mają niewielki wpływ na cały działający system. Ta rola miar pewności utwierdza również w przekonaniu, że „wiedza daje siłę", czyli rzetelność samej wiedzy najlepiej wspiera postawienie

prawidłowych diagnoz. Przesłanki każdej reguły składają się z i lub z szeregu faktów. Kiedy stosowana jest reguła produkcji, czynniki pewności związane z każdym warunkiem przesłanki są łączone w celu uzyskania miary pewności dla ogólnej przesłanki w następujący sposób. Dla P1 i P2 przesłanki reguły:

CF (P1 i P2) = MIN (CF (P1), CF (P2)) i

CF (P1 lub P2) = MAX (CF (P1), CF (P2)).

Połączone CF przesłanek, przy użyciu powyższych reguł, jest następnie mnożone przez CF samej reguły, aby uzyskać CF dla wniosków reguły, biorąc pod uwagę te przesłanki. Na przykład rozważ regułę w bazie wiedzy:

(P1 i P2) lub P3 → R1 (.7) i R2 (.3)

gdzie P1, P2 i P3 to przesłanki, a R1 i R2 to wnioski z reguły, mające odpowiednio współczynniki CF 0,7 i 0,3. Liczby te są dodawane do reguły podczas jej projektowania i reprezentują pewność eksperta co do wniosku, jeśli wszystkie przesłanki są znane z całkowitą pewnością. Jeśli uruchomiony program wygenerował P1, P2 i P3 z wartościami CF równymi odpowiednio 0,6, 0,4 i 0,2, to R1 i R2 mogą zostać dodane do zebranych wyników specyficznych dla przypadku z wartościami CF odpowiednio 0,28 i 0,12. Oto obliczenia dla tego przykładu:

CF (P1 (0,6) i P2 (0,4)) = MIN (0,6; 0,4) = 0,4. CF ((0,4) lub P3 (0,2)) = MAX (0,4,0,2) = 0,4. Współczynnik CF dla R1 w regule wynosi 0,7, więc R1 jest dodawany do zbioru wiedzy specyficznej dla przypadku z powiązanym współczynnikiem CF równym (0,7) × (0,4) = 0,28. Współczynnik CF dla R2 w regule wynosi 0,3, więc R2 jest dodawany do zbioru wiedzy specyficznej dla przypadku z powiązanym CF równym (0,3) × (0,4) = 0,12. Potrzebny jest jeszcze jeden środek: jak połączyć wiele CF, gdy dwie lub więcej reguł popiera ten sam wynik R. Reguła ta odzwierciedla teorię pewności analogiczną do procedury mnożenia miar prawdopodobieństwa w celu połączenia niezależnych dowodów. Używając tej reguły wielokrotnie, można łączyć wyniki dowolnej liczby reguł używanych do określania wyniku R. Załóżmy, że CF (R1) jest obecnym współczynnikiem pewności związanym z wynikiem R, a poprzednio nieużywana reguła daje wynik R (ponownie) z CF (R2); wtedy nowy CF R jest obliczany ze wzoru: CF (R1) + CF (R2) - (CF (R1) × CF (R2)), gdy CF (R1) i CF (R2) są dodatnie, CF (R1) + CF (R2) + (CF (R1) × CF (R2)), gdy CF (R1) i CF (R2) są ujemne, oraz

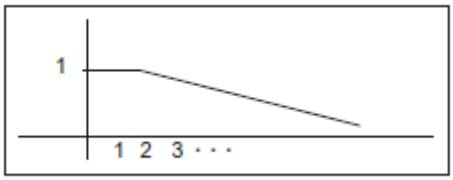
CF (R1) + CF (R2) / 1 - MIN (| CF (R1) |, | CF (R2) |)

w przeciwnym razie, gdzie „X” jest wartością bezwzględną X. Oprócz tego, że są łatwe do obliczenia, równania te mają inne pożądane właściwości. Po pierwsze, współczynniki CF, które wynikają z zastosowania tej zasady, zawsze mieszczą się w przedziale od 1 do -1. Po drugie, wynikiem łączenia sprzecznych CF jest to, że znoszą się one nawzajem, tak jak jest to pożądane. Wreszcie, połączona miara CF jest funkcją monotonicznie rosnącą (malejącą) w sposób, jakiego można by oczekiwać w przypadku łączenia dowodów. Wreszcie, miary zaufania w tradycji czynników pewności ze Stanford są ludzkimi (subiektywnymi) oszacowaniami miar prawdopodobieństwa symptomów / przyczyn. Jak zauważono w tradycji bayesowskiej, jeśli A, B i C wszystkie wpływają na D, musimy wyodrębnić i odpowiednio połączyć wszystkie wcześniejsze i późniejsze prawdopodobieństwa, w tym P (D), P (D | A), P (D | B), P (D | C), P (D | A, B) itd., Gdy chcemy uzasadnić kwestię D. Tradycja Stanford Certainty Factor pozwala inżynierowi wiedzy połączyć wszystkie te relacje w jeden czynnik zaufania, CF, dołączony do zasada; to znaczy, jeśli A, B i C, to D (CF). Niewątpliwie ta prosta algebra lepiej odzwierciedla sposób, w jaki eksperci-ludzie łączą i propagują wiele zestawów przekonań. Teoria pewności może być krytykowana jako nadmiernie ad hoc. Chociaż jest zdefiniowane w formalnej algebrze, znaczenie miar pewności nie jest tak rygorystycznie ugruntowane, jak formalna teoria prawdopodobieństwa. Jednak teoria

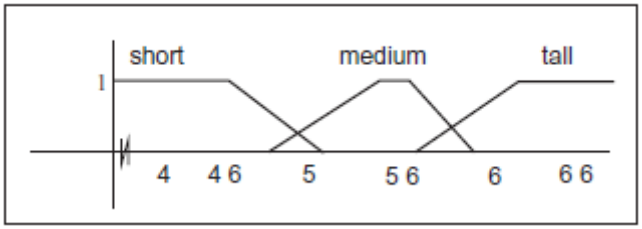
pewności nie próbuje stworzyć algebry dla „poprawnego” rozumowania. Raczej to „smarowanie” pozwala systemowi eksperckiemu łączyć zwierzenia, gdy porusza się w przestrzeni poszukiwań. Jego pomiary są ad hoc w tym samym sensie, w jakim zaufanie eksperta będącego człowiekiem do jego wyników jest przybliżone, heurystyczne i nieformalne. Kiedy MYCIN jest uruchomiony, CF są używane w wyszukiwaniu heurystycznym, aby nadać priorytet celom do osiągnięcia i punkt odcięcia, kiedy cel nie musi być dalej rozważany. Ale nawet jeśli CF jest używany do utrzymania działania programu i zbierania informacji, siła programu pozostaje zainwestowana w jakość reguł.

9.2.2 Rozumowanie z użyciem zbiorów rozmytych

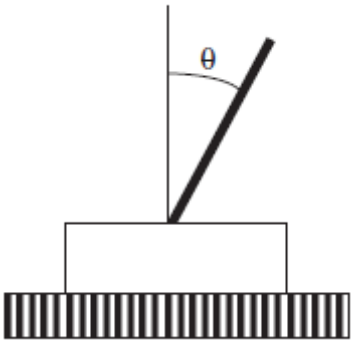
Istnieją dwa założenia, które są niezbędne do zastosowania formalnej teorii mnogości. Pierwsza odnosi się do przynależności do zbioru: dla dowolnego elementu i zbioru należącego do jakiegoś wszechświata, element jest albo członkiem zbioru, albo jest członkiem dopełnienia tego zbioru. Drugie założenie, zwane prawem wyłączonego środka, głosi, że element nie może należeć zarówno do zbioru, jak i do jego uzupełnienia. Oba te założenia zostały naruszone w teorii zbiorów rozmytych Lotfi Zadeha. W rzeczywistości zbiory i prawa rozumowania tradycyjnej teorii mnogości są określane jako ostre, z rozmytego punktu widzenia zbiorów. Głównym argumentem Zadeha jest to, że chociaż teoria prawdopodobieństwa tak jest odpowiedni do mierzenia losowości informacji, jest odpowiedni do pomiaru znaczenia informacji. Rzeczywiście, wiele nieporozumień związanych ze stosowaniem angielskich słów i zwrotów jest związanych raczej z brakiem jasności (niejasności) niż z przypadkowością. Jest to kluczowy punkt do analizy struktur językowych i może być również ważny w tworzeniu miary zaufania do reguł produkcji. Zadeh proponuje teorię możliwości jako miarę niejasności, tak jak teoria prawdopodobieństwa mierzy losowość. Teoria Zadeha wyraża brak precyzji w sposób ilościowy, wprowadzając funkcję przynależności do zbioru, która może przyjmować wartości rzeczywiste od 0 do 1. Pojęcie zbioru rozmytego można opisać następująco: niech S będzie zbiorem i sa członkiem tego zbioru . Rozmyty podzbiór F z S jest zdefiniowany przez funkcję przynależności mF (s), która mierzy „stopień” do którego s należy do F. Standardowy przykład zbioru rozmytego, jak pokazano na rysunku 6, polega na tym, że S jest zbiorem dodatnich liczb całkowitych, a F jest rozmytym podzbiorem S, zwanym małymi liczbami całkowitymi.



Teraz różne wartości całkowite mogą mieć „możliwość” rozkładu określającego ich „rozmyte członkostwo” w zbiorze małych liczb całkowitych: mF (1) = 1,0, mF (2) = 1,0, mF (3) = 0,9, mF (4) = 0,8, ..., mF (50) = 0,001 itd. Dla stwierdzenia, że dodatnia liczba całkowita X jest małą liczbą całkowitą, mF tworzy rozkład możliwości na wszystkie dodatnie liczby całkowite (S). Teoria zbiorów rozmytych nie zajmuje się sposobem tworzenia tych rozkładów możliwości, ale raczej regułami obliczania połączonych możliwości dla wyrażeń, które zawierają zmienne rozmyte. W związku z tym zawiera zasady łączenia miar możliwości dla wyrażeń zawierających zmienne rozmyte; w rzeczywistości prawa dla or, i, a nie tych wyrażeń są podobne do tych właśnie przedstawionych dla algebry współczynników pewności Stanforda; W przypadku rozmytej reprezentacji zbioru małych liczb całkowitych, każda liczba całkowita należy do tego zbioru z powiązaną miarą ufności. W tradycyjnej logice zbiorów „ostrych”, pewność elementu będącego w zbiorze musi wynosić 1 lub 0. Rysunek 7 przedstawia funkcję przynależności do zbioru dla pojęcia niskiego, średniego i wysokiego mężczyzny.

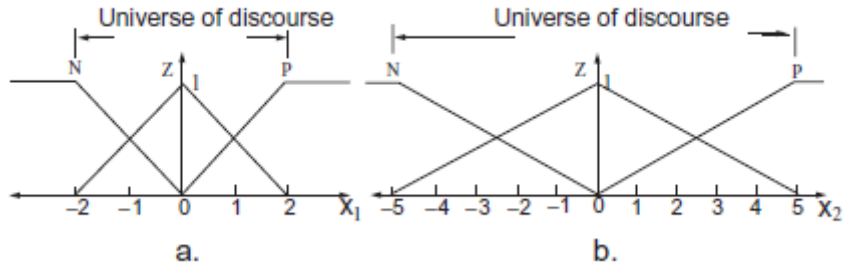


Zwróć uwagę, że każda osoba może należeć do więcej niż jednego zestawu, na przykład samiec 5 '9 "należy zarówno do zbioru średnich, jak i wysokich samców. Następnie pokażemy zasady łączenia i propagowania miar rozmytych przez przedstawianie części problemu, obecnie klasyczna w literaturze dotyczącej zbiorów rozmytych, reżim sterowania dla odwróconego wahadła. Rysunek 8 przedstawia wahadło, odwrócone, które chcemy zachować w równowadze i skierowane do góry.

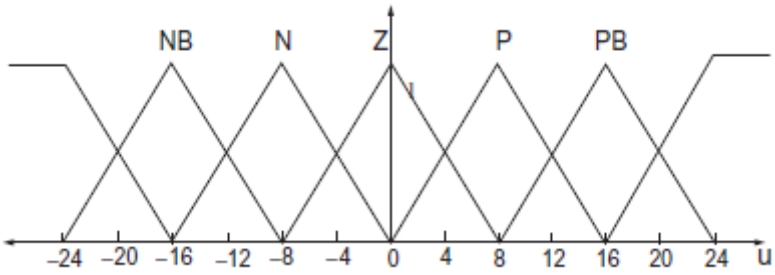


Utrzymujemy wahadło w równowadze, przesuwając podstawę układu, aby zrównoważyć siłę grawitacji działającą na wahadło. Istnieją zestawy równań różniczkowych, które mogą deterministycznie utrzymywać wahadło w równowadze. Zaletą rozmytego podejścia do sterowania tym układem wahadła jest to, że można ustalić algorytm do wydajnego sterowania systemem w czasie rzeczywistym. Następnie opiszemy ten reżim kontroli. Upraszczamy problem wahadła, przedstawiając go w dwóch wymiarach. Te dwa pomiary są używane jako wartości wejściowe do sterownika. Po pierwsze, kąt θ, odchylenie wahadła od pionu, a po drugie, prędkość dθ / dt, z jaką porusza się wahadło. Obie te miary są dodatnie w kwadrancie po prawej stronie od pionu i ujemne po lewej stronie. Te dwie wartości są przekazywane do kontrolera rozmytego w każdej iteracji systemu. Sygnał wyjściowy kontrolera to ruch i kierunek podstawy systemu, instrukcje mające na celu utrzymanie równowagi wahadła. Aby wyjaśnić działanie kontrolera rozmytego, opisujemy proces rozwiązania zbioru rozmytego. Dane opisujące stan wahadła, θ i dθ / dt, są interpretowane jako miary rozmyte i przedstawiane w zestawie reguł rozmytych. Ten krok jest często bardzo wydajny dzięki zastosowaniu struktury zwanej rozmytą macierzą asocjacyjną lub FAM, rysunek 9.12, gdzie relacje wejścia / wyjścia są kodowane bezpośrednio. Reguły nie są ze sobą powiązane, jak w przypadku tradycyjnego rozwiązywania problemów opartego na regułach. Zamiast tego wszystkie dopasowane reguły uruchamiają się, a ich wyniki są łączone. Wynik ten, zwykle reprezentowany przez obszar rozmytej przestrzeni parametrów wyjściowych, rysunek 9.10, jest następnie usuwany w celu zwrócenia odpowiedzi sterującej. Zauważ, że zarówno oryginalne wejście, jak i ostateczne wyjście kontrolera są wyraźnymi wartościami. Są to dokładne odczyty niektórych monitorów, wejść i precyzyjnych instrukcji dotyczących działań kontrolnych, wyjścia. Następnie opiszemy rozmyte regiony dla wartości wejściowych, θ i dθ / dt. Przykład ten upraszcza sytuację np. W ilości rozmytych obszarów wartości wejściowych, ale pokazuje pełny cykl stosowania reguły i reakcję kontrolera. Wartość wejściowa θ jest podzielona na trzy obszary, Ujemny, Zero i Dodatni, gdzie θ mieści się w zakresie od -2 do +2 radianów, jak widać na rysunku.9a. Rysunek 9b

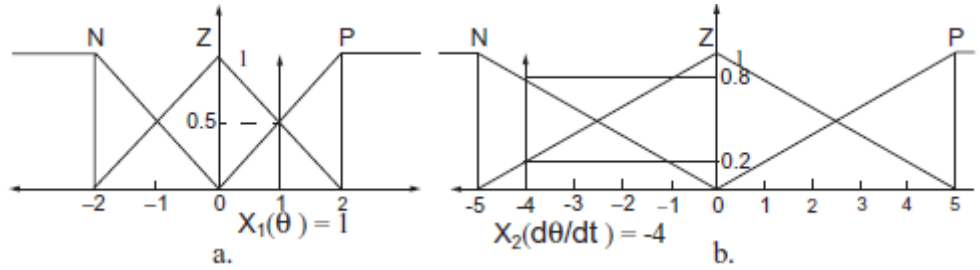
przedstawia trzy obszary, na które podzielona jest druga wartość wejściowa, dθ / dt, ponownie ujemna, zero i dodatnia, w zakresie od -5 do +5 stopni na sekundę.



Rysunek 10 przedstawia podział przestrzeni wyjściowej, w którym używamy środkowych pięciu regionów, Negative Big, Negative, Zero, Positive i Positive Big. Miara między -24 a +24 reprezentuje ruch i kierunek każdej odpowiedzi.



Załóżmy, że rozpoczyna się symulacja i pierwsze wartości podane kontrolerowi to θ = 1 idθ / dt = −4. Rysunek 11 odzwierciedla fuzifikację tych miar wejściowych.



W każdej sytuacji wartość wejściowa wpływa na dwa obszary rozmytej przestrzeni wejściowej. Dla θ wartości wynoszą zero, przy 0,5 i dodatnie, przy 0,5 miarach możliwości. Dla dθ / dt są one Ujemne przy 0,8 i Zero przy 0,2 miarach możliwości. Rysunek12 przedstawia uproszczoną postać rozmytej macierzy asocjacyjnej dla tego problemu.

x_1 \ x_2	P	Z	N
P	PB	P	Z
Z	P	Z	N
N	Z	N	NB

Wartości wejściowe do tabeli dla θ lub x1 znajdują się po lewej stronie, a dla dθ / dt lub x2 znajdują się w górnej części macierzy. W takim razie tabela 3 × 3 w prawym dolnym rogu FAM podaje wartości wyjściowe. Na przykład, jeśli θ jest dodatnie, a dθ / dt jest ujemne, funkcja FAM zwraca wartość zerowego ruchu układu wahadła. Zwróć uwagę, że odpowiedź nadal musi zostać rozszyfrowana przy użyciu obszaru wyjściowego Zero z rysunku 10. W takim przypadku, ponieważ każda wartość wejściowa dotyka dwóch obszarów przestrzeni wejściowej, należy zastosować cztery reguły. Jak wspomniano powyżej, reguły kombinacji dla układów rozmytych są podobne do reguł algebry współczynników pewności ze Stanford. W rzeczywistości Zadeh był pierwszym (historycznie), który zaproponował te reguły kombinacji dla algebry rozumowania rozmytego. Jeśli miary dwóch pomieszczeń są połączone operatorem AND, to minimum ich miar jest traktowane jako miara reguły. Jeśli dwa lokale są połączone, podejmowane są maksymalne środki. W naszym przykładzie wszystkie pary przesłanek są połączone operatorem AND, więc minimum ich miar jest traktowane jako miara wyniku reguły:

IF x1 = P AND x2 = Z THEN u = P

min(0.5, 0.2) = 0.2 P

IF x1 = P AND x2 = N THEN u = Z

min(0.5, 0.8) = 0.5 Z

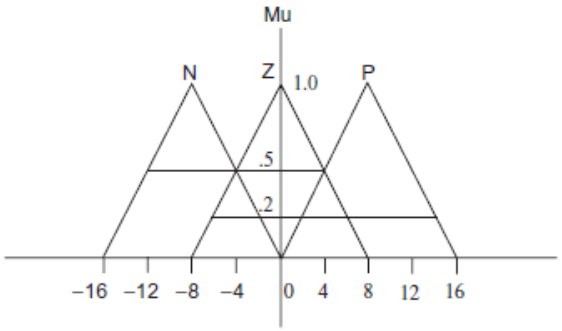
IF x1 = Z AND x2 = Z THEN u = Z

min(0.5, 0.2) = 0.2 Z

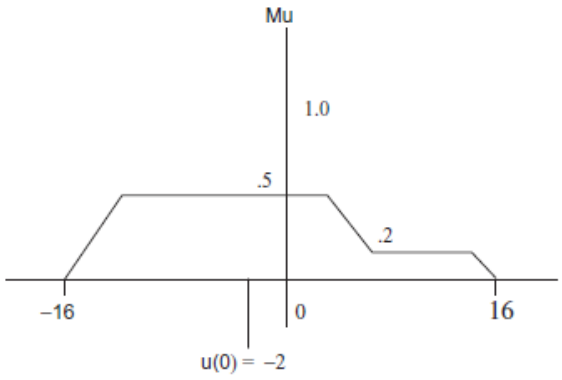
IF x1 = Z AND x2 = N THEN u = N

min(0.5, 0.8) = 0.5 N.

Następnie łączone są wyniki wyjściowe. W tym przykładzie łączymy razem dwa obszary z rysunku 10, wskazane przez wyniki tego zestawu dwóch reguł strzelania. Istnieje wiele możliwych technik defuzyfikacji. Wybraliśmy jedną z najpopularniejszych, metodę centroidu. Aby użyć tej metody, środek ciężkości sumy pól wartości wyjściowych staje się wartością końcową, którą regulator przykłada do wahadła. Związek, jak również środek ciężkości związku, przedstawiono na rysunku 13.



a.



b.

Po zastosowaniu tego wyjścia lub wyniku do systemu, θ idθ / dt są ponownie próbkowane i cykl kontrolny jest powtarzany. Istnieje wiele problemów, którymi nie zajęliśmy się przy opisywaniu systemów rozumowania rozmytego, w tym wzorce oscylacji w procesie konwergencji i optymalne częstotliwości próbkowania. Systemy rozmyte, szczególnie w dziedzinie sterowania, oferują inżynierom potężne i wydajne narzędzie do radzenia sobie z nieprecyzyjnymi pomiarami.

9.2.3 Teoria dowodów Dempstera-Shafera

Do tego momentu w naszej dyskusji na temat rozumowania w warunkach niepewności opisaliśmy techniki, które uwzględniają indywidualne zdania i przypisują każdemu wpływ przyczynowy lub liczbową ocenę stopnia przekonania, jaki moglibyśmy mieć, mając dane zestawy dowodów. Jednym z ograniczeń probabilistycznych podejść do niepewności jest użycie jednej wielkości do pomiaru sytuacji, która może być bardzo złożona. Często niepewność wynika z kombinacji brakujących dowodów, nieodłącznych ograniczeń reguł heurystycznych oraz ograniczeń naszej własnej wiedzy. Alternatywne podejście, zwane teorią dowodów Dempstera-Shafera, rozważa zbiory zdań i przypisuje każdemu z nich przedział [przekonanie, prawdopodobieństwo], w którym musi znajdować się stopień przekonania dla każdego zdania. Ta miara wiary, oznaczona bel, waha się od zera, co wskazuje na brak dowodów na poparcie zbioru zdań, do jednego, co oznacza pewność. Wiarygodność zdania p, pl (p), jest określona:

pl (p) = 1 - bel (not (p))

Zatem prawdopodobieństwo waha się również od zera do jednego i odzwierciedla, w jaki sposób dowód braku (p) odnosi się do możliwości wiary w p. Jeśli mamy pewne dowody na to, że nie (p), to bel (not (p)) będzie równe jeden, a pl (p) będzie równe zero. Jedyną możliwą wartością bel (p) jest również zero. Załóżmy, że mamy dwie konkurujące hipotezy h1 i h2. Kiedy nie mamy informacji

popierając obie hipotezy, każda z nich ma zakres przekonania / prawdopodobieństwa [0,1]. W miarę gromadzenia dowodów spodziewamy się, że te przedziały się skracają, co stanowi wzrost wiarygodności hipotez. W domenie bayesowskiej prawdopodobnie zaczęlibyśmy (bez dowodów) równo rozdzielić wcześniejsze prawdopodobieństwa między dwie hipotezy, dając każdej, powiedzmy, p (hi) = 0,5. Dempster – Shafer wyjaśnia, że nie mamy żadnych dowodów, kiedy zaczynamy; z drugiej strony podejście bayesowskie może skutkować taką samą miarą prawdopodobieństwa, niezależnie od tego, ile mamy danych. Zatem Dempster – Shafer może być bardzo przydatny, gdy jest ważne, aby podjąć decyzję na podstawie ilości zebranych dowodów. Podsumowując, Dempster i Shafer zajmują się problemem pomiaru pewności, dokonując fundamentalnego rozróżnienia między brakiem pewności a ignorancją. W teorii prawdopodobieństwa jesteśmy zmuszeni wyrazić zakres naszej wiedzy na temat hipotezy h za pomocą jednej liczby p(h). Problem z tym podejściem, mówią Dempster i Shafer, polega na tym, że po prostu nie zawsze możemy znać wartości jego potwierdzających prawdopodobieństw, a zatem jakikolwiek konkretny wybór p (h) może nie być uzasadniony. Funkcje przekonań Dempstera – Shafera spełniają aksjomaty słabsze niż te z teorii prawdopodobieństwa, to znaczy redukują się do teorii prawdopodobieństwa, gdy wszystkie prawdopodobieństwa są osiągalne. Funkcje wiary pozwalają nam wykorzystać naszą wiedzę do powiązania przypisania prawdopodobieństw zdarzeniom w przypadku braku dokładnych prawdopodobieństw. Teoria Dempstera – Shafera opiera się na dwóch ideach: po pierwsze, na idei uzyskiwania stopni przekonań dla jednego pytania z subiektywnych prawdopodobieństw dla pytań pokrewnych, a po drugie, na zastosowaniu reguły do łączenia tych stopni przekonań, gdy są one dowody. Ta reguła kombinacji została pierwotnie zaproponowana przez Dempstera. Następnie przedstawiamy nieformalny przykład rozumowania Dempstera-Shafera, następnie odrzucamy zasadę Dempstera i na koniec stosujemy tę regułę do bardziej realistycznej sytuacji. Załóżmy, że mam subiektywne prawdopodobieństwo wiarygodności mojej przyjaciółki Melissy. Prawdopodobieństwo, że jest wiarygodna wynosi 0,9, a niewiarygodne, 0,1. Przypuśćmy, że Melissa powie mi, że włamano się do mojego komputera. To stwierdzenie jest prawdziwe, jeśli Melissa jest wiarygodna, ale niekoniecznie jest fałszywe, jeśli jest niewiarygodna. Tak więc zeznanie Melissy w pewnym stopniu uzasadnia 0,9 przekonania, że włamano się do mojego komputera i 0,0 przekonania, że tak nie jest. Przekonanie o wartości 0,0 nie oznacza, że jestem pewien, że mój komputer nie został włamany, ponieważ prawdopodobieństwo wynosi 0,0. Oznacza to po prostu, że zeznania Melissy nie dają mi powodu, by sądzić, że mój komputer nie został włamany. Miarą prawdopodobieństwa, pl, w tej sytuacji jest:

pl (computer_broken_into) = 1 - bel (not (computer_broken_into)) = 1 - 0,0 lub 1,0, a moją miarą wiary dla zeznań Melissy jest [0,9 1,0]. Zwróć uwagę, że nadal nie ma dowodów na to, że mój komputer nie został włamany. Następnie rozważymy regułę Dempstera dotyczącą łączenia dowodów. Przypuśćmy, że mój przyjaciel Bill również mi powie, że włamano się do mojego komputera. Załóżmy, że prawdopodobieństwo, że Bill jest wiarygodny, wynosi 0,8, a niewiarygodnego, wynosi 0,2. Muszę również przypuszczać, że zeznania Billa i Melissy na temat mojego komputera są od siebie niezależne; to znaczy mają oddzielne powody, by mi powiedzieć, że ich zdaniem włamano się do mojego komputera. Fakt, że Bill jest wiarygodny, musi być również niezależny od wiarygodności Melissy Prawdopodobieństwo, że zarówno Bill, jak i Melissa są wiarygodni, jest iloczynem ich rzetelności, czyli 0,72; prawdopodobieństwo, że oba są niewiarygodne, wynosi 0,02. Prawdopodobieństwo, że przynajmniej jeden z nich jest wiarygodny, wynosi 1 - 0,02 lub 0,98. Ponieważ obaj powiedzieli, że włamano się do mojego komputera i istnieje prawdopodobieństwo 0,98, że przynajmniej jeden z nich jest niezawodny, przypiszę zdarzeniu włamania do mojego komputera na [0,98 1,0] stopień wiary. Załóżmy, że Bill i Melissa nie zgadzają się co do tego, czy włamano się do mojego komputera: Melissa mówi, że tak, a Bill mówi, że tak nie jest. W tym przypadku nie mogą one jednocześnie być poprawne i nie mogą być jednocześnie wiarygodne. Albo są niewiarygodne, albo tylko jeden jest niezawodny.

Wcześniejsze prawdopodobieństwo, że tylko Melissa jest wiarygodna, wynosi 0,9 × (1 - 0,8) = 0,18, że tylko Bill jest wiarygodny, wynosi 0,8 × (1 - 0,9) = 0,08, a żadne z nich nie wynosi 0,2 × 0,1 = 0,02. Biorąc pod uwagę, że co najmniej jeden nie jest wiarygodny (0,18 + 0,08 + 0,02) = 0,28, możemy również obliczyć prawdopodobieństwo późniejsze, że tylko Melissa jest wiarygodna jako 0,18 / 0,28 = 0,643, a mój komputer został włamany, lub prawdopodobieństwo późniejsze, że tylko Bill miał rację, 0,08 / 0,28 = 0,286, a mój komputer nie został włamany. Właśnie użyliśmy reguły Dempstera, aby połączyć przekonania. Kiedy Melissa i Bill zgłosili włamanie do komputera, zsumowaliśmy trzy hipotetyczne sytuacje, które wsparły włamanie: Bill i Melissa są wiarygodni; Bill jest godny zaufania, a Melissa nie; i Melissa jest wiarygodna, a Bill nie. Przekonanie, 0,98, było sumą tych możliwych wspierających hipotetycznych scenariuszy. Przy drugim zastosowaniu reguły Dempstera świadkowie nie zgodzili się. Ponownie podsumowaliśmy wszystkie możliwe scenariusze. Jedyną niemożliwą sytuacją było to, że obaj byli wiarygodni; tak więc albo Melissa była wiarygodna, a Bill nie, Bill był godny zaufania, a Melissa nie, albo też żaden nie był wiarygodny. Suma tych trzech daje prawdopodobieństwo włamania 0,64. Przekonanie, że mój komputer nie został włamany (zdaniem Billa), wynosiło 0,286; ponieważ prawdopodobieństwo włamania wynosi 1 - bel (nie (włamanie)) lub 0,714, miarą przekonania jest [0,28 0,714]. Aby zastosować regułę Dempstera, uzyskujemy stopnie wiary dla jednego pytania (Czy włamano się do mojego komputera?) Z prawdopodobieństwa dla innego pytania (Czy świadkowie są wiarygodni?). Reguła zaczyna się od założenia, że pytania, dla których mamy prawdopodobieństwa, są niezależne, ale ta niezależność jest tylko a priori. Znika, gdy mamy konflikt między różnymi dowodami. Zastosowanie podejścia Dempstera – Shafera w określonej sytuacji wiąże się z rozwiązaniem dwóch powiązanych problemów. Po pierwsze, dzielimy niepewność sytuacji na niezależne dowody a priori. Po drugie, realizujemy regułę Dempstera. Te dwa zadania są ze sobą powiązane: Przypuśćmy znowu, że Bill i Melissa powiedzieli mi niezależnie, że wierzyli, że włamano się do mojego komputera. Przypuśćmy również, że zadzwoniłem do osoby zajmującej się naprawami, aby sprawdzić mój komputer i że zarówno Bill, jak i Melissa byli tego świadkami. Z powodu tego powszechnego zdarzenia nie mogę już porównywać stopni przekonań. Jeśli jednak wyraźnie rozważę możliwość pracy osoby naprawiającej na moim komputerze, to mam trzy niezależne dowody: wiarygodność Melissy, wiarygodność Billa oraz dowody na obecność osoby naprawiającej, które mogę następnie połączyć z regułą Dempstera . Załóżmy, że mamy wyczerpujący zestaw wzajemnie wykluczających się hipotez, które nazywamy Q. Naszym celem jest dołączenie pewnej miary przekonania, m, do różnych podzbiorów Z Q; m jest czasami nazywane funkcją gęstości prawdopodobieństwa dla podzbioru Q. Realistycznie rzecz biorąc, nie wszystkie dowody bezpośrednio potwierdzają poszczególne elementy Q. W rzeczywistości dowody najczęściej wspierają różne podzbiory Z z Q. Ponadto, ponieważ elementy Q są zakładając, że wzajemnie się wykluczają, dowody na korzyść niektórych mogą mieć wpływ na naszą wiarę w innych. W systemie czysto bayesowskim, w sekcji 9.3, zajmujemy się obydwoma tymi sytuacjami, wymieniając wszystkie kombinacje prawdopodobieństw warunkowych. W systemie Dempster – Shafer radzimy sobie z tymi interakcjami poprzez bezpośrednie manipulowanie zestawami hipotez. Wielkość $m_n(Z)$ mierzy ilość przekonań przypisanych do podzbioru Z hipotez, an reprezentuje liczbę źródeł dowodów. Reguła Dempstera mówi:

$$m_n(Z) = \frac{\sum_{X \cap Y = Z} m_{n-2}(X) m_{n-1}(Y)}{1 - \sum_{X \cap Y = \phi} m_{n-2}(X) m_{n-1}(Y)}$$

Na przykład wiara w hipotezę Z, przy n = 3 źródłach dowodu, $m_3(Z)$, jest sumą iloczynów sytuacji hipotetycznych $m_1(X)$ i $m_2(Y)$, których współwystępowanie wspiera Z , to znaczy $X \cap Y = Z$. Mianownik reguły Dempstera uznaje, jak widać w poniższym przykładzie, że X i Y mogą mieć puste przecięcie, a suma ufności musi być znormalizowana o jeden minus suma te wartości. Następnie stosujemy regułę

Dempstera do sytuacji diagnozy medycznej. Załóżmy, że Q reprezentuje domenę naszego zainteresowania, zawierającą cztery hipotezy: że pacjent ma przeziębienie (C), grypę (F), migrenowe bóle głowy (H) lub zapalenie opon mózgowych (M). Naszym zadaniem jest powiązanie miar przekonań z zestawami hipotez w Q. Jak już wspomniano, są to zestawy hipotez, ponieważ dowody nie muszą wspierać wyłącznie indywidualnych hipotez. Na przykład gorączka może wspomóc {C, F, M}. Ponieważ elementy Q są traktowane jako wzajemnie wykluczające się hipotezy, dowody na korzyść niektórych mogą wpływać na wiarę w innych. Jak już wspomniano, podejście Dempstera – Shafera odnosi się do interakcji poprzez bezpośrednie obsługiwanie zestawów hipotez.

Dla funkcji gęstości prawdopodobieństwa m i wszystkich podzbiorów Z zbioru Q wielkość m(qi) reprezentuje przekonanie, które jest obecnie przypisane do każdego q_i z Q z sumą wszystkich m (q_i) równą jeden. Jeśli Q zawiera n elementów, to istnieją 2^n podzbiory Q. Chociaż adresowanie 2^n wartości może wydawać się zniechęcające, zwykle okazuje się, że wiele podzbiorów nigdy nie wystąpi. W związku z tym istnieje pewne uproszczenie procesu rozwiązywania, ponieważ wartości te można zignorować, ponieważ nie mają one użyteczności w dziedzinie problemu. Wreszcie wiarygodność Q to $pl(Q) = 1 - \Sigma m(q_i)$, gdzie q_i to zbiory hipotez, które mają pewne przekonanie wspierające. Jeśli nie mamy informacji o jakichkolwiek hipotezach, jak to często bywa, kiedy rozpoczynamy diagnozę, to pl(Q) = 1,0. Załóżmy, że pierwszym dowodem jest to, że nasz pacjent ma gorączkę i to potwierdza {C, F, M} na poziomie 0,6. Nazywamy to pierwsze przekonanie m1. Jeśli jest to nasza jedyna hipoteza, to m_1{C, F, M} = 0,6, gdzie m_1(Q) = 0,4, aby uwzględnić pozostałą dystrybucję przekonań. Należy zauważyć, że m_1 (Q) = 0,4 reprezentuje pozostałą część naszego rozkładu przekonań, to znaczy wszystkie inne możliwe przekonania w Q, a nie naszą wiarę w dopełnienie {C, F, M}. Załóżmy, że mamy teraz jakieś nowe dane do naszej diagnozy, powiedzmy, że pacjent ma silne nudności, co sugeruje {C, F, H} z poziomem wsparcia 0,7. Dla tego przekonania nazwijmy go m2, mamy m_2{C, F, H} = 0,7 i m_2(Q) = 0,3. Używamy reguły Dempstera, aby połączyć te dwa przekonania, m_1 i m_2. Niech X będzie zbiorem podzbiorów Q, którym m_1 przypisuje wartość różną od zera, a Y będzie zbiorem podzbiorów Q, którym m_2 przypisuje wartość niezerową. Następnie tworzymy przekonanie złożone, m_3 zdefiniowane w podzbiorach Z z Q przy użyciu reguły Dempstera. Stosując regułę Dempstera do diagnoz, najpierw zwróć uwagę, że nie ma zbiorów $X \cap Y$, które byłyby puste, więc mianownikiem jest 1. Rozkład przekonań dla m_3 przedstawiono w tabeli 1.

m_1	m_2	m_3
m_1{C,F,M} = 0.6	m_2{C,F,H} = 0.7	m_3{C,F} = 0.42
m_1(Q) = 0.4	m_2{C,F,H} = 0.7	m_3{C,F,H} = 0.28
m_1{C,F,M} = 0.6	m_2(Q) = 0.3	m_3{C,F,M} = 0.18
m_1(Q) = 0.4	m_2(Q) = 0.3	m_3(Q) = 0.12

Używając reguły Dempstera, cztery zbiory Z, wszystkie możliwe sposoby przecięcia X i Y, tworzą skrajną prawą kolumnę tabeli 1. Ich poziom przekonań jest obliczany przez pomnożenie przekonań dla odpowiednich elementów X i Y odpowiednio pod m_1 i m_2. Należy również zauważyć, że w tym przykładzie każdy zestaw w Z jest unikalny, co często nie ma miejsca. Rozszerzamy nasz przykład po raz ostatni, aby pokazać, w jaki sposób puste zestawy przekonań są uwzględniane w analizie. Załóżmy, że mamy nowy fakt, wyniki hodowli laboratoryjnej, które są związane z zapaleniem opon mózgowych. Mamy teraz m_4{M} = 0,8 i m_4(Q) = 0,2. Możemy użyć wzoru Dempstera, aby połączyć m_3, wyniki naszej poprzedniej analizy, z m4, aby otrzymać m5, jak widać w tabeli 2.

m_3	m_4	m_5 (without denominator)
$m_3\{C,F\} = 0.42$	$m_4\{M\} = 0.8$	$m_5\{\ \} = 0.336$
$m_3(Q) = 0.12$	$m_4\{M\} = 0.8$	$m_5\{M\} = 0.096$
$m_3\{C,F\} = 0.42$	$m_4(Q) = 0.2$	$m_5\{C,F\} = 0.084$
$m_3(Q) = 0.12$	$m_4(Q) = 0.2$	$m_5(Q) = 0.024$
$m_3\{C,F,H\} = 0.28$	$m_4\{M\} = 0.8$	$m_5\{\ \} = 0.224$
$m_3(C,F,M) = 0.18$	$m_4\{M\} = 0.8$	$m_5\{M\} = 0.144$
$m_3\{C,F,H\} = 0.28$	$m_4(Q) = 0.2$	$m_5\{C,F,H\} = 0.056$
$m_3(C.F,M) = 0.18$	$m_4(Q) = 0.2$	$m_5\{C,F,M\} = 0.036$

Po pierwsze, zauważ, że $m_5\{M\}$ jest tworzone przez przecięcia dwóch różnych par zbiorów, więc suma $m_5\{M\} = 0,240$. Mamy również przypadek, w którym kilka przecięć zbiorów daje pusty zbiór {}. Zatem mianownik równania Dempstera to 1 - (0,336 + 0,224) = 1 - 0,56 = 0,44. Ostateczna połączona funkcja przekonań dla m_5 to:

$m_5\{M\} = 0.545$	$m_5\{C,F\} = 0.191$	$m_5\{\ \} = 0.56$
$m_5\{C,F,H\} = 0.127$	$m_5\{C,F,M\} = 0.082$	$m_5(Q) = 0.055$

Trzy uwagi końcowe. Po pierwsze, duże przekonanie przypisane do pustego zbioru, jak w tym końcowym $m_5\{\} = 0,56$, wskazuje, że istnieją sprzeczne dowody w zestawie przekonań m_i. W rzeczywistości zaprojektowaliśmy nasz przykład tak, aby pokazać kilka cech rozumowania Dempstera-Shafera, a w konsekwencji poświęcić rzetelność medyczną. Po drugie, gdy istnieją duże zestawy hipotez, a także złożone zbiory dowodów, obliczenia dla zestawów przekonań mogą stać się kłopotliwe, mimo że, jak wskazano wcześniej, ilość obliczeń jest nadal znacznie mniejsza niż w przypadku rozumowania bayesowskiego. Wreszcie podejście Dempstera – Shafera jest bardzo przydatnym narzędziem, gdy silniejsze założenia bayesowskie mogą nie być uzasadnione. Dempster – Shafer jest przykładem algebry wspierającej użycie prawdopodobieństwa subiektywnego w rozumowaniu. Czasami wydaje się, że te subiektywne prawdopodobieństwa lepiej odzwierciedlają rozumowanie eksperta.

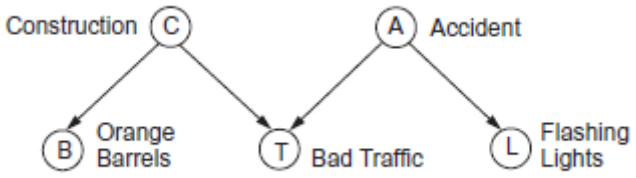
9.3 Stochastyczne podejście do niepewności

Korzystając z teorii prawdopodobieństwa, możemy określić, często z argumentu a priori, szanse zaistnienia zdarzeń. Możemy również opisać, jak kombinacje zdarzeń mogą na siebie wpływać. Chociaż ostatnie poprawki teorii prawdopodobieństwa czekały na matematyków początku XX wieku, w tym Fishera, Neymana i Pearsona, próba stworzenia algebry kombinatorycznej sięga średniowiecza do Greków, w tym Llulla, Porfiry'ego i Platona. Wgląd wspierający teorię prawdopodobieństwa jest taki, że jeśli potrafimy zrozumieć częstotliwość, z jaką zdarzenia miały miejsce w przeszłości, możemy wykorzystać te informacje (jako błąd indukcyjny) do interpretacji i wnioskowania na temat obecnych danych. W Części 5 zauważyliśmy, że istnieje wiele sytuacji, w których analiza probabilistyczna jest odpowiednia. Na przykład, gdy świat jest naprawdę losowy, jak podczas gry z dobrze wymieszanymi kartami lub kręcenia uczciwym kołem ruletki. Co więcej, kiedy wiele wydarzeń na świecie może nie być naprawdę przypadkowych, ale niemożliwe jest poznanie i zmierzenie wszystkich ich przyczyn i interakcji na tyle dobrze, aby przewidzieć zdarzenia; korelacje statystyczne są użytecznym wsparciem w interpretacji świata. Kolejną rolą statystyki jest podstawa automatycznej indukcji i uczenia maszynowego (na przykład algorytm ID3 z sekcji 10.3). Wreszcie w ostatnich pracach podjęto próbę bezpośredniego połączenia pojęć prawdopodobieństwa i przyczynowości. Podstawowym

mechanizmem wnioskowania w domenach stochastycznych jest pewna forma reguły Bayesa. Jak jednak zauważyliśmy, pełne wykorzystanie wnioskowania bayesowskiego w domenach złożonych szybko staje się trudne do opanowania. Probabilistyczne modele graficzne są specjalnie zaprojektowane w celu rozwiązania tej złożoności. Probabilistyczne modele graficzne to „... małżeństwo między teorią prawdopodobieństwa i teorią grafów". Probabilistyczne modele graficzne stają się również coraz ważniejsze w uczeniu maszynowym (Część 13), ponieważ mogą rozwiązywać problemy niepewności i złożoności, które są obecne i mogą być fundamentalnie ograniczające dla współczesnej sztucznej inteligencji. Jordan zwraca uwagę, że problem modułowości jest kamieniem węgielnym modułów graficznych: proste części modelu są łączone ze sobą za pomocą teorii prawdopodobieństwa, która wspiera spójność systemu jako całości i jednocześnie integruje model graficzny z danymi wniosek (Jordan 1999). Jednocześnie teoria grafów oferuje modele graficzne zarówno jako intuicyjny sposób przedstawiania wysoce oddziałujących zestawów komponentów, jak i strukturę danych, która wspiera wydajne algorytmy wnioskowania. Rozważymy dwa typy probabilistycznych modeli graficznych: ukierunkowane, bayesowskie sieci przekonań i różne formy modeli Markowa oraz nieukierunkowane drzewa klikowe i pola losowe Markowa. Pola Markowa i inne niekierowane modele graficzne są w stanie przedstawić wiele cyklicznych zależności, których nie można przedstawić za pomocą skierowanych grafów. Z drugiej strony sieci przekonań bayesowskich są ukierunkowanymi modelami graficznymi, które są w stanie uchwycić niejawnie wywnioskowane związki i zależności dokładniej i wydajniej niż w przypadku niekierowanych modeli graficznych. W następnych sekcjach przedstawiamy sieci przekonań bayesowskich i kilka technik wnioskowania zaprojektowanych specjalnie w celu rozwiązania złożoności obliczeniowej grafów nieukierunkowanych i / lub pełnego rozumowania bayesowskiego. Później, w sekcji 9.3.5, wprowadzimy procesy Markowa i wspomnimy o kilku ważnych odmianach reprezentacyjnych. Jedną z nich jest dynamiczna sieć bayesowska, a drugą dyskretny proces Markowa. Jednym z głównych ograniczeń BBN i tradycyjnych HMM jest ich wrodzona propozycja. Aby rozwiązać ten problem, podjęto kilka prób połączenia logiki przyimkowej, pełnego rachunku predykatów opartego na zmiennych; z probabilistycznymi modelami graficznymi. Podsumowujemy kilka z tych wysiłków w sekcji 9.3.7, a systemy probabilistyczne pierwszego rzędu ponownie omówimy w Części 13.

9.3.1 Ukierunkowany model graficzny: Bayesowska sieć przekonań

Chociaż Bayesowska teoria prawdopodobieństwa, oferuje matematyczną podstawę do wnioskowania w niepewnych warunkach, złożoność napotkana przy stosowaniu jej do realistycznych dziedzin problemowych może być wygórowana. Na szczęście często możemy zmniejszyć tę złożoność, koncentrując się na mniejszym zestawie bardziej istotnych wydarzeń i dowodów. Jedno podejście, bayesowskie sieci przekonań lub BBN (Pearl 1988), oferuje model obliczeniowy do rozumowania w celu najlepszego wyjaśnienia zbioru danych w kontekście oczekiwanych związków przyczynowych w dziedzinie problemowej. Sieci przekonań bayesowskich mogą radykalnie zmniejszyć liczbę parametrów pełnego modelu bayesowskiego i pokazać, w jaki sposób dane domeny (lub nawet brak danych) mogą podzielić i skupić się na rozumowaniu. Ponadto modułowość domeny problemowej często pozwala projektantowi programu przyjmować wiele założeń dotyczących niezależności, które są niedozwolone w pełnym leczeniu bayesowskim. W większości przypadków rozumowania nie jest konieczne tworzenie dużej wspólnej tabeli prawdopodobieństwa, w której wymienione są prawdopodobieństwa dla wszystkich możliwych kombinacji zdarzeń i dowodów. Raczej ludzcy eksperci są w stanie wybrać lokalne zjawiska, o których wiedzą, że będą oddziaływać i uzyskać wskaźniki prawdopodobieństwa lub wpływu, które odzwierciedlają tylko te grupy zdarzeń. Eksperci zakładają, że wszystkie inne zdarzenia są warunkowo niezależne lub ich korelacje są tak małe, że można je zignorować. BBN precyzują te intuicje. Jako przykład bayesowska sieć przekonań, rozważ ponownie problem ruchu, przedstawiony na rysunku 14.



Przypomnijmy, że budowa drogi to C, wypadek, A, obecność pomarańczowych beczek, B, zły ruch uliczny, T, a migające światła to L. Aby obliczyć łączne prawdopodobieństwo wszystkich parametrów z przykładu, przyjmując pełne podejście bayesowskie, wymagana wiedza lub pomiary dla wszystkich parametrów znajdujących się w poszczególnych stanach. Zatem wspólne prawdopodobieństwo, przy użyciu topologicznie posortowanej kolejności dla zmiennych, wynosi:

p (C, A, B, T, L) = p (C) p (A | C) p (B | C, A) p (T | C, A, B) p (L | C, A, B , T)

Liczba parametrów w tym wspólnym prawdopodobieństwie wynosi 2^5 lub 32. Ta tabela jest wykładnicza pod względem liczby parametrów. W przypadku problemu o dowolnej złożoności, powiedzmy z trzydziestoma lub więcej parametrami, wspólna tabela dystrybucji miałaby ponad miliard elementów! Należy jednak pamiętać, że jeśli możemy podtrzymać założenie, że parametry tego problemu zależą tylko od prawdopodobieństw ich rodziców, to znaczy możemy przyjąć, że węzły są niezależne od wszystkich nie-potomków, biorąc pod uwagę wiedzę ich rodziców, obliczenie p (C, A, B, T, L) staje się:

p (C, A, B, T, L) = p (C) * p (A) * p (B | C) * p (T | C, A) * p (L | A)

Aby lepiej zobaczyć wprowadzone przez nas uproszczenia, rozważ p (B | C, A) z poprzedniego równania. W naszym najnowszym równaniu zredukowaliśmy to do p (B | C). Opiera się to na założeniu, że budowa drogi nie jest skutkiem przyczynowym zaistnienia wypadku. Podobnie, obecność pomarańczowych beczek nie jest przyczyną złego ruchu, ale budowa i wypadek dają w rezultacie p (T | C, A), a nie p (T | C, A, B). Wreszcie, zależność probabilistyczna p (L | C, A, B, T) zostaje zredukowana do p (L | A). Rozkład prawdopodobieństwa dla p (C, A, B, T, L) ma teraz tylko 20 (zamiast 32) parametrów. Jeśli przejdziemy do bardziej realistycznego problemu, na przykład z 30 zmiennymi i jeśli każdy stan ma co najwyżej dwoje rodziców, w rozkładzie będzie najwyżej 240 elementów. Jeśli każdy stan ma trzech rodziców, maksymalna liczba elementów w rozkładzie wynosi 490: znacznie mniej niż miliard wymagany dla pełnego podejścia bayesowskiego! Musimy uzasadnić tę zależność węzła w sieci przekonań wyłącznie od jego rodziców. Powiązania między węzłami sieci przekonań reprezentują warunkowe prawdopodobieństwa wpływu przyczynowego. W rozumowaniu eksperckim za pomocą wnioskowania przyczynowego ukryte jest założenie, że wpływy te są ukierunkowane, to znaczy, że obecność jakiegoś zdarzenia powoduje w jakiś sposób inne zdarzenia w sieci. Co więcej, rozumowanie wpływu przyczynowego nie jest koliste, ponieważ jakiś skutek nie może zawrócić, by sam spowodować. Z tych powodów sieci przekonań bayesowskich będą miały naturalną reprezentację jako skierowany graf acykliczny lub DAG, w którym spójne wzorce rozumowania są odzwierciedlane jako ścieżki przez związki przyczynowo-skutkowe. Bayesowskie sieci przekonań są jednym z przykładów tego, co często nazywa się modelami graficznymi.

W przypadku przykładu ruchu mamy jeszcze silniejszą sytuację, bez cykli nieukierunkowanych. To pozwala nam w bardzo prosty sposób obliczyć rozkład prawdopodobieństwa w każdym węźle. Rozkład węzłów bez rodziców jest bezpośrednio sprawdzany. Wartości węzłów potomnych są obliczane przy użyciu tylko rozkładów prawdopodobieństwa rodziców każdego dziecka, wykonując odpowiednie obliczenia na tabeli prawdopodobieństwa warunkowego dziecka i rozkładach rodziców. Jest to

możliwe, ponieważ nie musimy martwić się o korelacje między rodzicami innych węzłów niebędących potomkami. Powoduje to naturalną separację abdukcyjną, w której wypadek nie ma żadnego związku z obecnością pomarańczowych beczek, jak widać na rysunku 14. Podsumowujemy naszą dyskusję na temat BBN za pomocą następującej definicji.

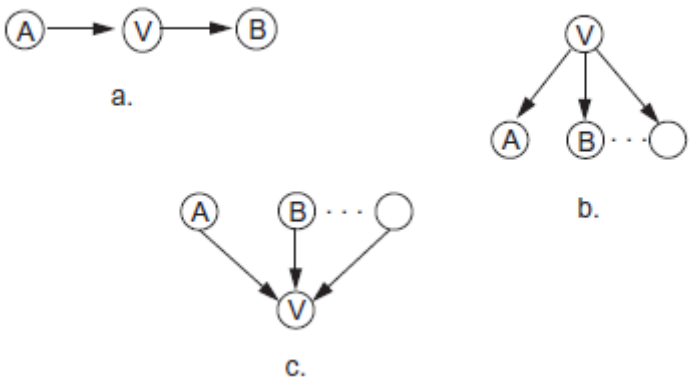
DEFINICJA

Bayesowka sieć przekonań

Model graficzny nazywa się siecią przekonań bayesowskich (BBN), jeśli jego wykres, opatrzony adnotacjami prawdopodobieństw warunkowych, jest ukierunkowany i acykliczny. Ponadto BBN zakładają, że węzły są niezależne od wszystkich ich nie-potomków, biorąc pod uwagę wiedzę o ich rodzicach. Dynamiczna (lub czasowa) sieć bayesowska (DBN) to sekwencja identycznych sieci bayesowskich, których węzły są połączone w (ukierunkowanym) wymiarze czasu. Ogólny DBM rozważymy szerzej w sekcji 9.3.4 oraz w Części 13. W następnej sekcji rozważymy założenie ukryte w większości ludzkiego rozumowania ekspertów: że obecność lub brak danych w domenie (niejawnie przyczynowej) może podzielić i ukierunkować poszukiwanie wyjaśnień w tej dziedzinie. Fakt ten ma również poważne konsekwencje dla złożoności eksploracji przestrzeni poszukiwań.

9.3.2 Ukierunkowane modele graficzne: separacja d

Ważną zaletą reprezentowania domen aplikacji jako modeli graficznych jest to, że obecność lub brak informacji może prowadzić do podziału modelu na partycje, a co za tym idzie do kontrolowania złożoności wyszukiwania. Następnie przedstawiamy kilka przykładów tego, a następnie podajemy definicję separacji d, która wspiera te intuicje. Rozważmy diagnozę problemów z olejem w silniku samochodowym: załóżmy, że zużyte pierścienie tłokowe powodują nadmierne zużycie oleju, co z kolei powoduje niski odczyt poziomu oleju w samochodzie. Sytuację tę ilustruje rysunek 15a, gdzie A to zużyte pierścienie tłokowe, V to nadmierne zużycie oleju, a B to niski poziom oleju. Jeśli nie wiemy nic o nadmiernym zużyciu oleju, to mamy związek przyczynowy między zużytymi pierścieniami tłokowymi a niskim poziomem oleju. Jeśli jednak jakiś test da wiedzę o stanie nadmiernego zużycia oleju, to zużyte pierścienie tłokowe i niski poziom oleju są od siebie niezależne. W drugim przykładzie: załóżmy, że zużyte pierścienie tłokowe mogą powodować zarówno niebieski wydech, jak i niski poziom oleju. Sytuację tę przedstawiono na rysunku 15b, gdzie V to zużyte pierścienie tłokowe, A to niebieski wydech, a B to niski poziom oleju.



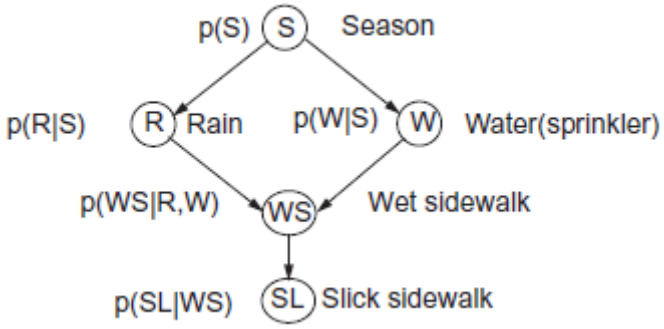
Jeśli wiemy, że zużyte pierścienie tłokowe są prawdziwe lub fałszywe, to nie wiemy, czy niebieski wydech i niski poziom oleju są skorelowane; jeśli nie mamy informacji o zużytych pierścieniach tłokowych, wówczas niebieski wydech i niski poziom oleju są skorelowane. Wreszcie, jeśli niski poziom

oleju może być spowodowany albo nadmiernym zużyciem oleju, albo wyciekiem oleju, to mając wiedzę o tym, czy jest niski poziom oleju, są skorelowane jego dwie możliwe przyczyny. Jeśli stan niskiego poziomu oleju nie jest znany, to dwie możliwe przyczyny są niezależne. Ponadto, jeśli prawdziwy jest niski poziom oleju, ustalenie wycieku oleju jako prawdziwego wyjaśni nadmierne zużycie oleju. W obu przypadkach informacja o niskim poziomie oleju jest kluczowym elementem w procesie wnioskowania. Widzimy tę sytuację na rysunku 15c, gdzie A to nadmierne zużycie oleju, B to wyciek oleju, a V to niski poziom oleju. Doprecyzowujemy te intuicje, definiując rozdzielenie d węzłów w sieci przekonań lub innym modelu graficznym:

DEFINICJA

d-SEPARACJA

Dwa węzły A i B w skierowanym acyklicznym grafie są rozdzielone d, jeśli każda ścieżka między nimi jest zablokowana. Ścieżka to dowolna ciągła seria połączeń na grafie (węzły łączące w dowolnym kierunku, np. Istnieje ścieżka od A do B na rysunku 15b). Ścieżka jest blokowana, jeśli na ścieżce znajduje się węzeł pośredni V z jedną z właściwości: połączenie jest szeregowe lub rozbieżne, a stan V jest znany, lub połączenie jest zbieżne i ani V, ani żadne z dzieci V nie mają dowodów. Podajemy dalsze przykłady relacji szeregowych, rozbieżnych i zbieżnych węzłów, a także jak separacja d wpływa na ścieżki argumentacji w przykładzie na rysunku 16.



Zanim opuścimy wykresy z rysunku 15, pokażemy, w jaki sposób założenia sieci przekonań bayesowskich mogą uprościć obliczanie prawdopodobieństw warunkowych. Korzystając z prawa Bayesa, każdy wspólny rozkład prawdopodobieństwa można rozłożyć na iloczyn prawdopodobieństw warunkowych. Tak więc na rysunku 15a. wspólne prawdopodobieństwo trzech zmiennych A, V, B wynosi:

p (A, V, B) = p (A) p (V | A) p (B | A, V).

Korzystamy z założenia sieci przekonań bayesowskich, że prawdopodobieństwo warunkowe zmiennej danej wiedzy wszystkich jej poprzedników jest równe prawdopodobieństwu warunkowemu danej wiedzy tylko jej rodziców. W rezultacie w powyższym równaniu p (B | A, V) staje się p (B | V), ponieważ V jest bezpośrednim rodzicem B, a A nie. Łączne rozkłady prawdopodobieństwa dla trzech sieci na rysunku 15 są następujące:

a) p (A, V, B) = p (A) p (V | A) p (B | V),

b) p (V, A, B) = p (V) p (A | V) p (B | V) i

c) p (A, B, V) = p (A) p (B) p (V | A, B).

Jak pokazał przykład ruchu, w przypadku większych sieci przekonań bayesowskich można wyeliminować znacznie więcej zmiennych z prawdopodobieństw warunkowych. To właśnie to uproszczenie sprawia, że sieci przekonań bayesowskich i inne modele graficzne są znacznie łatwiejsze do opracowania statystycznego niż pełna analiza bayesowska. Następnie przedstawiamy bardziej złożony model graficzny, zawierający cykl nieukierunkowany, i proponujemy skuteczny algorytm wnioskowania, propagację drzewa klikowego.

9.3.3 Ukierunkowane modele graficzne: algorytm wnioskowania

Następny przykład, zaadaptowany z Pearl (1988), pokazuje bardziej złożoną sieć bayesowską. Pora roku na rysunku 16 determinuje prawdopodobieństwo wystąpienia opadów deszczu oraz wody z instalacji tryskaczowej. Mokry chodnik będzie skorelowany z deszczem lub wodą z tryskacza. Wreszcie, chodnik będzie śliski w zależności od tego, czy jest to mokry chodnik, czy nie. Na rysunku przedstawiliśmy relację prawdopodobieństwa, jaką każdy z tych parametrów ma ze swoimi rodzicami. Należy również zauważyć, że w porównaniu z przykładem ruchu, przykład śliskiego chodnika ma cykl, jeśli wykres nie jest ukierunkowany. Zadajemy teraz pytanie, jak można opisać prawdopodobieństwo wystąpienia mokrego chodnika, p (WS)? Nie da się tego zrobić jak poprzednio, gdzie p(W) = p(W | S) p (S) lub p(R) = p(R | S) p(S). Dwie przyczyny WS nie są od siebie niezależne; na przykład, jeśli S = lato, to p(W) i p(R) mogą wzrosnąć. Tak więc pełne korelacje dwóch zmiennych, wraz z ich dalszą korelacją z S, należy obliczyć. W tej sytuacji możemy to zrobić, ale, jak zobaczymy, obliczenie to jest wykładnicze pod względem liczby możliwych przyczyn WS. Obliczenie przedstawiono w tabeli 4.

R	W	p(WS)
t	t	x } S=hot
t	f	} S=cold
f	t	
f	f	

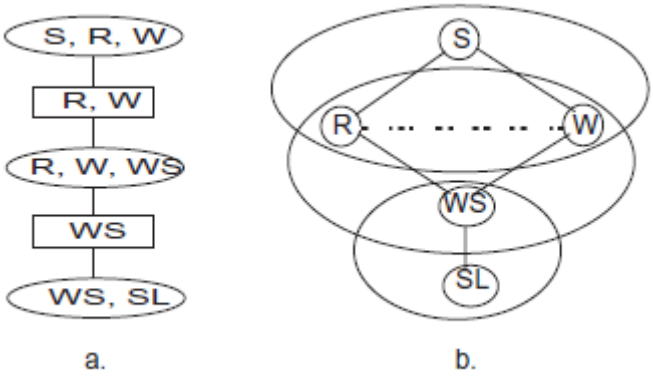
Teraz obliczamy jeden wpis w tej tabeli, x, gdzie R i W są prawdą; aby życie było prostsze, zakładamy, że sezon S jest albo hot , albo cold.

x = p(R = t, W = t) for the two (hot, cold) conditions of S, season

= p(S = hot) p(R = t | S = hot) p(W = t | S = hot) +

p(S = cold) p(R = t | S = cold) p(W = t | S = cold)

W podobny sposób można uzupełnić pozostałą część tabeli 4. Stanowi to wspólne prawdopodobieństwo wystąpienia opadów deszczu i wody z tryskacza. Ten większy „element makro” reprezentuje p (WS) = p (WS | R, W) p (R, W). Uciekliśmy tutaj z dość rozsądną kalkulacją; problem polega na tym, jak wspomniano powyżej, że to wyliczenie jest wykładnicze pod względem liczby rodziców państwa. Nazywamy ten element makra połączoną zmienną lub kliką do obliczania p (WS). Używamy tej koncepcji kliki w celu zastąpienia propagacji ograniczeń DAG z rysunku 16 acyklicznym drzewem kliki, jak widać na rysunku 17.



Prostokątne prostokąty na rysunku 17a odzwierciedlają zmienne, które dzielą grupy powyżej i poniżej. Tabela, która przekazuje odpowiednie parametry do następnej kliki, jest wykładnicza pod względem liczby tych parametrów. Należy również zauważyć, że zmienna łącząca wraz ze wszystkimi jej rodzicami musi być obecna w klice. Dlatego przy tworzeniu sieci przekonań lub innego modelu graficznego (procesu inżynierii wiedzy) powinniśmy uważać, ile zmiennych jest rodzicami dowolnego stanu. Kliki będą się również nakładać, jak widać na rysunku 17b, aby przekazywać informacje przez pełne drzewo klik, zwane drzewem skrzyżowań. Następnie przedstawiamy algorytm opracowany przez Laurintzena i Spiegelhaltera (1988), który tworzy drzewo połączeń z dowolnej sieci przekonań bayesowskich.

1. Dla wszystkich węzłów w sieci przekonań ustaw wszystkie skierowane linki jako nieukierunkowane.

2. Dla dowolnego węzła narysuj połączenia między wszystkimi jego rodzicami (przerywana linia między R i W na rysunku 17b).

3. Na otrzymanym wykresie poszukaj dowolnego cyklu o długości większej niż trzy i dodaj kolejne łącza, które skracają ten cykl do trzech. Ten proces nazywa się triangulacją i na przykładzie z rysunku 17b nie jest konieczny.

4. Utworzyć drzewo węzłowe z wynikową strukturą triangulacyjną. Odbywa się to poprzez znalezienie maksymalnych klik (klik, które są kompletnymi podgrafami, a nie podgrafami większej kliki) Zmienne w tych klikach są umieszczane w węzłach, a wynikowe drzewo węzłów jest tworzone przez połączenie dowolnych dwóch węzłów, które mają co najmniej jedną zmienną , jak na rysunku 17a.

Proces triangulacji opisany w kroku 3 powyżej jest krytyczny, ponieważ chcemy, aby wynikowe drzewo węzłów miało minimalny koszt obliczeniowy podczas propagowania informacji. Niestety, decyzja o stworzeniu optymalnych drzew węzłów kosztów jest NP trudna. Często na szczęście do uzyskania użytecznych wyników może wystarczyć prosty zachłanny algorytm. Zwróć uwagę, że rozmiary tabel wymagane do przekazania informacji w drzewie połączeń na rysunku 9.17 to 2 * 2 * 2, 2 * 2 * 2 i 2 * 2. Na koniec bierzemy przykładową sieć z rysunku 16 i wracamy do kwestii separacji d. Pamiętaj, że celem rozdzielenia d jest to, że przy niektórych informacjach części sieci przekonań można zignorować podczas obliczania rozkładów prawdopodobieństwa.

1. SL jest oddzielone d od R, S i W, biorąc pod uwagę, że WS jest znane.

2. D-separacja jest symetryczna, to znaczy S jest również d-oddzielona od (i nie jest to możliwe wyjaśnienie SL), biorąc pod uwagę wiedzę o WS.

3. R i W są zależne z powodu S, ale wiedząc, że S, R i W są rozdzielone d.

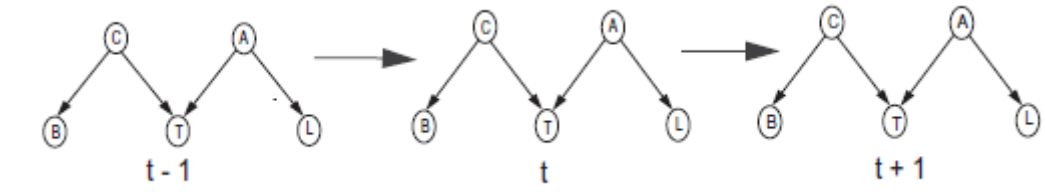
4. Jeśli znamy WS, to R i W nie są rozdzielone d; jeśli nie znamy S, to R i W są.

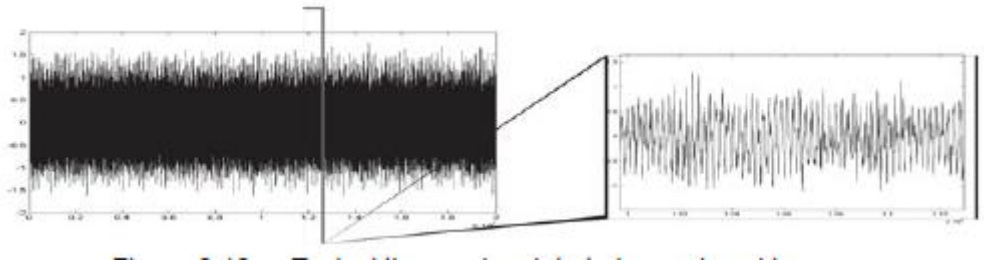
5. Mając łańcuch R -> WS -> SL, jeśli znamy WS, to R i SL są rozdzielone d.

Musimy być ostrożni, gdy znamy informacje o potomkach danego państwa. Na przykład, jeśli znamy SL, to R i W NIE są rozdzielone d, ponieważ SL jest skorelowane z WS, a wiedząc, że WS, R i W nie są rozdzielone d. Ostatni komentarz: Bayesowskie sieci przekonań wydają się odzwierciedlać sposób, w jaki ludzie rozumują w złożonych domenach, w których niektóre czynniki są znane i powiązane a priori z innymi. W miarę jak rozumowanie postępuje poprzez stopniowe tworzenie instancji informacji, wyszukiwanie jest jeszcze bardziej ograniczone iw rezultacie bardziej wydajne. Ta efektywność wyszukiwania silnie kontrastuje z podejściem wspieranym przez zastosowanie pełnego wspólnego rozkładu, gdzie więcej informacji wymaga wykładniczo większej potrzeby relacji statystycznych i wynikającego z tego szerszego wyszukiwania. Dostępnych jest wiele algorytmów służących do budowania sieci przekonań i propagowania argumentów w miarę pozyskiwania nowych dowodów. Szczególnie polecamy podejście z przekazywaniem wiadomości Pearl'a oraz metodę triangulacji drzew kliki zaproponowaną przez Lauritzena i Spiegelhaltera. Druzdel i Henrion również zaproponowali algorytmy propagacji wpływu w sieci. Dechter (1996) przedstawia algorytm eliminacji wiadra jako ujednolicającą strukturę wnioskowania probabilistycznego. W następnej sekcji wprowadzimy dynamiczne sieci bayesowskie.

9.3.4 Ukierunkowane modele graficzne: dynamiczne sieci bayesowskie

Następnie rozważymy uogólnienie sieci przekonań bayesowskich, dynamicznej (lub czasowej) sieci bayesowskiej lub DBN. Ideą wspierającą DBN jest przedstawienie stanu domeny lub czasu obserwacji za pomocą zestawu zmiennych losowych, a nie tylko pojedynczej zmiennej losowej. Dzięki temu rozszerzeniu sieci przekonań bayesowskich mogą być używane do reprezentowania warunkowej niezależności między zmiennymi z różnych perspektyw, na przykład w różnych okresach. Większość wydarzeń, z którymi stykamy się my, ludzie, i na które oczekuje się od nas inteligentnej reakcji, ma charakter czasowy: pojawiają się one w różnych okresach czasu. Dwa przykłady można zobaczyć na rysunkach18 i 19.



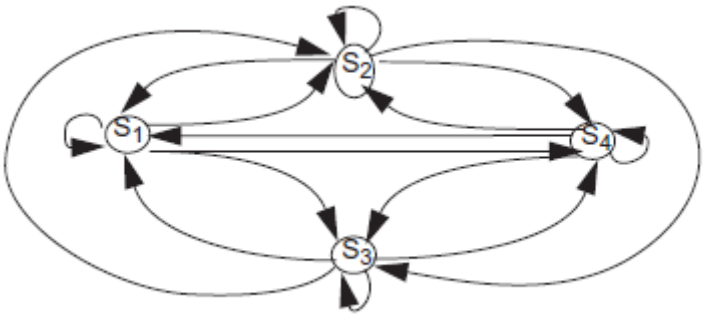


Na rysunku 18 wzięliśmy nasz model ruchu z rysunku 14 i zmapowaliśmy go w różnych okresach czasu. Może to uchwycić zmieniający się punkt widzenia kierowcy w czasie, powiedzmy co minutę, zostanie uchwycony przez zmieniające się parametry sieci bayesowskiej. Za pierwszym razem kierowca jest świadomy zwolnienia, za drugim zauważa obecność pomarańczowych beczek, w trzecim jest ich jeszcze więcej itd. Oczekuje się, że DBN uchwyci rosnącą świadomość. budowy dróg, wraz z (znacznie) mniejszą pewnością wypadku. Rysunek 9.19 przedstawia przykładowe dane sygnałów pochodzących

ze złożonego środowiska. (W rzeczywistości są to wielokrotne sygnały cieplne, wibracje i inne sygnały pochodzące z układu wirnika helikoptera). To, co DBN musi mierzyć w tej sytuacji, to jak te sygnały zmieniają się w czasie i informują kontrolera o aktualnym stanie pracy systemu. Rozważymy ten przykład dokładniej. Sieci DBN są najczęściej wykorzystywane do śledzenia dwóch typów systemów zmiennych w czasie. Po pierwsze, sekwencje zależne od czasu, takie jak opisane na rysunkach 18 i 19, a po drugie zjawiska naturalnie występujące w czasie, takie jak próbkowanie fonemów lub słów w aplikacji do analizy języka naturalnego. Należy zauważyć, że możliwe jest przedstawienie korelacji między zmiennymi losowymi w tym samym przedziale czasu (korelacje chwilowe) zarówno z skierowanymi, jak i niekierowanymi krawędziami modeli graficznych. Jednak korelacje między elementami wykresów odzwierciedlających dane szeregów czasowych muszą być uchwycone za pomocą ukierunkowanych modeli graficznych. Podsumowując: Jeśli każda krawędź łącząca serię modeli graficznych reprezentujących dane związane z czasem ma odzwierciedlać wymiar czasu, model nazywa się dynamiczną siecią Bayesa. DBN może również przechwytywać nieczasowe dane sekwencyjne, takie jak język, gdzie nowe informacje odzwierciedlają zmianę stanu przetwarzania.

9.3.5 Modele Markowa: dyskretny proces Markowa

Przedstawiliśmy maszyny skończone jako reprezentacje graficzne, w których stany są zmieniane w zależności od zawartości strumienia wejściowego. Stany i ich przejścia odzwierciedlają właściwości języka formalnego. Następnie przedstawiliśmy akceptor stanów skończonych (maszynę Moore'a), maszynę stanową, która potrafiła „rozpoznawać" łańcuchy o różnych właściwościach. Przedstawiliśmy probabilistyczną maszynę skończoną, maszynę stanu, w której następna funkcja stanu była reprezentowana przez rozkład prawdopodobieństwa stanu bieżącego. Dyskretny proces Markowa jest specjalizacją tego podejścia, w którym system ignoruje wartości wejściowe. Rysunek 20 to maszyna stanu Markowa, czasami nazywana łańcuchem Markowa, z czterema różnymi stanami.



Ta ogólna klasa systemu może być opisana w dowolnym okresie jako będąca w jednym ze zbioru S odrębnych stanów, s_1, s_2, s_3, ..., s_n. System podlega zmianom stanu, z możliwością pozostawania w tym samym stanie w regularnych, dyskretnych odstępach czasu. Opisujemy uporządkowany zbiór czasów T, które są powiązane z dyskretnymi przedziałami jako t_1, t_2, t_3, ..., t_t. System zmienia stan zgodnie z rozkładem prawdopodobieństw związanych z każdym stanem. Oznaczymy aktualny stan maszyny w czasie t jako σ_t. Pełny opis probabilistyczny tego systemu wymaga w ogólnym przypadku określenia stanu obecnego σ_t w odniesieniu do wszystkich jego stanów poprzedzających. Zatem prawdopodobieństwo, że system znajdzie się w jakimkolwiek określonym stanie σ_t, wynosi: $p(\sigma_t) = p(\sigma_t \mid \sigma_t - 1, \sigma_t - 2, \sigma_t - 3, ...)$ gdzie σ_t - i są poprzednimi stanami σ_t. W łańcuchu Markowa pierwszego rzędu prawdopodobieństwo obecnego stanu jest funkcją tylko jego bezpośredniego stanu poprzednika:

$p(\sigma_t) = p(\sigma_t \mid \sigma_t - 1)$

gdzie σ_t - 1 poprzedza σ_t. (Ogólnie rzecz biorąc, stan w łańcuchu Markowa rzędu n jest zależny od poprzedzających go n stanów). Zakładamy również, że prawa strona tego równania jest niezmienna w czasie, tj. Stawiamy hipotezę, że we wszystkich okresach czasu układu przejścia między określonymi stanami zachowują te same relacje probabilistyczne. Na podstawie tych założeń możemy teraz stworzyć zbiór prawdopodobieństw przejścia stanów a_{ij} między dowolnymi dwoma stanami s_i i s_j:

$a_{ij} = p(\sigma_t = s_i \mid \sigma_t - 1 = s_j), 1 \leq i, j \leq N$

Przypomnij sobie, że mogę równać się j, w którym to przypadku system pozostaje w tym samym stanie. Na rozkładach prawdopodobieństwa pozostają tradycyjne ograniczenia; dla każdego stanu s_i:

$a_{ij} \geq 0$ oraz

System, który właśnie opisaliśmy, nazywany jest obserwowalnym modelem Markowa pierwszego rzędu, ponieważ wyjściem systemu jest zbiór stanów w każdym dyskretnym przedziale czasu, a każdy stan systemu odpowiada zdarzeniu fizycznemu (obserwowalnemu). Sprawiamy, że obserwowalny model Markowa jest bardziej formalny z definicją, a następnie podajemy przykład.

DEFINICJA

(OBSERWOWALNY) MODEL MARKOWA

Model graficzny nazywamy (obserwowalnym) modelem Markowa, jeśli jego wykres jest ukierunkowany, a prawdopodobieństwo dotarcia do dowolnego stanu s_t ze zbioru stanów S w dyskretnym czasie t jest funkcją rozkładów prawdopodobieństwa jego bycia w poprzednich stanach S w poprzednich czasach. Każdy stan st S odpowiada fizycznie obserwowalnej sytuacji.

Obserwowalny model Markowa jest pierwszego rzędu, jeśli prawdopodobieństwo, że znajdzie się on w stanie obecnym s_t w dowolnym momencie t jest jedynie funkcją tego, że był w stanie poprzednim s_t - 1 w chwili t - 1, gdzie s_t i s_t - 1 należą do zbioru obserwowalnych stanów S. Zauważ, że każdy rozkład prawdopodobieństwa ma właściwość bycia modelem Markowa. Siła tego podejścia wynika z założeń pierwszego rzędu. Jako przykład obserwowalnego modelu Markowa pierwszego rzędu, rozważmy pogodę w południe, powiedzmy, dla określonej lokalizacji. Zakładamy, że ta lokalizacja ma cztery różne stany dyskretne dla zmiennej pogody: s_1 = słońce, s_2 = pochmurno, s_3 = mgła, s_4 = opady. Przedziały czasowe dla modelu Markowa będą w południe każdego kolejnego dnia. Zakładamy również, że rany między stanami pogody pozostają stałe w czasie (nie jest to prawda dla większości lokalizacji!), A obserwowalna pogoda może pozostawać w tym samym stanie przez wiele dni. Ta sytuacja jest przedstawiona na rysunku 20 i jest obsługiwana przez macierz przejść stanów a_{ij}:

a_{ij} =

	s_1	s_2	s_3	s_4
s_1	0.4	0.3	0.2	0.1
s_2	0.2	0.3	0.2	0.3
s_3	0.1	0.3	0.3	0.3
s_4	0.2	0.3	0.3	0.2

W tej macierzy przejść a_{ij}, , pierwszy rząd przedstawia przejścia od s_1 do każdego ze stanów , w tym przebywanie w tym samym stanie; drugi rząd to przejścia z s_2 i tak dalej. Właściwości wymagane, aby przejścia były rozkładami prawdopodobieństwa z każdego stanu są spełnione (ich suma wynosi 1,0). Teraz zadajemy pytania dotyczące naszego modelu. Załóżmy, że dzisiaj, s_1, jest słonecznie; jakie jest

prawdopodobieństwo, że następnych pięć dni pozostanie słonecznych? Albo jakie jest prawdopodobieństwo następnych pięciu dni są słoneczne, słoneczne, pochmurne, pochmurne, opady? Aby rozwiązać ten drugi problem, określamy prawdopodobieństwo i określamy, że pierwszy dzień, s1, to dzisiejsze obserwowane słońce: O = s1, s1, s1, s2, s2, s4

Prawdopodobieństwo tych obserwowanych stanów, biorąc pod uwagę model Markowa pierwszego rzędu, M, wynosi:

$$
\begin{aligned}
p(O \mid M) &= p(s_1, s_1, s_1, s_2, s_2, s_4 \mid M) \\
&= p(s_1) \times p(s_1 \mid s_1) \times p(s_1 \mid s_1) \times p(s_2 \mid s_1) \times p(s_2 \mid s_2) \times p(s_4 \mid s_2) \\
&= 1 \times a_{11} \times a_{11} \times a_{12} \times a_{22} \times a_{24} \\
&= 1 \times (.4) \times (.4) \times (.3) \times (.3) \times (.3) \\
&= .00432
\end{aligned}
$$

Wynika to z założenia Markowa pierwszego rzędu, gdzie pogoda każdego dnia jest funkcją (tylko) pogody z poprzedniego dnia. Obserwujemy też, że dziś jest słońce. Możemy rozszerzyć ten przykład, aby określić, biorąc pod uwagę, że znamy dzisiejszą pogodę, prawdopodobieństwo, że pogoda będzie taka sama przez dokładnie następnych t dni, tj. że pogoda pozostaje taka sama do dnia t + 1, w którym to czasie jest inna. Dla dowolnego stanu pogody si i modelu Markowa M mamy obserwację O:

O = $\{s_i, s_i, s_i, ..., s_i, s_j\}$, gdzie jest dokładnie (t + 1) s_i i gdzie $s_i \neq s_j$, to:

$p(O|M) = 1 \times a_{ii}^{t} \times (1 - a_{ii})$

gdzie a_{ii} to prawdopodobieństwo przejścia stanu si do siebie. Wartość ta nazywana jest dyskretną funkcją gęstości prawdopodobieństwa dla czasu trwania t okresów w stanie si modelu M. Ta funkcja gęstości czasu trwania wskazuje na czas trwania stanu w modelu Markowa. Na podstawie tej wartości możemy obliczyć, w ramach modelu M, oczekiwaną liczbę obserwacji lub czas trwania di w dowolnym stanie si, zakładając, że pierwsza obserwacja jest w tym stanie:

$$
\begin{aligned}
d_i &= \sum_{d=1}^{n} d(a_{ii})^{(d-1)}(1 - a_{ii}) \quad \text{where n approaches } \infty, \text{ or:} \\
&= \frac{1}{1 - a_{ii}}
\end{aligned}
$$

Na przykład oczekiwana liczba kolejnych dni opadów, w tym modelu, wynosi 1 / (1 - 0,3) lub 1,43. Podobnie liczba kolejnych słonecznych dni, których można by się spodziewać w przypadku tego modelu, wynosi 1,67. Następnie podsumujemy różne formy modeli Markowa

9.3.6 Modele Markowa: wariacje

W modelach Markowa, które widzieliśmy do tej pory, każdy stan odpowiada dyskretnemu fizycznemu - i możliwemu do zaobserwowania - zdarzeniu, takim jak wartość pogody o określonej porze dnia. Ta klasa modelu jest naprawdę dość ograniczona i teraz uogólniamy ją na szerszą klasę modeli. Opisujemy pokrótce kilka z tych podejść. Rozdział 13.1 oferuje bardziej wszechstronną prezentację ukrytych modeli Markowa wraz z kilkoma ważnymi odmianami.

Ukryte modele Markowa

W przedstawionych już modelach Markowa każdy stan odpowiada dyskretnemu fizycznemu i obserwowalnemu wydarzeniu. W ukrytym modelu Markowa lub HMM zakłada się, że obserwowane wartości są funkcjami probabilistycznymi bieżącego stanu ukrytego. Na przykład, obserwowany dźwięk mowy ludzkiej jest odbiciem telefonu, który ma zamiar odtworzyć rozmówca. Wypowiadany dźwięk jest tylko probabilistycznie powiązany z rzeczywistym stanem lub intencjami mówiącego. Zatem HMM jest podwójnie osadzonym procesem stochastycznym, w którym obserwowalny proces stochastyczny (hałas mówiącego) jest wspomagany przez nieobserwowalny proces stochastyczny (stan lub intencje mówiącego). HMM jest szczególnym przypadkiem DBN

Modele semi-Markowa

Model semi-Markov to dwuskładnikowy łańcuch Markowa. Pierwsza składowa odpowiada za wybór kolejnego przejścia stanu, a druga za czas przejścia. Kiedy model semi-Markowa wchodzi w stan s_i pozostaje w tym stanie dla czasu t_i, który jest pobierany z rozkładu prawdopodobieństwa trwania stanu. Po upływie czasu t_i proces przechodzi do następnego stanu s_{i+1} zgodnie z rozkładem prawdopodobieństwa przejścia między stanami i ponownie wybiera się czas przejścia t_{i+1}. Model semi-Markowa obsługuje dowolne rozkłady prawdopodobieństwa czasu trwania stanu w porównaniu ze stałymi przejściami czasowymi określonymi przez tradycyjne modele Markowa.

Procesy decyzyjne Markowa

Dwa inne warianty modeli Markowa, szeroko stosowane w uczeniu się ze wzmocnieniem, to proces decyzyjny Markowa (MDP) i częściowo obserwowalny proces decyzyjny Markowa, r POMDP. MDP jest zdefiniowany w dwóch przestrzeniach: przestrzeni stanów dla rozpatrywanego problemu oraz przestrzeni możliwych działań. Przejście do nowego stanu w przestrzeni stanów jest zależne od stanu obecnego i aktualnego działania i jest kierowane przez warunkowy rozkład prawdopodobieństwa. W każdym stanie obliczana jest nagroda na podstawie obecnego stanu i działania. Zwykle zadaniem uczenia się przez wzmacnianie jest maksymalizacja skumulowanej funkcji nagrody w obecnych warunkach. Podczas gdy MDP działa ze stanem obserwowanym (deterministycznym) przed użyciem macierzy przejścia opartej na prawdopodobieństwie do wybrania następnego stanu, POMDP ma tylko probabilistyczną wiedzę o swoim obecnym stanie (a także o macierzy probabilistycznej do określenia następnego stanu). W ten sposób POMDP może okazać się trudny obliczeniowo w złożonych środowiskach.

9.3.7 Alternatywy pierwszego rzędu dla BBN w modelowaniu probabilistycznym

Do tego punktu w sekcji 9.3 przedstawiliśmy reprezentacje oparte na rachunku zdań dla wnioskowania z niepewnością. Naturalne jest pytanie, w jakim sensie modele oparte na logice predykatów probabilistycznych mogą być użyte do wnioskowania. Siłą pełnej reprezentacji logiki predykatów jest semantyka deklaratywna wraz z ramami reprezentacji opartymi na zmiennych. Ograniczenia reprezentacji tradycyjnych podejść opartych na logice zdań obejmują ograniczoną zdolność radzenia sobie z szumem i niepewnością, a także statyczny obraz reprezentacji. Zatem modele Baysian i Markovian przedstawione w sekcjie 9.3 są ograniczone do reprezentacji na poziomie propozycji, gdzie przedstawienie ogólnych (opartych na zmiennych) relacji z rozkładem, na przykład, ∀X male (X) → smart(X) (0,49 0,51). Wiele złożonych dziedzin problemowych wymaga tego poziomu ekspresji. W przypadku złożonych dziedzin, szczególnie w obszarach diagnozy i rokowania, ważne jest również, aby móc reprezentować powtarzalne, rekurencyjne i (potencjalnie) nieskończone struktury. Dlatego jeśli chcemy reprezentować systemy, które zmieniają stan w czasie, musimy mieć tę reprezentacyjną moc. Wymaga to rekurencyjnego lub powtarzanego do czasu wykonania reżimu kontroli typu. Jeśli chcemy dynamicznie przebudować tradycyjny BBN, jesteśmy zmuszeni do rekonstrukcji całej sieci w okresach czasu, ponieważ tym strukturom brakuje deklaratywnej i zorientowanej na czas semantyki. Niedawna

działalność badawcza doprowadziła do powstania wielu ważnych rozszerzeń modeli graficznych, w tym szeregu systemów pierwszego rzędu (opartych na zmiennych i rekurencyjnych). Istnieje również szereg hierarchicznych i rozkładalnych modeli bayesowskich. Wymieniamy kilka z tych nowych formalizmów modelowania, które wspierają taką reprezentację.

Budowa sieci bayesowskiej na podstawie baz wiedzy

Haddawy, Ngo i współpracownicy łączą razem system logiczny pierwszego rzędu z sieciami przekonań bayesowskimi. W rezultacie tworzą logikę probabilistyczną pierwszego rzędu używaną jako język reprezentacji do określania pewnej klasy sieci w postaci bazy wiedzy. Semantyka formalna bazy wiedzy (bayesowskiej) jest egzekwowana za pomocą ściśle określonego języka reprezentacji określającego zdania bazy. Haddawy i Ngo dostarczyły sprawdzony, poprawny algorytm generowania sieci bayesowskiej, implementujący kombinację semantyki formalnej dla bazowego języka logicznego, a także semantyki niezależności dla bazy wiedzy. Jedną z cech algorytmu generującego jest to, że unika on tworzenia nieistotnych węzłów dla sieci przy użyciu dowodów określonych przez użytkownika. Haddawy i Ngo argumentują, że rozumowanie staje się bardzo nieefektywne, gdy wiedza podstawa staje się bardzo duża. Jako środek zaradczy proponują wykorzystanie informacji kontekstowych do indeksowania probabilistycznych zdań bazy wiedzy. Kiedy algorytm generowania modelu konstruuje sieć, pomija zdania z bazy wiedzy, których kontekst nie jest istotny do bieżącego zadania. Wynikowy model jest znacznie mniejszy, niż gdyby do budowy modelu wykorzystano całą bazę wiedzy. Podejście Haddawy, Ngo i współpracowników jest jedną z pierwszych prób badawczych w dziedzinie stochastycznego modelowania logicznego, które jawnie wykorzystuje kontekstowe informacje o domenie jako sposób na skoncentrowanie bazy wiedzy na odpowiednich informacjach i zmniejszenie rozmiaru modelu. konieczne do rozwiązania określonego zadania rozumowania.

Programy logiki bayesowskiej, BLP.

Programy logiki bayesowskiej oferują ramy reprezentacyjne, w tym programowanie logiczne i modele graficzne. Dokładniej rzecz biorąc, BLP łączą sieci przekonań bayesowskich z logiką klauzuli Horn. Na podstawie określonego zapytania system generuje sieć bayesowską w celu udzielenia odpowiedzi na zapytanie przy użyciu zestawu reguł pierwszego rzędu z parametrami niepewności. Wynikowa reprezentacja jest łatwa do zinterpretowania, a jej projektanci przypisują jej teoretyczną semantykę modelu podobną do tej przedstawionej w Cześci 2.

Probabilistyczne modele relacyjne, PRM

Friedman, Getoor i ich współpracownicy opracowali inne podejście do systemów probabilistycznych pierwszego rzędu. Zamiast tworzyć deklaratywny język podobny do logiki, PRM określają model prawdopodobieństwa dla klas obiektów. Osoby o ograniczonej możliwości poruszania się definiują ograniczenia prawdopodobieństwa na poziomie klasy, dzięki czemu ograniczenia te mogą być używane z dowolnym obiektem z tej klasy. W przypadku PRM możliwe jest również określenie probabilistycznych zależności między atrybutami powiązanych klas. Szereg rozszerzeń PRM pozwala na podklasy wspierające probabilistykę zależności na odpowiednich poziomach szczegółowości. Hierarchie klas pozwalają na modelowanie zarówno na poziomie jednostek (tradycyjna sieć przekonań bayesowskich), jak i klas. Dalszym rozszerzeniem PRM jest traktowanie struktur relacyjnych samych modeli jako niepewnych, a nie ustalonych. Z tej perspektywy istnieje ścisły związek między PRM a relacyjnymi bazami danych, ponieważ mogą one mieć podobną strukturę.

Sieci logiczne Markowa, sieci MLN.

Richardson i Domingos proponują inny system oparty na logice probabilistycznej, zwany sieciami logicznymi Markowa (MLN). Większość poprzednich opartych na logice podejść do modelowania stochastycznego wykorzystuje ograniczone podzbiory logiki ogólnej, przy czym klauzule Horn są najbardziej typową reprezentacją. Aby usunąć ograniczenia logiki, Richardson i Domingos używają ogólnej logiki pierwszego rzędu, której zdania są konwertowane na spójną formę normalną (CNF). Wykorzystują również pola losowe Markowa jako probabilistyczny odpowiednik logiki, co jest kolejną istotną różnicą w stosunku do wcześniej opisanych systemów (sieci przekonań bayesowskich są zdecydowanie najczęściej używaną reprezentacją). W konsekwencji mapowanie zdań logicznych w CNF na pola losowe Markowa jest proste. MLN to pierwsze ramy teoretyczne z pełnym mapowaniem od rachunku predykatów pierwszego rzędu z symbolami funkcji do rozkładów prawdopodobieństwa.

Loopy Logic

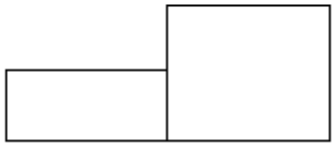
Logika pętli jest deklaratywnym językiem probabilistycznym pierwszego rzędu i kompletnym według Turinga. Swoją nazwę zawdzięcza wykorzystaniu zapętlonego algorytmu propagacji przekonań Pearl do wnioskowania. Jego oparte na logice deklaratywne reprezentacje są najpierw tłumaczone na pola losowe Markowa. Uczenie się parametrów z algorytmem maksymalizacji oczekiwań (EM) dobrze współgra z zapętloną propagacją przekonań. Zapętlony system logiczny, wraz z wykorzystaniem pola losowego Markowa i uczenia parametrów w oparciu o EM. Istnieje wiele dalszych rozszerzeń tradycyjnych BBN, które sprawiają, że są one pierwszym zamówieniem i Turingiem. Dalsze przykłady modelowania stochastycznego pierwszego rzędu obejmują język IBAL Pffefera oraz Stochastic Lambda Calculus of Pless i Luger. Zorientowane obiektowo reprezentacje stochastyczne obejmują (Koller i Pfeffer 1997, 1998; Pfeffer i wsp. 1999; Xiang i wsp. 2000; Pless i wsp. 2000). Metody stochastyczne są ważne w całej dziedzinie sztucznej inteligencji; patrz, na przykład, czynniki probabilistyczne.

Sztuczna inteligencja : Uczenie maszynowe : Oparte na symbolach

(X / XVI)

ĆWICZENIA

1. Rozważ zachowanie programu Winston do uczenia się koncepcji podczas nauki pojęcia „krok”, gdzie krok składa się z krótkiego pudełka i wysokiego pudełka umieszczonych w kontakcie, jak na rysunku



Stwórz semantyczne reprezentacje sieci trzech lub czterech przykładów i potencjalnych błędów i pokaż rozwój koncepcji.

2. Przebieg algorytmu eliminacji kandydatów pokazany na rysunku 9 nie pokazuje koncepcji kandydatów, które zostały stworzone, ale wyeliminowane, ponieważ były albo zbyt ogólne, zbyt szczegółowe, albo podciągnięte przez jakąś inną koncepcję. Ponownie wykonaj ślad wykonania, pokazując te koncepcje i powody, dla których każdy z nich został wyeliminowany.

3. Zbuduj algorytmy przeszukiwania przestrzeni wersji w Prologu lub w wybranym przez siebie języku. Jeśli używasz Prologu lub Javy, zapoznaj się ze wskazówkami dotyczącymi wyszukiwania w przestrzeni wersji w materiałach pomocniczych.

4. Korzystając z teoretycznej funkcji selekcji informacji przedstawionej w sekcji 10.4.3, pokaż szczegółowo, jak ID3 konstruuje drzewo z rysunku 14 z przykładów w tabeli 1. Pamiętaj, aby pokazać obliczenia użyte do obliczenia przyrostu informacji dla każdego testu i wyniku wyboru testoweo.

5. Korzystając ze wzoru Shannona, pokaż, czy komunikat o wyniku obrotu koła ruletki zawiera więcej informacji niż komunikat o wyniku rzutu monetą. Co się stanie, jeśli komunikat koła ruletki to „nie 00”?

6. Opracuj prostą tabelę przykładów w jakiejś dziedzinie, takiej jak klasyfikacja zwierząt według gatunku, i prześledź konstrukcję drzewa decyzyjnego za pomocą algorytmu ID3.

7. Zaimplementuj ID3 w wybranym przez siebie języku i uruchom go na przykładzie historii kredytowej z tekstu.

8. Omów problemy, które mogą wyniknąć z używania atrybutów ciągłych w danych, takich jak koszt pieniężny, dolary i centy lub wysokość, liczba rzeczywista, jednostki. Zaproponuj metodę rozwiązania tego problemu ciągłych danych.

9. Inne problemy ID3 to złe lub brakujące dane. Dane są złe, jeśli jeden zestaw atrybutów ma dwa różne wyniki. Brakuje danych, jeśli nie ma części atrybutu, być może dlatego, że uzyskanie ich było zbyt drogie. Jak można rozwiązać te problemy podczas opracowywania algorytmów ID3?

10. Z Quinlana uzyskaj algorytm drzewa decyzyjnego C4.5 i przetestuj go na zbiorze danych. Istnieją kompletne programy i zbiory danych dla C4.5 dostępne z tego źródła.

11. Opracuj teorię dziedzinową dla uczenia się opartego na wyjaśnieniach w wybranym obszarze problemowym. Prześledź zachowanie ucznia opartego na wyjaśnieniach, stosując tę teorię do kilku przypadków szkoleniowych.

12) Opracuj algorytm uczenia się oparty na wyjaśnieniach w wybranym przez siebie języku. Jeśli używasz Prologu, rozważ algorytmy opracowane w materiale pomocniczym.

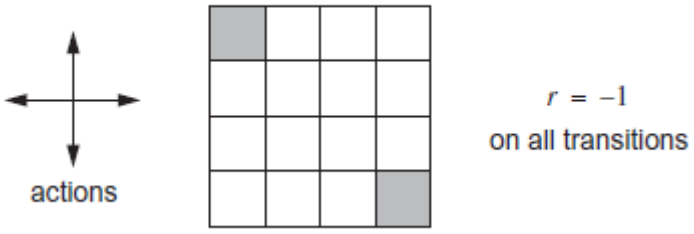
13. Rozważmy przykład gry w kółko i krzyżyk w sekcji 10.7.2. Zaimplementuj algorytm uczenia się różnic czasowych w wybranym przez siebie języku. Jeśli zaprojektowałeś algorytm tak, aby uwzględniał symetrie problemu, czego się spodziewasz? Jak może to ograniczyć nasze rozwiązanie?

14. Co się stanie, jeśli algorytm różnicy czasowej z Problemu 13 gra przeciwko sobie w kółko i krzyżyk?

15. Przeanalizuj program gry w warcaby Samuela z perspektywy uczenia się przez wzmacnianie.

16. Czy możesz przeanalizować problem odwróconego wahadła, z perspektywy uczenia się przez wzmacnianie? Zbuduj kilka prostych miar nagrody i zastosuj algorytm różnic czasowych w swojej analizie.

17. Innym typem problemu, który jest doskonały do uczenia się przez wzmacnianie, jest tak zwany świat sieci. Na rysunku 26 przedstawiamy prosty świat siatki 4 x 4. Dwa szare rogi to żądane stany terminala agenta. Ze wszystkich innych stanów ruch agenta jest w górę, w dół, w lewo lub w prawo. Agent nie może wyjść z sieci: próbując, pozostawia stan bez zmian. Nagroda za wszystkie przejścia, z wyjątkiem stanów końcowych, wynosi -1. Przeanalizuj sekwencję siatek, które tworzą rozwiązanie oparte na algorytmie różnic czasowych przedstawionym w sekcji 10.7.2.



UCZENIE MASZYNOWE: OPARTE NA SYMBOLACH

10.0 Wprowadzenie

Zdolność uczenia się musi być częścią każdego systemu, który twierdzi, że posiada ogólną inteligencję. Rzeczywiście, w naszym świecie symboli i interpretacji samo pojęcie niezmiennego intelektu wydaje się sprzecznością terminów. Inteligentni agenci muszą być zdolni do zmiany w trakcie swoich interakcji

ze światem, a także poprzez doświadczenie własnych stanów i procesów wewnętrznych. Przedstawiamy cztery rozdziały poświęcone uczeniu maszynowemu, odzwierciedlające cztery podejścia do problemu: pierwszy, oparty na symbolach, drugi, koneksjonistyczny, trzeci, genetyczny / ewolucyjny i wreszcie dynamiczny / stochastyczny. Uczenie się jest ważne dla praktycznych zastosowań sztucznej inteligencji. Feigenbaum i McCorduck (1983) nazwali „wąskie gardło inżynierii wiedzy" główną przeszkodą w powszechnym stosowaniu inteligentnych systemów. Tym „wąskim gardłem" jest koszt i trudność tworzenia systemów przy użyciu tradycyjnych technik pozyskiwania wiedzy. Jednym z rozwiązań tego problemu byłoby rozpoczęcie programów z minimalną ilością wiedzy i uczenie się na przykładach, poradach wysokiego poziomu lub własnych eksploracjach domeny aplikacji. Herbert Simon definiuje uczenie się jako:

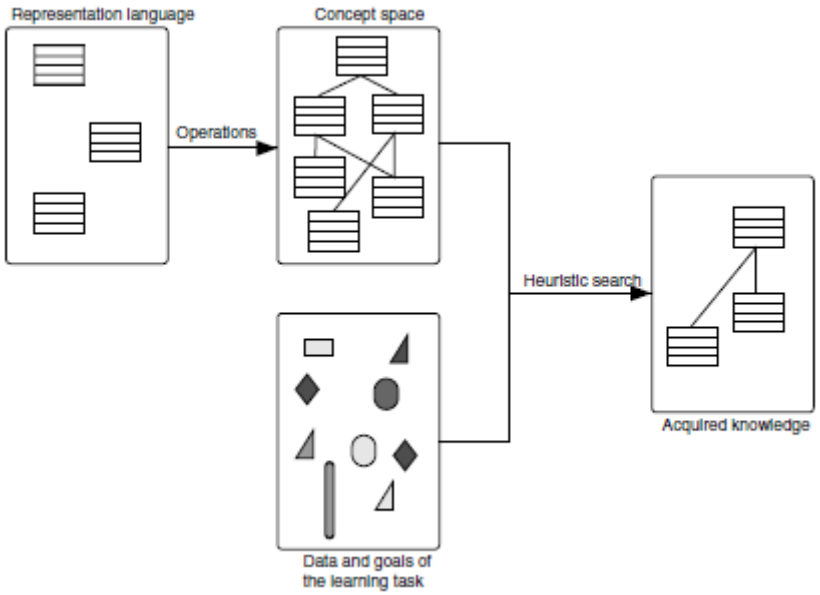
jakakolwiek zmiana w systemie, która pozwala mu działać lepiej za drugim razem przy powtarzaniu tego samego zadania lub przy innym zadaniu z tej samej populacji.

Ta definicja, choć zwięzła, sugeruje wiele kwestii związanych z opracowywaniem programów uczących się. Nauka wymaga uogólnienia na podstawie doświadczenia: wydajność powinna się poprawić nie tylko w przypadku „powtarzania tego samego zadania", ale także w przypadku podobnych zadań w domenie. Ponieważ interesujące domeny są zwykle rozległe, uczący się zazwyczaj analizuje tylko ułamek wszystkich możliwych przykładów; z tego ograniczonego doświadczenia uczący się musi poprawnie uogólniać na niewidoczne przypadki domeny. Na tym polega problem integracji zawodowej, który ma zasadnicze znaczenie dla uczenia się. W większości problemów z uczeniem się dostępne dane nie są wystarczające, aby zagwarantować optymalne uogólnienie, niezależnie od zastosowanego algorytmu. Uczniowie muszą uogólniać heurystycznie, to znaczy muszą wybierać te aspekty swojego doświadczenia, które z największym prawdopodobieństwem okażą się skuteczne w przyszłości. Takie kryteria wyboru nazywane są błędami indukcyjnymi. Definicja Simona opisuje uczenie się jako pozwalające systemowi „działać lepiej za drugim razem". Jak wskazuje poprzedni akapit, wybranie możliwych zmian w systemie, które pozwolą na jego usprawnienie, jest trudnym zadaniem. Badania naukowe muszą uwzględniać możliwość, że zmiany mogą faktycznie obniżyć wydajność. Zapobieganie i wykrywanie takich problemów to kolejna kwestia dotycząca algorytmu uczenia się. Uczenie się wiąże się ze zmianami ucznia; to jest jasne. Jednak dokładny charakter tych zmian i najlepszy sposób ich przedstawienia nie są oczywiste. Jedno podejście modeluje uczenie się jako zdobywanie jawnie reprezentowanej wiedzy dziedzinowej. W oparciu o swoje doświadczenie uczeń konstruuje lub modyfikuje wyrażenia w języku formalnym, takim jak logika, i zachowuje tę wiedzę do wykorzystania w przyszłości. Podejścia symboliczne, scharakteryzowane przez algorytmy z rozdziału 10, opierają się na założeniu, że głównym wpływem na zachowanie programu jest jego zbiór jawnie reprezentowanej wiedzy dziedzinowej. Sieci neuronowe lub koneksjonistyczne, nie uczą się poprzez przyswajanie zdań w języku symbolicznym. Podobnie jak mózg zwierzęcy, który składa się z dużej liczby połączonych ze sobą neuronów, sieci łącznikowe są systemami połączonych ze sobą sztucznych neuronów. Wiedza jest zawarta w organizacji i interakcji tych neuronów. Sieci neuronowe nie uczą się, dodając reprezentacje do bazy wiedzy, ale modyfikując swoją ogólną strukturę, aby dostosować się do okoliczności świata, w którym zamieszkują. Rozważymy uczenie się genetyczne i ewolucyjne. Ten model uczenia się, jaki posiadamy, można zaobserwować w systemach ludzkich i zwierzęcych, które ewoluowały w kierunku równowagi ze światem. To podejście do uczenia się poprzez adaptację znajduje odzwierciedlenie w algorytmach genetycznych, programowaniu genetycznym i badaniach nad sztucznym życiem. Wreszcie, przedstawiamy dynamiczne i probabilistyczne podejście do uczenia się. Wiele złożonych lub nie do końca zrozumiałych sytuacji na świecie, takich jak generowanie i rozumienie ludzkiego języka, można najlepiej „zrozumieć" za pomocą narzędzi probabilistycznych. Podobnie można zdiagnozować układy dynamiczne, takie jak usterki w pracującym silniku mechanicznym. W tej Części rozpoczynamy badanie uczenia maszynowego od podejścia

opartego na symbolach, w którym algorytmy różnią się pod względem celów, dostępnych danych szkoleniowych oraz stosowanych przez nie strategii uczenia się i języków reprezentacji wiedzy. Jednak wszystkie te algorytmy uczą się, przeszukując przestrzeń możliwych koncepcji w celu znalezienia akceptowalnego uogólnienia. W sekcji 10.1 zarysujemy ramy uczenia maszynowego opartego na symbolach, które podkreślają wspólne założenia leżące u podstaw całej tej pracy. Chociaż sekcja 10.1 przedstawia w zarysie różnorodne zadania związane z uczeniem się, cały rozdział skupia się głównie na nauczaniu indukcyjnym. Indukcja, która polega na uczeniu się uogólnień na podstawie zestawu przykładów, jest jednym z najbardziej podstawowych zadań edukacyjnych. Uczenie się pojęć jest typowym problemem związanym z uczeniem się indukcyjnym: biorąc pod uwagę przykłady pewnych pojęć, takich jak „kot", „choroba soi" lub „dobra inwestycja w akcje", próbujemy wywnioskować definicję, która pozwoli uczniowi poprawnie rozpoznać przyszłe przypadki tę koncepcję. W sekcjach 10.2 i 10.3 omówiono dwa algorytmy używane do wprowadzania pojęć, przeszukiwania przestrzeni wersji i ID3. Sekcja 10.4 dotyczy roli indukcyjnego uprzedzenia w uczeniu się. Przestrzenie poszukiwań napotykane podczas uczenia się są zazwyczaj bardzo duże, nawet jak na standardy rozwiązywania problemów w oparciu o wyszukiwanie. Te złożone problemy są zaostrzone przez problem wyboru wśród różnych uogólnień wspieranych przez dane szkoleniowe. Odchylenie indukcyjne odnosi się do każdej metody, którą program uczący się używa do ograniczenia przestrzeni możliwych uogólnień. Algorytmy z sekcji 10.2 i 10.3 są oparte na danych. Nie wykorzystują wcześniejszej wiedzy z dziedziny uczenia się, ale opierają się na dużej liczbie przykładów, aby zdefiniować podstawowe właściwości ogólnej koncepcji. Algorytmy, które generalizują na podstawie wzorców w danych uczących, nazywane są podobieństwami. W przeciwieństwie do metod opartych na podobieństwach, uczący się może wykorzystać wcześniejszą wiedzę w tej dziedzinie do kierowania generalizacją. Na przykład ludzie nie potrzebują dużej liczby przykładów, aby skutecznie się uczyć. Często pojedynczy przykład, analogia lub ogólna rada wystarczy przekazać ogólną koncepcję. Skuteczne wykorzystanie takiej wiedzy może pomóc agentowi uczyć się skuteczniej i przy mniejszym prawdopodobieństwie popełnienia błędu. Rozdział 10.5 dotyczy uczenia się opartego na wyjaśnieniach, uczenia się przez analogię i innych technik wykorzystujących wcześniejszą wiedzę do uczenia się na podstawie ograniczonych danych szkoleniowych. Algorytmy przedstawione w sekcjach od 10.2 do 10.5, mimo że różnią się strategiami wyszukiwania, językami reprezentacji i ilością wykorzystanej wcześniejszej wiedzy, zakładają, że dane szkoleniowe są klasyfikowane przez nauczyciela lub w inny sposób. Uczący się dowiaduje się, czy dana instancja jest pozytywnym czy negatywnym przykładem docelowej koncepcji. To poleganie na instancjach szkoleniowych o znanej klasyfikacji definiuje zadanie nadzorowanego uczenia się. Rozdział 10.6 kontynuuje badania nad wprowadzaniem poprzez badanie uczenia się bez nadzoru, które dotyczy tego, jak inteligentny agent może zdobyć użyteczną wiedzę w przypadku braku prawidłowo sklasyfikowanych danych szkoleniowych. Tworzenie kategorii lub grupowanie pojęciowe jest podstawowym problemem w uczeniu się bez nadzoru. Biorąc pod uwagę zbiór obiektów wykazujących różne właściwości, w jaki sposób agent może podzielić te obiekty na przydatne kategorie? Skąd w ogóle wiemy, czy kategoria będzie przydatna? W tej sekcji przeanalizujemy CLUSTER / 2 i COBWEB, dwa algorytmy tworzenia kategorii. Wreszcie w sekcji 10.7 przedstawiamy uczenie się przez wzmacnianie. W tym przypadku agent znajduje się w środowisku i otrzymuje informacje zwrotne z tego kontekstu. Uczenie się wymaga od agenta działania, a następnie interpretowania informacji zwrotnych z tych działań. Uczenie się ze wzmocnieniem różni się od uczenia się pod nadzorem tym, że nie ma „nauczyciela" bezpośrednio reagującego na każde działanie; raczej agent sam musi stworzyć strategię interpretowania wszystkich informacji zwrotnych. Uczenie się ze wzmocnieniem dobrze pasuje do konstruktywistycznej epistemologii, jak opisano w sekcji 16.2. Całe nauczanie przedstawione w tym rozdziale ma jedną wspólną cechę: jest postrzegane jako różnorodne przeszukiwanie przestrzeni stanów. Nawet uczenie się ze wzmocnieniem wywodzi funkcję wartości z przestrzeni stanów. Następnie przedstawiamy ogólną strukturę opartą na wyszukiwaniu do pracy w uczeniu maszynowym

10.1 Struktura uczenia się za pomocą symboli

Algorytmy uczące się można scharakteryzować w kilku wymiarach, jak pokazano na rysunku 1:



1. Dane i cele zadania edukacyjnego. Jednym z głównych sposobów charakteryzowania problemów w uczeniu się jest uwzględnienie celów osoby uczącej się i danych, które otrzymuje. Na przykład algorytmy uczenia się pojęć w sekcjach 10.2 i 10.3 rozpoczynają się od zbioru pozytywnych (i zazwyczaj negatywnych) przykładów klasy docelowej; celem jest wywnioskowanie ogólnej definicji, która pozwoli uczniowi rozpoznać przyszłe wystąpienia tej klasy. W przeciwieństwie do podejścia wykorzystującego duże ilości danych stosowanego przez te algorytmy, uczenie się oparte na wyjaśnieniach (sekcja 10.5) próbuje wywnioskować ogólną koncepcję na podstawie pojedynczego przykładu szkoleniowego i wcześniejszej bazy wiedzy specyficznej dla domeny. Koncepcyjne algorytmy grupowania omówione w sekcji 10.6 ilustrują inną odmianę problemu indukcji: zamiast zestawu instancji uczących o znanej kategoryzacji, algorytmy te zaczynają się od zbioru niesklasyfikowanych instancji. Ich zadaniem jest odkrycie kategoryzacji, które mogą być przydatne dla ucznia.

Przykłady nie są jedynym źródłem danych treningowych. Na przykład ludzie często uczą się z porad wysokiego szczebla. Nauczając programowania, profesorowie na ogół mówią swoim studentom, że wszystkie pętle muszą osiągnąć warunek kończący. Ta rada, choć jest poprawna, nie jest bezpośrednio przydatna: musi zostać przetłumaczona na określone zasady manipulowania licznikami pętli lub warunkami logicznymi w języku programowania. Analogie to kolejny rodzaj danych uczących, które muszą być poprawnie zinterpretowane, zanim będą mogły być użyte. Jeśli nauczyciel mówi uczniowi, że elektryczność jest jak woda, uczeń musi wywnioskować prawidłowy cel analogii: kiedy woda przepływa przez rurę, prąd przepływa przez drut. Podobnie jak w przypadku płynącej wody, możemy zmierzyć ilość energii elektrycznej (natężenie prądu) i ciśnienie za przepływem (napięcie). Jednak w przeciwieństwie do wody energia elektryczna nie zmoczy rzeczy ani nie pomoże nam myć rąk. Interpretacja analogii polega na znajdowaniu znaczących podobieństw i unikaniu fałszywych lub pozbawionych znaczenia wniosków.

Możemy również scharakteryzować algorytm uczenia się na podstawie celu lub celu ucznia. Celem wielu algorytmów uczenia się jest koncepcja lub ogólny opis klasy obiektów. Algorytmy uczące się mogą również pozyskiwać plany, heurystyki rozwiązywania problemów lub inne formy wiedzy proceduralnej. Właściwości i jakość samych danych szkoleniowych to kolejny wymiar, według którego

klasyfikujemy zadania edukacyjne. Dane mogą pochodzić od nauczyciela ze środowiska zewnętrznego lub mogą być generowane przez sam program. Dane mogą być wiarygodne lub mogą zawierać szum. Może być prezentowany w dobrze zorganizowany sposób lub składać się z niezorganizowanych danych. Może zawierać zarówno przykłady pozytywne, jak i negatywne lub tylko przykłady pozytywne. Dane mogą być łatwo dostępne, program może wymagać konstruowania eksperymentów lub wykonywania innej formy zbierania danych.

2. Reprezentacja zdobytej wiedzy. Programy uczenia maszynowego wykorzystywały wszystkie języki reprezentacji omówione w tym tekście. Na przykład programy, które uczą się klasyfikować obiekty, mogą przedstawiać te pojęcia jako wyrażenia w rachunku predykatów lub mogą używać reprezentacji strukturalnej, takiej jak ramki lub obiekty. Plany można opisać jako sekwencję operacji lub jako tabelę trójkątną. Heurystyki można przedstawić jako reguły rozwiązywania problemów. Proste sformułowanie problemu uczenia się pojęć przedstawia przypadki pojęcia jako zdania łączące zawierające zmienne. Na przykład dwa przypadki słowa „piłka” (niewystarczające do opanowania pojęcia) mogą być reprezentowane przez:

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)

size(obj2, large) ∧ color(obj2, red)∧ shape(obj2, round)

Ogólne pojęcie „piłki” można zdefiniować następująco:

size(X, Y) ∧ color(X, Z) ∧ shape(X, round)

gdzie każde zdanie, które łączy się z tą ogólną definicją, reprezentuje piłkę.

3. Zestaw operacji. Biorąc pod uwagę zestaw instancji szkoleniowych, uczeń musi skonstruować uogólnienie, regułę heurystyczną lub plan, który spełni jego cele. Wymaga to umiejętności manipulowania reprezentacjami. Typowe operacje obejmują uogólnianie lub specjalizowanie wyrażeń symbolicznych, dostosowywanie wag w sieci neuronowej lub modyfikowanie reprezentacji programu w inny sposób.

W przedstawionym właśnie przykładzie uczenia się koncepcji uczący się może uogólnić definicję, zastępując stałe zmiennymi. Jeśli zaczniemy od koncepcji:

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)

zastąpienie pojedynczej stałej zmienną daje uogólnienia:

size(obj1, X) ∧ color(obj1, red) ∧ shape(obj1, round)

size(obj1, small) ∧ color(obj1, X) ∧ shape(obj1, round)

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, X)

size(X, small) ∧ color(X, red) ∧ shape(X, round)

4. Przestrzeń pojęciowa. Język reprezentacji, wraz z opisanymi powyżej operacjami, definiuje przestrzeń potencjalnych definicji pojęć. Uczący się musi przeszukać tę przestrzeń, aby znaleźć pożądane pojęcie. Złożoność tej przestrzeni pojęciowej jest podstawową miarą trudności problemu uczenia się.

5. Przeszukiwanie heurystyczne. Programy uczenia się muszą uwzględniać kierunek i kolejność wyszukiwania, a także wykorzystywać dostępne dane szkoleniowe i heurystykę w celu wydajnego wyszukiwania. W naszym przykładzie uczenia się pojęcia „piłka” wiarygodny algorytm może przyjąć

pierwszy przykład jako potencjalną koncepcję i uogólnić go, uwzględniając kolejne przykłady. Na przykład po jednym przykładzie szkolenia

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)

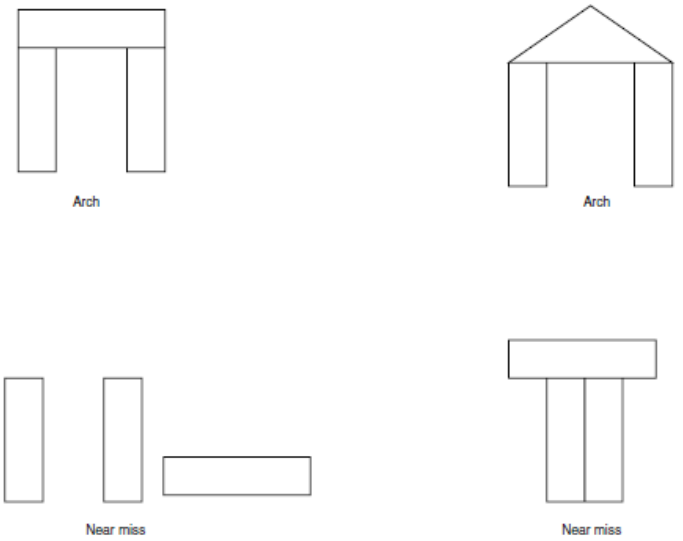
uczący się uczyni ten przykład koncepcją kandydata; ta koncepcja poprawnie klasyfikuje jedyny pozytywny przypadek. Jeśli algorytm otrzyma drugą pozytywną instancję

size(obj2, large) ∧ color(obj2, red) ∧ shape(obj2, round)

uczący się może uogólniać koncepcję kandydata, zastępując stałe zmiennymi w razie potrzeby, aby stworzyć koncepcję pasującą do obu przypadków. W rezultacie otrzymujemy bardziej ogólną koncepcję kandydata, która jest bliższa naszej docelowej koncepcji „piłki”

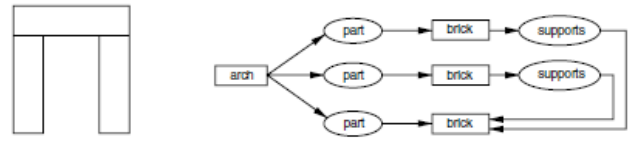
size(X, Y) ∧ color(X, red) ∧ shape(X, round)

Praca Patricka Winstona dotycząca uczenia się pojęć na podstawie pozytywnych i negatywnych przykładów ilustruje te elementy. Jego program uczy się ogólnych definicji pojęć strukturalnych, takich jak „łuk”, w świecie bloków. Dane szkoleniowe to seria pozytywnych i negatywnych przykładów koncepcji: przykłady blokowych struktur świata, które pasują do kategorii, wraz z bliskimi trafieniami. Te ostatnie to przypadki, które prawie należą do kategorii, ale zawodzą w przypadku jednej właściwości lub relacji. Zdarzenia potencjalnie wypadkowe umożliwiają programowi wyodrębnienie cech, których można użyć do wykluczenia negatywnych wystąpień z docelowej koncepcji. Rysunek 2 przedstawia pozytywne przykłady i sytuacje potencjalnie wypadkowe dla pojęcia „łuk”.

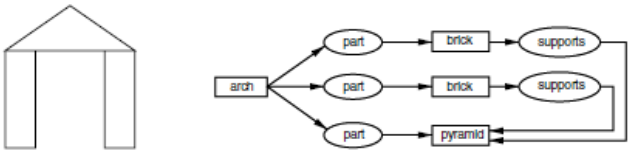


Program przedstawia koncepcje jako sieci semantyczne, jak na rysunku 3.

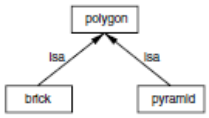
a. An example of an arch and its network description



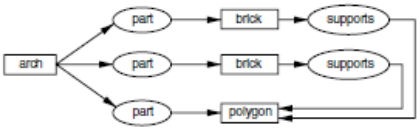
b. An example of another arch and its network description



c. Given background knowledge that bricks and pyramids are both types of polygons

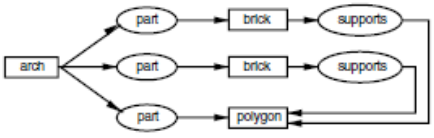


d. Generalization that includes both examples

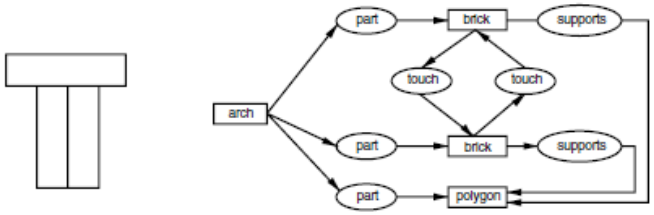


Uczy się poprzez udoskonalanie kandydata opisu koncepcji docelowej w miarę prezentowania instancji szkoleniowych. Program Winston udoskonala opisy kandydatów poprzez uogólnienie i specjalizację. Uogólnienie zmienia wykres, aby umożliwić uwzględnienie nowych przykładów koncepcji. Rysunek 3a przedstawia łuk zbudowany z trzech cegieł i opisujący go wykres. Następny przykład treningowy, Rysunek 3b, to łuk z piramidą, a nie cegłą na górze. Ten przykład nie pasuje do opisu kandydata. Program dopasowuje te wykresy, próbując znaleźć częściowy izomorfizm między nimi. Graf dopasowujący wykorzystuje nazwy węzłów do kierowania procesem dopasowywania. Gdy program dopasuje wykresy, może wykryć różnice między nimi. Na rysunku 3 wykresy są zgodne na wszystkich komponentach, z wyjątkiem tego, że górny element na pierwszym wykresie jest ceglany, a odpowiadający mu węzeł w drugim przykładzie to piramida. Część podstawowej wiedzy programu stanowi hierarchia uogólnień tych pojęć, rysunek 3c. Program uogólnia wykres, zastępując ten węzeł najmniej powszechnym nadtypem cegły i piramidy; w tym przykładzie jest to wielokąt. Wynikiem jest koncepcja z rysunku 3d. W przypadku przedstawienia bliskiego braku, przykładu, który różni się od koncepcji docelowej w pojedynczej właściwości, program specjalizuje się w opisie kandydata, aby wykluczyć przykład. Rysunek 10.4a przedstawia opis kandydata. Różni się od bliskiego zdarzenia z rysunku 10.4b relacjami dotykowymi przykładu bliskiego chybienia. Program specjalizuje się w wykresie, dodając łącza, których nie wolno dotykać, aby wykluczyć sytuację potencjalnie niebezpieczną, rysunek 10.4c. Zauważ, że algorytm zależy w dużej mierze od bliskości negatywnych przykładów z koncepcją docelową. Różniąc się od celu tylko jedną właściwością, bliskie chybienie pomaga algorytmowi dokładnie określić, w jaki sposób wyspecjalizować koncepcję kandydata.

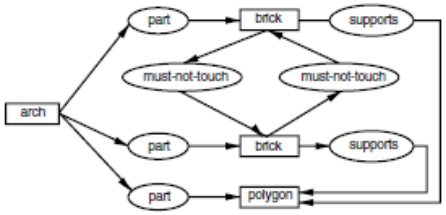
a. Candidate description of an arch



b. A near miss and its description



c. Arch description specialized to exclude the near miss



Te operacje - specjalizacja sieci poprzez dodawanie łączy i uogólnianie jej poprzez zastępowanie nazw węzłów lub łączy bardziej ogólną koncepcją - definiują przestrzeń możliwych definicji pojęć. Program Winstona polega na wyszukiwaniu wspinaczki górskiej w przestrzeni koncepcyjnej na podstawie danych treningowych. Ponieważ program nie cofa się, jego wydajność jest bardzo wrażliwa na kolejność przykładów szkoleniowych; zła kolejność może doprowadzić program do ślepych zaułków w przestrzeni wyszukiwania. Instancje szkoleniowe muszą być prezentowane w programie w takiej kolejności, która pomaga w nauce pożądanej koncepcji, podobnie jak nauczyciel organizuje lekcje, aby pomóc uczniowi w nauce. Jakość i kolejność przykładów szkoleniowych są również ważne dla algorytmu dopasowywania wykresów programu; efektywne dopasowywanie wymaga, aby wykresy nie były zbyt różne. Chociaż program Winstona jest wczesnym przykładem uczenia indukcyjnego, ilustruje cechy i problemy wspólne dla większości technik uczenia maszynowego z rozdziału 10: wykorzystanie operacji generalizacji i specjalizacji do zdefiniowania przestrzeni pojęciowej, wykorzystanie danych do kierowania wyszukiwaniem w tej przestrzeni oraz wrażliwość algorytmu uczenia się na jakość danych szkoleniowych. W kolejnych sekcjach omówiono te problemy i techniki opracowane przez uczenie maszynowe w celu ich rozwiązania.

10.2 Przeszukiwanie przestrzeni wersji

Przeszukiwanie przestrzeni wersji ilustruje implementację uczenia indukcyjnego jako przeszukiwania przestrzeni pojęć. Przeszukiwanie przestrzeni wersji wykorzystuje fakt, że operacje uogólniające narzucają uporządkowanie pojęć w przestrzeni, a następnie wykorzystuje to uporządkowanie do kierowania wyszukiwaniem.

10.2.1 Operatory uogólnienia i przestrzeń pojęciowa

Uogólnienie i specjalizacja to najpowszechniejsze typy operacji służących do definiowania przestrzeni koncepcji. Podstawowe operacje generalizacji używane w uczeniu maszynowym to:

1. Zastąpienie stałych zmiennymi. Na przykład,

color(ball, red)

uogólnia do

color(X, red)

2. Usuwanie warunków z wyrażenia łączącego.

shape(X, round) ∧ size(X, small) ∧ color(X, red)

uogólnia

shape(X, round) ∧ color(X, red)

3. Dodawanie rozłącznika do wyrażenia.

shape(X, round) ∧ size(X, small) ∧ color(X, red)

uogólnia

shape(X, round) ∧ size(X, small) ∧ (color(X, red) ∨ color(X, blue))

4. Zastąpienie właściwości jej rodzicem w hierarchii klas. Jeśli wiemy, że primary_color jest superklasą czerwieni, to

color(X, red)

uogólnia

color(X, primary_color)

Możemy myśleć o uogólnieniu w terminach teorii zbiorów: niech P i Q będą zbiorami zdań pasujących do wyrażeń rachunku predykatów, odpowiednio, p i q. Wyrażenie p jest bardziej ogólne niż q iff P ⊇ Q. W powyższych przykładach zestaw zdań pasujących do koloru (X, czerwony) zawiera zestaw elementów pasujących do koloru (piłka, czerwony). Podobnie w przykładzie 2, możemy myśleć o zbiorze okrągłych, czerwonych rzeczy jako o zbiorze małych, czerwonych, okrągłych rzeczy. Zauważ, że relacja „bardziej ogólna niż" definiuje częściowe uporządkowanie w przestrzeni zdań logicznych. Wyrażamy to za pomocą symbolu „≥", gdzie p ≥ q oznacza, że p jest bardziej ogólne niż q. To uporządkowanie jest potężnym źródłem ograniczeń wyszukiwania przeprowadzanego przez algorytm uczący się. Formalizujemy ten związek poprzez pojęcie pokrycia. Jeśli pojęcie p jest bardziej ogólne niż pojęcie q, mówimy, że p obejmuje q. Definiujemy relację pokryw: niech p (x) i q (x) będą opisami, które klasyfikują przedmioty jako pozytywne przykłady pojęcia. Innymi słowy, dla obiektu x, p (x) → dodatni (x) i q (x) → dodatni (x). p obejmuje q iff q (x) → dodatni (x) jest logiczną konsekwencją p (x) → dodatni (x). Na przykład kolor (X, Y) obejmuje kolor (kula, Z), który z kolei pokrywa kolor (kula, czerwony). Jako prosty przykład rozważ dziedzinę obiektów, które mają właściwości i wartości:

Sizes = {large, small}

Colors = {red, white, blue}

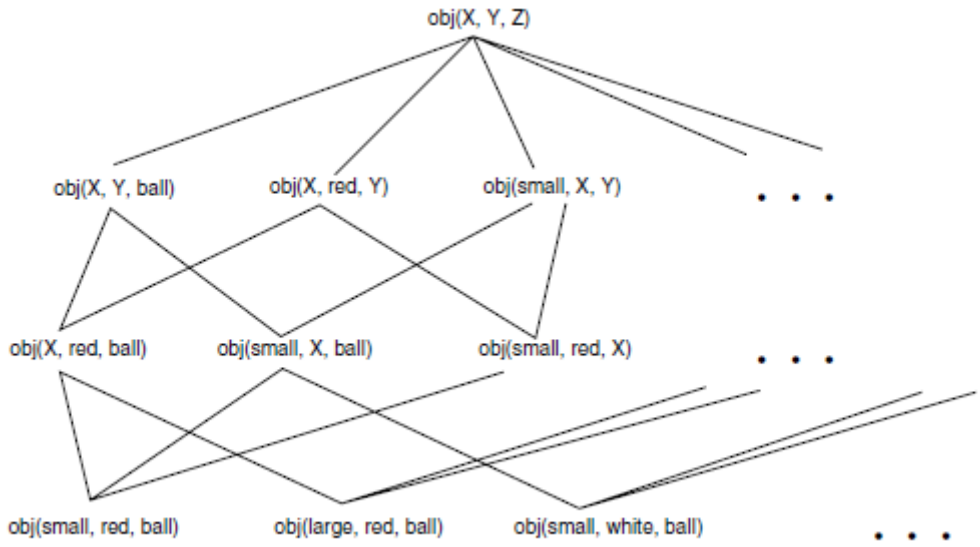
Shapes = {ball, brick, cube}

Obiekty te można przedstawić za pomocą predykatu obj (Sizes, Color, Shapes). Operacja uogólniająca polegająca na zamianie stałych na zmienne definiuje przestrzeń z rysunku 10.5. Możemy postrzegać uczenie się indukcyjne jako poszukiwanie w tej przestrzeni koncepcji zgodnej ze wszystkimi przykładami szkoleniowymi.

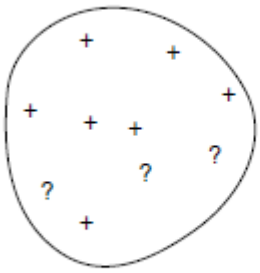
10.2.2 Algorytm eliminacji kandydatów

W tej sekcji przedstawiono trzy algorytmy do przeszukiwania przestrzeni pojęć. Algorytmy te opierają się na pojęciu przestrzeni wersji, która jest zbiorem wszystkich opisów koncepcji zgodnych z przykładami szkoleniowymi. Algorytmy te działają na zasadzie zmniejszania rozmiaru przestrzeni wersji, gdy dostępnych jest więcej przykładów. Pierwsze dwa algorytmy redukują przestrzeń wersji odpowiednio w określonym kierunku i ogólnym do określonego kierunku. Trzeci algorytm, zwany eliminacją kandydatów, łączy te podejścia w przeszukiwanie dwukierunkowe. Następnie opiszemy i oceniamy te algorytmy.

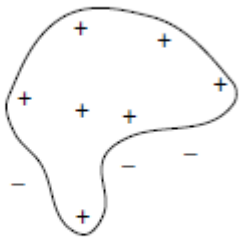
Algorytmy te są oparte na danych: uogólniają na podstawie prawidłowości znalezionych w danych szkoleniowych. Ponadto, korzystając z danych szkoleniowych o znanej klasyfikacji, algorytmy te wykonują różne nadzorowane uczenie. Podobnie jak w przypadku programu Winston do nauki opisów strukturalnych, przeszukiwanie przestrzeni wersji wykorzystuje zarówno pozytywne, jak i negatywne przykłady koncepcji docelowej. Chociaż można uogólniać tylko na podstawie pozytywnych przykładów, negatywne przykłady są ważne, aby zapobiec nadmiernemu uogólnianiu algorytmu. Idealnie, wyuczona koncepcja musi być na tyle ogólna, aby objąć wszystkie pozytywne przykłady, a także musi być wystarczająco szczegółowa, aby wykluczyć wszystkie negatywne przykłady. W miejscu na rysunku 5,



jedną koncepcją, która obejmowałaby wszystkie zbiory wyłącznie pozytywnych instancji, byłaby po prostu obj (X, Y, Z). Jednak ta koncepcja jest prawdopodobnie zbyt ogólna, ponieważ oznacza, że wszystkie instancje należą do koncepcji docelowej. Jednym ze sposobów uniknięcia nadmiernej generalizacji jest jak najmniejsze uogólnianie, aby uwzględnić pozytywne przykłady; innym jest użycie negatywnych przypadków w celu wyeliminowania zbyt ogólnych pojęć. Jak ilustruje rysunek 6



Concept induced from
positive examples only



Concept induced from
positive and negative examples

, negatywne przypadki zapobiegają nadmiernej generalizacji, zmuszając uczącego się do specjalizacji pojęć w celu wykluczenia negatywnych przypadków. Algorytmy w tej sekcji wykorzystują obie te techniki. Specyficzne dla wyszukiwania ogólnego, dla zbioru hipotez S definiujemy jako:

Begin

For every s ∈ S, if s does not match p, replace s with its most specific

generalization that matchs p;

Delete from S all hypotheses more general than some other hypothesis in S;

Delete from S all hypotheses that match a previously observed negative

instance in N;

End;

For every negative instance n

Begin

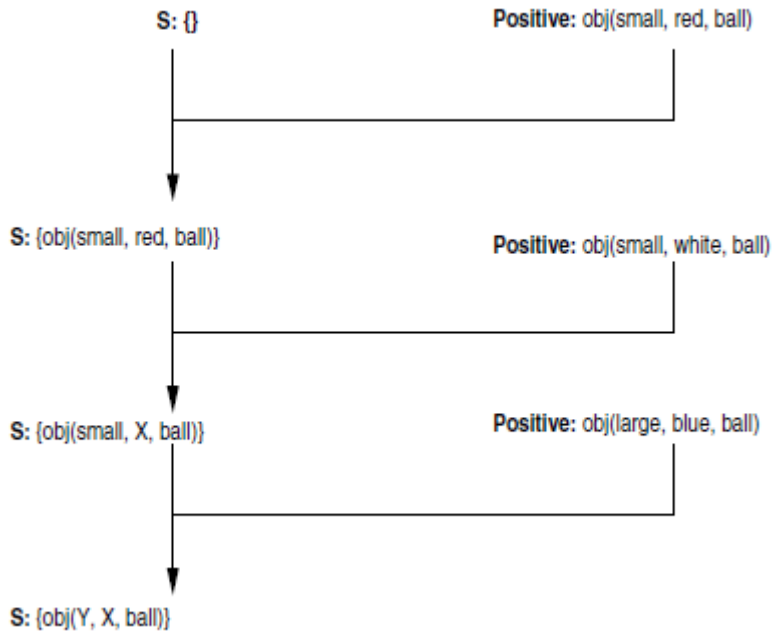
Delete all members of S that match n;

Add n to N to check future hypotheses for overgeneralization;

End;

End

Specyficzne dla wyszukiwania ogólnego zachowuje zbiór, S, hipotez lub potencjalnych definicji pojęć. Aby uniknąć nadmiernej uogólnienia, te definicje kandydatów są maksymalnie szczegółowymi uogólnieniami z danych szkoleniowych. Pojęcie c jest maksymalnie specyficzne, jeśli obejmuje wszystkie przykłady pozytywne, żadnego z przykładów negatywnych, a dla każdego innego pojęcia c ’, które obejmuje przykłady pozytywne, c ≤ c ’. Rysunek 7 przedstawia przykład zastosowania tego algorytmu do przestrzeni wersji z rysunku 5.



Specyficzny dla ogólnego algorytmu wyszukiwania w przestrzeni wersji jest wbudowany w Prolog w naszych materiałach pomocniczych. Możemy również wyszukiwać od ogólnego do konkretnego kierunku. Algorytm ten utrzymuje zbiór, G, maksymalnie ogólnych pojęć, które obejmują wszystkie pozytywne i żadne negatywne przypadki. Pojęcie c jest maksymalnie ogólne, jeśli nie obejmuje żadnego z negatywnych instancji szkoleniowych, a dla każdego innego pojęcia c ’, które nie obejmuje negatywnej instancji szkoleniowej, c ≥ c ’. W tym algorytmie negatywne przypadki prowadzą do specjalizacji koncepcji kandydatów; algorytm wykorzystuje pozytywne instancje, aby wyeliminować zbyt wyspecjalizowane koncepcje.

Begin

Initialize G to contain the most general concept in the space;

P contains all positive examples seen so far;

For each negative instance n

Begin

For each g ∈ G that matches n, replace g with its most general specializations

that do not match n;

Delete from G all hypotheses more specific than some other hypothesis in G;

Delete from G all hypotheses that fail to match some positive example in P;

End;

For each positive instance p

Begin

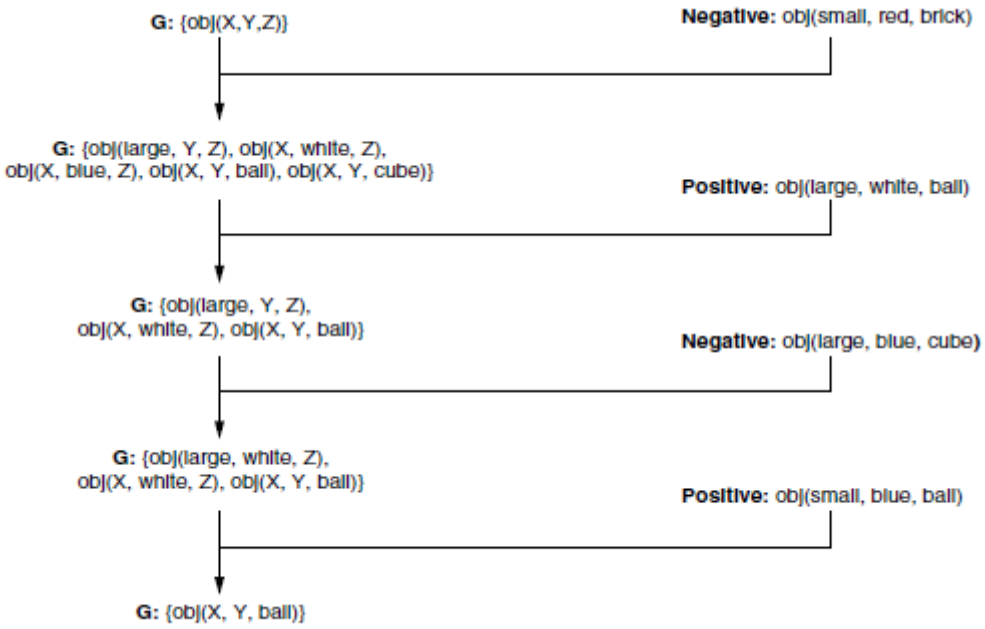
Delete from G all hypotheses that fail to match p;

Add p to P;

End;

End

Rysunek 8 przedstawia przykład zastosowania tego algorytmu do przestrzeni wersji z rysunku 10.5.



W tym przykładzie algorytm wykorzystuje wiedzę podstawową, że rozmiar może mieć wartości {large, small}, kolor może mieć wartości {red, white, blue}, a kształt może mieć wartości {ball, brick, cube}. Ta wiedza jest niezbędna, jeśli algorytm ma specjalizować koncepcje poprzez zastępowanie zmiennych stałych. Algorytm eliminacji kandydatów łączy te podejścia w przeszukiwanie dwukierunkowe. To dwukierunkowe podejście ma wiele zalet w nauce. Algorytm utrzymuje dwa zbiory pojęć kandydatów: G - zbiór maksymalnie ogólnych pojęć kandydatów i S - zbiór maksymalnie konkretnych kandydatów. Algorytm specjalizuje się w G i generalizuje S, dopóki nie zbiegają się one z koncepcją docelową. Algorytm jest zdefiniowany:

Begin

Initialize G to be the most general concept in the space;

Initialize S to the first positive training instance;

For each new positive instance p

Begin

Delete all members of G that fail to match p;

For every s ∈ S, if s does not match p, replace s with its most specific

generalizations that match p;

Delete from S any hypothesis more general than some other hypothesis in S;

Delete from S any hypothesis more general than some hypothesis in G;

End;

For each new negative instance n

Begin

Delete all members of S that match n;

For each g ∈ G that matches n, replace g with its most general specializations

that do not match n;

Delete from G any hypothesis more specific than some other hypothesis in G;

Delete from G any hypothesis more specific than some hypothesis in S;

End;

If G = S and both are singletons, then the algorithm has found a single concept that

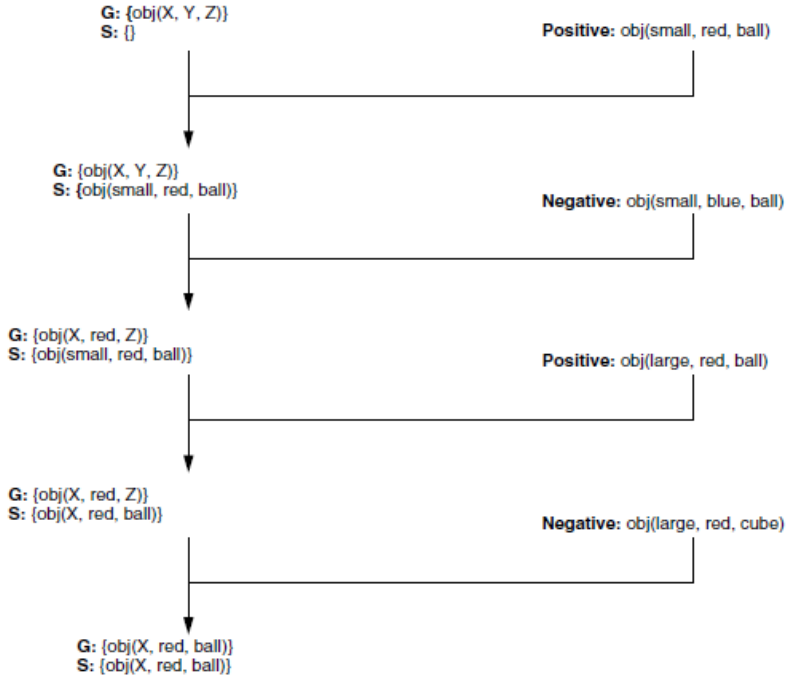
is consistent with all the data and the algorithm halts;

If G and S become empty, then there is no concept that covers all positive instances

and none of the negative instances;

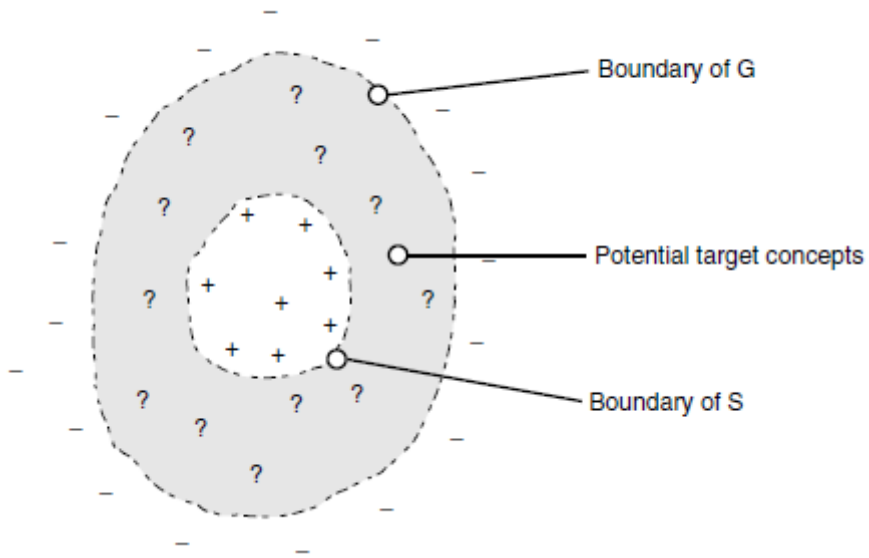
End

Rysunek 9 ilustruje zachowanie algorytmu eliminacji kandydatów podczas przeszukiwania przestrzeni wersji z rysunku 5.



Należy zauważyć, że rysunek nie przedstawia pojęć, które powstały w wyniku uogólnienia lub specjalizacji, ale zostały wyeliminowane jako zbyt ogólne lub szczegółowe. Opracowanie tej części algorytmu zostawiamy jako ćwiczenie i pokazujemy częściową implementację w Prologu w naszych materiałach pomocniczych. Połączenie dwóch kierunków wyszukiwania w jednym algorytmie ma kilka

zalet. Zestawy G i S podsumowują informacje odpowiednio w negatywnych i pozytywnych instancjach treningowych, eliminując potrzebę zapisywania tych instancji. Na przykład, po uogólnieniu S w celu objęcia pozytywnej instancji, algorytm używa G do wyeliminowania pojęć w S, które nie obejmują żadnych negatywnych instancji. Ponieważ G jest zbiorem maksymalnie ogólnych pojęć, które nie pasują do żadnych negatywnych instancji, każdy członek S, który jest bardziej ogólny niż jakikolwiek członek G, musi pasować do jakiejś negatywnej instancji. Podobnie, ponieważ S jest zbiorem maksymalnie szczegółowych uogólnień, które obejmują wszystkie pozytywne instancje, każdy nowy członek G, który jest bardziej szczegółowy niż członek S, musi nie obejmować jakiegoś pozytywnego przypadku i może również zostać wyeliminowany. Rysunek 10 przedstawia abstrakcyjny opis algorytmu eliminacji kandydatów.



Znaki „+” reprezentują pozytywne przykłady treningu; Znaki „-” wskazują negatywne instancje. Najbardziej wewnętrzny okrąg obejmuje zbiór znanych pozytywnych instancji objętych koncepcjami w S. Najbardziej zewnętrzny okrąg obejmuje instancje objęte G; każda instancja poza tym kręgiem jest ujemna. Zacieniona część grafiki zawiera koncepcję docelową wraz z koncepcjami, które mogą być zbyt ogólne lub szczegółowe (?). Wyszukiwanie „zawęża” najbardziej zewnętrzną koncepcję, aby wykluczyć negatywne przypadki; „poszerza” najbardziej wewnętrzną koncepcję o nowe pozytywne przykłady. Ostatecznie oba zestawy zbiegają się w ramach koncepcji docelowej. W ten sposób eliminacja kandydatów może wykryć, kiedy znalazła jedną, spójną koncepcję docelową. Kiedy zarówno G, jak i S zbiegają się w tę samą koncepcję, algorytm może się zatrzymać. Jeśli G i S staną się puste, nie ma koncepcji, która obejmowałaby wszystkie pozytywne i żadne negatywne przypadki. Może się tak zdarzyć, jeśli dane treningowe są niespójne lub jeśli koncepcja celu może nie zostać wyrażona w języku reprezentacji. Ciekawym aspektem eliminacji kandydatów jest jej przyrostowy charakter. Algorytm uczenia przyrostowego akceptuje pojedynczo instancje szkoleniowe, tworząc użyteczne, choć prawdopodobnie niekompletne, uogólnienie po każdym przykładzie. Kontrastuje to z algorytmami wsadowymi, które wymagają, aby wszystkie przykłady uczące były obecne przed rozpoczęciem uczenia się. Nawet zanim algorytm eliminacji kandydatów zbiega się do jednej koncepcji, zbiory G i S zapewniają użyteczne ograniczenia dla tej koncepcji: jeśli c jest pojęciem celu, to dla wszystkich g ∈ G i s ∈ S, s c g. Każde pojęcie, które jest bardziej ogólne niż niektóre pojęcia w G, obejmie negatywne przypadki; każda koncepcja, która jest bardziej szczegółowa niż niektóre pojęcia w S, nie obejmuje niektórych pozytywnych przykładów. Sugeruje to, że przypadki, które „dobrze pasują” do pojęć ograniczonych przez G i S, są co najmniej prawdopodobnymi przykładami tego pojęcia. W następnej

sekcji wyjaśnimy tę intuicję na przykładzie programu, który wykorzystuje eliminację kandydatów do nauki heurystyki wyszukiwania. LEX uczy się heurystyki rozwiązywania problemów integracji symbolicznej. Ta praca nie tylko demonstruje użycie G i S do definiowania częściowych pojęć, ale także ilustruje takie dodatkowe kwestie, jak złożoność uczenia się wieloetapowych zadań, przypisywanie punktów / winy oraz związek między uczeniem się a komponentami rozwiązywania problemów skomplikowany system.

10.2.3 LEX: wywoływanie heurystyki wyszukiwania

LEX uczy się heurystyki rozwiązywania problemów integracji symbolicznej. LEX integruje wyrażenia algebraiczne poprzez przeszukiwanie heurystyczne, zaczynając od wyrażenia, które ma zostać zintegrowane, i poszukując celu: wyrażenia, które nie zawiera znaków całkowych. Składnik uczenia się systemu wykorzystuje dane od osoby rozwiązującej problem do wywoływania heurystyk, które poprawiają wydajność osoby rozwiązującej problem. LEX przeszukuje przestrzeń zdefiniowaną przez operacje na wyrażeniach algebraicznych. Jego operatory to typowe transformacje używane podczas wykonywania integracji. Zawierają:

OP1: $\int r\, f(x)\, dx \rightarrow r \int f(x)\, dx$

OP2: $\int u\, dv \rightarrow uv - \int v\, du$

OP3: $1 * f(x) \rightarrow f(x)$

OP4: $\int (f1(x) + f2(x))\, dx \rightarrow \int f1(x)\, dx + \int f2(x)\, dx$

Operatory to reguły, których lewa strona określa, kiedy mogą być stosowane. Chociaż lewa strona definiuje okoliczności, w których operator może być używany, nie zawiera heurystyki określającej, kiedy operator powinien być używany. LEX musi nauczyć się użytecznych heurystyk na podstawie własnego doświadczenia. Heurystyki to wyrażenia postaci:

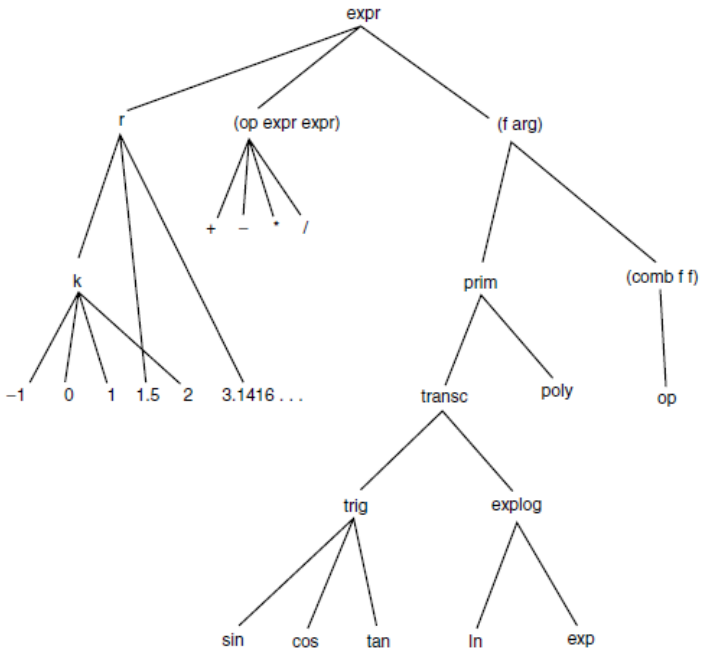
Jeśli bieżący stan problemu pasuje do P, zastosuj operator O z powiązaniami B. Na przykład typowa heurystyka, której może się nauczyć LEX, to:

Jeśli stan problemu pasuje do $\int x\, \text{transcendental}(x)\, dx$, zastosuj OP2 z wiązaniami

$u = x$

$dv = \text{transcendentalny}(x)\, dx$

Tutaj heurystyka sugeruje zastosowanie całkowania przez części w celu rozwiązania całki x razy pewnej funkcji transcendentalnej, np. Funkcji trygonometrycznej w x. Język LEX do przedstawiania pojęć składa się z symboli opisanych na rysunku 11.



Zauważ, że symbole istnieją w hierarchii uogólnień, gdzie każdy symbol pasuje do któregokolwiek z jego potomków w hierarchii. LEX uogólnia wyrażenia według zastąpienie symbolu jego przodkiem w tej hierarchii. Na przykład, biorąc pod uwagę wyrażenie:

∫ 3x cos (x) dx

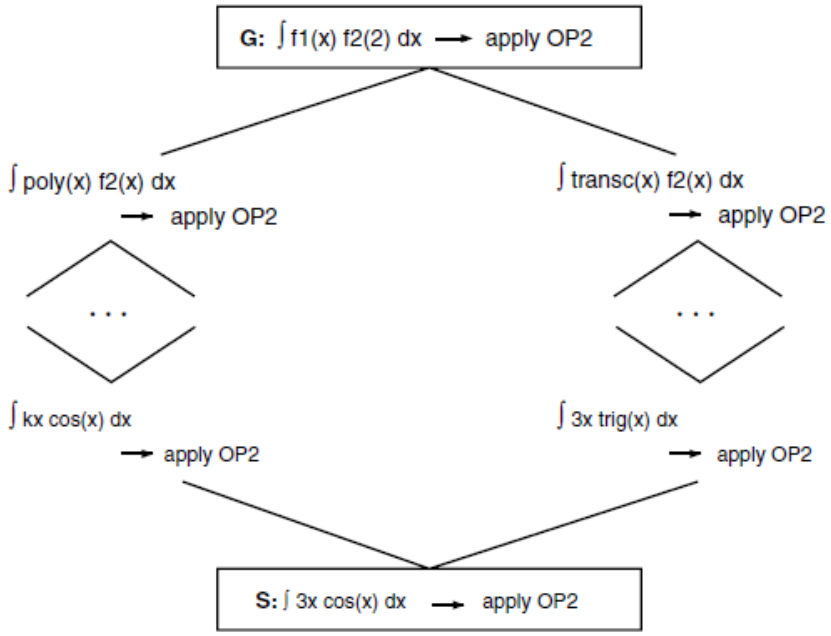
LEX może zamienić cos na trig. To daje wyrażenie:

∫ 3x trig (x) dx

Alternatywnie, może zastąpić 3 symbolem k, który reprezentuje dowolną liczbę całkowitą:

∫ kx cos (x) dx

Rysunek 10.12 przedstawia przestrzeń wersji dla OP2 zdefiniowaną przez te uogólnienia.



Ogólna architektura LEX składa się z czterech komponentów:

1. uogólniacz, który wykorzystuje eliminację kandydatów do znalezienia heurystyki

2. osoba rozwiązująca problemy, która tworzy ślady rozwiązań problemów

3. krytyk, który na podstawie śladu problemu wytwarza pozytywne i negatywne przykłady

4. generator problemów, który tworzy nowe problemy kandydatów

LEX utrzymuje zbiór przestrzeni wersji. Każda przestrzeń wersji jest powiązana z operatorem i reprezentuje częściowo wyuczoną heurystykę dla tego operatora. Generalizator aktualizuje te przestrzenie wersji przy użyciu pozytywnych i negatywnych przykładów aplikacji operatora, wygenerowanych przez krytyka. Po otrzymaniu pozytywnej instancji LEX określa, czy przestrzeń wersji związana z tym operatorem zawiera instancję. Przestrzeń wersji zawiera instancję pozytywną, jeśli instancja jest objęta niektórymi pojęciami w G. LEX używa instancji dodatniej do aktualizacji tej heurystyki. Jeśli żadna istniejąca heurystyka nie pasuje do instancji, LEX tworzy nową przestrzeń wersji, używając tej instancji jako pierwszego pozytywnego przykładu. Może to prowadzić do tworzenia wielu przestrzeni wersji dla różnych heurystyk i jednego operatora. Narzędzie do rozwiązywania problemów LEX tworzy drzewo przestrzeni poszukiwanej podczas rozwiązywania problemu integracji. Ogranicza czas procesora, który osoba rozwiązująca problem może wykorzystać do rozwiązania problemu. LEX przeprowadza wyszukiwanie w pierwszej kolejności, używając własnej, rozwijającej się heurystyki. Ciekawym aspektem wydajności LEX-a jest użycie G i S jako częściowych definicji heurystyki. Jeśli do danego stanu może odnosić się więcej niż jeden operator, LEX wybiera ten, który wykazuje najwyższy stopień częściowego dopasowania do stanu problemu. Stopień częściowego dopasowania definiuje się jako procent wszystkich pojęć zawartych między G i S, które pasują do bieżącego stanu. Ponieważ koszt obliczeniowy testowania stanu pod kątem wszystkich takich kandydujących koncepcji byłby wygórowany, LEX szacuje stopień dopasowania jako procent wpisów faktycznie w G i S, które pasują do stanu. Zauważ, że wydajność powinna stale rosnąć, ponieważ LEX ulepsza swoją heurystykę. Wyniki empiryczne potwierdziły tę hipotezę. LEX uzyskuje pozytywne i negatywne przykłady zastosowań

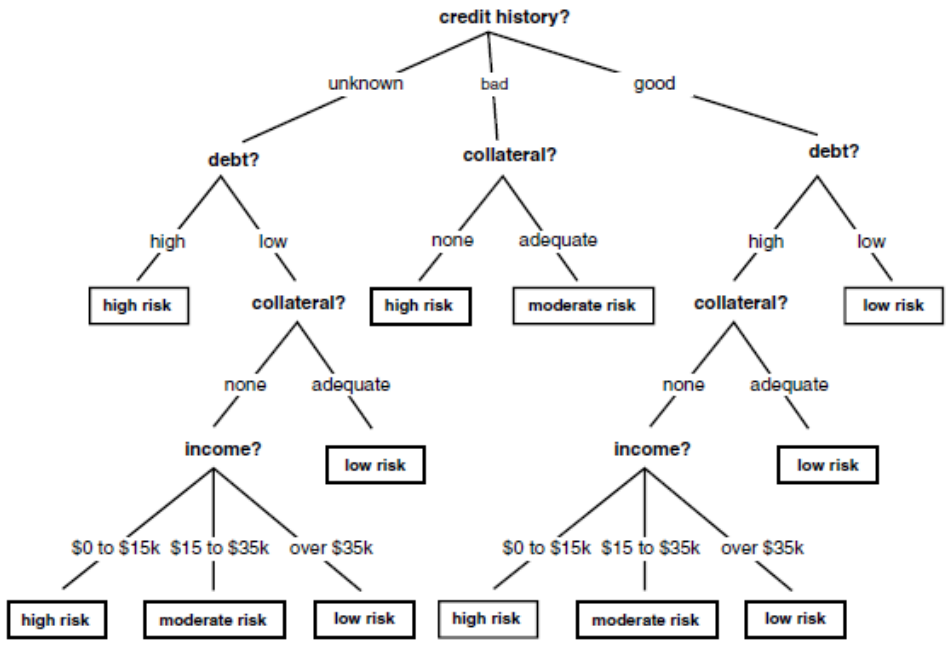
operatora ze śladu rozwiązania wygenerowanego przez rozwiązującego problem. W przypadku braku nauczyciela LEX musi klasyfikować aplikacje operatora jako pozytywne lub negatywne; to jest przykład problemu przypisania kredytu. Kiedy uczenie się jest podejmowane w kontekście wieloetapowego rozwiązywania problemów, często nie jest jasne, które działanie w sekwencji powinno odpowiadać za wynik. Jeśli osoba rozwiązująca problem poda złą odpowiedź, skąd wiemy, który z kilku kroków faktycznie spowodował błąd? Krytyk LEX podchodzi do tego problemu, zakładając, że ślad rozwiązania zwrócony przez osobę rozwiązującą problem reprezentuje najkrótszą drogę do celu. LEX klasyfikuje aplikacje operatorów wzdłuż tej (zakładanej) najkrótszej ścieżki jako pozytywne instancje; aplikacje operatorów, które odbiegają od tej ścieżki, są traktowane jako negatywne wystąpienia. Jednakże, traktując ślad osoby rozwiązującej problem jako rozwiązanie najkrótszej ścieżki, krytyk musi odnieść się do faktu, że ewoluujące heurystyki LEX nie są gwarantowane jako dopuszczalne (sekcja 4.3). Ścieżka rozwiązania znaleziona przez osobę rozwiązującą problem może w rzeczywistości nie być najkrótszą ścieżką. Aby upewnić się, że nie sklasyfikował błędnie aplikacji operatora jako negatywnej, LEX najpierw rozszerza ścieżki rozpoczęte przez takie operatory, aby upewnić się, że nie prowadzą one do lepszego rozwiązania. Zazwyczaj rozwiązanie problemu daje od 2 do 20 instancji szkoleniowych. LEX przekazuje te dodatnie i ujemne instancje do generalizatora, który używa ich do aktualizacji przestrzeni wersji dla powiązanych operatorów. Generator problemów jest najmniej rozwiniętą częścią programu. Chociaż do automatyzacji wyboru problemu zastosowano różne strategie, większość przykładów dotyczyła przypadków wybranych ręcznie. Skonstruowano jednak generator problemów, który badał różne strategie. Jedno podejście generuje instancje, które były objęte częściową heurystyką dla dwóch różnych operatorów, aby LEX nauczył się ich rozróżniać. Testy empiryczne pokazują, że LEX skutecznie uczy się przydatnych heurystyk. W jednym teście LEX otrzymał 5 zadań testowych i 12 zadań szkoleniowych. Przed szkoleniem rozwiązał 5 zadań testowych w średnio 200 krokach; te rozwiązania nie wykorzystywały heurystyki do kierowania wyszukiwaniem. Po opracowaniu heurystyk na podstawie 12 problemów szkoleniowych, rozwiązano te same problemy testowe w średnio 20 krokach. LEX zajmuje się szeregiem problemów w uczeniu się, w tym takimi jak przypisywanie punktów, wybór instancji szkoleniowych oraz związek między komponentami rozwiązywania problemów i generalizacji algorytmu uczenia się. LEX podkreśla również znaczenie odpowiedniej reprezentacji pojęć. Skuteczność LEX-a w dużej mierze wynika z hierarchicznej organizacji pojęć. Ta hierarchia jest na tyle mała, że ogranicza przestrzeń potencjalnych heurystyk i umożliwia wydajne wyszukiwanie, a jednocześnie jest wystarczająco bogata, aby reprezentować efektywną heurystykę.

10.3 Algorytm indukcji drzewa decyzyjnego ID3

ID3 , podobnie jak eliminacja kandydatów, wywołuje koncepcje z przykładów. Jest szczególnie interesujący ze względu na reprezentację wyuczonej wiedzy, podejście do zarządzania złożonością, heurystykę wybierania koncepcji kandydatów oraz potencjał do obsługi zaszumionych danych. ID3 przedstawia koncepcje jako drzewa decyzyjne, reprezentację, która pozwala nam określić klasyfikację obiektu poprzez testowanie jego wartości dla określonych właściwości. Na przykład rozważmy problem szacowania ryzyka kredytowego osoby na podstawie takich właściwości, jak historia kredytowa, bieżące zadłużenie, zabezpieczenie i dochód. Tabela 10.1 wymienia próbkę osób o znanym ryzyku kredytowym.

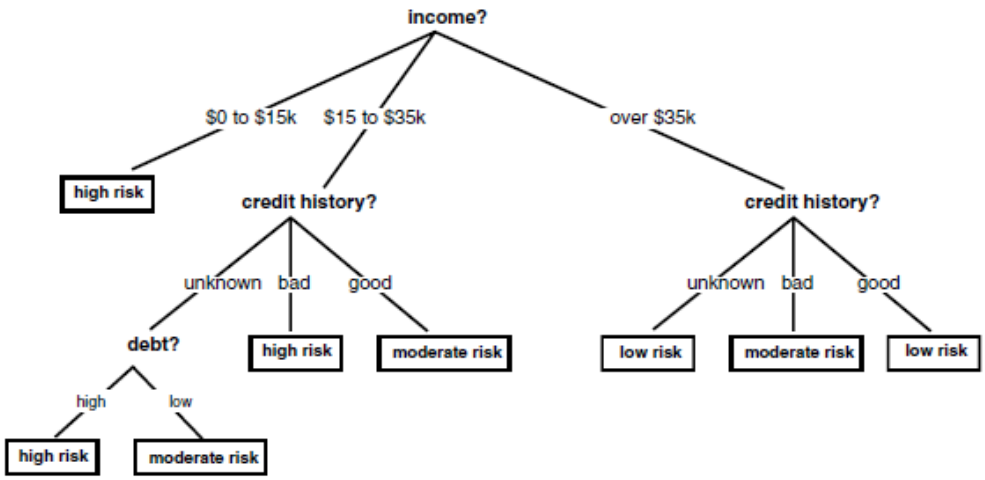
NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	$0 to $15k
2.	high	unknown	high	none	$15 to $35k
3.	moderate	unknown	low	none	$15 to $35k
4.	high	unknown	low	none	$0 to $15k
5.	low	unknown	low	none	over $35k
6.	low	unknown	low	adequate	over $35k
7.	high	bad	low	none	$0 to $15k
8.	moderate	bad	low	adequate	over $35k
9.	low	good	low	none	over $35k
10.	low	good	high	adequate	over $35k
11.	high	good	high	none	$0 to $15k
12.	moderate	good	high	none	$15 to $35k
13.	low	good	high	none	over $35k
14.	high	bad	high	none	$15 to $35k

Drzewo decyzyjne na rysunku 13 przedstawia klasyfikacje w tabeli 1, ponieważ drzewo to może poprawnie sklasyfikować wszystkie obiekty w tabeli.



W drzewie decyzyjnym każdy węzeł wewnętrzny reprezentuje test pewnej właściwości, takiej jak historia kredytowa lub dług; każda możliwa wartość tej właściwości odpowiada gałęzi drzewa. Węzły liści reprezentują klasyfikacje, takie jak niskie lub umiarkowane ryzyko. Osobę nieznanego typu można sklasyfikować, przechodząc przez to drzewo: w każdym węźle wewnętrznym przetestuj wartość danej

osoby dla tej właściwości i wybierz odpowiednią gałąź. Trwa to aż do osiągnięcia węzła liścia i klasyfikacji obiektu. Należy zauważyć, że przy klasyfikowaniu dowolnej instancji drzewo to nie wykorzystuje wszystkich właściwości przedstawionych w tabeli 10.1. Na przykład, jeśli dana osoba ma dobrą historię kredytową i niskie zadłużenie, możemy, zgodnie z drzewem, zignorować jej zabezpieczenie i dochód i sklasyfikować ją jako osobę o niskim ryzyku. Pomimo pominięcia niektórych testów, drzewo to poprawnie klasyfikuje wszystkie przykłady. Generalnie rozmiar drzewa niezbędnego do sklasyfikowania danego zestawu przykładów zmienia się w zależności od kolejności testowania właściwości. Rysunek 14 przedstawia drzewo, które jest znacznie prostsze niż to z rysunku 13, ale które również poprawnie klasyfikuje przykłady w tabeli 1.



Biorąc pod uwagę zestaw instancji szkoleniowych i szereg różnych drzew decyzyjnych, które prawidłowo je klasyfikują, możemy zapytać, które drzewo ma największe prawdopodobieństwo prawidłowego sklasyfikowania niewidocznych instancji populacji. Algorytm ID3 zakłada, że jest to najprostsze drzewo decyzyjne obejmujące wszystkie przykłady szkoleniowe. Powodem tego założenia jest uświęcona tradycją heurystyka preferowania prostoty i unikania niepotrzebnych założeń. Zasada ta, znana jako Brzytwa Ockhama, została po raz pierwszy wyartykułowana przez średniowiecznego logika Williama z Ockhama w 1324 roku: próżno jest robić więcej, co można zrobić za mniej. . . . Podmioty nie powinny być mnożone poza koniecznością. Bardziej współczesna wersja Occam's Razor twierdzi, że zawsze powinniśmy akceptować najprostszą odpowiedź, która poprawnie pasuje do naszych danych. W tym przypadku jest to najmniejsze drzewo decyzyjne poprawnie klasyfikuje wszystkie podane przykłady. Chociaż Brzytwa Ockhama sprawdziła się jako ogólna heurystyka dla wszelkiego rodzaju aktywności intelektualnej, jej użycie tutaj ma bardziej szczegółowe uzasadnienie. Jeśli przyjmiemy, że podane przykłady są wystarczające do skonstruowania ważnego uogólnienia, to naszym problemem staje się odróżnienie niezbędnych własności od obcych. Najprostsze drzewo decyzyjne obejmujące wszystkie przykłady powinno z najmniejszym prawdopodobieństwem zawierać niepotrzebne ograniczenia. Chociaż pomysł ten jest intuicyjnie pociągający, jest to założenie, które musi zostać przetestowane empirycznie; Część 10.3.3 przedstawia niektóre z tych wyników empirycznych. Jednak przed zbadaniem tych wyników przedstawiamy algorytm ID3 do wywoływania drzew decyzyjnych na podstawie przykładów.

10.3.1 Odgórna indukcja drzewa decyzyjnego

ID3 konstruuje drzewa decyzyjne w sposób odgórny. Zauważ, że dla dowolnej właściwości możemy podzielić zestaw przykładów szkoleniowych na rozłączne podzbiory, gdzie wszystkie przykłady w partycji mają wspólną wartość dla tej właściwości. ID3 wybiera właściwość do przetestowania w

bieżącym węźle drzewa i używa tego testu do podzielenia zestawu przykładów; algorytm następnie rekurencyjnie konstruuje poddrzewo dla każdej partycji. Trwa to do momentu, gdy wszyscy członkowie partycji będą w tej samej klasie; ta klasa staje się węzłem liścia drzewa. Ponieważ kolejność testów ma kluczowe znaczenie dla skonstruowania prostego drzewa decyzyjnego, ID3 w dużym stopniu opiera się na kryteriach wyboru testu w korzeniu każdego poddrzewa. Aby uprościć naszą dyskusję, w tej sekcji opisano algorytm konstruowania drzew decyzyjnych przy założeniu odpowiedniej funkcji wyboru testu. W sekcji 10.3.2 przedstawiamy heurystykę wyboru algorytmu ID3. Na przykład rozważmy sposób, w jaki ID3 konstruuje drzewo z rysunku 14 z tabeli 1. Rozpoczynając od pełnej tabeli przykładów, ID3 wybiera dochód jako główną właściwość za pomocą funkcji wyboru opisanej w sekcji 10.3.2. Spowoduje to podział zestawu przykładów, jak pokazano na rysunku 15, przy czym elementy każdej partycji są wymienione według numerów w tabeli. Algorytm indukcji rozpoczyna się od próby prawidłowo sklasyfikowanych członków kategorii docelowych. ID3 następnie konstruuje drzewo decyzyjne zgodnie z następującym algorytmem:

```
function induce_tree (example_set, Properties)

begin

if all entries in example_set are in the same class

then return a leaf node labeled with that class

else if Properties is empty

then return leaf node labeled with disjunction of all classes in example_set

else begin

select a property, P, and make it the root of the current tree;

delete P from Properties;

for each value, V, of P,

begin

create a branch of the tree labeled with V;

let partitionv be elements of example_set with values V for property P;

call induce_tree(partitionv, Properties), attach result to branch V

end

end

end
```

ID3 stosuje funkcję induce_tree rekurencyjnie do każdej partycji. Na przykład podział {1, 4, 7, 11} składa się wyłącznie z osób wysokiego ryzyka; ID3 odpowiednio tworzy węzeł liścia. ID3 wybiera właściwość historii kredytowej jako katalog główny poddrzewa dla partycji {2, 3, 12, 14}. Na rysunku 10.16 historia kredytowa dalej dzieli ten czteroelementowy podział na {2, 3}, {14} i {12}. Kontynuując wybieranie testów i konstruowanie poddrzew w ten sposób, ID3 ostatecznie tworzy drzewo z rysunku 14. Przed przedstawieniem heurystyki wyboru testów ID3 warto zbadać związek między algorytmem konstrukcji drzewa a naszym spojrzeniem na uczenie się jako przeszukiwanie przestrzeni pojęć. Możemy myśleć o zbiorze wszystkich możliwych drzew decyzyjnych jako definiujących przestrzeń wersji. Nasze operacje

poruszania się po tej przestrzeni polegają na dodawaniu testów do drzewa. ID3 implementuje rodzaj chciwego wyszukiwania w przestrzeni wszystkich możliwych drzew: dodaje poddrzewo do aktualnego drzewa i kontynuuje jego przeszukiwanie; nie cofa się. To sprawia, że algorytm jest bardzo wydajny; uzależnia ją również od kryteriów wyboru właściwości do przetestowania.

10.3.2 Wybór testu teoretycznego informacji

Możemy myśleć o każdej właściwości instancji jako o wkładzie pewnej ilości informacji do jej klasyfikacji. Na przykład, jeśli naszym celem jest określenie gatunku zwierzęcia, odkrycie, że składa ono jaja, dostarcza pewnej ilości informacji do tego celu. ID3 mierzy informacje uzyskane przez uczynienie każdej właściwości korzeniem bieżącego poddrzewa. Następnie wybiera właściwość, która zapewnia największy zysk z informacji. Teoria informacji zapewnia matematyczną podstawę pomiaru zawartości informacyjnej wiadomości. Możemy myśleć o wiadomości jako o przykładzie we wszechświecie możliwych wiadomości; czynność przesłania wiadomości jest taka sama, jak wybranie jednej z tych możliwych wiadomości. Z tego punktu widzenia rozsądne jest zdefiniowanie treści informacyjnej wiadomości jako zależnej zarówno od rozmiaru tego wszechświata, jak i częstotliwości, z jaką występuje każdy możliwy komunikat. Znaczenie liczby możliwych komunikatów jest ewidentne na przykładzie hazardu: porównaj komunikat poprawnie przewidujący wynik zakręcenia koła ruletki z komunikatem przewidującym wynik rzutu uczciwą monetą. Ponieważ ruletka może mieć więcej wyników niż rzut monetą, przesłanie dotyczące jej wyniku ma dla nas większą wartość: wygrana w ruletce również jest lepsza niż wygrana w rzucie monetą. W związku z tym powinniśmy traktować tę wiadomość jako przekazującą więcej informacji. Wpływ prawdopodobieństwa każdej wiadomości na ilość informacji jest oczywisty w innym przykładzie hazardu. Załóżmy, że ustawiłem monetę w taki sposób, aby w 3/4 wypadała orzeł. Ponieważ wiem już wystarczająco dużo o monecie, aby prawidłowo obstawić w 3/4 czasu, komunikat informujący mnie o wyniku danego rzutu jest dla mnie mniej wart niż w przypadku uczciwej monety. Shannon sformalizował to, definiując ilość informacji w wiadomości jako funkcję prawdopodobieństwa wystąpienia p każdej możliwej wiadomości, a mianowicie $-\log_2 p$. Biorąc pod uwagę zbiór komunikatów, $M = \{m_1, m_2, \ldots m_n\}$ i prawdopodobieństwo, $p(m_i)$, dla wystąpienia każdego komunikatu, oczekiwana zawartość informacyjna komunikatu M jest określona wzorem:

$$I[M] = \left(\sum_{i=1}^{n} - p(m_i) \log_2 (p(m_i)) \right) = E[-\log_2 p(m_i)]$$

Informacje zawarte w wiadomości są mierzone w bitach. Na przykład treść informacyjna wiadomości informującej o wyniku rzutu uczciwą monetą to:

$$\begin{aligned} I[\text{Coin toss}] &= -p(\text{heads})\log_2(p(\text{heads})) - p(\text{tails})\log_2(p(\text{tails})) \\ &= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \\ &= 1 \text{ bit} \end{aligned}$$

Jeśli jednak moneta została sfałszowana tak, aby w 75% przypadków pojawiała się reszka, wówczas treść wiadomości jest następująca:

$$I[\text{Coin toss}] = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right)$$
$$= -\frac{3}{4}*(-0.415) - \frac{1}{4}*(-2)$$
$$= 0.811 \text{ bits}$$

Ta definicja formalizuje wiele naszych intuicji dotyczących treści informacyjnych komunikatów. Teoria informacji jest szeroko stosowana w informatyce i telekomunikacji, w tym w takich zastosowaniach, jak określanie pojemności kanałów komunikacyjnych do przenoszenia informacji, opracowywanie algorytmów kompresji danych i opracowywanie strategii komunikacji odpornych na zakłócenia. ID3 wykorzystuje teorię informacji, aby wybrać test, który daje największy zysk w klasyfikacji przykładów szkoleniowych. Możemy myśleć o drzewie decyzyjnym jako o przekazywaniu informacji o klasyfikacji przykładów w tabeli decyzyjnej; zawartość informacyjna drzewa jest obliczana na podstawie prawdopodobieństw różnych klasyfikacji. Na przykład, jeśli przyjmiemy, że wszystkie przykłady w tabeli 10.1 występują z równym prawdopodobieństwem, to:

p (ryzyko jest wysokie) = 6/14, p (ryzyko jest umiarkowane) = 3/14, p (ryzyko jest niskie) = 5/14

Wynika z tego, że rozkład opisany w tabeli 10.1, $D_{9.1}$ i, w konsekwencji, każde drzewo, które obejmuje te przykłady, jest następujące:

$$I[D_{9.1}] = -\frac{6}{14}\log_2\left(\frac{6}{14}\right) - \frac{3}{14}\log_2\left(\frac{3}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right)$$
$$= -\frac{6}{14}*(-1.222) - \frac{3}{14}*(-2.222) - \frac{5}{14}*(-1.485)$$
$$= 1.531 \text{ bits}$$

Zysk informacji uzyskany przez wykonanie testu w korzeniu bieżącego drzewa jest równy sumie informacji w drzewie minus ilość informacji potrzebnych do ukończenia klasyfikacji po wykonaniu testu. Ilość informacji potrzebnych do uzupełnienia drzewo definiuje się jako średnią ważoną informacji we wszystkich jego poddrzewach. Średnią ważoną obliczamy, mnożąc zawartość informacyjną każdego poddrzewa przez procent przykładów obecnych w tym poddrzewie i sumując te produkty. Załóżmy zbiór instancji uczących C. Jeśli uczynimy właściwość P z n wartościami korzeniem bieżącego drzewa, podzieli to C na podzbiory, $\{C_1, C_2, \ldots C_n\}$. Oczekiwane informacje potrzebne do ukończenia drzewa po ustawieniu P jako katalogu głównego to:

$$E[P] = \sum_{i=1}^{n} \frac{|C_i|}{|C|} I[C_i]$$

Zysk z właściwości P jest obliczany przez odjęcie oczekiwanych informacji, które posłużą do uzupełnienia drzewa, od całkowitej zawartości informacji w drzewie:

gain (P) = I [C] - E [P]

W przykładzie z tabeli 1, jeśli uczynimy dochód właściwością sprawdzaną u podstawy drzewa, to podzieli tabelę przykładów na podziały C_1 = {1,4,7,11}, C_2 = {2,3,12,14} i C_3 = {5,6,8,9,10,13}. Oczekiwane informacje potrzebne do ukończenia drzewa to:

$$E[income] = \frac{4}{14} * I[C_1] + \frac{4}{14} * I[C_2] + \frac{6}{14} * I[C_3]$$
$$= \frac{4}{14} * 0.0 + \frac{4}{14} * 1.0 + \frac{6}{14} * 0.650$$
$$= 0.564 \text{ bits}$$

Zysk informacyjny dotyczący dystrybucji tabeli 1 to:

$$gain(income) = I[D_{9.1}] - E[income]$$
$$= 1.531 - 0.564$$
$$= 0.967 \text{ bits}$$

Podobnie możemy to pokazać

$$gain(credit\ history) = 0.266$$
$$gain(debt) = 0.063$$
$$gain\ (collateral) = 0.206$$

Podobnie możemy pokazać, że ponieważ dochód zapewnia największy przyrost informacji, ID3 wybierze go jako korzeń drzewa. Algorytm w dalszym ciągu stosuje tę analizę rekurencyjnie do każdego poddrzewa, dopóki nie ukończy drzewa.

10.3.3 Ocena ID3

Chociaż algorytm ID3 tworzy proste drzewa decyzyjne, nie jest oczywiste, że takie drzewa będą skuteczne w przewidywaniu klasyfikacji nieznanych przykładów. ID3 został oceniony zarówno w kontrolowanych testach, jak i aplikacjach i udowodnił, że działa dobrze w praktyce. Quinlan, na przykład, ocenił wyniki ID3 w kwestii uczenia się klasyfikowania szachownic w końcówce szachowej (Quinlan 1983). W końcówce białe grali królem i wieżą, przeciw czarnym, królem i skoczkiem. Celem ID3 było nauczenie się rozpoznawania desek, które doprowadziły do straty czarnych w ciągu trzech ruchów. Atrybuty to różne wysokopoziomowe właściwości desek, takie jak „niemożność bezpiecznego poruszania króla". W teście wykorzystano 23 takie cechy. Po uwzględnieniu symetrii szachownic cała domena problemowa składała się z 1,4 miliona różnych szachownic, z czego 474 000 to strata czarnych w trzech ruchach. ID3 został przetestowany, dając mu losowo wybrany zestaw treningowy, a następnie testując go na 10000 różnych tablicach, również wybranych losowo. Testy Quinlana dały wyniki przedstawione w tabeli 2. Przewidywane maksymalne błędy pochodzą ze statystycznego modelu zachowania ID3 w tej dziedzinie. Dalszą analizę i szczegóły można znaleźć w Quinlan (1983). Wyniki te są poparte dalszymi badaniami empirycznymi oraz anegdotycznymi wynikami z dalszych zastosowań. Odmiany ID3 zostały opracowane, aby radzić sobie z takimi problemami, jak hałas i zbyt duże zestawy treningowe

Size of Training Set	Percentage of Whole Universe	Errors in 10,000 Trials	Predicted Maximum Errors
200	0.01	199	728
1,000	0.07	33	146
5,000	0.36	8	29
25,000	1.79	6	7
125,000	8.93	2	1

10.3.4 Kwestie danych drzewa decyzyjnego: pakowanie, przyspieszanie

Quinlan jako pierwszy zaproponował wykorzystanie teorii informacji do tworzenia poddrzew w uczeniu się drzew decyzyjnych, a jego praca była podstawą naszej prezentacji. Nasze przykłady były jednak czyste, a ich użycie proste. Istnieje wiele problemów, których nie rozwiązaliśmy, a każdy z nich często występuje w dużym zestawie danych:

1. Dane są złe. Może się tak zdarzyć, gdy dwa (lub więcej) identyczne zestawy atrybutów dają różne wyniki. Co możemy zrobić, jeśli nie mamy a priori powodu, aby pozbyć się danych?

2. Brakuje danych z niektórych zestawów atrybutów, być może dlatego, że ich uzyskanie jest zbyt drogie. Czy ekstrapolujemy? Czy możemy stworzyć nową wartość „nieznana”? Jak możemy wyrównać tę nieprawidłowość?

3. Niektóre zestawy atrybutów są ciągłe. Poradziliśmy sobie z tym, dzieląc „dochód” z wartości ciągłej na dogodne podzbiory wartości, a następnie wykorzystaliśmy te grupy. Czy są lepsze podejścia?

4. Zestaw danych może być zbyt duży dla algorytmu uczącego. Jak sobie z tym radzisz?

Rozwiązanie tych problemów stworzyło nowe generacje algorytmów uczenia drzew decyzyjnych po ID3. Najbardziej znanym z nich jest C4,5. Te problemy doprowadziły również do takich technik, jak workowanie i przyspieszanie. Ponieważ dane dla systemów uczących się klasyfikatorów są wektorami lub instancjami wartości atrybutów, kuszące jest manipulowanie danymi w celu sprawdzenia, czy tworzone są różne klasyfikatory. Pakowanie tworzy repliki zestawów szkoleniowych poprzez pobieranie próbek z wymianą z instancji szkoleniowych. Zwiększanie używa wszystkich instancji w każdej replikacji, ale zachowuje wagę dla każdej instancji w zestawie uczącym. Ta waga ma odzwierciedlać znaczenie tego wektora. Kiedy wagi są korygowane, tworzone są różne klasyfikatory, ponieważ wagi powodują, że uczący się koncentruje się na różnych instancjach. W obu przypadkach wiele utworzonych klasyfikatorów jest łączonych przez głosowanie w celu utworzenia klasyfikatora złożonego. Podczas pakowania każdy klasyfikator składników ma ten sam głos, podczas gdy wzmacnianie przypisuje różne siły głosu do klasyfikatorów składników na podstawie ich dokładności. Podczas pracy z bardzo dużymi zbiorami danych często dzieli się dane na podzbiory, buduje drzewo decyzyjne na jednym podzbiorze, a następnie testuje jego dokładność na innych podzbiorach. Literatura dotycząca uczenia się drzew decyzyjnych jest obecnie dość obszerna, zawiera szereg zbiorów danych on-line, a szereg opublikowanych wyników empirycznych pokazuje wyniki stosowania różnych wersji algorytmów drzew decyzyjnych na tych danych. Wreszcie, łatwo jest przekonwertować drzewo decyzyjne na porównywalny zestaw reguł. To, co robimy, to przekształcanie każdej możliwej ścieżki w drzewie decyzyjnym w jedną regułę. Wzór reguły, jej lewa strona, składa się z decyzji prowadzących do węzła liścia. Akcja lub prawa strona to węzeł liścia lub wynik drzewa. Ten zestaw reguł można

dodatkowo dostosować, aby przechwytywać poddrzewa w drzewie decyzyjnym. Ten zestaw reguł można następnie uruchomić w celu sklasyfikowania nowych danych.

10.4 Indukcyjne odchylenie i zdolność uczenia się

Jak dotąd nasza dyskusja kładła nacisk na wykorzystywanie danych empirycznych do kierowania generalizacją. Jednak pomyślna indukcja zależy również od wcześniejszej wiedzy i założeń dotyczących natury poznanych pojęć. Indukcyjne nastawienie odnosi się do wszelkich kryteriów, których uczący się używa do ograniczania przestrzeni pojęć lub wybierania pojęć w tej przestrzeni. W następnej części przeanalizujemy potrzebę uprzedzeń i rodzaje uprzedzeń, które zazwyczaj występują w programach nauczania. W sekcji 10.4.2 przedstawiono teoretyczne wyniki dotyczące ilościowego określania skuteczności obciążeń indukcyjnych.

10.4.1 Odchylenie indukcyjne

Przestrzenie do nauki są zwykle duże; bez jakiegoś sposobu ich przycinania uczenie się oparte na wyszukiwaniu byłoby praktyczną niemożliwością. Na przykład rozważmy problem uczenia się klasyfikacji ciągów bitów (ciągów zer i jedynek) na podstawie przykładów pozytywnych i negatywnych. Ponieważ klasyfikacja jest po prostu podzbiorem zbioru wszystkich możliwych ciągów, całkowita liczba możliwych klasyfikacji jest równa zbiorem potęg lub zbiorowi wszystkich podzbiorów całej populacji. Jeśli istnieje m instancji, możliwe są 2^m klasyfikacji. Ale dla łańcuchów n bitów są 2^n różnych łańcuchów. Zatem do potęgi 2^n istnieją 2 różne klasyfikacje ciągów bitów o długości n. Dla n = 50 liczba ta jest większa niż liczba cząsteczek w znanym wszechświecie! Bez pewnych ograniczeń heurystycznych uczący się nie byłby w stanie efektywnie przeszukiwać takich przestrzeni we wszystkich dziedzinach oprócz najbardziej trywialnych. Innym powodem konieczności uprzedzenia jest natura samego indukcyjnego uogólnienia. Uogólnienie nie oznacza zachowania prawdy. Na przykład, jeśli napotkamy uczciwego polityka, czy mamy prawo zakładać, że wszyscy politycy są uczciwi? Ilu uczciwych polityków musimy spotkać, zanim będziemy mogli przyjąć takie założenie? Hume omówił ten problem, znany jako problem indukcji, kilkaset lat temu: Mówisz, że jedno zdanie jest wnioskiem z drugiego; ale musisz przyznać, że wnioskowanie to nie jest intuicyjne, ani też nie jest demonstracyjne. Jaka to jest natura? Stwierdzenie, że jest to eksperymentalne, rodzi pytanie. Wszystkie wnioski płynące z doświadczenia zakładają jako podstawę, że przyszłość będzie przypominać przeszłość i że podobne moce zostaną połączone z podobnymi zmysłami. Nawiasem mówiąc, w XVIII wieku dzieło Hume'a było postrzegane jako intelektualne zagrożenie, zwłaszcza dla prób matematycznego udowodnienia istnienia bóstwa przez wspólnotę religijną. Wśród przeciwnych teorii proponowanych w celu ratowania „pewności" była teoria wielebnego Bayesa, angielskiego duchownego, patrz rozdziały 5, 9 i 13. Ale wracając do wprowadzenia. W uczeniu indukcyjnym dane szkoleniowe stanowią tylko podzbiór wszystkich instancji w domenie; w konsekwencji każdy zestaw treningowy może wspierać wiele różnych uogólnień. W naszym przykładzie klasyfikatora ciągów bitowych załóżmy, że uczący się otrzymał ciągi {1100, 1010} jako pozytywne przykłady pewnej klasy ciągów. Wiele uogólnień jest zgodnych z tymi przykładami: zbiór wszystkich ciągów zaczynających się od „1" i kończących się na „0", zbiór wszystkich ciągów zaczynających się od „1", zbiór wszystkich ciągów o parzystości lub dowolny inny podzbiór cała populacja obejmująca {1100, 1010}. Z czego może skorzystać uczeń, aby dokonać wyboru spośród tych uogólnień? Same dane nie są wystarczające; wszystkie te wybory są zgodne z danymi. Uczący się musi przyjąć dodatkowe założenia dotyczące „prawdopodobnych" pojęć. W uczeniu się te założenia często przybierają postać heurystyki wyboru gałęzi przestrzeni poszukiwań. Przykładem takiej heurystyki jest funkcja wyboru testu teorii informacji używana przez ID3 (Rozdział 10.3.2). ID3 przeprowadza przeszukiwanie wzgórz w przestrzeni możliwych drzew decyzyjnych. Na każdym etapie wyszukiwania sprawdza wszystkie testy, które można wykorzystać do rozszerzenia drzewa, i wybiera test, który

uzyska najwięcej informacji. Jest to „chciwa” heurystyka: faworyzuje gałęzie przestrzeni poszukiwań, które wydają się przesuwać największą odległość w kierunku stanu celu.

Ta heurystyka umożliwia ID3 efektywne przeszukiwanie przestrzeni drzew decyzyjnych, a także rozwiązuje problem wyboru wiarygodnych uogólnień z ograniczonych danych. ID3 zakłada, że najmniejsze drzewo, które poprawnie klasyfikuje wszystkie podane przykłady, będzie najbardziej skłonne do poprawnej klasyfikacji przyszłych instancji szkoleniowych. Uzasadnieniem tego założenia jest to, że małe drzewa rzadziej przyjmują założenia, które nie są poparte danymi. Jeśli zbiór uczący jest wystarczająco duży i naprawdę reprezentatywny dla populacji, takie drzewa powinny obejmować wszystkie i tylko niezbędne testy do określenia przynależności do klasy. Jak omówiono w sekcji 10.3.3, oceny empiryczne wykazały, że założenie to jest całkiem uzasadnione. Ta preferencja dla prostych definicji pojęć jest używana w wielu algorytmach uczących się, takich jak algorytm CLUSTER / 2 z sekcji 10.6.2. Inną formą błędu indukcyjnego są ograniczenia składniowe reprezentacji wyuczonych pojęć. Takie uprzedzenia nie są heurystykami przy wyborze gałęzi przestrzeni pojęciowej. Zamiast tego ograniczają rozmiar samej przestrzeni, wymagając, aby wyuczone koncepcje były wyrażane w ograniczonym języku reprezentacji. Na przykład drzewa decyzyjne są językiem znacznie bardziej ograniczonym niż pełny rachunek predykatów. Odpowiednie zmniejszenie rozmiaru przestrzeni koncepcyjnej ma zasadnicze znaczenie dla wydajności ID3. Przykład błędu składniowego, który mógłby okazać się skuteczny w klasyfikowaniu ciągów bitowych, ograniczyłby opisy koncepcji do wzorców symboli ze zbioru {0, 1, #}. Wzorzec definiuje klasę wszystkich pasujących ciągów, gdzie dopasowanie jest określane zgodnie z następującymi regułami: Jeśli wzorzec ma 0 na określonej pozycji, to ciąg docelowy musi mieć 0 na odpowiedniej pozycji. Jeśli wzorzec ma 1 w określonej pozycji, to ciąg docelowy musi mieć 1 w odpowiedniej pozycji. # Na danej pozycji może odpowiadać 1 lub 0. Na przykład wzorzec „1 ## 0” definiuje zestaw ciągów {1110, 1100, 1010, 1000}. Uwzględnienie tylko tych klas, które mogłyby być reprezentowane jako pojedynczy taki wzorzec, znacznie zmniejsza rozmiar przestrzeni koncepcji. Dla łańcuchów o długości n możemy zdefiniować 3n różnych wzorów. Jest to znacznie mniej niż 2 do potęgi 2n możliwych koncepcji w nieograniczonej przestrzeni. To odchylenie pozwala również na prostą implementację przeszukiwania przestrzeni wersji, gdzie uogólnienie polega na zastąpieniu 1 lub 0 we wzorcu kandydującym znakiem #. Jednak koszt, jaki ponosimy za to nastawienie, to niezdolność do reprezentowania (a tym samym uczenia się) pewnych pojęć. Na przykład pojedynczy wzorzec tego typu nie może reprezentować klasy wszystkich ciągów o parzystości. Ten kompromis między wyrazistością a wydajnością jest typowy. Na przykład LEX nie rozróżnia nieparzystych lub parzystych liczb całkowitych w taksonomii symboli. W konsekwencji nie może nauczyć się żadnej heurystyki, która zależy od tego rozróżnienia. Chociaż wykonano prace w programach, które mogą zmieniać swoje nastawienie w odpowiedzi na dane, większość programów edukacyjnych zakłada stałe obciążenie indukcyjne. Uczenie maszynowe zbadało szereg uprzedzeń reprezentacyjnych: tendencyjność sprzężona ogranicza wyuczoną wiedzę do koniunkcji literałów. Jest to szczególnie powszechne, ponieważ użycie dysjunkcji w opisach pojęć stwarza problemy do uogólnienia. Na przykład załóżmy, że pozwalamy na dowolne użycie rozłączników w reprezentacji pojęć w algorytmie eliminacji kandydatów. Ponieważ maksymalnie specyficzne uogólnienie zbioru pozytywnych instancji jest po prostu dysjunkcją wszystkich instancji, uczący się w ogóle nie uogólnia. Doda rozłączenia w nieskończoność, wprowadzając formę uczenia się na pamięć (Mitchell 1980). Ograniczenia liczby rozłączników. W wielu zastosowaniach błędy czysto koniunkturalne są zbyt ograniczone. Jednym podejściem, które zwiększa wyrazistość reprezentacji podczas rozwiązywania problemów dysjunkcji, jest dopuszczenie małej, ograniczonej liczby dysjunkcji. Wektory cech to reprezentacja opisująca obiekty jako zbiór cech, których wartości różnią się w zależności od obiektu. Obiekty przedstawione w tabeli 10.1 są reprezentowane jako zbiory cech. Drzewa decyzyjne to reprezentacja koncepcji, która okazała się skuteczna w algorytmie ID3. Klauzule Horn wymagają

ograniczenia formy implikacji, która została wykorzystana w automatycznym rozumowaniu, a także w wielu programach do uczenia się reguł z przykładów. Oprócz błędów składniowych omówionych w tej sekcji, wiele programów wykorzystuje wiedzę specyficzną dla domeny do rozważenia znanej lub zakładanej semantyki domeny. Taka wiedza może zapewnić niezwykle skuteczne nastawienie. W sekcji 10.5 omówiono te podejścia oparte na wiedzy. Zanim jednak rozważymy rolę wiedzy w uczeniu się, krótko przeanalizujemy teoretyczne wyniki określające ilościowo błąd indukcyjny.

10.4.2 Teoria zdolności uczenia się

Celem błędu indukcyjnego jest ograniczenie zbioru pojęć docelowych w taki sposób, abyśmy mogli zarówno efektywnie przeszukiwać zbiór, jak i znajdować definicje pojęć wysokiej jakości. Ciekawy zbiór prac teoretycznych dotyczy problemu ilościowego określania skuteczności obciążenia indukcyjnego. Jakość pojęć definiujemy pod kątem ich zdolności do poprawnej klasyfikacji obiektów, które nie znalazły się w zestawie instancji szkoleniowych. Nie jest trudno napisać algorytm uczący się, który tworzy koncepcje, które będą poprawnie klasyfikować wszystkie przykłady, które widział; wystarczyłoby do tego uczenie się na pamięć. Jednak ze względu na dużą liczbę instancji w większości domen lub fakt, że niektóre instancje nie są dostępne, algorytmy mogą sobie tylko pozwolić. przeanalizować część możliwych przykładów. Zatem realizacja wyuczonej koncepcji w nowych instancjach jest niezwykle ważna. Podczas testowania algorytmów uczących się zwykle dzielimy zbiór wszystkich instancji na nieprzecinające się zbiory instancji szkoleniowych i instancji testowych. Po przeszkoleniu programu na zbiorze uczącym testujemy go na zestawie testowym. Przydatne jest myślenie o wydajności i poprawności jako o właściwościach języka do wyrażania pojęć, tj. O nastawieniu indukcyjnym, a nie o konkretnym algorytmie uczenia się. Algorytmy uczące się przeszukują przestrzeń pojęć; jeśli ta przestrzeń jest możliwa do zarządzania i zawiera koncepcje, które działają dobrze, to każdy rozsądny algorytm uczenia się powinien znaleźć te definicje; jeśli przestrzeń jest bardzo złożona, sukces algorytmu będzie ograniczony. Skrajny przykład wyjaśni tę kwestię.

Pojęcia piłki można się nauczyć, mając odpowiedni język do opisu właściwości przedmiotów. Po obejrzeniu stosunkowo niewielkiej liczby piłek, osoba będzie w stanie określić je zwięźle: piłki są okrągłe. Porównaj to z koncepcją, której nie można się nauczyć: załóżmy, że zespół ludzi biegnie po całej planecie i wybiera zestaw kilku milionów obiektów całkowicie losowo, nazywając wynikową klasę bunch_of_stuff. Nie tylko koncepcja wywołana przez jakąkolwiek próbkę pliku bunch_of_stuff wymagałaby niezwykle złożonej reprezentacji, ale jest również mało prawdopodobne, aby ta koncepcja poprawnie zaklasyfikowała niewidoczne elementy zbioru. Te obserwacje nie zakładają żadnych założeń dotyczących zastosowanych algorytmów uczenia się, po prostu mogą znaleźć koncepcję w przestrzeni zgodną z danymi. kuli można się nauczyć, ponieważ możemy ją zdefiniować za pomocą kilku cech: pojęcie można wyrazić w języku stronniczym. Próba opisania pojęcia bunch_of_stuff wymagałaby definicji pojęcia tak długo, jak lista wszystkich właściwości wszystkich obiektów w klasie. Dlatego zamiast definiować zdolność uczenia się w kategoriach określonych algorytmów, definiujemy ją w kategoriach języka używanego do przedstawiania pojęć. Ponadto, aby osiągnąć ogólność, nie definiujemy możliwości uczenia się w określonych domenach problemowych, takich jak uczenie się pęczka_zamówienia. Zamiast tego definiujemy go w kategoriach właściwości składniowych języka definicji pojęć. Definiując zdolność uczenia się, musimy nie tylko uwzględniać efektywność; musimy również uporać się z faktem, że mamy ograniczone dane. Generalnie nie możemy mieć nadziei na znalezienie dokładnie poprawnej koncepcji na podstawie losowej próbki instancji. Raczej, tak jak przy szacowaniu średniej zbioru w statystykach, próbujemy znaleźć pojęcie, które jest bardzo prawdopodobne, że jest prawie poprawne. W konsekwencji poprawność pojęcia to prawdopodobieństwo, w całej populacji instancji, że poprawnie zaklasyfikuje instancję.

Oprócz poprawności wyuczonych pojęć musimy również wziąć pod uwagę prawdopodobieństwo, że algorytm je znajdzie. Oznacza to, że istnieje niewielka szansa, że próbki, które widzimy, są tak nietypowe, że nauka jest niemożliwa. Zatem określona dystrybucja pozytywnych instancji lub określony zbiór uczący wybrany z tych instancji może, ale nie musi być wystarczający do wybrania koncepcji wysokiej jakości. Dlatego mamy do czynienia z dwoma prawdopodobieństwami: prawdopodobieństwem, że nasze próbki nie są nietypowe, oraz prawdopodobieństwem, że algorytm znajdzie pojęcie jakości, czyli normalnym błędem oszacowania. Te dwa prawdopodobieństwa są ograniczone odpowiednio przez δ i ε, w definicji uczulalności PAC, którą podajemy poniżej. Podsumowując, zdolność uczenia się jest właściwością przestrzeni pojęciowych i jest określana przez język wymagany do reprezentowania pojęć. Oceniając te przestrzenie, musimy wziąć pod uwagę zarówno prawdopodobieństwo, że dane są przez przypadek całkiem zubożone, jak i prawdopodobieństwo, z jakim wynikająca z nich koncepcja poprawnie sklasyfikuje niewidoczne przypadki Valiant (1984) sformalizował te intuicje w teorii prawdopodobnie w przybliżeniu poprawnej (PAC) uczenie się.

Klasy pojęć można nauczyć się PAC, jeśli istnieje algorytm, który działa wydajnie i ma duże prawdopodobieństwo znalezienia w przybliżeniu poprawnej koncepcji. Przez w przybliżeniu poprawne rozumiemy, że koncepcja poprawnie klasyfikuje wysoki odsetek nowych instancji. Dlatego wymagamy zarówno, aby algorytm znalazł z dużym prawdopodobieństwem koncepcję, która jest prawie poprawna, jak i sam algorytm był wydajny. Interesującym aspektem tej definicji jest to, że niekoniecznie zależy ona od rozmieszczenia pozytywnych przykładów w przestrzeni instancji. Zależy to od natury języka koncepcyjnego, to znaczy od uprzedzeń i pożądanego stopnia poprawności. ostatecznie, przyjmując założenia dotyczące przykładowych rozkładów, często można uzyskać lepszą wydajność, to znaczy zadowolić się mniejszą liczbą próbek, niż wymaga tego teoria. Formalnie Valiant definiuje zdolność uczenia się PAC w następujący sposób. Niech C będzie zbiorem pojęć c, a X zbiorem instancji. Koncepcjami mogą być algorytmy, wzorce lub inne sposoby dzielenia X na pozytywne i negatywne instancje. C jest PAC, którego można się nauczyć, jeśli istnieje algorytm o następujących właściwościach:

1. Jeżeli dla błędu koncepcji ε i prawdopodobieństwa uszkodzenia δ, to istnieje algorytm, który przy losowej próbie wystąpień o wielkości n = | X | wielomian w 1 / ε i 1 / δ, algorytm tworzy pojęcie c, element C, takie, że prawdopodobieństwo, że c ma błąd uogólnienia większy niż ε, jest mniejsze niż δ. To znaczy dla y pobranego z tego samego rozkładu, z którego pobrano próbki w X:

P [P [y jest błędnie sklasyfikowane przez c] ≥ ε] ≤ δ.

2. Czas wykonania algorytmu jest wielomianowy w n, 1 / ε i 1 / δ.

Korzystając z tej definicji zdolności uczenia się PAC, badacze wykazali podatność na działanie kilku błędów indukcyjnych. Na przykład Valiant udowadnia, że klasy wyrażeń k-CNF można się nauczyć. Wyrażenia k-CNF są zdaniami w postaci normalnej koniunkcji z ograniczeniem liczby rozłączników; wyrażenia są tworzone z koniunkcji zdań, $c_1 \Lambda c_2 \Lambda . . . c_n$, gdzie każdy ci jest dysjunkcją nie więcej niż k literałów. Ten teoretyczny wynik potwierdza powszechne ograniczenie pojęć do formy łącznikowej używanej w wielu algorytmach uczenia się. Nie powielamy tutaj dowodu, ale odsyłamy czytelnika do artykułu Valianta, w którym udowadnia ten wynik, wraz z możliwością nauczenia się innych uprzedzeń.

10.5 Wiedza i nauka

ID3 i algorytm eliminacji kandydatów uogólniają na podstawie prawidłowości w danych szkoleniowych. Takie algorytmy są często określane jako oparte na podobieństwie, ponieważ uogólnienie jest przede wszystkim funkcją podobieństw w przykładach szkoleniowych. Błędy stosowane przez te algorytmy są

ograniczone do składniowych ograniczeń formy uczenia się , umiejętności; nie czynią mocnych założeń co do semantyki domen. W tej sekcji przyjrzymy się algorytmom, takim jak uczenie się oparte na wyjaśnieniach, które wykorzystują wcześniejszą wiedzę dziedzinową do kierowania generalizacją. Początkowo pogląd, że wcześniejsza wiedza jest niezbędna do uczenia się, wydaje się sprzeczna. Jednak zarówno uczenie maszynowe, jak i badacze kognitywni przedstawili argumenty przemawiające za tym właśnie pojęciem, argumentując, że najskuteczniejsze uczenie się ma miejsce, gdy uczący się ma już znaczną wiedzę w tej dziedzinie. Jednym z argumentów przemawiających za istotnością wiedzy w uczeniu się jest opieranie się technik uczenia się opartych na podobieństwach na stosunkowo dużej ilości danych szkoleniowych. Z drugiej strony ludzie mogą formułować wiarygodne uogólnienia z zaledwie jednej instancji szkoleniowej, a wiele praktycznych zastosowań wymaga, aby program nauczania robił to samo. Kolejny argument przemawiający za ważnością wcześniejszej wiedzy zakłada, że każdy zestaw przykładów szkoleniowych może wspierać nieograniczoną liczbę uogólnień, z których większość jest albo nieistotna, albo bezsensowna. Indukcyjne nastawienie jest jednym ze sposobów dokonania tego rozróżnienia. W tej sekcji przyjrzymy się algorytmom, które wykraczają poza czysto syntaktyczne uprzedzenia, aby rozważyć rolę silnej wiedzy dziedzinowej w uczeniu się.

10.5.1 Meta-DENDRAL

Meta-DENDRAL jest jednym z najwcześniejszych i wciąż jednym z najlepszych przykładów wykorzystania wiedzy w uczeniu się indukcyjnym. Meta-DENDRAL pozyskuje reguły, które będą wykorzystywane przez program DENDRAL do analizy danych spektrograficznych. DENDRAL wnioskuje o strukturze cząsteczek organicznych na podstawie ich wzoru chemicznego i danych spektrograficznych. Spektrograf mas bombarduje cząsteczki elektronami, powodując zerwanie niektórych wiązań chemicznych. Chemicy mierzą wagę uzyskanych kawałków i interpretują wyniki, aby uzyskać wgląd w strukturę związku. DENDRAL wykorzystuje wiedzę w postaci reguł interpretacji danych spektrograficznych. Założeniem reguły DENDRAL jest wykres pewnej części struktury molekularnej. Konkluzją reguły jest ten wykres z zaznaczoną lokalizacją rozszczepienia. Meta-DENDRAL wywodzi te zasady z wyników spektrograficznych na cząsteczkach o znanej strukturze. Meta-DENDRAL otrzymuje strukturę znanego związku, wraz z masą i względną liczebnością fragmentów wytworzonych metodą spektrografii. Interpretuje je, konstruując opis, gdzie wystąpiły przerwy. Te wyjaśnienia przerw w określonych cząsteczkach służą jako przykłady do konstruowania ogólnych reguł. Określając miejsce rozszczepienia w biegu treningowym, DENDRAL wykorzystuje „teorię półrzędu" chemii organicznej. Ta teoria, choć nie jest wystarczająco silna, aby wspierać bezpośrednią konstrukcję reguł DENDRAL, wspiera interpretację rozszczepień w znanych cząsteczkach. Teoria półrzędu składa się z reguł, ograniczeń i heurystyk, takich jak:

Wiązania podwójne i potrójne nie pękają.

W danych pojawiają się tylko fragmenty większe niż dwa atomy węgla.

Korzystając z teorii półrzędu, meta-DENDRAL konstruuje wyjaśnienia rozszczepienia. Te wyjaśnienia wskazują prawdopodobne miejsca rozszczepień wraz z możliwymi migracjami atomów przez przerwę.

Te wyjaśnienia stają się zbiorem pozytywnych przykładów dla programu wprowadzania reguł. Ten komponent wprowadza ograniczenia w przesłankach reguł DENDRAL poprzez wyszukiwanie ogólne do szczegółowego. Rozpoczyna się całkowicie ogólnym opisem rozszczepienia: $X_1 * X_2$. Ten wzór oznacza, że rozszczepienie, wskazane gwiazdką, może nastąpić między dwoma dowolnymi atomami. Specjalizuje się we wzornictwie poprzez:

dodawanie atomów: $X_1 * X_2 \rightarrow X_3 - X_1 * X_2$

gdzie operator „-” wskazuje na wiązanie chemiczne, czyli tworzenie instancji atomów lub atrybutów atomów: $X_1 * X_2 \rightarrow C * X_2$

Meta-DENDRAL uczy się z pozytywnych przykładów tylko a d wykonuje wspinaczkowe przeszukiwanie przestrzeni koncepcyjnej. Zapobiega nadmiernemu uogólnieniu, ograniczając zasady kandydatów do około połowy instancji szkoleniowych. Kolejne komponenty programu oceniają i udoskonalają te reguły, szukając zbędnych lub modyfikujących reguły, które mogą być zbyt ogólne lub szczegółowe. Siła meta-DENDRAL polega na wykorzystaniu wiedzy dziedzinowej do zmiany surowych danych w bardziej użyteczną formę. Daje to programowi odporność na hałas, poprzez wykorzystanie jego teorii do eliminacji obcych lub potencjalnie błędnych danych, oraz możliwość uczenia się na podstawie stosunkowo niewielu instancji szkoleniowych. Podstawą uczenia się opartego na wyjaśnieniach jest przekonanie, że dane szkoleniowe muszą być tak interpretowane, aby były w pełni użyteczne.

10.5.2 Uczenie się oparte na wyjaśnieniach

Uczenie się oparte na wyjaśnieniach wykorzystuje jawnie reprezentowaną teorię domeny do konstruowania wyjaśnienia przykładu szkoleniowego, zwykle dowodu, że przykład logicznie wynika z teorii. Uogólniając na podstawie wyjaśnienia instancji, a nie samej instancji, uczenie się oparte na wyjaśnieniach filtruje szum, wybiera odpowiednie aspekty doświadczenia i organizuje dane szkoleniowe w systematyczną i spójną strukturę. Istnieje kilka alternatywnych sformułowań tego pomysłu. Na przykład program STRIPS do reprezentowania ogólnych operatorów planowania (patrz sekcja 8.4) wywarł silny wpływ na te badania. Meta-DENDRAL, jak widzieliśmy w sekcji 10.5.1, ustalił moc opartej na teorii interpretacji instancji szkoleniowych. Ostatnio wielu autorów (DeJong i Mooney 1986, Minton 1988) zaproponowało alternatywne sformułowania tej idei. Algorytm generalizacji opartej na wyjaśnieniach Mitchella i innych jest również typowy dla tego gatunku. W tej części przyjrzymy się wariacji algorytmu uczenia się opartego na wyjaśnieniach (EBL) opracowanego przez DeJonga i Mooneya. EBL zaczyna się od:

1. Koncepcja celu. Zadaniem ucznia jest określenie skutecznej definicji tego pojęcia. W zależności od konkretnego zastosowania, pojęciem docelowym może być klasyfikacja, twierdzenie do udowodnienia, plan osiągnięcia celu lub heurystyka dla rozwiązania problemu.

2. Przykład szkolenia, przykład celu.

3. Teoria domeny, zestaw reguł i faktów, które są wykorzystywane do wyjaśnienia, w jaki sposób przykład szkolenia jest przykładem koncepcji celu.

4. Kryteria operacyjności, niektóre sposoby opisu formy, jaką mogą przybrać definicje pojęć.

Aby zilustrować EBL, przedstawiamy przykład uczenia się, kiedy przedmiot jest filiżanką. Jest to odmiana problemu zbadanego przez Winstona i inni i przystosowany do uczenia się opartego na wyjaśnianiu przez Mitchell et al. (1986). Pojęcie celu to reguła, której można użyć, aby wywnioskować, czy przedmiot jest filiżanką:

premise(X) → cup(X)

gdzie przesłanka jest wyrażeniem łączącym zawierającym zmienną X. Przyjmijmy teorię dziedzinową, która zawiera następujące zasady dotyczące kubków:

liftable(X) ∧ holds_liquid(X) → cup(X)

part(Z, W) ∧ concave(W) ∧ points_up(W) → holds_liquid(Z)

light(Y) ∧ part(Y, handle) → liftable(Y)

small(A) → light(A)

made_of(A, feathers) → light(A)

Przykład szkolenia jest przykładem koncepcji celu. Oznacza to, że otrzymujemy:

cup(obj1)

small(obj1)

part(obj1, handle)

owns(bob, obj1)

part(obj1, bottom)

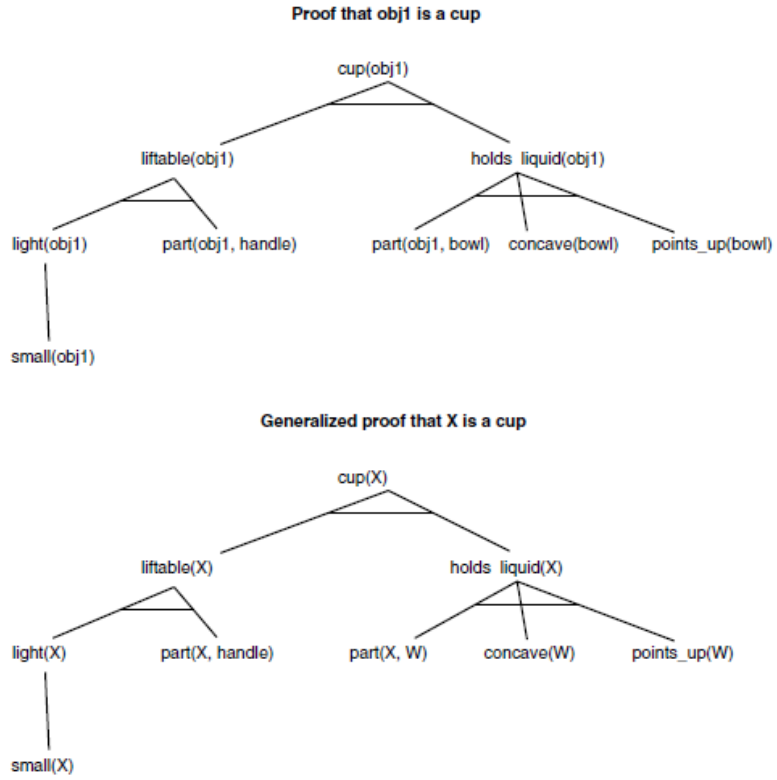
part(obj1, bowl)

points_up(bowl)

concave(bowl)

color(obj1, red)

Na koniec załóżmy, że kryteria operacyjne wymagają, aby pojęcia docelowe były zdefiniowane w kategoriach obserwowalnych, strukturalnych właściwości obiektów, takich jak część i points_up. Możemy zapewnić reguły dziedzinowe, które umożliwią uczącemu się wywnioskować, czy opis działa, lub możemy po prostu wyszczególnić predykaty operacyjne. Korzystając z tej teorii, sprawdzający twierdzenie może skonstruować wyjaśnienie, dlaczego przykład jest rzeczywiście przykładem koncepcji szkoleniowej: dowód, że koncepcja docelowa logicznie wynika z przykładu, jak w pierwszym drzewie na rysunku 17. Należy zauważyć, że to wyjaśnienie eliminuje takie nieistotne pojęcia, jak kolor (obj1, czerwony) z danych treningowych i obejmuje te aspekty przykładu, o którym wiadomo, że są istotne dla celu. Następny etap uczenia się opartego na wyjaśnianiu uogólnia wyjaśnienie, aby utworzyć definicję pojęcia, która może być wykorzystana do rozpoznania innych kubków. EBL osiąga to poprzez podstawianie zmiennych na te stałe w drzewie dowodowym, które zależą wyłącznie od konkretnej instancji uczącej, jak na rysunku 17.

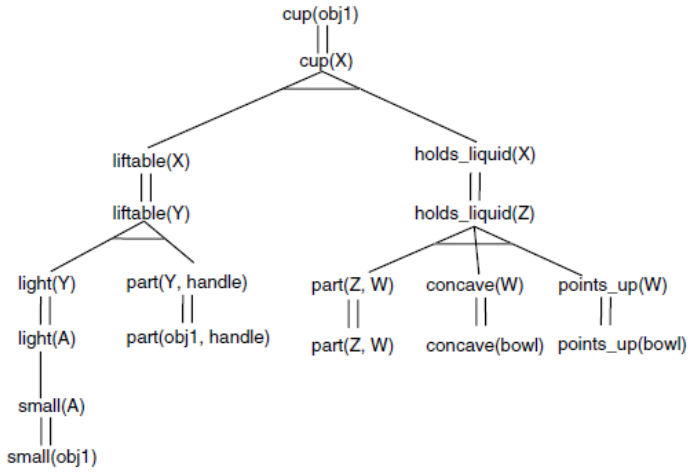


W oparciu o uogólnione drzewo, EBL definiuje nową regułę, której konkluzją jest korzeń drzewa i której przesłanką jest koniunkcja liści:

small(X) ∧ part(X, handle) ∧ part(X, W) ∧ concave(W) ∧ points_up(W) → cup(X)

Konstruując uogólnione drzewo dowodowe, naszym celem jest zastąpienie zmiennych tymi stałymi, które są częścią instancji szkoleniowej, przy jednoczesnym zachowaniu tych stałych i ograniczeń, które są częścią teorii domeny. W tym przykładzie stały uchwyt pochodzi z teorii domeny, a nie z instancji uczącej. Zachowaliśmy to jako zasadnicze ograniczenie nabytej reguły .

Możemy zbudować uogólnione drzewo dowodu na wiele sposobów, używając instancji szkoleniowej jako przewodnika. Mitchell i inni osiągają to poprzez najpierw skonstruowanie drzewa dowodu, które jest specyficzne dla przykładu szkoleniowego, a następnie uogólnienie dowodu poprzez proces zwany regresją celu. Regresja celu dopasowuje uogólniony cel (w naszym przykładzie puchar (X)) do korzenia drzewa dowodu, zastępując stałe zmiennymi zgodnie z wymaganiami dopasowania. Algorytm stosuje te podstawienia rekurencyjnie w drzewie, dopóki wszystkie odpowiednie stałe nie zostaną uogólnione. DeJong i Mooney proponują alternatywne podejście, które zasadniczo równolegle buduje uogólnione i specyficzne drzewa. Osiąga się to poprzez utrzymywanie wariacji drzewa dowodu składającego się z reguł używanych do dowodzenia celu różniącego się od podstawień zmiennych używanych w rzeczywistym dowodzie. Nazywa się to strukturą wyjaśnienia, jak na rysunku 18,



i przedstawia abstrakcyjną strukturę dowodu. Uczeń utrzymuje dwie odrębne listy zastępstw dla struktury wyjaśnienia: listę konkretnych zmian wymaganych do wyjaśnienia przykładu szkoleniowego oraz listę ogólnych podstawień wymaganych do wyjaśnienia uogólnionego celu. Tworzy te listy podstawień podczas budowania struktura wyjaśnienia. Listę podstawień ogólnych i szczegółowych konstruujemy w następujący sposób: niech s_s i s_g będą odpowiednio listami podstawień szczegółowych i ogólnych. Dla każdego dopasowania między wyrażeniami e_1 i e_2 w strukturze wyjaśnienia zaktualizuj s_s i s_g zgodnie z następującą regułą:

jeśli e_1 znajduje się w przesłance reguły domeny, a e_2 jest zakończeniem reguły domeny, to zacznij

T_s = najbardziej ogólny unifikator e1ss i e2ss% unify e_1 i e_2 pod s_s

$s_s = s_s T_s$% update s_s, komponując go za pomocą T_s

T_g = najbardziej ogólny unifikator e1sg i e2sg% ujednolicić e_1 i e_2 pod s_g

$s_g = s_g T_g$% update s_g, komponując go z T_g

koniec

jeśli e_1 znajduje się w przesłance reguły domeny, a e_2 jest faktem w instancji uczącej

następnie rozpocznij% only update s_s

T_s = najbardziej ogólny unifikator $e_1 s_s$ i $e_2 s_s$% unify e_1 i e_2 pod s_s

$s_s = s_s T_s$% update s_s, komponując go za pomocą T_s

koniec

W przykładzie z rysunku 18:

s_s = {obj1 / X, obj1 / Y, obj1 / A, obj1 / Z, bowl / W}

s_g = {X / Y, X / A, X / Z}

Zastosowanie tych podstawień do struktury wyjaśnień z rysunku 18 daje specyficzne i ogólne drzewa dowodowe z rysunku 17. Nauka oparta na wyjaśnieniach oferuje szereg korzyści:

1. Przykłady treningowe często zawierają nieistotne informacje, takie jak kolor filiżanki w poprzednim przykładzie. Teoria domeny pozwala uczestnikowi na wybranie odpowiednich aspektów instancji szkoleniowej.

2. Podany przykład może pozwolić na wiele możliwych uogólnień, z których większość jest albo bezużyteczna, bez znaczenia, albo błędna. EBL tworzy uogólnienia, o których wiadomo, że mają znaczenie dla określonych celów i które są logicznie zgodne z teorią domeny.

3. Wykorzystując wiedzę dziedzinową, EBL umożliwia uczenie się z jednej instancji szkoleniowej.

4. Konstrukcja wyjaśnienia pozwala uczniowi postawić hipotezę o nieokreślonych związkach między jego celami a jego doświadczeniem, na przykład wydedukować definicję filiżanki na podstawie jej właściwości strukturalnych.

EBL zastosowano do wielu problemów w nauce. Na przykład Mitchell i in. (1983) omawiają dodanie EBL do algorytmu LEX. Załóżmy, że pierwszym pozytywnym przykładem użycia OP1 jest rozwiązanie wystąpienia $\int 7x^2 dx$. LEX uczyni tę instancję członkiem S, zbioru maksymalnie szczegółowych uogólnień. Jednak człowiek by to zrobił natychmiast rozpoznaj, że techniki użyte do rozwiązania tego wystąpienia nie zależą od konkretnych wartości współczynnika i wykładnika, ale będą działać dla dowolnych wartości rzeczywistych, o ile wykładnik nie jest równy -1. Uczeń ma prawo wnioskować, że OP1 należy zastosować do dowolnego wystąpienia postaci $\int r_1 x (r^{2 \neq -1}) dx$, gdzie r_1 i r_2 są dowolnymi liczbami rzeczywistymi. LEX został rozszerzony, aby wykorzystać swoją wiedzę z algebry z uczeniem się opartym na wyjaśnianiu, aby dokonać tego typu uogólnień. Na koniec, w materiałach pomocniczych do tej książki zaimplementowaliśmy algorytm uczenia się w oparciu o wyjaśnienia w Prologu.

10.5.3 EBL i nauka na poziomie wiedzy

Chociaż jest to eleganckie sformułowanie roli wiedzy w uczeniu się, EBL stawia szereg ważnych pytań. Jeden z bardziej oczywistych dotyczy tego, czego faktycznie uczy się uczeń oparty na wyjaśnieniach. Czysta EBL może nauczyć się tylko reguł, które mieszczą się w dedukcyjnym zamknięciu istniejącej teorii. Oznacza to, że wyuczone reguły można było wywnioskować z bazy wiedzy bez korzystania w ogóle z instancji szkoleniowej. Jedyną funkcją instancji szkoleniowej jest skupienie się na dowodzeniu twierdzenia na odpowiednich aspektach dziedziny problemu. W konsekwencji EBL jest często postrzegany jako forma przyspieszenia uczenia się lub przeformułowania bazy wiedzy; EBL może przyspieszyć pracę ucznia, ponieważ nie musi rekonstruować drzewa dowodu leżącego u podstaw nowej reguły. Jednakże, ponieważ zawsze mógł odtworzyć dowód, EBL nie może zmusić uczącego się do zrobienia niczego nowego. To rozróżnienie zostało sformalizowane przez Dietterich a w jego omówieniu uczenia się na poziomie wiedzy . EBL przyjmuje informacje zawarte w zbiorze reguł i czyni je jawnymi. Rozważmy na przykład grę w szachy: minimalna znajomość reguł szachów, w połączeniu z możliwością wykonywania nieograniczonego przewidywania stanów na szachownicy, pozwoliłaby komputerowi na bardzo dobrą grę. Niestety, szachy są zbyt skomplikowane dla tego podejścia. Uczeń oparty na wyjaśnieniach, który potrafiłby opanować strategie szachowe, rzeczywiście nauczyłby się czegoś, co ze wszystkich praktycznych względów było nowe. EBL pozwala nam również zrezygnować z wymogu posiadania przez uczącego się pełnej i poprawnej teorii domeny i skupić się na technikach udoskonalania niekompletnych teorii w kontekście EBL. Tutaj uczeń konstruuje częściowe drzewo rozwiązań. Te gałęzie dowodu, których nie można uzupełnić, wskazują na braki w teorii. W tej dziedzinie do zbadania pozostaje szereg interesujących pytań. Obejmują one rozwój heurystyk do wnioskowania z niedoskonałymi teoriami, metodologie przypisywania punktów oraz wybór, który z kilku nieudanych dowodów powinien zostać naprawiony. Kolejnym zastosowaniem uczenia się opartego na wyjaśnianiu jest zintegrowanie go z podejściem do uczenia się opartym na podobieństwie. Ponownie, sugeruje się kilka podstawowych schematów, takich jak wykorzystanie EBL do

udoskonalenia danych szkoleniowych, do których ma zastosowanie teoria, a następnie przekazanie tych częściowo uogólnionych danych do ucznia opartego na podobieństwie w celu dalszego uogólnienia. Alternatywnie moglibyśmy użyć nieudanych wyjaśnień jako sposobu na celowanie w braki w teorii, kierując tym samym gromadzeniem danych dla ucznia opartego na podobieństwie. Inne zagadnienia w badaniach EBL obejmują techniki wnioskowania z błędnymi teoriami, alternatywy dla dowodzenia twierdzeń jako sposób konstruowania wyjaśnień, metody radzenia sobie z hałaśliwymi lub brakującymi danymi szkoleniowymi oraz metody określania, które wygenerowane reguły zapisać. Podczas gdy „czysty" EBL ogranicza się do uczenia się dedukcyjnego, analogie oferują bardziej elastyczną metodę wykorzystania istniejącej wiedzy. Rozumowanie analogiczne zakłada, że jeśli wiadomo, że dwie sytuacje są podobne pod pewnymi względami, prawdopodobnie będą podobne pod innymi. Na przykład, jeśli dwa domy mają podobną lokalizację, konstrukcję i stan, prawdopodobnie mają mniej więcej taką samą wartość sprzedaży. W przeciwieństwie do dowodów używanych w EBL, analogia nie jest logicznie uzasadniona. W tym sensie przypomina indukcję. Jak zauważył Russell (1989) i inni, analogia jest rodzajem indukcji pojedynczego wystąpienia: w naszym przykładzie domowym indukujemy właściwości jednego domu na podstawie tego, co wiemy o innym. Jak omówiliśmy w naszej prezentacji rozumowania opartego na przypadkach (sekcja 7.3), analogia jest bardzo przydatna do zastosowania istniejącej wiedzy w nowych sytuacjach. Na przykład załóżmy, że uczeń stara się poznać zachowanie elektryczności i załóżmy, że nauczyciel mówi jej, że elektryczność jest analogiczna do wody, przy czym napięcie odpowiada ciśnieniu, natężenie prądu odpowiada natężeniu przepływu i opór pojemności. rura. Korzystając z analogicznego rozumowania, student może łatwiej zrozumieć takie pojęcia, jak prawo Ohma. Standardowy model obliczeniowy analogii definiuje źródło analogii jako rozwiązanie problemu, przykład lub teorię, która jest stosunkowo dobrze zrozumiana. Cel nie jest całkowicie zrozumiały Analogia konstruuje mapowanie między odpowiednimi elementami celu i źródła. Analogiczne wnioski rozszerzają to mapowanie na nowe elementy domeny docelowej. Kontynuując analogię „elektryczność jest jak woda", jeśli wiemy, że ta analogia odwzorowuje przełączniki na zawory, natężenie prądu na wielkość przepływu i napięcie na ciśnienie wody, możemy rozsądnie wywnioskować, że powinna istnieć jakaś analogia do pojemności (tj. pole przekroju poprzecznego) rury wodociągowej; może to prowadzić do zrozumienia oporu elektrycznego. Wielu autorów zaproponowało ujednolicające ramy dla modeli obliczeniowych rozumowania analogicznego
.

Typowa struktura składa się z następujących etapów:

1. Odzyskiwanie. Biorąc pod uwagę problem docelowy, konieczne jest wybranie potencjalnego analogu źródłowego. Problemy w wyszukiwaniu analogicznym obejmują wybór tych cech celu i źródła, które zwiększają prawdopodobieństwo odzyskania przydatnego źródła analogowego i indeksowanie wiedzy zgodnie z tymi cechami. Ogólnie, wyszukiwanie ustala początkowe elementy analogicznego odwzorowania.

2. Opracowanie. Po odzyskaniu źródła często konieczne jest wyprowadzenie dodatkowych cech i relacji źródła. Na przykład może być konieczne opracowanie konkretnego śladu rozwiązywania problemów (lub wyjaśnienia) w domenie źródłowej jako podstawy dla analogii z celem.

3. Mapowanie i wnioskowanie. Ten etap obejmuje opracowanie mapowania atrybutów źródłowych na domenę docelową. Dotyczy to zarówno znanych podobieństw, jak i analogicznych wniosków.

4. Uzasadnienie. Tutaj ustalamy, że mapowanie jest rzeczywiście prawidłowe. Ten etap może wymagać modyfikacji mapowania.

5. Uczenie się. Na tym etapie zdobyta wiedza jest przechowywana w formie, która będzie przydatna w przyszłości.

Etapy te zostały rozwinięte w wielu modelach obliczeniowych rozumowania analogicznego. Na przykład teoria mapowania struktury nie tylko rozwiązuje problem konstruowania użytecznych analogii, ale także dostarcza wiarygodnego modelu rozumienia analogii przez ludzi. Głównym pytaniem w stosowaniu analogii jest to, jak odróżnić ekspresyjne, głębokie analogie od bardziej powierzchownych porównań. Gentner twierdzi, że prawdziwe analogie powinny podkreślać systematyczne, strukturalne cechy domeny zamiast bardziej powierzchownych podobieństw. Na przykład analogia, że „atom jest jak układ słoneczny" jest głębsza niż „słonecznik jest jak słońce", ponieważ ten pierwszy obejmuje cały system związków przyczynowych między ciałami na orbicie, podczas gdy drugi opisuje powierzchowne podobieństwa, takie jak fakt że zarówno słoneczniki, jak i słońce są okrągłe i żółte. Ta właściwość analogicznego odwzorowania nazywana jest systematycznością. Odwzorowanie struktury formalizuje tę intuicję. Rozważmy przykład atomu / słońca analogia systemowa, jak na rysunku 19, jak wyjaśnił Gentner. Domena źródłowa zawiera predykaty:

żółty (słońce)

niebieski (ziemia)

gorętszy niż (słońce, ziemia)

przyczyny (bardziej masywne (słońce, ziemia), przyciąganie (słońce, ziemia))

przyczyny (przyciąganie (słońce, ziemia), obraca się wokół (ziemia, słońce))

Dziedzina docelowa, którą ma wyjaśnić analogia, obejmuje

bardziej masywny (jądro, elektron)

obraca się wokół (elektron, jądro)

Mapowanie struktury ma na celu przeniesienie struktury przyczynowej źródła na cel. Mapowanie jest ograniczone następującymi regułami:

1. Właściwości są usuwane ze źródła. Ponieważ analogia sprzyja układom relacji, pierwszym etapem jest eliminacja predykatów, które opisują powierzchowne właściwości źródła. Odwzorowanie struktury formalizuje to, eliminując predykaty pojedynczego argumentu (predykaty jednoargumentowe) ze źródła. Uzasadnieniem tego jest to, że predykaty wyższej aryczności, z racji opisywania relacji między dwoma lub więcej bytami, z większym prawdopodobieństwem uchwycą systematyczne relacje, których celem jest analogia. W naszym przykładzie eliminuje to takie stwierdzenia, jak żółty (słońce) i niebieski (ziemia). Zauważ, że źródło może nadal zawierać stwierdzenia, takie jak gorętszy niż (słońce, ziemia), które nie są istotne dla analogii.

2. Mapa relacji niezmieniona od źródła do celu; argumenty dotyczące relacji mogą się różnić. W naszym przykładzie takie relacje, jak obraca się wokół i bardziej masywne, są takie same zarówno w źródle, jak i celu. To ograniczenie jest używane przez wiele teorii analogii i znacznie zmniejsza liczbę możliwych odwzorowań. Jest to również zgodne z heurystyką nadawania pierwszeństwa relacjom w mapowaniu.

3. Przy konstruowaniu odwzorowania preferowane są relacje wyższego rzędu jako centrum mapowania. W naszym przykładzie przyczyny są relacjami wyższego rzędu, ponieważ jako argumenty przyjmują inne relacje. Ten błąd odwzorowania nazywany jest zasadą systematyczności.

Te ograniczenia prowadzą do mapowania:

słońce → jądro

ziemia → elektron

Rozszerzenie mapowania prowadzi do wniosku:

przyczyny (bardziej masywne (jądro, elektron), przyciąganie (jądro, elektron))

przyczyny (przyciąganie (jądro, elektron), obroty wokół (elektron, jądro))

Teoria mapowania struktury została wdrożona i przetestowana w wielu dziedzinach. Choć daleki jest od pełnej teorii analogii, nie rozwiązując takich problemów, jak wyszukiwanie analogów źródłowych, okazał się zarówno praktyczny pod względem obliczeniowym, jak i zdolny do wyjaśnienia wielu aspektów ludzkiego rozumowania analogicznego. Wreszcie, jak zauważyliśmy w naszej prezentacji rozumowania opartego na przypadkach, analogia odgrywa kluczową rolę w tworzeniu i stosowaniu użytecznej podstawy przypadku

10.6 Uczenie się bez nadzoru

Omówione dotychczas algorytmy uczenia implementują formy uczenia się nadzorowanego. Zakładają istnienie nauczyciela, jakąś miarę sprawności lub inną zewnętrzną metodę klasyfikacji instancji szkoleniowych. Uczenie się bez nadzoru eliminuje nauczyciela i wymaga od uczniów samodzielnego tworzenia i oceny koncepcji. Nauka jest prawdopodobnie najlepszym przykładem uczenia się bez nadzoru u ludzi. Naukowcy nie mają korzyści nauczyciela. Zamiast tego proponują hipotezy wyjaśniające obserwacje; oceniać swoje hipotezy za pomocą takich kryteriów, jak prostota, ogólność i elegancja; i testować hipotezy za pomocą eksperymentów własnego projektu.

10.6.1 Odkrywanie i uczenie się bez nadzoru

AM jest jednym z najwcześniejszych i odnoszących największe sukcesy programów odkrywczych, wyprowadzającym wiele interesujących, nawet jeśli nie oryginalnych, koncepcji matematycznych. AM rozpoczął od koncepcji teorii mnogości, operacji tworzenia nowej wiedzy poprzez modyfikację i łączenie istniejących koncepcji oraz zestawu heurystyk do wykrywania „interesujących" koncepcji. Przeszukując tę przestrzeń pojęć matematycznych, AM odkrył liczby naturalne oraz kilka ważnych koncepcji teorii liczb, takich jak istnienie liczb pierwszych. Na przykład AM odkrył liczby naturalne, modyfikując swoje pojęcie „torby". Torba to uogólnienie zbioru, które pozwala na wielokrotne występowanie tego samego elementu. Na przykład {a, a, b, c, c} to worek. Specjalizując się w definicji worka, tak aby dopuszczał tylko jeden rodzaj elementu, AM odkrył analogię liczb naturalnych. Na przykład worek {1, 1, 1, 1} odpowiada liczbie 4. Połączenie worków doprowadziło do pojęcia dodawania: {1,1} ∪ {1,1} = {1,1,1,1 }, czyli 2 + 2 = 4. Badając dalsze modyfikacje tych pojęć, AM odkrył mnożenie jako serię dodawania. Korzystając z heurystyki, która definiuje nowe operatory poprzez odwracanie istniejących operatorów, AM odkrył dzielenie liczb całkowitych. Znalazł pojęcie liczb pierwszych, zauważając, że niektóre liczby mają dokładnie dwa dzielniki (siebie i 1). Tworząc nową koncepcję, AM ocenia ją zgodnie z szeregiem heurystyk, zachowując te koncepcje, które okazują się „interesujące". AM ustalił, że liczby pierwsze są interesujące na podstawie częstotliwości ich występowania. Oceniając koncepcje przy użyciu tej heurystyki, AM generuje wystąpienia koncepcji, testując każdą, aby sprawdzić, czy koncepcja jest prawdziwa. Jeśli koncepcja jest prawdziwa we wszystkich przypadkach, jest to tautologia, a AM nadaje jej niską ocenę. Podobnie AM odrzuca koncepcje, które nie są prawdziwe w przypadku braku instancji. Jeśli koncepcja jest prawdziwa dla znacznej części przykładów (jak ma to miejsce w przypadku liczb pierwszych), AM ocenia je jako interesujące i wybiera do dalszej modyfikacji. Chociaż AM odkrył liczby pierwsze i kilka innych interesujących koncepcji, nie wyszedł daleko poza elementarną teorię liczb. W późniejszej analizie tej pracy Lenat i Brown badają przyczyny sukcesu programu i jego ograniczenia. Chociaż Lenat

początkowo uważał, że heurystyka AM była głównym źródłem jego mocy, późniejsza ocena przypisała znaczną część sukcesu programu językowi używanemu do reprezentowania pojęć matematycznych. AM reprezentował koncepcje jako struktury rekurencyjne w odmianie języka programowania Lisp. Ze względu na podstawę w dobrze zaprojektowanym języku programowania reprezentacja ta definiowała przestrzeń, która zawierała dużą gęstość interesujących koncepcji. Było to szczególnie prawdziwe na wczesnych etapach poszukiwań. W miarę kontynuowania eksploracji przestrzeń powiększała się kombinatorycznie, a procent interesujących koncepcji „przerzedzał się". Ta obserwacja dodatkowo podkreśla związek między reprezentacją i wyszukiwaniem. Innym powodem, dla którego AM nie zdołał utrzymać imponującego tempa swoich wczesnych odkryć, jest jego niezdolność do „nauki uczenia się". Nie nabył nowych heurystyk w miarę zdobywania wiedzy matematycznej; w konsekwencji jakość jego poszukiwań pogorszyła się, gdy matematyka stała się bardziej złożona. W tym sensie AM nigdy nie rozwinęła głębokiego zrozumienia matematyki. Lenat zajął się tym problemem w późniejszych pracach nad programem o nazwie EURISKO, który próbuje nauczyć się nowych heurystyk. Szereg programów kontynuowało badanie problemów automatycznego wykrywania. IL stosuje różne techniki uczenia się do odkrywania matematyki, w tym metody takie jak dowodzenie twierdzeń i uczenie się oparte na wyjaśnieniach. BACON opracował modele obliczeniowe tworzenia ilościowych praw naukowych. Na przykład, korzystając z danych dotyczących odległości planet od Słońca i okresu ich orbit, BACON „ponownie odkrył" prawa ruchu planet Keplera. Dostarczając wiarygodny model obliczeniowy tego, w jaki sposób ludzie mogli dokonać odkrycia w różnych dziedzinach, BACON dostarczył użytecznego narzędzia i metodologii do badania procesu ludzkiego odkrywania naukowego. SCAVENGER użył odmiany algorytmu ID3 w celu poprawy jego zdolności do tworzenia użytecznych analogii. Shrager i Langley opisują szereg innych systemów wykrywania. Chociaż odkrycia naukowe są ważnym obszarem badań, dotychczasowe postępy są niewielkie. Bardziej podstawowy i być może bardziej owocny problem w uczeniu się bez nadzoru dotyczy odkrywania kategorii. Lakoff sugeruje, że kategoryzacja ma fundamentalne znaczenie dla ludzkiego poznania: wiedza teoretyczna wyższego poziomu zależy od umiejętności uporządkowania szczegółów naszego doświadczenia w spójne taksonomie. Większość naszej przydatnej wiedzy dotyczy kategorii obiektów, takich jak krowy, a nie konkretnych pojedynczych krów, takich jak Blossom czy Ferdinand. Nordhausen i Langley położyli nacisk na tworzenie kategorii jako podstawę jednolitej teorii odkryć naukowych. Przy opracowywaniu wyjaśnień, dlaczego chemikalia reagują w sposób, w jaki reagują, chemia opierała się na wcześniejszych pracach nad klasyfikowaniem związków na kategorie, takie jak „kwas" i „zasada". W następnej sekcji przyjrzymy się grupowaniu pojęciowemu, które jest problemem wykrywania przydatnych kategorii w niesklasyfikowanych danych.

10.6.2 Klastrowanie koncepcyjne

Problem grupowania zaczyna się od zbioru niesklasyfikowanych obiektów i środków do pomiaru podobieństwa obiektów. Celem jest zorganizowanie obiektów w klasy, które spełniają pewne standardy jakości, takie jak maksymalizacja podobieństwa obiektów w tej samej klasie. Taksonomia numeryczna jest jednym z najstarszych podejść do problemu grupowania. Metody numeryczne opierają się na reprezentacji obiektów jako zbioru cech, z których każda może mieć jakąś wartość liczbową. Miara rozsądnego podobieństwa traktuje każdy obiekt (wektor n wartości cech) jako punkt w przestrzeni n-wymiarowej. Podobieństwo dwóch obiektów to odległość euklidesowa między nimi w tej przestrzeni. Korzystając z tej miary podobieństwa, wspólny algorytm klastrowania tworzy klastry w sposób oddolny. Podejście to, często nazywane strategią klastrowania aglomeracyjnego, tworzy kategorie według:

1. Zbadanie wszystkich par obiektów, wybranie pary o najwyższym stopniu podobieństwa i uczynienie z tej pary skupienia.

2. Zdefiniowanie cech klastra jako pewnych funkcji, takich jak średnia cech elementów składowych, a następnie zastąpienie obiektów składowych tą definicją klastra.

3. Powtarzanie tego procesu na kolekcji obiektów, aż wszystkie obiekty zostaną zredukowane do jednego klastra.

4. Wiele algorytmów uczenia się bez nadzoru można postrzegać jako wykonujących estymacje maksymalnej gęstości prawdopodobieństwa, co oznacza znalezienie rozkładu, z którego najprawdopodobniej zostały narysowane dane. Przykładem jest interpretacja zestawu fonemów w aplikacji języka naturalnego, patrz rozdział 15.

Wynikiem tego algorytmu jest drzewo binarne, którego węzły-liścia są instancjami, a węzły wewnętrzne to skupiska rosnących rozmiarów. Możemy rozszerzyć ten algorytm na obiekty reprezentowane jako zbiory cech symbolicznych, a nie numerycznych. Jedynym problemem jest pomiar podobieństwa obiektów definiowanych za pomocą wartości symbolicznych, a nie liczbowych. Rozsądne podejście definiuje podobieństwo dwóch obiektów jako proporcję cech, które mają one wspólne. Biorąc pod uwagę przedmioty

object1 = {small, red, rubber, ball}

object2 = {small, blue, rubber, ball}

object3 = {large, black, wooden, ball}

ta metryka obliczyłaby wartości podobieństwa:

podobieństwo (obiekt1, obiekt2) = 3/4

podobieństwo (obiekt1, obiekt3) = podobieństwo (obiekt2, obiekt3) = 1/4

Jednak algorytmy tworzenia klastrów oparte na podobieństwie nie uwzględniają odpowiednio podstawowej roli wiedzy semantycznej w tworzeniu klastrów. Na przykład konstelacje gwiazd są opisywane zarówno na podstawie ich bliskości na niebie, jak i na podstawie istniejących ludzkich pojęć, takich jak „wielki wóz”. Przy definiowaniu kategorii nie możemy jednakowo traktować wszystkich cech. W danym kontekście pewne cechy obiektu są ważniejsze niż inne; proste wskaźniki podobieństwa traktują wszystkie funkcje jednakowo. Kategorie ludzkie zależą znacznie bardziej od celów kategoryzacji i wcześniejszej wiedzy w danej dziedzinie niż podobieństwa powierzchniowego. Weźmy na przykład pod uwagę klasyfikację wielorybów jako ssaków zamiast ryb. Podobieństwa powierzchni nie mogą wyjaśnić tej klasyfikacji, która zależy od szerszych celów klasyfikacji biologicznej i obszernych dowodów fizjologicznych i ewolucyjnych. Tradycyjne algorytmy grupowania nie tylko nie uwzględniają celów i podstawowej wiedzy pod uwagę, ale nie dają też znaczących semantycznych wyjaśnień powstałych kategorii. Te algorytmy reprezentują klastry w sposób rozszerzający, co oznacza wyliczenie wszystkich ich członków. Algorytmy nie generują definicji intencjonalnej lub nie ogólna reguła definiująca semantykę kategorii, która może być używana do klasyfikowania zarówno znanych, jak i przyszłych członków kategorii. Na przykład ekstensjonalna definicja zbioru osób, które służyły jako sekretarz generalny ONZ, po prostu wymienia te osoby. Definicja intensywna, taka jak:

{X | X został wybrany na Sekretarza Generalnego ONZ}

miałoby dodatkowe korzyści w postaci semantycznego definiowania klasy i umożliwienia nam rozpoznawania przyszłych członków kategorii. Klastrowanie koncepcyjne rozwiązuje te problemy, wykorzystując techniki uczenia maszynowego do tworzenia ogólnych definicji pojęć i stosując wiedzę podstawową do tworzenia kategorii. CLUSTER / 2 (Michalski i Stepp 1983) jest dobrym przykładem

tego podejścia. Wykorzystuje wiedzę podstawową w postaci uprzedzeń dotyczących języka używanego do reprezentowania kategorii. CLUSTER / 2 tworzy k kategorii poprzez konstruowanie indywiduów wokół k obiektów nasion. k to parametr, który może być dostosowywany przez użytkownika. CLUSTER / 2 ocenia wynik klastry, selekcję nowych nasion i powtarzanie procesu aż do spełnienia kryteriów jakościowych. Algorytm jest zdefiniowany:

1. Wybierz k nasion ze zbioru obserwowanych obiektów. Można to zrobić losowo lub zgodnie z jakąś funkcją wyboru.

2. Dla każdego ziarna, używając tego ziarna jako pozytywnej instancji, a wszystkich innych nasion jako negatywnych, stwórz maksymalnie ogólną definicję, która obejmuje wszystkie pozytywne i żadne negatywne instancje. Zwróć uwagę, że może to prowadzić do wielu klasyfikacji innych, niepasowanych obiektów.

3. Sklasyfikuj wszystkie obiekty w próbce zgodnie z tymi opisami. Zastąp każdy maksymalnie ogólny opis maksymalnie szczegółowym opisem obejmującym wszystkie obiekty w kategorii. Zmniejsza to prawdopodobieństwo, że klasy nakładają się na niewidoczne obiekty.

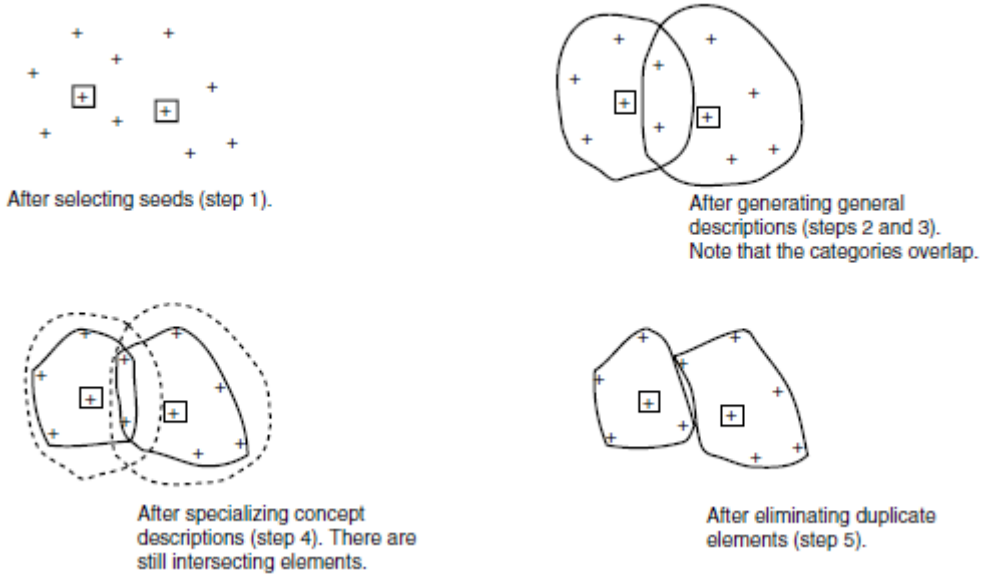
4. Zajęcia mogą nadal nakładać się na określone obiekty. CLUSTER / 2 zawiera algorytm dostosowywania nakładających się definicji.

5. Korzystając z metryki odległości, wybierz element najbliżej środka każdej klasy. Metryka odległości może być podobna do metryki podobieństwa omówionej powyżej.

6. Używając tych centralnych elementów jako nowych nasion, powtórz kroki 1–5. Zatrzymaj się, gdy klastry będą satysfakcjonujące. Typową miarą jakości jest złożoność ogólnych opisów klas. Na przykład odmiana Brzytwy Occam'a może preferować klastry, które dają proste syntaktycznie definicje, takie jak te z niewielką liczbą spójników.

7. Jeśli klastry są niezadowalające i po kilku iteracjach nie następuje poprawa, wybierz nowe ziarna znajdujące się najbliżej krawędzi klastra, a nie te w środku.

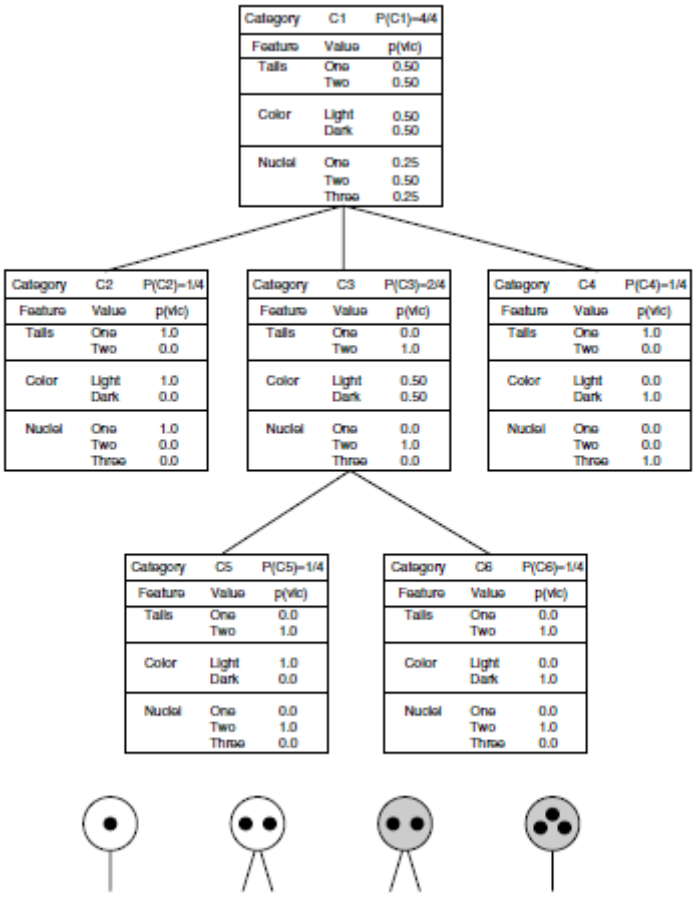
Rysunek 20 przedstawia etapy wykonywania CLUSTER / 2.



10.6.3 COBWEB i struktura wiedzy taksonomicznej

Wiele algorytmów grupujących, a także wiele algorytmów uczenia nadzorowanego, takich jak ID3, definiuje kategorie pod względem niezbędnych i wystarczających warunków członkostwa. Warunki te są zbiorem właściwości posiadanych przez wszystkich członków kategorii i tylko przez członków kategorii. Chociaż wiele kategorii, takich jak zbiór wszystkich delegatów ONZ, może być tak zdefiniowanych, kategorie ludzkie nie zawsze pasują do tego modelu. Rzeczywiście, kategoryzacja ludzi charakteryzuje się większą elastycznością i znacznie bogatszą strukturą niż dotychczas badaliśmy. Na przykład, jeśli ludzkie kategorie byłyby rzeczywiście zdefiniowane przez niezbędne i wystarczające warunki członkostwa, nie moglibyśmy rozróżnić stopni przynależności do kategorii, jednak psychologowie zauważyli silne poczucie prototypowości w kategoryzacji ludzi. Na przykład, generalnie myślimy o rudzie jako lepszym przykładzie ptaka niż kurczaka; dąb jest bardziej typowym przykładem drzewa niż palma (przynajmniej na północnych szerokościach geograficznych). Teoria podobieństwa rodzinnego wspiera te pojęcia prototypowości, argumentując, że kategorie są definiowane przez złożone systemy podobieństw między członkami, a nie przez konieczne i wystarczające warunki członkostwa. Takie kategorie nie mogą mieć żadnych właściwości wspólnych dla wszystkich ich członków. Wittgenstein podaje przykład gier: nie wszystkie gry wymagają dwóch lub więcej graczy, jak na przykład pasjans; nie wszystkie gry są zabawne dla graczy, jak np. rochambeau; nie wszystkie gry mają dobrze sformułowane zasady, takie jak gry w udawanie dla dzieci; i nie wszystkie gry obejmują zawody, takie jak skakanka. Niemniej uważamy, że kategoria jest dobrze zdefiniowana i jednoznaczna. Kategorie ludzkie różnią się również od większości formalnych hierarchii dziedziczenia tym, że nie wszystkie poziomy taksonomii ludzi są równie ważne. Psychologowie (Rosch 1978) wykazali istnienie kategorii na poziomie podstawowym. Kategoria poziomu podstawowego jest klasyfikacją najczęściej używaną przy opisywaniu obiektów, terminologią, której dzieci nauczyły się jako pierwsza, oraz poziomem, który w pewnym sensie obejmuje najbardziej fundamentalną klasyfikację obiektu. Na przykład kategoria „krzesło” jest bardziej podstawowa niż jej uogólnienia, takie jak „meble”, lub jej specjalizacje, takie jak „krzesło biurowe”. „Samochód” jest bardziej podstawowy niż „sedan” lub „pojazd”. Powszechne metody reprezentowania przynależności do klas i hierarchii, takie jak logika, systemy dziedziczenia, wektory cech lub drzewa decyzyjne, nie uwzględniają tych efektów. Jednak zrobienie tego jest ważne nie tylko dla kognitywistów, których celem jest zrozumienie ludzkiej inteligencji; jest również cenny dla inżynierii użytecznych aplikacji AI. Użytkownicy oceniają program pod kątem jego elastyczności, solidności i zdolności do zachowywania się w sposób, który wydaje się rozsądny według ludzkich standardów. Chociaż nie wymagamy, aby algorytmy sztucznej inteligencji były równoległe z architekturą ludzkiego umysłu, każdy algorytm, który proponuje odkrywanie kategorii, musi spełniać oczekiwania użytkownika co do struktury i zachowania tych kategorii. COBWEB porusza te kwestie. Chociaż nie jest pomyślany jako model ludzkiego poznania, uwzględnia kategoryzację na poziomie podstawowym i stopnie przynależności do kategorii. Ponadto COBWEB uczy się stopniowo: nie wymaga, aby wszystkie instancje były obecne przed rozpoczęciem uczenia się. W wielu aplikacjach uczeń uzyskuje dane w czasie. W takich sytuacjach musi stworzyć użyteczne opisy pojęć na podstawie początkowego zbioru danych i aktualizować te opisy, gdy będzie dostępnych więcej danych. COBWEB rozwiązuje również problem określenia prawidłowej liczby klastrów. CLUSTER / 2 stworzył określoną liczbę kategorii. Chociaż użytkownik może zmieniać tę liczbę lub algorytm mógłby wypróbować różne wartości, starając się poprawić kategoryzację, takie podejścia nie są szczególnie elastyczne. COBWEB wykorzystuje globalne metryki jakości do określenia liczby klastrów, głębokości hierarchii i przynależności do kategorii nowych instancji. W przeciwieństwie do algorytmów, które widzieliśmy do tej pory, COBWEB reprezentuje kategorie probabilistycznie. Zamiast definiować przynależność do kategorii jako zbiór wartości, które muszą być obecne dla każdej cechy obiektu, COBWEB reprezentuje prawdopodobieństwo, z jakim każda wartość cechy jest obecna. p (fi = vij□ck) to prawdopodobieństwo warunkowe (sekcja 5.2), z którym cecha fi będzie miała wartość vij, biorąc pod uwagę, że obiekt należy do kategorii ck

Rysunek 21 przedstawia taksonomię COBWEB zaczerpniętą z Gennariego i innych.



W tym przykładzie algorytm utworzył kategoryzację czterech zwierząt jednokomórkowych u dołu rysunku. Każde zwierzę jest zdefiniowane przez wartość cech: kolor oraz liczbę ogonów i jąder. Na przykład kategoria C3 ma prawdopodobieństwo 1,0 posiadania 2 ogonów, 0,5 prawdopodobieństwa posiadania jasnego koloru i 1,0 prawdopodobieństwo posiadania 2 jąder. Jak pokazano na rysunku, każda kategoria w hierarchii zawiera prawdopodobieństwa wystąpienia wszystkich wartości wszystkich cech. Jest to niezbędne zarówno do kategoryzowania nowych instancji, jak i do modyfikowania struktury kategorii w celu lepszego dopasowania do nowych instancji. Rzeczywiście, jako algorytm przyrostowy COBWEB nie oddziela tych działań. W przypadku otrzymania nowej instancji COBWEB bierze pod uwagę ogólną jakość umieszczenia instancji w istniejącej kategorii lub zmodyfikowania hierarchii w celu dostosowania jej do instancji. Kryterium stosowane przez COBWEB do oceny jakości klasyfikacji nazywa się użytecznością kategorii. Użyteczność kategorii została opracowana w badaniach nad kategoryzacją ludzi. Uwzględnia efekty poziomu podstawowego i inne aspekty struktury kategorii ludzi. Użyteczność kategorii próbuje zmaksymalizować zarówno prawdopodobieństwo, że dwa obiekty w tej samej kategorii mają wspólne wartości, jak i prawdopodobieństwo, że obiekty z różnych kategorii będą miały różne wartości właściwości. Kategoria użyteczność jest zdefiniowana:

$$\sum_k \sum_i \sum_j p(f_i=v_{ij}) p(f_i=v_{ij}|c_k) p(c_k|f_i=v_{ij})$$

Suma ta jest brana ze wszystkich kategorii, c_k, wszystkich cech, i wszystkich wartości cech, v_{ij}. p ($f_i = v_{ij} \mid c_k$), zwane przewidywalnością, to prawdopodobieństwo, że obiekt ma wartość v_{ij} dla cechy f_i przy założeniu, że obiekt należy do kategorii c_k. Im wyższe to prawdopodobieństwo, tym bardziej prawdopodobne jest, że dwa obiekty w kategorii mają te same wartości cech. p ($c_k | f_i = v^ij$), zwane predyktywnością, to prawdopodobieństwo, z jakim obiekt należy do kategorii ck, biorąc pod uwagę, że ma wartość v_{ij} dla cechy f_i. Im większe to prawdopodobieństwo, tym mniej prawdopodobne, że obiekty spoza kategorii będą miały te wartości cech. p (f_i = vij) służy jako waga, zapewniając, że często występujące wartości cech będą miały silniejszy wpływ na ocenę. Łącząc te wartości, miary użyteczności wysokiej kategorii wskazują na duże prawdopodobieństwo, że obiekty z tej samej kategorii będą miały wspólne właściwości, jednocześnie zmniejszając prawdopodobieństwo, że obiekty z różnych kategorii mają wspólne właściwości. Algorytm COBWEB jest zdefiniowany:

cobweb(Node, Instance)

begin

if Node is a leaf

then begin

create two children of Node, L_1 and L_2;

set the probabilities of L_1 to those of Node;

initialize the probabilities for L_2 to those of Instance;

add Instance to Node, updating Node s probabilities;

end

else begin

add Instance to Node, updating Node s probabilities;

for each child, C, of Node, compute the category utility of the clustering

achieved by placing Instance in C;

let S_1 be the score for the best categorization, C_1;

let S_2 be the score for the second best categorization, C_2;

let S_3 be the score for placing instance in a new category;

let S_4 be the score for merging C1 and C2 into one category;

let S_5 be the score for splitting C1 (replacing it with its child categories)

end

If S_1 is the best score

then cobweb(C_1, Instance) % place the instance in C_1

else if S_3 is the best score

then initialize the new category s probabilities to those of Instance

else if S_4 is the best score

then begin

let Cm be the result of merging C_1 and C_2;

cobweb(C_m, Instance)

end

else if S_5 is the best score

then begin

split C_1;

cobweb(Node, Instance)

end;

end

COBWEB przeprowadza wspinaczkowe przeszukiwanie przestrzeni możliwych taksonomii, wykorzystując użyteczność kategorii do oceny i wyboru możliwych kategoryzacji. Inicjalizuje taksonomię do pojedynczej kategorii, której cechy są cechami pierwszej instancji. Dla każdej kolejnej instancji algorytm rozpoczyna się od kategorii głównej i przechodzi przez drzewo. Na każdym poziomie wykorzystuje użyteczność kategorii do oceny taksonomii wynikających z:

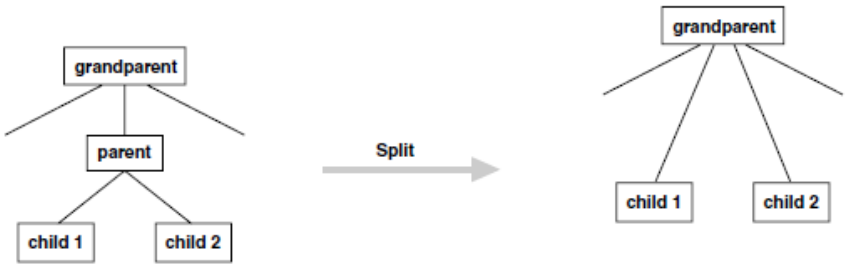
1. Umieszczenie instancji w najlepszej istniejącej kategorii.

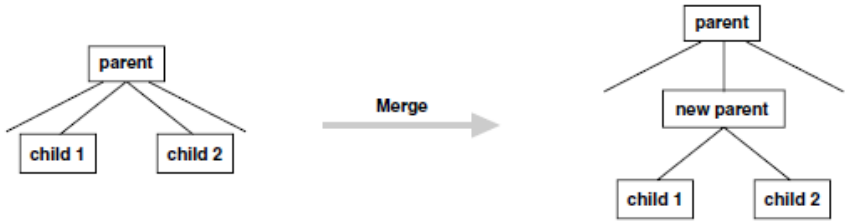
2. Dodanie nowej kategorii zawierającej tylko instancję.

3. Połączenie dwóch istniejących kategorii w jedną nową i dodanie instancji do tej kategorii.

4. Podział istniejącej kategorii na dwie i umieszczenie instancji w najlepszej nowej kategorii wynikowej.

Rysunek 10.22 ilustruje procesy scalania i dzielenia węzłów.





Aby scalić dwa węzły, algorytm tworzy nowy węzeł i czyni istniejące węzły dziećmi tego węzła. Oblicza prawdopodobieństwa dla nowego węzła, łącząc prawdopodobieństwa dla dzieci. Podział zastępuje węzeł jego potomkami. Ten algorytm jest wydajny i tworzy taksonomie z rozsądną liczbą klas. Ponieważ umożliwia członkostwo probabilistyczne, jego kategorie są elastyczne i solidne. Ponadto wykazał efekty kategorii na poziomie podstawowym i, poprzez pojęcie częściowego dopasowania kategorii, wspiera pojęcia prototypowości i stopnia członkostwa. Zamiast polegać na logice dwuwartościowej, COBWEB, podobnie jak logika rozmyta, postrzega „niejasność" przynależności do kategorii jako niezbędny element uczenia się i rozumowania w sposób elastyczny i inteligentny. Następnie przedstawiamy uczenie się ze wzmocnieniem, które, podobnie jak systemy klasyfikatorów, interpretuje informacje zwrotne ze środowiska, aby nauczyć się optymalnych zestawów relacji warunek / odpowiedź do rozwiązywania problemów w tym środowisku.

10.7 Uczenie się ze wzmocnieniem

My, ludzie (zwykle!) Uczymy się na podstawie interakcji z naszym środowiskiem. Chwila namysłu przypomina jednak, że informacje zwrotne z naszych działań na świecie nie zawsze są natychmiastowe i proste. Na przykład w relacjach międzyludzkich często potrzeba trochę czasu, aby w pełni docenić rezultaty naszych działań. Interakcja ze światem pokazuje nam przyczynę i skutek (Pearl 2000), konsekwencje naszych działań, a nawet sposób osiągania złożonych celów. Jako inteligentni agenci tworzymy zasady dotyczące pracy w naszym świecie i za jego pośrednictwem. „Świat" jest nauczycielem, ale jej lekcje są często subtelne, a czasem trudno je wygrać!

10.7.1 Elementy uczenia się ze wzmocnieniem

W uczeniu się ze wzmocnieniem projektujemy algorytmy obliczeniowe do przekształcania sytuacji na świecie w działania w sposób maksymalizujący miarę nagrody. Naszemu agentowi nie mówi się bezpośrednio, co ma robić ani jakie działania podjąć; raczej agent odkrywa poprzez eksplorację, które działania oferują najwięcej korzyści. Działania agentów wpływają nie tylko na natychmiastową nagrodę, ale także na późniejsze działania i ostateczne nagrody. Te dwie cechy, wyszukiwanie metodą prób i błędów oraz opóźnione wzmacnianie, to dwie najważniejsze cechy uczenia się przez wzmacnianie. W konsekwencji uczenie się przez wzmacnianie jest bardziej ogólną metodologią niż nauka opisana wcześniej w tym rozdziale. Uczenie się ze wzmocnieniem nie jest definiowane przez konkretne metody uczenia się, ale przez działania w obrębie środowiska i reakcje z niego. Każda metoda uczenia się tworząca tę interakcję jest dopuszczalną metodą uczenia się ze wzmocnieniem. Uczenie się ze wzmocnieniem również nie jest nadzorowane, co widać w naszych rozdziałach o uczeniu maszynowym. W uczeniu się nadzorowanym „nauczyciel" używa przykładów, aby bezpośrednio instruować lub szkolić ucznia. W uczeniu się ze wzmocnieniem sam agent uczący się, metodą prób, błędów i informacji zwrotnych, uczy się optymalnej polityki dla osiągnięcia celów w swoim środowisku. Kolejny kompromis, który uczący się wzmacniania musi wziąć pod uwagę, to używanie tylko tego, co obecnie wie, a dalsze odkrywanie jego świata. Aby zoptymalizować swoje możliwości nagradzania, agent musi nie tylko robić to, co już wie, ale także eksplorować te części swojego świata, które są nadal nieznane. Eksploracja umożliwia agentowi (prawdopodobnie) dokonywanie lepszych wyborów w przyszłości; zatem oczywiście agent, który zawsze lub nigdy nie bada, zwykle zawiedzie. Agent musi zbadać różne opcje i jednocześnie faworyzować te, które wydają się najlepsze. W przypadku zadań z parametrami stochastycznymi działania eksploracyjne muszą być często wykonywane wielokrotnie, aby uzyskać wiarygodne oszacowania nagród. Wiele algorytmów rozwiązywania problemów przedstawionych wcześniej w tej książce, w tym planiści, decydenci i algorytmy wyszukiwania, można rozpatrywać w kontekście uczenia się przez wzmacnianie. Na przykład, możemy stworzyć plan ze

sterownikiem teloreaktywnym (rozdział 7.4), a następnie ocenić jego powodzenie za pomocą algorytmu uczenia się przez wzmocnienie. W rzeczywistości algorytm wzmocnienia DYNA-Q (Sutton 1990, 1991) integruje uczenie się modelu z planowaniem i działaniem. Zatem uczenie się ze wzmocnieniem oferuje metodę oceny zarówno planów, jak i modeli oraz ich przydatności do wykonywania zadań w złożonych środowiskach. Przedstawiamy teraz terminologię dotyczącą uczenia się ze wzmocnieniem:

t jest dyskretnym krokiem w procesie rozwiązywania problemu

s_t jest stanem problemu w momencie t, zależnym od s_{t-1} i przy a_{t-1}

a_t to działanie w czasie t, zależne od s_t

r_t jest nagrodą w momencie t, zależną od s_{t-1} i a_{t-1}

π to zasada podejmowania działań w państwie. Zatem π jest odwzorowaniem stanów na działania

π * to optymalna zasada

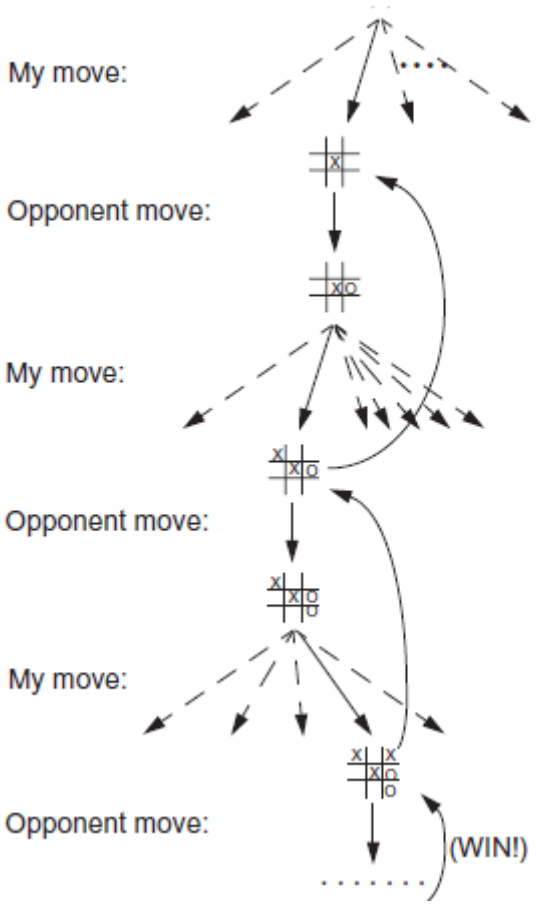
V mapuje stan na jego wartość. Zatem V^{π} (s) jest wartością stanu s w ramach polityki π

W sekcji 10.7.2, uczenie się różnic czasowych uczy się V dla każdego s przy statycznym π. Istnieją cztery komponenty uczenia się przez wzmacnianie, polityka π, funkcja nagrody r, mapowanie wartości V i dość często model środowiska. Polityka określa wybory i metody działania agenta uczenia się w danym momencie. W ten sposób polityka może być reprezentowana przez zestaw reguł produkcji lub prostą tabelę przeglądową. Polityka dotycząca konkretnej sytuacji, jak właśnie wspomniano, może być również wynikiem szeroko zakrojonych poszukiwań, konsultacji modelu lub procesu planowania. Może też być stochastyczny. Polityka jest krytycznym elementem agenta uczenia się, ponieważ sama w sobie wystarcza do wytworzenia zachowania w dowolnym momencie. Funkcja nagrody r_t definiuje relacje stan / cel problemu w czasie t. Mapuje każde działanie, a dokładniej każdą parę stan-reakcja, na miarę nagrody, wskazując na celowość tego działania dla osiągnięcia celu. Zadaniem agenta w uczeniu się przez wzmacnianie jest maksymalizacja całkowitej nagrody, jaką otrzymuje za wykonanie swojego zadania. Funkcja wartości V jest właściwością każdego stanu środowiska wskazującą nagrodę, jakiej system może oczekiwać za działania kontynuowane z tego stanu. Podczas gdy funkcja nagrody mierzy natychmiastową pożądalność par stan-odpowiedź, funkcja wartości wskazuje na długoterminową pożądalność stanu środowiska. Państwo czerpie swoją wartość zarówno z własnej wewnętrznej jakości, jak i z jakości stanów, które mogą z niego wynikać, tj. z nagrody za bycie w tych stanach. Na przykład stan / działanie może mieć niską natychmiastową nagrodę, ale mieć wysoką wartość, ponieważ zwykle następuje po nim inne stany, które przynoszą wysoką nagrodę. Niska wartość może również wskazywać stany, które nie są skojarzone z ostatecznie udanymi ścieżkami rozwiązań. Bez funkcji nagrody nie ma wartości, a jedynym celem szacowania wartości jest uzyskanie większej liczby nagród. W podejmowaniu decyzji najbardziej interesują nas jednak wartości, ponieważ wartości wskazują stany i kombinacje stanów, które przynoszą największe korzyści. Jednak znacznie trudniej jest określić wartości dla stanów niż określić nagrody. Nagrody są przyznawane bezpośrednio przez środowisko, podczas gdy wartości są szacowane, a następnie ponownie szacowane na podstawie sukcesów i porażek w czasie. W rzeczywistości najbardziej krytycznym i najtrudniejszym aspektem uczenia się przez wzmacnianie jest stworzenie metody efektywnego określania wartości. Jedną metodę, regułę uczenia się różnic czasowych, przedstawiamy w sekcji 10.7.2. Ostatnim - i opcjonalnym - elementem uczenia się przez wzmacnianie jest model środowiska. Model jest mechanizmem służącym do wychwytywania aspektów zachowania środowiska. Jak widzieliśmy w rozdziale 7.3, modele mogą być wykorzystywane nie tylko do określania błędów, jak w rozumowaniu

diagnostycznym, ale także jako część określania planu działania. Modele pozwalają nam ocenić możliwe przyszłe działania, nie doświadczając ich w rzeczywistości. Planowanie oparte na modelach jest niedawnym dodatkiem do paradygmatu uczenia się przez wzmacnianie, ponieważ wczesne systemy miały tendencję do tworzenia parametrów nagrody i wartości w oparciu o czystą metodę prób i błędów agenta.

10.7.2 Przykład: ponownie w kółko i krzyżyk

Następnie przedstawiamy algorytm uczenia się ze wzmocnieniem dla kółko i krzyżyk, problem, który już rozważaliśmy, oraz jeden, którym zajmowali się Suttton i Barto . Ważne jest, aby porównać i kontrastować podejście uczenia się przez wzmacnianie z innymi metodami rozwiązania, na przykład mini-max. Przypominamy, że gra w kółko i krzyżyk to gra dwuosobowa rozgrywana na siatce 3x3. Gracze, X i O, naprzemiennie umieszczają swoje znaki na siatce, a pierwszy gracz, który otrzyma trzy znaki z rzędu, poziomo, pionowo lub po przekątnej, jest zwycięzcą. Jak czytelnik jest świadomy, kiedy ta gra jest rozgrywana przy użyciu doskonałych informacji i potwierdzonych wartości, zawsze jest to remis. Jednak dzięki uczeniu się przez wzmacnianie będziemy mogli zrobić coś znacznie bardziej interesującego. Pokażemy, jak możemy uchwycić grę niedoskonałego przeciwnika i stworzyć politykę, która pozwoli nam zmaksymalizować naszą przewagę nad tym przeciwnikiem. Nasza polityka może również ewoluować, gdy nasz przeciwnik ulepszy swoją grę, a za pomocą modelu będziemy w stanie generować rozwidlenia i inne ruchy atakujące! Najpierw musimy ustawić tabelę liczb, po jednej dla każdego możliwego stanu gry. Te liczby, czyli wartość tego stanu, będą odzwierciedlać aktualne oszacowanie prawdopodobieństwa wygrania z tego stanu. Będzie to wspierać politykę ściśle wygrywającej strategii, tj. Jeśli wygrana naszego przeciwnika lub remis będzie dla nas uznany za przegraną. To podejście polegające na remisie-przegranej pozwala nam na ustanowienie polityki ukierunkowanej na wygraną i różni się od naszego doskonałego modelu informacyjnego wygrana-przegrana-remis .W rzeczywistości jest to ważna różnica; staramy się uchwycić umiejętności rzeczywistego przeciwnika, a nie doskonałe informacje o jakimś wyidealizowanym przeciwniku. W ten sposób zainicjujemy naszą tabelę wartości z 1 za każdą wygraną dla nas pozycję, 0 za każdą przegraną lub remisowaną planszę i 0,5 wszędzie indziej, odzwierciedlając początkowe przypuszczenie, że mamy 50% szans na wygraną w tych stanach. . Gramy teraz wiele meczów z tym przeciwnikiem. Dla uproszczenia załóżmy, że jesteśmy X i naszym przeciwnikiem 0. Rysunek 23 odzwierciedla sekwencję możliwych ruchów, oba rozważanych i wybranych w danej sytuacji w grze. Aby wygenerować ruch, najpierw rozważamy każdy stan, który jest legalnym przesunięciem z naszego obecnego stanu, to znaczy każdy stan otwarty, do którego możemy się zgłosić za pomocą naszego X. Sprawdzamy bieżącą miarę wartości tego stanu przechowywaną w naszej tabeli. W większości przypadków możemy wykonać chciwy ruch, czyli przyjąć stan, który ma najlepszą funkcję wartości. Czasami będziemy chcieli wykonać ruch eksploracyjny i wybrać losowo spośród innych stanów. Te ruchy eksploracyjne mają na celu rozważenie alternatyw możemy nie dostrzec w sytuacji w grze, rozszerzając możliwe optymalizacje wartości. Podczas gry zmieniamy funkcje wartości każdej z wybranych przez nas statystyk. Staramy się, aby ich najnowsze wartości lepiej odzwierciedlały prawdopodobieństwo znalezienia się na zwycięskiej ścieżce. Wcześniej nazwaliśmy to funkcją nagrody dla stanu. Aby to zrobić, wykonujemy kopię zapasową stanu, który wybraliśmy, jako funkcję wartości następnego wybranego stanu. Jak widać za pomocą „strzałek w górę” na rysunku 23,



ta akcja rezerwowa pomija wybór naszego przeciwnika, a mimo to odzwierciedla zestaw wartości, do których skierowała nas przy następnym wyborze stanu. W ten sposób bieżąca wartość wcześniejszego stanu, który wybieramy, jest dostosowywana tak, aby lepiej odzwierciedlała wartość późniejszego stanu (i ostatecznie oczywiście wartości wygrywającej lub tracącej). Zwykle robimy to, przenosząc poprzedni stan o pewien ułamek różnicy wartości między nim a nowszym stanem, który wybraliśmy. Ta miara ułamkowa, zwana parametrem wielkości kroku, jest odzwierciedlona przez mnożnik c w równaniu:

$V(s_n) = V(s_n) + c\,(V(s_{n+1}) - V(s_n))$

W tym równaniu sn reprezentuje stan wybrany w czasie n, a s_{n+1} stan wybrany w czasie n + 1. To równanie aktualizacji jest przykładem reguły uczenia się różnicy czasowej, ponieważ jej zmiany są funkcją różnicy $V(s_{n+1}) - V(s_n)$ między oszacowaniami wartości w dwóch różnych momentach, n i n +1. Omówimy te zasady uczenia się dalej w następnej sekcji. Reguła różnicy czasowej działa całkiem dobrze w przypadku kółko i krzyżyk. Chcemy skrócić czas pokrycia parametru wielkości kroku tak, aby w miarę uczenia się systemu dokonywane były sukcesywnie mniejsze korekty wartości stanu. Gwarantuje to zbieżność funkcji wartości dla każdego stanu z prawdopodobieństwem wygranej, biorąc pod uwagę naszego przeciwnika. Poza tym, z wyjątkiem okresowych posunięć eksploracyjnych, dokonane wybory będą de facto posunięciami optymalnymi, czyli optymalną polityką wobec tego przeciwnika. Jednak jeszcze bardziej interesujący jest fakt, że jeśli wielkość kroku nigdy nie spadnie do zera, polityka ta będzie się nieustannie zmieniać, aby odzwierciedlić wszelkie zmiany / ulepszenia w grze przeciwnika . Nasz przykład „kółko i krzyżyk” ilustruje wiele ważnych cech uczenia się przez wzmacnianie. Najpierw

uczenie się podczas interakcji z otoczeniem, tutaj naszym przeciwnikiem. Po drugie, istnieje wyraźny cel (odzwierciedlony w szeregu stanów celu), a optymalne zachowanie wymaga planowania i patrzenia w przyszłość, które uwzględniają opóźnione efekty poszczególnych ruchów. Na przykład, algorytm uczenia się przez wzmocnienie w efekcie ustawia pułapki na wiele ruchów dla naiwnego przeciwnika. Ważną cechą uczenia się przez wzmacnianie jest to, że efekty patrzenia w przyszłość i planowania można w rzeczywistości osiągnąć bez wyraźnego modelu przeciwnika lub poprzez rozszerzone poszukiwania. W naszym przykładzie „kółko i krzyżyk" nauka rozpoczęła się bez wcześniejszej wiedzy wykraczającej poza zasady gry. (Po prostu zainicjowaliśmy wszystkie stany nieterminalne na 0.5.) Uczenie się ze wzmocnieniem z pewnością nie wymaga tego widoku „pustej karty". Wszelkie dostępne wcześniejsze informacje można wbudować w wartości stanu początkowego. Możliwe jest również zajmowanie się państwami, w których nie ma dostępnych informacji. Wreszcie, jeśli model sytuacji jest dostępny, wynikowe informacje oparte na modelu można wykorzystać jako wartości stanu. Należy jednak pamiętać, że uczenie się przez wzmacnianie można zastosować w każdej sytuacji: żaden model nie jest wymagany, ale modele mogą być używane, jeśli są dostępne lub można się ich nauczyć. W przykładzie „kółko i krzyżyk" nagroda była amortyzowana za każdą decyzję dotyczącą działania państwa. Nasz agent był krótkowzroczny, zainteresowany maksymalizacją tylko natychmiastowych nagród. Rzeczywiście, jeśli użyjemy głębszego spojrzenia w przód z uczeniem się przez wzmocnienie, będziemy musieli zmierzyć zdyskontowany zwrot ostatecznej nagrody. Przyjmujemy, że stopa dyskontowa γ reprezentuje wartość bieżącą przyszłej nagrody: nagroda otrzymana k kroków czasowych w przyszłości jest warta tylko γ^{k-1} razy więcej niż byłaby warta, gdyby została otrzymana natychmiast. To dyskontowanie miary nagrody jest ważne przy stosowaniu podejścia programowania dynamicznego do uczenia się przez wzmacnianie przedstawione w następnej sekcji. Kółko i krzyżyk był przykładem gry dwuosobowej. Uczenie się ze wzmocnieniem może być również wykorzystywane w sytuacjach, w których nie ma przeciwników, a jedynie informacje zwrotne z otoczenia.

Przykład gry w kółko i krzyżyk miał również ograniczoną (i właściwie dość małą) przestrzeń stanów. Uczenie się ze wzmocnieniem można również zastosować, gdy przestrzeń stanów jest bardzo duża, a nawet nieskończona. W późniejszym przypadku wartości stanu są generowane tylko wtedy, gdy ten stan zostanie napotkany i użyty w rozwiązaniu. Na przykład Tesauro (1995) użył opisanej właśnie reguły różnicy czasowej, wbudowanej w sieć neuronową, aby nauczyć się grać w backgammona. Mimo że szacowany rozmiar przestrzeni stanów tryktraka to 1020 stanów, program Tesauro gra na poziomie najlepszych graczy.

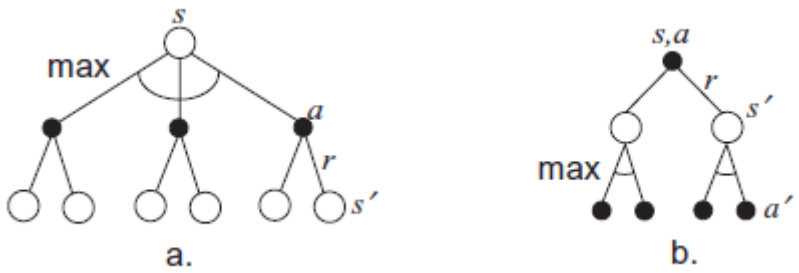
10.7.3 Algorytmy wnioskowania i aplikacje do uczenia się ze wzmocnieniem

Według Suttona i Barto istnieją trzy różne rodziny algorytmów wnioskowania uczenia się przez wzmocnienie: uczenie się z różnicami czasowymi, programowanie dynamiczne i metody Monte Carlo. Te trzy elementy stanowią podstawę praktycznie wszystkich obecnych podejść do uczenia się przez wzmacnianie. Metody różnic czasowych uczą się z próbkowanych trajektorii i tworzą kopie zapasowe wartości ze stanu do stanu. W poprzedniej sekcji widzieliśmy przykład uczenia się różnic czasowych za pomocą kółka i krzyżyka. Metody programowania dynamicznego obliczają funkcje wartości poprzez tworzenie kopii zapasowych wartości ze stanów następnych do stanów poprzedników. Metody programowania dynamicznego systematycznie aktualizują jeden stan po drugim, w oparciu o model rozkładu kolejnego stanu. Podejście do programowania dynamicznego opiera się na fakcie, że dla dowolnej polityki π i dowolnego stanu s zachodzi następujące rekurencyjne równanie spójności:

$$V\pi (s) = \Sigma\pi (a \mid s) * \Sigma\pi (s \rightarrow s \mid a) * (Ra (s \rightarrow s) + \gamma (V\pi (s))$$

π (a | s) jest prawdopodobieństwem zadziałania danego państwa w ramach polityki stochastycznej π. π (s → s | a) jest prawdopodobieństwem wystąpienia s pod działaniem a. To jest równanie Bellmana

dla V^{π}. Wyraża związek między wartością stanu a rekurencyjnie obliczonymi wartościami jego kolejnych stanów. Na rysunku 24a przedstawiamy obliczenie pierwszego kroku, w którym ze stanu s oczekujemy trzech możliwych następnych stanów.



W przypadku polityki π działanie a ma prawdopodobieństwo wystąpienia π (a | s). Z każdego z tych trzech stanów środowisko może odpowiedzieć jednym z kilku stanów, powiedzmy s nagrodą r. Równanie Bellmana uśrednia wszystkie te możliwości, ważąc każdą z nich według prawdopodobieństwa wystąpienia. Stwierdza, że wartość stanu początkowego s musi być równa zdyskontowanej wartości γ oczekiwanych kolejnych stanów plus nagroda wygenerowana na ścieżce.

Klasyczne modele programowania dynamicznego mają ograniczoną użyteczność ze względu na założenie idealnego modelu. Jeśli n i m oznaczają liczbę stanów i akcji, metoda programowania dynamicznego gwarantuje znalezienie optymalnej polityki w czasie wielomianowym, nawet jeśli całkowita liczba polityk deterministycznych wynosi nm. W tym sensie programowanie dynamiczne jest wykładniczo szybsze niż jakiekolwiek bezpośrednie wyszukiwanie w przestrzeni polityki, ponieważ bezpośrednie wyszukiwanie musiałoby wyczerpująco ocenić każdą politykę, aby dać taką samą gwarancję. Metody Monte Carlo nie wymagają pełnego modelu. Zamiast tego próbkują całe trajektorie stanów, aby zaktualizować funkcję wartości na podstawie końcowych wyników odcinków. Metody Monte Carlo wymagają doświadczenia, czyli przykładowych sekwencji stanów, działań i nagród z on-line lub symulowanych interakcji z otoczeniem. Doświadczenie on-line jest interesujące, ponieważ nie wymaga wcześniejszej znajomości środowiska, ale nadal może być optymalne. Uczenie się na podstawie symulowanego doświadczenia jest również potężne. Model jest wymagany, ale może być raczej generatywny niż analityczny, to znaczy model zdolny do generowania trajektorii, ale nie zdolny do obliczania jawnych prawdopodobieństw. W związku z tym nie musi tworzyć pełnych rozkładów prawdopodobieństwa wszystkich możliwych przejść, które są wymagane w programowaniu dynamicznym. Zatem metody Monte Carlo rozwiązują problem uczenia się ze wzmocnieniem poprzez uśrednianie przykładowego zwrotu. Aby zapewnić dobrze zdefiniowane zwroty, metody Monte Carlo są definiowane tylko dla pełnych odcinków, to znaczy wszystkie odcinki muszą ostatecznie zostać zakończone. Co więcej, dopiero po zakończeniu odcinka szacunki wartości i zasady ulegają zmianie. Tak więc metody Monte Carlo są przyrostowe w sensie odcinka po odcinku, a nie krok po kroku. Termin „Monte Carlo" jest często używany szerzej dla każdej metody szacowania, której działanie obejmuje istotny składnik losowy. Tutaj jest używany specjalnie dla metod opartych na uśrednianiu pełnych zwrotów. Ponownie rozważymy metody Monte Carlo w Części 13. Istnieją inne metody stosowane do uczenia się przez wzmacnianie, z których najważniejsza to Qlearning (Watkins 1989), wariant podejścia opartego na różnicach czasowych. W Q-learningu Q jest funkcją par stan-działanie względem wyuczonych wartości. Dla wszystkich stanów i działań:

Q: (stan x akcja) → wartość

Dla jednoetapowego Q-learningu:

$Q(s_t, a_t) \leftarrow (1 - c) * Q(s_t, a_t) + c * [r_t + 1 + \gamma * \max Q(s_{t+1}, a) - Q(s_t, a_t)]$

a gdzie zarówno c, jak i γ są ≤ 1, a r_{t+1} jest nagrodą przy s_{t+1}. Możemy wizualizować podejście Q na rysunku 24b i porównać je z rysunkiem 24a, gdzie węzeł początkowy to sytuacja-akcja . Ta reguła tworzenia kopii zapasowych aktualizuje pary stan-akcja, tak że górny stan na rysunku 24b, główny element kopii zapasowej, jest węzłem akcji połączonym ze stanem, który go utworzył. W Q-learningu kopia zapasowa pochodzi z węzłów akcji, maksymalizując wszystkie możliwe akcje z następnych stanów, wraz z ich nagrodami. W pełnym rekurencyjnym Q-learningu dolne węzły drzewa kopii zapasowych to wszystkie węzły końcowe, do których można dotrzeć za pomocą sekwencji działań rozpoczynających się od węzła głównego, wraz z nagrodami wynikającymi z tych działań następczych. Q-learning on-line, wykraczający poza możliwe działania, nie wymaga budowania pełnego modelu świata. Q-learning może być również przeprowadzany w trybie offline. Jak widać, Q-learning jest rodzajem podejścia opartego na różnicach czasowych. Dalsze szczegóły można znaleźć w Watkins oraz Sutton i Barto . Obecnie istnieje wiele istotnych problemów rozwiązanych dzięki uczeniu się ze wzmocnieniem, w tym Backgammon. Sutton i Barto również analizują Samuela program sprawdzający, sekcja 4.3, z punktu widzenia uczenia się przez wzmocnienie. Omawiają również podejście do uczenia się ze wzmocnieniem w akrobacie, przydzielanie wind, dynamiczną alokację kanałów, harmonogramowanie warsztatów pracy i inne klasyczne obszary problemowe

Sztuczna inteligencja : Uczenie maszynowe : Konekcjonizm

(XI / XVI)

ĆWICZENIA

1. Stwórz neuron McCullocha-Pittsa, który może obliczyć implikowaną funkcję logiczną, ⇒.

2. Zbuduj sieć perceptronów w LISP i uruchom ją na przykładzie klasyfikacji z sekcji 11.2.2.

a. Wygeneruj inny zestaw danych podobny do tego z tabeli 3 i uruchom na nim swój klasyfikator.

b. Weź wyniki działania klasyfikatora i użyj wag do określenia specyfikacji linii oddzielającej zbiory.

3. Zbuduj sieć propagacji wstecznej w LISP lub C ++ i użyj jej do rozwiązania problemu z wyłącznością lub z sekcji 11.3.3. Rozwiąż problem z wyłącznością lub problemem za pomocą innej architektury wstecznej propagacji, być może z dwoma ukrytymi węzłami i bez węzłów odchylenia. Porównaj prędkości konwergencji przy użyciu różnych architektur.

4. Napisz sieć Kohonena w LISP lub C ++ i użyj jej do sklasyfikowania danych z tabeli 3. Porównaj swoje wyniki z wynikami z sekcji 11.2.2 i 11.4.2.

5. Napisz sieć kontrpropagacji, aby rozwiązać problem na wyłączność. Porównaj swoje wyniki z wynikami uzyskanymi w sieci propagacji wstecznej w sekcji 11.3.3. Użyj swojej sieci kontrpropagacji, aby rozróżnić klasy z tabeli 3.

6. Użyj siatki wstecznej, aby rozpoznać dziesięć (narysowanych ręcznie) cyfr. Jednym podejściem byłoby zbudowanie tablicy punktów 4 x 6. Kiedy cyfra zostanie narysowana na tej siatce, pokryje ona niektóre elementy, nadając im wartość 1, a pomijając inne, wartość 0. Ten 24-elementowy wektor byłby wartością wejściową dla twojej sieci. Zbudowałbyś własne wektory szkoleniowe. Wykonaj to samo zadanie z siatką przeciwpropagacyjną; porównaj swoje wyniki.

7. Wybierz inny wzorzec wejściowy niż ten, którego użyliśmy w sekcji 11.5.2. Aby rozpoznać ten wzorzec, użyj nienadzorowanego algorytmu uczenia hebbijskiego.

8. W sekcji 11.5.4 wykorzystano algorytm asocjatora liniowego do stworzenia asocjacji dwóch par wektorów. Wybierz trzy (nowe) skojarzenia par wektorów i rozwiąż to samo zadanie. Sprawdź, czy Twój asocjator liniowy jest interpolacyjny; to znaczy, czy może kojarzyć bliskie chybienia wzorców? Ustaw asocjator liniowy na autoasocjacyjny.

9. Rozważmy dwukierunkową pamięć asocjacyjną (BAM) z Rozdziału 11.6.3. Zmień pary asocjacji podane w naszym przykładzie i utwórz macierz wag dla asocjacji. Wybierz nowe wektory i przetestuj swojego asocjatora BAM.

10. Opisz różnice między pamięcią BAM a asocjatorem liniowym. Co to jest przesłuch i jak można mu zapobiec?

11. Napisz sieć Hopfielda, aby rozwiązać problem podróżujących sprzedawców w dziesięciu miastach.

UCZENIE MASZYNOWE: KONEKCJONIZM

11.0 Wprowadzenie

W Części 10 położyliśmy nacisk na podejście do uczenia się oparte na symbolach. Centralnym aspektem tej hipotezy jest użycie symboli w odniesieniu do obiektów i relacji w dziedzinie. W niniejszym rozdziale przedstawiamy podejścia do uczenia się inspirowane neurologią lub biologią. Modele inspirowane neuronami, czasami nazywane równoległym przetwarzaniem rozproszonym (PDP) lub systemami koneksjonistycznymi, pomniejszają nacisk na jawne użycie symboli w rozwiązywaniu problemów. Zamiast tego utrzymują, że inteligencja powstaje w systemach prostych, oddziałujących ze sobą elementów (biologicznych lub sztucznych neuronów) w wyniku procesu uczenia się lub adaptacji, w wyniku którego dostosowywane są połączenia między komponentami. Przetwarzanie w tych systemach jest rozproszone w zbiorach lub warstwach neuronów. Rozwiązywanie problemów jest równoległe w tym sensie, że wszystkie neurony w kolekcji lub warstwie przetwarzają swoje dane wejściowe jednocześnie i niezależnie. Systemy te mają również tendencję do łagodnego degradacji, ponieważ informacje i przetwarzanie są rozproszone w węzłach i warstwach sieci.

W modelach koneksjonistycznych istnieje jednak silny charakter reprezentacyjny zarówno przy tworzeniu parametrów wejściowych, jak i przy interpretacji wartości wyjściowych. Na przykład, aby zbudować sieć neuronową, projektant musi stworzyć schemat kodowania wzorców świata na wielkości numeryczne w sieci. Wybór schematu kodowania może odegrać kluczową rolę w ostatecznym sukcesie lub niepowodzeniu uczenia się sieci. W systemach łącznikowych przetwarzanie jest równoległe i rozproszone bez manipulacji symbolami jako symbolami. Wzorce w domenie są kodowane jako wektory numeryczne. Połączenia między komponentami są również reprezentowane przez wartości liczbowe. Wreszcie transformacja wzorców jest wynikiem operacji numerycznych, zwykle mnożenia macierzy. Te „wybory projektanta" dla architektury koneksjonistycznej stanowią indukcyjne nastawienie systemu.

Algorytmy i architektury, które implementują te techniki, są zwykle uczone lub uwarunkowane, a nie jawnie zaprogramowane. Rzeczywiście, jest to główna siła tego podejścia: odpowiednio zaprojektowana architektura sieci i algorytm uczenia się mogą często uchwycić niezmienności na świecie, nawet w postaci dziwnych ciągników i, bez bezpośredniego zaprogramowania ich do rozpoznawania. Sposób, w jaki to się dzieje, stanowi materiał tej części. Zadania, do których dobrze nadaje się podejście neuronowe / koneksjonistyczne, obejmują: klasyfikację, określenie kategorii lub grupowania, do których należy wartość wejściowa; rozpoznawanie wzorców, identyfikowanie struktury lub wzorców w danych; przywoływanie pamięci, w tym problem pamięci adresowalnej treści; przewidywanie, takie jak identyfikowanie choroby na podstawie objawów, przyczyn na podstawie skutków; optymalizacja, znalezienie „najlepszej" organizacji ograniczeń; oraz filtrowanie szumów, czyli oddzielanie sygnału od tła, z uwzględnieniem nieistotnych składników sygnału. Metody opisane w tym rozdziale najlepiej sprawdzają się w przypadku zadań, które mogą być trudne do sformułowania w modelach symbolicznych. Zwykle obejmuje to zadania, w których dziedzina problemu wymaga umiejętności opartych na percepcji lub nie ma jasno zdefiniowanej składni. W części 11.1 przedstawiamy modele uczenia się inspirowane neuronami z historycznego punktu widzenia. Przedstawiamy komponenty uczenia się sieci neuronowej, w tym neuron „mechaniczny", i opisujemy niektóre historycznie ważne wczesne prace, w tym neuron McCullocha-Pittsa. Ewolucja paradygmatów

szkolenia sieciowego w ciągu ostatnich 60 lat dostarcza ważnych informacji na temat obecnego stanu tej dyscypliny. W sekcji 11.2 kontynuujemy prezentację historyczną, wprowadzając uczenie się przez perceptron i regułę delta. Przedstawiamy przykład perceptronu używanego jako klasyfikator. W sekcji 11.3 przedstawiamy sieci neuronowe z ukrytymi warstwami oraz regułę uczenia się wstecznej propagacji. Te innowacje wprowadzono w sztucznych sieciach neuronowych, aby przezwyciężyć problemy, jakie miały wczesne systemy przy uogólnianiu punktów danych, których nie można było rozdzielić liniowo. Propagacja wsteczna to algorytm przypisywania „winy" za nieprawidłowe odpowiedzi do węzłów systemu wielowarstwowego z ciągłym progowaniem. W sekcji 11.4 przedstawiamy modele konkurencyjnego uczenia się opracowane przez Kohonena i Hecht-Nielsen. W tych modelach wektory wagi sieci są używane do reprezentowania wzorców, a nie siły połączenia. Algorytm uczenia się typu zwycięzca bierze wszystko wybiera węzeł, którego wzór wag jest najbardziej podobny do wektora wejściowego i dostosowuje go, aby bardziej przypominał wektor wejściowy. Nie jest to nadzorowane, ponieważ wygrywanie polega po prostu na zidentyfikowaniu węzła, którego aktualny wektor wagi najbardziej przypomina wektor wejściowy. Połączenie warstw Kohonena z Grossberga w jednej sieci oferuje interesujący model uczenia się w odpowiedzi na bodziec, zwany uczeniem się z przeciwną propagacją. W części 11.5 przedstawiamy model Hebba uczenia się przez wzmacnianie. Hebb stwierdził, że za każdym razem, gdy jeden neuron przyczynia się do odpalenia innego neuronu, tzw. zwiększa się siła ścieżki między neuronami. Uczenie się Hebba jest modelowane przez prosty algorytm dostosowywania wag połączeń. Prezentujemy zarówno nienadzorowane, jak i nadzorowane wersje nauki języka Hebba. Przedstawiamy również stowarzyszenie liniowe, Hebba model oparty na odzyskiwaniu wzorców z pamięci. Część 11.6 wprowadza bardzo ważną rodzinę sieci zwanych sieciami atraktorowymi. Sieci te wykorzystują połączenia sprzężenia zwrotnego w celu wielokrotnego przełączania sygnału w sieci. Za wyjście sieci uważa się stan sieci po osiągnięciu równowagi.

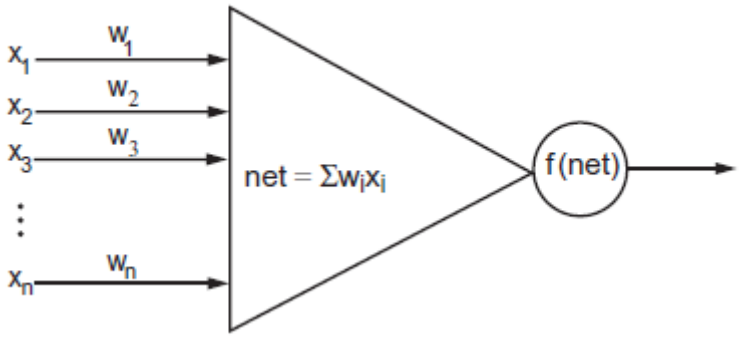
Wagi sieci są tak skonstruowane, że tworzony jest zestaw atraktorów. Wzorce wejściowe w basenie atraktora osiągają równowagę w tym atraktorze. Dlatego atraktory mogą być używane do przechowywania wzorów w pamięci. Mając wzorzec wejściowy, pobieramy albo najbliższy zapisany wzorzec w sieci, albo wzorzec powiązany z najbliższym przechowywanym wzorcem. Pierwszy typ pamięci nazywany jest autoasocjatywnym, drugi typ heteroasocjacyjny. John Hopfield, fizyk teoretyczny, zdefiniował klasę sieci atraktorów, których zbieżność może być reprezentowana przez minimalizację energii. Sieci Hopfielda mogą służy do rozwiązywania problemów związanych ze spełnieniem ograniczeń, takich jak problem podróżującego sprzedawcy, poprzez odwzorowanie funkcji optymalizacji na funkcję energii. W Części 12 przedstawiamy ewolucyjne modele uczenia się, w tym algorytmy genetyczne i sztuczne życie. Część 13 przedstawia dynamiczne i stochastyczne modele uczenia się. Omówimy kwestie reprezentacyjne i stronniczość w uczeniu się.

11.1 Podstawy sieci koekcjonistycznych

11.1.1 Wczesna historia

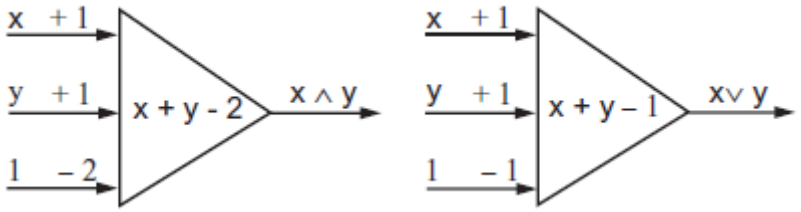
Architektury koneksjonistyczne są często postrzegane jako niedawny rozwój, jednak możemy prześledzić ich początki do wczesnych prac w dziedzinie informatyki, psychologii i filozofii. Na przykład John von Neumann był zafascynowany zarówno automatami komórkowymi, jak i inspirowanymi neurologicznie podejściami do obliczeń. Na wczesne prace nad uczeniem się neuronów wpływ miały psychologiczne teorie uczenia się zwierząt, zwłaszcza teoria Hebba. W tej sekcji przedstawiamy podstawowe komponenty uczenia się sieci neuronowych i przedstawiamy ważne historycznie prace w tej dziedzinie. Podstawą sieci neuronowych jest sztuczny neuron, jak na rysunku 1. Sztuczny neuron składa się z: Sygnałów wejściowych, xi. Dane te mogą pochodzić ze środowiska lub z aktywacji innych

neuronów. Różne modele różnią się w dopuszczalnym zakresie wartości wejściowych; zazwyczaj wejścia są dyskretne, ze zbioru {0, 1} lub {-1, 1} lub liczb rzeczywistych.



Zestaw rzeczywistych odważników, w_i. Wagi opisują siłę połączenia. Poziom aktywacji $\Sigma w_i x_i$. Poziom aktywacji neuronu jest określony przez skumulowaną siłę jego sygnałów wejściowych, przy czym każdy sygnał wejściowy jest skalowany przez wagę połączenia w_i wzdłuż tej linii wejściowej. Poziom aktywacji oblicza się w ten sposób, biorąc sumę ważonych danych wejściowych, to znaczy $\Sigma w_i x_i$. Funkcja progowa, f. Ta funkcja oblicza końcowy lub wyjściowy stan neuronu poprzez określenie, jak daleko poziom aktywacji neuronu jest poniżej lub powyżej pewnej wartości progowej. Funkcja progowa ma na celu wytworzenie stanu włączenia / wyłączenia rzeczywistych neuronów. Oprócz tych właściwości poszczególnych neuronów, sieć neuronowa charakteryzuje się również właściwościami globalnymi, takimi jak: Topologia sieci. Topologia sieci to wzór połączeń między poszczególnymi neuronami. Ta topologia jest głównym źródłem obciążenia indukcyjnego sieci. Zastosowany algorytm uczenia się. W tym rozdziale przedstawiono szereg algorytmów uczenia się. Schemat kodowania. Obejmuje to interpretację danych przesyłanych do sieci i wyniki ich przetwarzania. Najwcześniejszym przykładem obliczeń neuronowych jest neuron McCullocha -Pittsa. Sygnały wejściowe do neuronu McCullocha - Pittsa są pobudzające (+1) lub hamujące (-1). Funkcja aktywacji mnoży każde wejście przez odpowiadającą mu wagę i sumuje wyniki; jeśli suma jest większa lub równa zero, neuron zwraca 1, w przeciwnym razie −1. McCulloch i Pitts pokazali, w jaki sposób te neurony można skonstruować, aby obliczały dowolną funkcję logiczną, wykazując, że systemy tych neuronów zapewniają kompletny model obliczeniowy. Rysunek 2 przedstawia neurony McCullocha-Pittsa do obliczania funkcji logicznych. Neuron i ma trzy wejścia: x i y to wartości do połączenia; trzecia, czasami nazywana odchyleniem, ma stałą wartość +1. Dane wejściowe i odchylenie mają odpowiednio wagi +1, +1 i −2. Zatem dla dowolnych wartości x i y neuron oblicza wartość x + y - 2: jeśli ta wartość jest mniejsza niż 0, zwraca −1, w przeciwnym razie a 1. Tabela 11.1 przedstawia neuron obliczający x ^ y. W podobny sposób, ważona suma danych wejściowych dla neuronu lub, patrz rysunek 11.2, jest większa lub równa 0, chyba że oba x i y są równe -1.

x	y	x + y – 2	Output
1	1	0	1
1	0	–1	–1
0	1	–1	–1
0	0	–2	–1



Chociaż McCulloch i Pitts wykazali moc obliczeń neuronowych, zainteresowanie tym podejściem zaczęło rozkwitać dopiero wraz z rozwojem praktycznych algorytmów uczenia się. Modele wczesnego uczenia się w dużym stopniu opierały się na pracy psychologa D. O. Hebba, który spekulował, że uczenie się zachodzi w mózgach poprzez modyfikację synaps. Hebb wysunął teorię, że powtarzające się odpalenia synapsy zwiększają jej czułość i prawdopodobieństwo jej odpalenia w przyszłości. Jeśli dany bodziec wielokrotnie powodował aktywność w grupie komórek, komórki te są silnie powiązane. W przyszłości podobne bodźce będą miały tendencję do pobudzania tych samych ścieżek neuronowych, co spowoduje rozpoznanie bodźca. Model uczenia się Hebba działał wyłącznie na wzmocnieniu używanych ścieżek i zignorowaniu zahamowania, kary za błąd lub wyczerpania. Współcześni psychologowie próbowali wdrożyć model Hebba, ale nie uzyskali ogólnych wyników bez dodania mechanizmu hamującego (Rochester et al. 1988, Quinlan 1991). Rozważymy Hebbowski model uczenia się wsekcji 11.5. W następnej sekcji rozszerzamy model neuronowy McCullocha -Pittsa, dodając warstwy połączonych mechanizmów neuronowych i algorytmy ich interakcji. Pierwsza wersja nazywała się perceptronem.

11.2 Nauka perceptronu

11.2.1 Algorytm uczenia perceptronu

Frank Rosenblatt opracował algorytm uczenia się dla typu sieci jednowarstwowej zwanej perceptronem. W propagacji sygnału perceptron był podobny do neuronu McCullocha-Pittsa, co zobaczymy w sekcji 11.2.2. Wartości wejściowe i poziomy aktywacji perceptronu wynoszą -1 lub 1; wagi mają wartość rzeczywistą. Poziom aktywacji perceptronu podaje się poprzez zsumowanie ważonych wartości wejściowych, $\Sigma w_i x_i$. Perceptrony wykorzystują prostą funkcję sztywnego ograniczania progu, gdzie aktywacja powyżej progu daje wartość wyjściową 1, aw innym przypadku -1. Biorąc pod uwagę wartości wejściowe x_i, wagi w_i i próg t, perceptron oblicza swoją wartość wyjściową jako:

$$\begin{aligned} 1 \quad & \text{if } \Sigma x_i w_i \geq t \\ -1 \quad & \text{if } \Sigma x_i w_i < t \end{aligned}$$

Perceptron wykorzystuje prostą formę nadzorowanego uczenia się. Po próbie rozwiązania problemu nauczyciel podaje poprawny wynik. Następnie perceptron zmienia swoje wagi, aby zmniejszyć błąd. Stosowana jest następująca reguła. Niech c będzie stałą, której rozmiar określa szybkość uczenia się, a d będzie pożądaną wartością wyjściową. Korekta wagi na i-tym składniku wektora wejściowego, Δw_i, jest wyrażona wzorem:

$$\Delta w_i = c(d - \text{sign}(\Sigma x_i w_i))\, x_i$$

sign ($\Sigma w_i x_i$) to wartość wyjściowa perceptronu. To jest +1 lub −1. Różnica między wymaganą mocą a rzeczywistymi wartościami wyjściowymi będzie zatem wynosić 0, 2 lub −2. Dlatego dla każdego składnika wektora wejściowego: Jeśli żądane wartości wyjściowe i rzeczywiste wartości wyjściowe są

równe, nie rób nic. Jeśli rzeczywista wartość wyjściowa wynosi −1, a powinna wynosić 1, zwiększ wagi w i-tym wierszu o $2cx_i$. Jeśli rzeczywista wartość wyjściowa wynosi 1 i powinna wynosić −1, zmniejsz wagi w i-tym wierszu o $2cx_i$. Ta procedura skutkuje wytworzeniem zestawu wag, które mają na celu zminimalizowanie średniego błędu w całym zbiorze uczącym. Jeśli istnieje zestaw wag, które dają prawidłowe wyjście dla każdego członka zbioru uczącego, procedura uczenia się perceptronu nauczy się go. Początkowo perceptrony były witane z entuzjazmem. Jednak Nils Nilsson i inni przeanalizowali ograniczenia modelu perceptronowego. Wykazali, że perceptrony nie mogą rozwiązać pewnej trudnej klasy problemów, a mianowicie problemów, w których punkty danych nie są rozłączne liniowo. Chociaż w tamtym czasie przewidywano różne ulepszenia modelu perceptronu, w tym perceptrony wielowarstwowe, Marvin Minsky i Seymour Papert w swojej książce Perceptrons (1969) argumentowali, że problemu liniowej rozdzielności nie da się przezwyciężyć żadną formą sieci perceptronów. Przykładem nieliniowo rozdzielnej klasyfikacji jest wyłączność-lub. Wyłącznie - lub może być reprezentowane przez tabelę prawdy:

x_1	x_2	Output
1	1	0
1	0	1
0	1	1
0	0	0

Rozważmy perceptron z dwoma wejściami, x_1, x_2, dwoma wagami, w_1, w_2 i progiem t. Aby nauczyć się tej funkcji, sieć musi znaleźć przypisanie wagi, które spełnia następujące nierówności, przedstawione graficznie na rysunku 3:

$w_1 * 1 + w_2 * 1 < t$, z linii 1 tabeli prawdy.

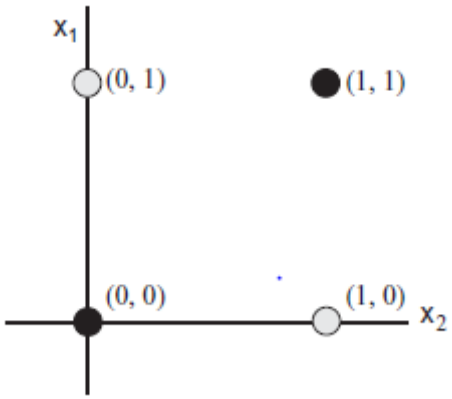
$w_1 * 1 + 0 > t$, z linii 2 tabeli prawdy.

$0 + w_2 * 1 > t$, z linii 3 tabeli prawdy.

$0 + 0 < t$ lub t musi być dodatnia, począwszy od ostatniego wiersza tabeli.

Ta seria równań na w_1, w_2 i t nie ma rozwiązania, co dowodzi, że perceptron, który rozwiązuje wyłączność lub jest niemożliwy. Chociaż ostatecznie powstałyby sieci wielowarstwowe, które mogłyby rozwiązać problem lub wyłączność, algorytm uczenia się perceptronu działał tylko w sieciach jednowarstwowych. Tym, co sprawia, że perceptron jest wykluczony lub niemożliwy, jest to, że dwie klasy, które należy rozróżnić, nie dają się rozdzielić liniowo. Można to zobaczyć na rysunku 3. Niemożliwe jest narysowanie prostej w dwóch wymiarach oddzielającej punkty danych {(0,0), (1,1)} od {(0,1), (1,0)}. Możemy myśleć o zbiorze wartości danych dla sieci jako o definiowaniu przestrzeni. Każdy parametr danych wejściowych odpowiada jednemu wymiarowi, przy czym każda wartość wejściowa określa punkt w przestrzeni. W przykładzie na wyłączność lub cztery wartości wejściowe, indeksowane przez współrzędne x_1, x_2 , tworzą punkty danych z rysunku 3. Problem poznania binarnej klasyfikacji instancji szkoleniowych sprowadza się do rozdzielenia tych punktów na dwie grupy. W przypadku przestrzeni o n wymiarach klasyfikację można rozdzielić liniowo, jeśli jej klasy mogą być oddzielone n-1 wymiarową hiperpłaszczyzną. (W dwóch wymiarach n-wymiarowa hiperpłaszczyzna jest linią; w trzech wymiarach jest to płaszczyzna itp.). W wyniku ograniczenia liniowej rozdzielności badania przesunęły się w kierunku pracy w architekturach opartych na symbolach, spowalniając postęp w

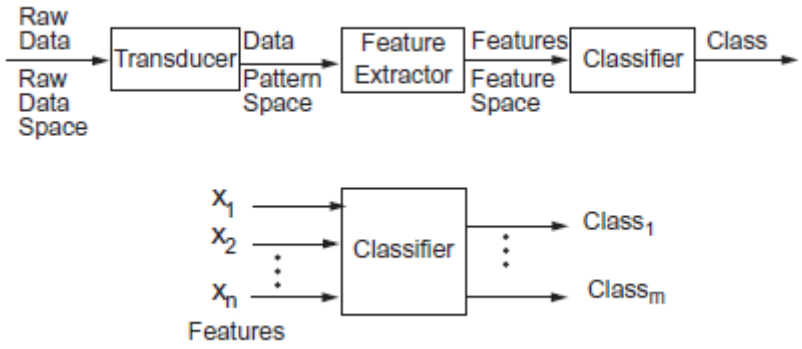
metodologii koneksjonizmu. Późniejsza praca w latach 80. i 90. pokazała jednak, że problemy te są możliwe do rozwiązania.



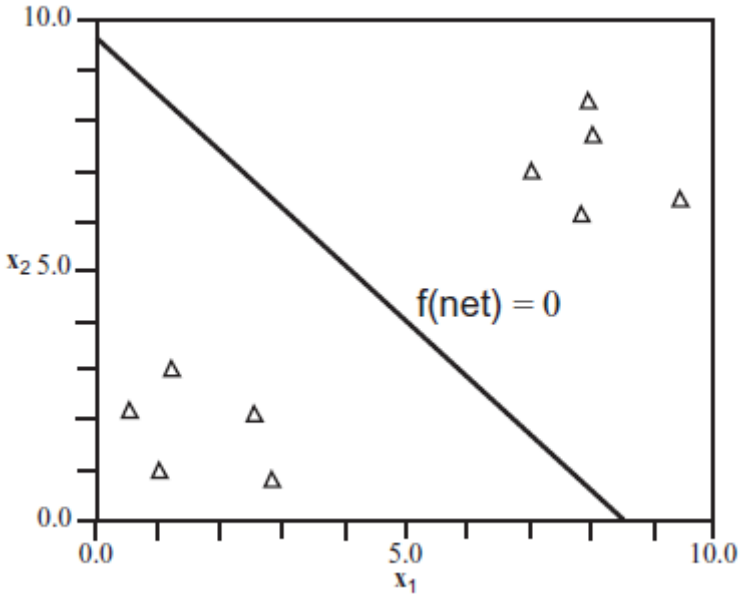
W sekcji 11.3 omówimy propagację wsteczną, rozszerzenie uczenia się przez perceptron, które działa w sieciach wielowarstwowych. Przed zbadaniem wstecznej propagacji podajemy przykład sieci perceptronowej, która wykonuje klasyfikacje. Sekcja 11.2 kończy się zdefiniowaniem uogólnionej reguły delta, uogólnienia algorytmu uczenia się przez perceptron, który jest używany w wielu architekturach sieci neuronowych, w tym w propagacji wstecznej. 1

1.2.2 Przykład: wykorzystanie sieci perceptronów do klasyfikacji

Rysunek 4 przedstawia przegląd problemu klasyfikacji.



Surowe dane z przestrzeni możliwych punktów są wybierane i przenoszone do nowej przestrzeni danych / wzorców. W tym nowym obszarze wzoru elementy są identyfikowane, a na koniec jednostka, którą te elementy reprezentują, jest klasyfikowana. Przykładem mogą być fale dźwiękowe nagrane na cyfrowym urządzeniu rejestrującym. Stamtąd sygnały akustyczne są tłumaczone na zestaw parametrów amplitudy i częstotliwości. Wreszcie system klasyfikatora może rozpoznać te wzorce cech jako dźwięczną mowę określonej osoby. Innym przykładem jest przechwytywanie informacji przez sprzęt medyczny, taki jak defibrylatory serca. Cechy znalezione w tej przestrzeni wzorców byłyby następnie wykorzystywane do klasyfikowania zestawów objawów do różnych kategorii chorób. W naszym przykładzie klasyfikacji przetwornik i ekstraktor cech z rysunku 4 tłumaczy informacje o problemie na parametry dwuwymiarowej przestrzeni kartezjańskiej. Rysunek 5 przedstawia analizę dwu-cechową perceptronu informacji z tabeli 3.



Pierwsze dwie kolumny tabeli przedstawiają punkty danych, na których została przeszkolona sieć. Trzecia kolumna reprezentuje klasyfikację +1 lub -1, używaną jako informacja zwrotna w uczeniu sieci. Rysunek 5 to wykres danych uczących problemu, pokazujący liniową separację klas danych utworzonych, gdy wytrenowana sieć została uruchomiona w każdym punkcie danych. Omówimy najpierw ogólną teorię klasyfikacji. Każda grupa danych, którą identyfikuje klasyfikator, jest reprezentowana przez region w przestrzeni wielowymiarowej. Każda klasa R_i ma funkcję dyskryminacyjną g_i mierzącą przynależność do tego regionu. W obszarze R_i i-ta funkcja dyskryminacyjna ma największą wartość:

$g_i(x) > g_j(x)$ dla wszystkich j, $1 < j < n$.

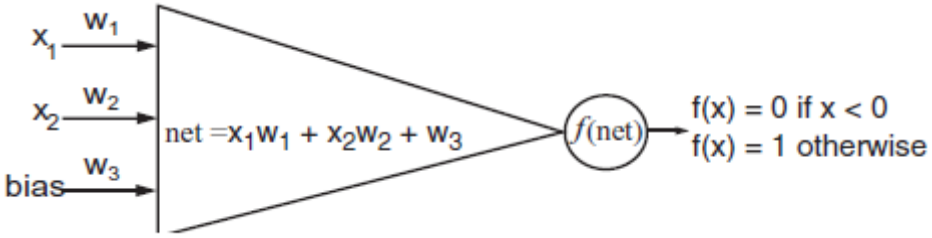
W prostym przykładzie z tabeli 3, dwa parametry wejściowe tworzą dwa oczywiste obszary lub klasy w przestrzeni, jeden reprezentowany przez 1, a drugi przez -1.

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	–1
2.5	2.1	1
8.0	7.7	–1
0.5	2.2	1
7.9	8.4	–1
7.0	7.0	–1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	–1

Ważnym szczególnym przypadkiem funkcji dyskryminacyjnych jest taki, który ocenia przynależność do klasy na podstawie odległości od jakiegoś centralnego punktu w regionie. Klasyfikacja oparta na tej funkcji dyskryminacyjnej nazywana jest klasyfikacją minimalnej odległości. Prosty argument pokazuje, że jeśli klasy można rozdzielić liniowo, istnieje klasyfikacja minimalnej odległości. Jeśli regiony R_i i R_j sąsiadują ze sobą, podobnie jak dwa regiony na rysunku 11.5, istnieje region graniczny, w którym funkcje dyskryminacyjne są równe:

$g_i(x) = g_j(x)$ lub $g_i(x) - g_j(x) = 0$.

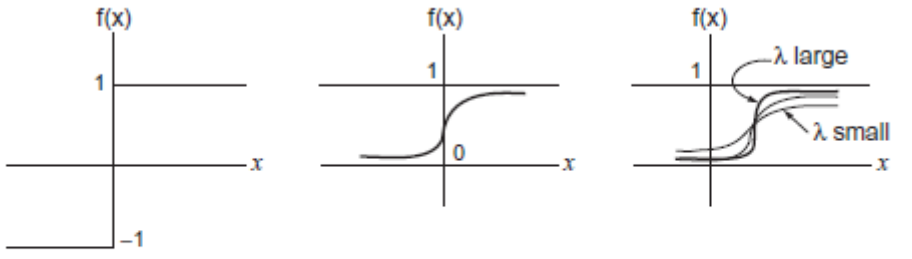
Jeśli klasy można rozdzielić liniowo, jak na rysunku 11.5, funkcja dyskryminacyjna oddzielająca regiony jest linią prostą lub $g_i(x) - g_j x)$ jest liniowa. Ponieważ prosta jest umiejscowieniem punktów w równej odległości od dwóch stałych punktów, funkcje dyskryminacyjne, $g_i(x)$ i $g_j(x)$, są funkcjami odległości minimalnej, mierzonymi od kartezjańskiego środka każdego z regionów. Perceptron z rysunku 6 obliczy tę funkcję liniową.



Potrzebujemy dwóch parametrów wejściowych i będziemy mieć odchylenie o stałej wartości 1. Perceptron oblicza:

$f(net) = f(w_1 * x_1 + w_2 * x_2 + w_3 * 1)$, gdzie $f(x)$ jest znakiem x.

Gdy f(x) wynosi +1, x jest interpretowane jako należące do jednej klasy, a gdy wynosi −1, x należy do drugiej. To progowanie do +1 lub -1 nazywane jest liniowym progowaniem bipolarnym.



a. A hard limiting and bipolar linear threshold.

b. A sigmoidal and unipolar threshold.

c. The sigmoidal, biased and squashed. As λ gets larger the sigmoid approximates a linear threshold.

Odchylenie służy do przesunięcia funkcji progowania na osi poziomej. Zakres tej zmiany można poznać, dostosowując ciężar w_3 podczas treningu. Teraz używamy punktów danych z tabeli 3 do trenowania perceptronu z rysunku 6. Załóżmy losową inicjalizację wag do [.75, .5, −.6] i użyj algorytmu uczenia perceptronowego z rozdziału 11.2.1. Indeksy górne, np. 1 w $f(net)^1$, reprezentują bieżący numer iteracji algorytmu. Zaczynamy od pierwszego punktu danych w tabeli:

$f(net)^1 = f(.75 * 1 + .5 * 1 - .6 * 1) = f(.65) = 1$

Ponieważ $f(net)^1 = 1$, poprawna wartość wyjściowa, nie dostosowujemy ciężary. Zatem $W^2 = W^1$. Dla naszego drugiego punktu danych: $f(net)^2 = f(0,75 * 9,4 + .5 * 6,4 - .6 * 1) = f(9,65) = 1$

Tym razem nasz wynik powinien wynosić -1, więc musimy zastosować regułę uczenia się opisaną w sekcji 11.1.1:

$W^t = W^{t-1} + c\,(d^{t-1} - znak\,(W^{t-1} * X^{t-1}))\,X^{t-1}$

gdzie c jest stałą uczenia się, X i W są wektorami wejściowymi i wektorami wagi, a t iteracją sieci. d^{t-1} jest pożądanym wynikiem w czasie t - 1, lub w naszej sytuacji, w t = 2. Produkcja netto w t = 2 wynosi 1. Zatem różnica między pożądaną i rzeczywistą produkcją netto, d^2 - sign ($W^2 * X^2$) wynosi -2. W rzeczywistości, w twardym, ograniczonym bipolarnym perceptronie, przyrost uczenia się zawsze będzie wynosił + 2c lub −2c razy wektor uczący. Niech stała uczenia się będzie małą dodatnią liczbą rzeczywistą 0,2. Aktualizujemy wektor wagi:

$$W^3 = W^2 + 0.2(-1-1)X^2 = \begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix} - 0.4 \begin{bmatrix} 9.4 \\ 6.4 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix}$$

Rozważmy teraz trzeci punkt danych z nowo dostosowanymi wagami:

$f(net)^3 = f(- 3,01 * 2,5 - 2,06 * 2,1 - 1,0 * 1) = f(-12,84) = -1$

Ponownie, wynik netto nie jest pożądanym wyjściem. Pokazujemy regulację W^4

$$W^4 = W^3 + 0.2(1-(-1))X^3 = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix} + 0.4 \begin{bmatrix} 2.5 \\ 2.1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -2.01 \\ -1.22 \\ -0.60 \end{bmatrix}$$

Po 10 iteracjach sieci perceptronowej uzyskuje się separację liniową z rysunku 5. Po wielokrotnym uczeniu zbioru danych, w sumie około 500 iteracji, wektor wagi jest zbieżny do [−1,3, −1,1, 10,9]. Interesuje nas linia oddzielająca te dwie klasy. W odniesieniu do funkcji dyskryminacyjnych g_i i g_j, linia jest zdefiniowana jako zbiór punktów, w których $g_i(x) = g_j(x)$ lub $g_i(x) - g_j(x) = 0$, czyli gdzie wynik netto równa się 0. Równanie produkcji netto jest podane w postaci wag. To jest:

wyjście = $w_1x_1 + w_2x_2 + w_3$.

W konsekwencji linia oddzielająca dwie klasy jest określona równaniem liniowym: $-1,3 * x_1 + -1,1 * x_2 + 10,9 = 0$.

11.2.3 Uogólniona reguła delta

Prostym sposobem uogólnienia sieci perceptronów jest zastąpienie jej twardej funkcji ograniczania progów innymi typami funkcji aktywacji. Na przykład funkcje ciągłej aktywacji oferują możliwość bardziej wyrafinowanych algorytmów uczenia się pozwalając na dokładniejszą szczegółowość pomiaru błędów. Rysunek 7 przedstawia wykres niektórych funkcji progowych: liniowej bipolarnej funkcji progowej, rysunek 7a, podobnej do tej używanej przez perceptron, oraz szeregu funkcji sigmoidalnych. Funkcje sygmoidalne są tak nazywane, ponieważ ich wykres jest krzywą w kształcie litery „S", jak na rysunku 7b. Powszechna sigmoidalna funkcja aktywacji, zwana logistyką funkcja jest określona równaniem: $f(net) = 1/(1 + e^{-\lambda * net})$, gdzie $net = \Sigma x_i w_i$ .Podobnie jak w przypadku wcześniej zdefiniowanych funkcji, xi jest wejściem w linii i, w_i jest wagą w linii i oraz λ „parametr zgniatania" używany do precyzyjnego dostrojenia krzywej sigmoidalnej. Gdy λ staje się duże, sigmoida zbliża się do

liniowej funkcji progowej powyżej {0,1}; zbliżając się do 1, zbliża się do prostej. Te wykresy progowe przedstawiają wartości wejściowe, poziom aktywacji neuronu, w funkcji skalowanej aktywacji lub wyjścia neuronu. Sigmoidalna funkcja aktywacji jest ciągła, co pozwala na dokładniejszy pomiar błędu. Podobnie jak w przypadku twardej ograniczającej funkcji progowej, sigmoidalna funkcja aktywacji odwzorowuje większość wartości w swojej domenie na regiony bliskie 0 lub 1. Istnieje jednak obszar szybkiego, ale ciągłego przejścia między 0 a 1. W pewnym sensie przybliża ona zachowanie progowe zapewniając ciągłą funkcję wyjściową. Użycie λ w wykładniku reguluje nachylenie esicy w obszarze przejściowym. Ważone odchylenie przesuwa próg wzdłuż osi X. Historyczne pojawienie się sieci z funkcjami ciągłej aktywacji zasugerowało nowe podejście do uczenia się z redukcją błędów. Reguła uczenia Widrow-Hoff jest niezależna od funkcji aktywacji, minimalizując kwadratowy błąd między pożądaną wartością wyjściową a aktywacją sieci, $net_i = WX_i$. Być może najważniejszą regułą uczenia się dla funkcji ciągłej aktywacji jest reguła delta. Intuicyjnie reguła delta opiera się na idei powierzchni błędu, jak pokazano na rysunku 8. Ta powierzchnia błędu przedstawia skumulowany błąd w zestawie danych jako funkcję wag sieci. Każda możliwa konfiguracja wagi sieci jest reprezentowana przez punkt na tej n-wymiarowej powierzchni błędu. Biorąc pod uwagę konfigurację wagi, chcemy, aby nasz algorytm uczenia się znalazł kierunek na tej powierzchni, który najszybciej zmniejszy błąd. To podejście jest nazywane uczeniem się zstępującym gradientem, ponieważ gradient jest miarą nachylenia, jako funkcji kierunku, od punktu na powierzchni. Aby skorzystać z reguły delta, sieć musi używać funkcji aktywacji, która jest ciągła, a zatem zróżnicowana. Tę właściwość ma przedstawiony wzór logistyczny. Formuła uczenia się reguły delta dla dostosowania wagi na j-tym wejściu do i-tego węzła to:

$c\,(d_i - O_i)\,f\,'(net_i)\,x_j,$

gdzie c jest stałą kontrolującą szybkość uczenia się, d_i i O_i to pożądane i rzeczywiste wartości wyjściowe i-tego węzła. Pochodna funkcji aktywacji dla i-tego węzła to f', a x_j jest j-tym wejściem do węzła i. Pokażemy teraz wyprowadzenie tego wzoru. Średni kwadrat błędu sieci można znaleźć, sumując kwadrat błędu dla każdego węzła:

$$\text{Error} = (1/2)\sum_i (d_i - O_i)^2$$

gdzie d_i to żądana wartość dla każdego węzła wyjściowego, a O_i to rzeczywista wartość wyjściowa węzła. Podważamy każdy błąd tak, aby poszczególne błędy, niektóre prawdopodobnie ujemne, a inne dodatnie, nie znosiły się nawzajem. Rozważamy tutaj przypadek, w którym węzeł znajduje się w warstwie wyjściowej; opisujemy ogólny przypadek, gdy przedstawiamy sieci z ukrytymi warstwami w sekcji 11.3. Chcemy najpierw zmierzyć szybkość zmian błędu sieci w odniesieniu do wyjścia każdego węzła. W tym celu posługujemy się pojęciem pochodnej cząstkowej, które daje nam tempo zmian wielu zmiennych funkcji w odniesieniu do określonej zmiennej. Pochodna częściowa błędu całkowitego w odniesieniu do każdej jednostki wyjściowej i wynosi:

$$\frac{\partial \text{Error}}{\partial O_i} = \frac{\partial (1/2) * \Sigma (d_i - O_i)^2}{\partial O_i} = \frac{\partial (1/2) * (d_i - O_i)^2}{\partial O_i}$$

Drugie uproszczenie jest możliwe, ponieważ rozważamy węzeł w warstwie wyjściowej, gdzie jego błąd nie wpłynie na żaden inny węzeł. Biorąc pochodną tej wielkości, otrzymujemy:

$$\frac{\partial(1/2)*(d_i - O_i)^2}{\partial O_i} = -(d_i - O_i)$$

Chcemy tempa zmiany błędu sieci w funkcji zmiany wag w węźle i. Aby uzyskać zmianę określonej wagi, w_k, polegamy na zastosowaniu pochodnej cząstkowej, tym razem biorąc pochodną cząstkową błędu w każdym węźle w odniesieniu do wagi w_k w tym węźle. Rozwinięcie po prawej stronie znaku równości daje nam reguła łańcuchowa dla pochodnych cząstkowych:

$$\frac{\partial Error}{\partial w_k} = \frac{\partial Error}{\partial O_i} * \frac{\partial O_i}{\partial w_k}$$

To daje nam elementy potrzebne do rozwiązania równania. Korzystając z naszego wcześniejszego wyniku, otrzymujemy:

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i) * \frac{\partial O_i}{\partial w_k}$$

Kontynuujemy rozważając najbardziej właściwy czynnik, pochodną cząstkową rzeczywistego wyniku w i-tym węźle, biorąc pod uwagę każdą wagę w tym węźle. Wzór na wynik węzła i w funkcji jego wag jest następujący:

$$O_i = f(W_i X_i), \text{ where } W_i X_i = net_i.$$

Ponieważ f jest funkcją ciągłą, biorąc pochodną otrzymujemy:

$$\frac{\partial O_i}{\partial w_k} = x_k * f'(W_i X_i) = f'(net_i) * x_k$$

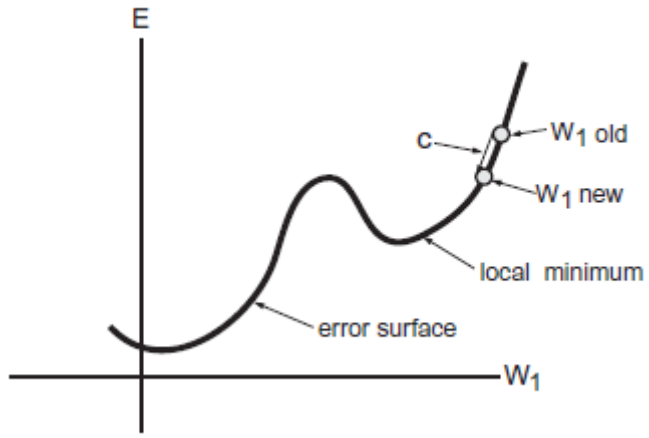
Podstawiając w poprzednim równaniu:

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i) f'(net_i) * x_k$$

Minimalizacja błędu wymaga, aby zmiany masy następowały w kierunku ujemnej składowej gradientu. W związku z tym:

$$\Delta w_k = -c\frac{\partial Error}{\partial w_k} = -c[-(d_i - O_i) * f'(net_i) * x_k] = c(d_i - O_i) f'(net_i) * x_k$$

Zauważamy, że reguła delta jest jak wspinaczka po wzgórzach, ponieważ na każdym kroku stara się zminimalizować lokalną miarę błędu, używając pochodnej do znalezienia nachylenia przestrzeni błędów w regionie lokalnym dla określonego punktu. To sprawia, że uczenie się delta jest podatne na problem rozróżnienia minimów lokalnych od globalnych w przestrzeni błędów. Stała uczenia się c wywiera istotny wpływ na działanie reguły delta, co ilustruje dalsza analiza rysunku 8.

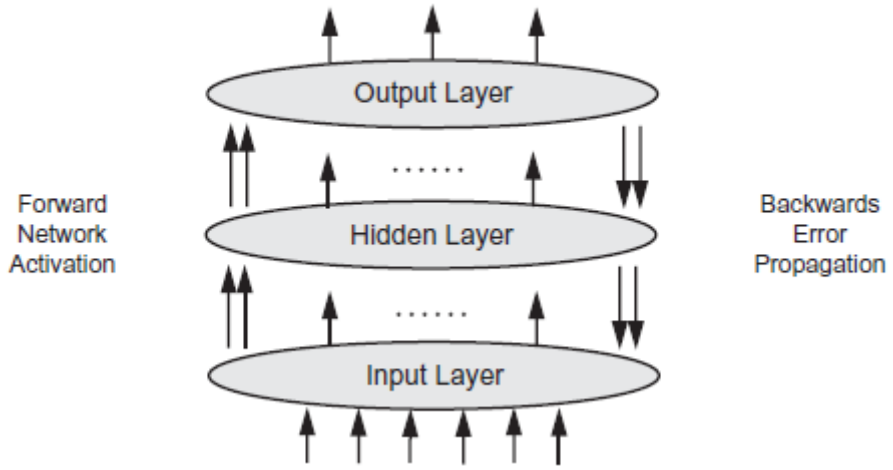


Wartość c określa, o ile wartości wagi zmieniają się w jednym odcinku uczenia się. Im większa wartość c, tym szybciej wagi zbliżają się do wartości optymalnej. Jeśli jednak c jest zbyt duże, algorytm może przekroczyć minimum lub oscylować wokół optymalnych wag. Mniejsze wartości c są mniej podatne na ten problem, ale nie pozwalają systemowi nauczyć się tak szybko. Optymalna wartość współczynnika uczenia się, czasami zwiększana przez współczynnik pędu, jest parametrem dostosowywanym do konkretnego zastosowania w drodze eksperymentu. Chociaż reguła delta sama w sobie nie pokonuje ograniczeń sieci jednowarstwowych, jej uogólniona postać ma kluczowe znaczenie dla funkcjonowania algorytmu wstecznej propagacji, algorytmu uczenia się w sieci wielowarstwowej. Algorytm ten został przedstawiony w następnej sekcji.

11.3 Uczenie się wstecznej propagacji

11.3.1 Wyprowadzenie algorytmu propagacji wstecznej

Jak widzieliśmy, jednowarstwowe sieci perceptronowe są ograniczone pod względem klasyfikacji, które mogą wykonywać. W sekcjach 11.3 i 11.4 pokazujemy, że dodanie wielu warstw może przezwyciężyć wiele z tych ograniczeń. Zauważamy, że sieci wielowarstwowe są obliczeniowo kompletne, to znaczy równoważne klasie maszyn Turinga. Wcześni badacze nie byli jednak w stanie zaprojektować algorytmu uczenia się do ich użytku. W tej sekcji przedstawiamy uogólnioną regułę delta, która oferuje jedno rozwiązanie tego problemu. Neurony w sieci wielowarstwowej, jak na rysunku 9, są połączone warstwami, przy czym jednostki w warstwie n przekazują swoje aktywacje tylko neuronom w warstwie n + 1.



Wielowarstwowe przetwarzanie sygnału oznacza, że błędy w głębi sieci mogą rozprzestrzeniać się i ewoluować w sposób złożony , nieprzewidziane drogi przez kolejne warstwy. Zatem analiza źródła błędu w warstwie wyjściowej jest złożona. Propagacja wsteczna zapewnia algorytm przypisywania winy i odpowiedniego dostosowywania wagi. Podejście przyjęte przez algorytm wstecznej propagacji polega na rozpoczęciu od warstwy wyjściowej i propagowaniu błędu wstecz przez warstwy ukryte. Kiedy przeanalizowaliśmy uczenie się za pomocą reguły delta, zauważyliśmy, że wszystkie informacje potrzebne do aktualizacji wag na neuronie były lokalne dla tego neuronu, z wyjątkiem wielkości błędu. W przypadku węzłów wyjściowych można to łatwo obliczyć jako różnicę między pożądanymi a rzeczywistymi wartościami wyjściowymi. W przypadku węzłów w warstwach ukrytych znacznie trudniej jest określić błąd, za który odpowiada węzeł. Funkcja aktywacji dla propagacji wstecznej jest zwykle funkcją logistyczną:

$$f(net) = 1/(1 + e^{-\lambda * net}), \text{ where } net = \Sigma x_i w_i.$$

Ta funkcja jest używana z czterech powodów: Po pierwsze, ma kształt sigmoidalny. Po drugie, jako funkcja ciągła ma wszędzie pochodną. Po trzecie, ponieważ wartość pochodnej jest największa tam, gdzie funkcja sigmoidalna zmienia się najszybciej, przypisanie największego błędu przypisuje się tym węzłom, których aktywacja była najmniej pewna. Wreszcie pochodną można łatwo obliczyć przez odejmowanie i mnożenie:

$$f'(net) = (1/(1 + e^{-\lambda * net})) = \lambda(f(net) * (1 - f(net))).$$

Trening wstecznej propagacji wykorzystuje uogólnioną regułę delta. Wykorzystuje to to samo podejście gradientowe, które przedstawiono w sekcji 11.2. W przypadku węzłów w warstwie ukrytej przyjrzymy się ich wkładowi w błąd w warstwie wyjściowej. Wzory na obliczenie korekty wagi wki na ścieżce od k-tego do i-tego węzła w treningu wstecznej propagacji są następujące:

1) $\Delta w_{ki} = -c(d_i - O_i) * O_i (1 - O_i) x_k$, for nodes on the output layer, and

2) $\Delta w_{ki} = -c * O_i (1 - O_i) \sum_j (-delta_j * w_{ij}) x_k$, for nodes on hidden layers.

W 2) j jest indeksem węzłów w następnej warstwie, do której rozchodzą się sygnały i i:

$$\text{delta}_j = -\frac{\partial \text{Error}}{\partial \text{net}_j} = (d_i - O_i) * O_i (1 - O_i).$$

Pokażemy teraz wyprowadzenie tych formuł. Najpierw wyprowadzamy 1), wzór na dostosowanie wagi węzłów w warstwie wyjściowej. Tak jak poprzednio, chcemy, aby szybkość zmiany błędu sieci była funkcją zmiany k-tej wagi, w_k, węzła i. Sytuację tę potraktowaliśmy w wyprowadzeniu reguły delta w sekcji 11.2.3 i wykazaliśmy, że:

$$\frac{\partial \text{Error}}{\partial w_k} = -((d_i - O_i) * f'(\text{net}_i) * x_k)$$

Ponieważ f, która może być dowolną funkcją, jest teraz funkcją aktywacji logistycznej, mamy:

$$f'(\text{net}) = f'(1/(1 + e^{-\lambda * \text{net}})) = f(\text{net}) * (1 - f(\text{net})).$$

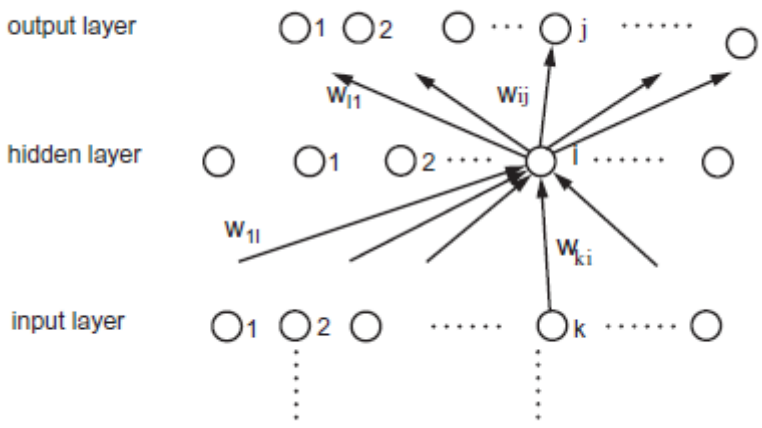
Przypomnij sobie, że $f(\text{net}_i)$ to po prostu O_i. Zastępując w poprzednim równaniu, otrzymujemy:

$$\frac{\partial \text{Error}}{\partial w_k} = -(d_i - O_i) * O_i * (1 - O_i) * x_k$$

Ponieważ minimalizacja błędu wymaga, aby zmiany wagi były w kierunku ujemnej składowej gradientu, mnożymy przez −c, aby uzyskać korektę wagi dla i-tego węzła warstwy wyjściowej:

$$\Delta w_k = c(d_i - O_i) * O_i * (1 - O_i) * x_k.$$

Następnie wyprowadzamy korektę wagi dla ukrytych węzłów. Ze względu na przejrzystość początkowo zakładamy jedną ukrytą warstwę. Bierzemy pojedynczy węzeł i w warstwie ukrytej i analizujemy jego udział w całkowitym błędzie sieci. Robimy to, rozważając początkowo udział węzła i w błędzie w węźle j warstwy wyjściowej. Następnie sumujemy te udziały we wszystkich węzłach w warstwie wyjściowej. Na koniec opiszemy udział k-tej wagi wejściowej w węźle i w błędzie sieci. Rysunek 10 ilustruje tę sytuację.



Najpierw przyjrzymy się częściowej pochodnej błędu sieci w odniesieniu do wyjścia węzła i w warstwie ukrytej. Uzyskujemy to, stosując regułę łańcucha:

$$\frac{\partial Error}{\partial O_i} = \frac{\partial Error}{\partial net_j} * \frac{\partial net_j}{\partial O_i}$$

Ujemnik pierwszego członu po prawej stronie, (δError) / (δnet$_j$), nazywa się delta$_j$, dlatego możemy przepisać równanie jako:

$$\frac{\partial Error}{\partial O_i} = -delta_j * \frac{\partial net_j}{\partial O_i}$$

Przypomnijmy, że aktywacja węzła j, net$_j$, na warstwie wyjściowej jest sumą iloczynu jego własnych wag i wartości wyjściowych pochodzących z węzłów na warstwie ukrytej:

$$net_j = \sum_i w_{ij} O_i$$

Ponieważ bierzemy pochodną cząstkową w odniesieniu do tylko jednego składnika sumy, a mianowicie połączenia między węzłem i a węzłem j, otrzymujemy:

$$\frac{\partial net_j}{\partial O_i} = w_{ij}$$

gdzie wij jest ciężarem połączenia z węzła i w warstwie ukrytej do węzła j w warstwie wyjściowej. Podstawiając ten wynik:

$$\frac{\partial Error}{\partial O_i} = -delta_j * w_{ij}$$

Teraz zsumujemy wszystkie połączenia węzła i z warstwą wyjściową:

$$\frac{\partial Error}{\partial O_i} = \sum_j -delta_j * w_{ij}$$

Daje nam to wrażliwość błędu sieciowego na dane wyjściowe węzła i w warstwie ukrytej. Następnie określamy wartość delta$_i$, wrażliwość błędu sieci na aktywację sieci w ukrytym węźle i. Daje to wrażliwość błędu sieciowego na przychodzące wagi węzła i. Używając ponownie reguły łańcucha:

$$-delta_i = \frac{\partial Error}{\partial net_i} = \frac{\partial Error}{\partial O_i} * \frac{\partial O_i}{\partial net_i}$$

Ponieważ używamy funkcji aktywacji logistycznej

$$\frac{\partial O_i}{\partial net_i} = O_i * (1 - O_i)$$

Teraz podstawiamy tę wartość w równaniu na delta$_i$, aby otrzymać:

$$-delta_i = O_i * (1 - O_i) * \sum_j -delta_j * w_{ij}$$

Wreszcie możemy ocenić wrażliwość błędu sieciowego w warstwie wyjściowej na przychodzące wagi w ukrytym węźle i. Sprawdzamy k-tą wagę w węźle i, w_k. Zgodnie z zasadą łańcucha:

$$\frac{\partial Error}{\partial w_{ki}} = \frac{\partial Error}{\partial net_i} * \frac{\partial net_j}{\partial w_{ki}} = -delta_i * \frac{\partial net_j}{\partial w_{ki}} = -delta_i * x_k$$

gdzie xk jest k-tym wejściem do węzła i. Podstawiamy do równania wartość $-delta_i$:

$$\frac{\partial Error}{\partial w_{ki}} = O_i(1 - O_i)\sum_j (-delta_j * w_{ij})x_k$$

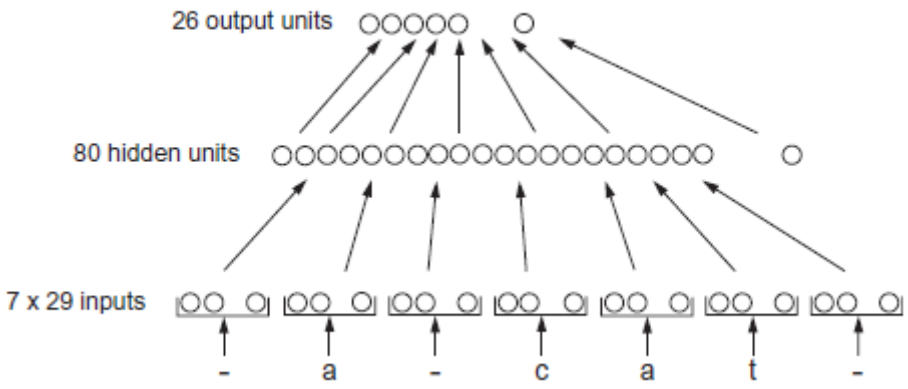
Ponieważ minimalizacja błędu wymaga, aby zmiany wagi były w kierunku ujemnej składowej gradientu, korektę wagi dla k-tego ciężaru i otrzymujemy przez pomnożenie przez ujemną wartość stałej uczenia:

$$\Delta w_{ki} = -c\frac{\partial Error}{\partial w_{ki}} = c * O_i(1 - O_i)\sum_j (delta_j * w_{ij})x_k \ .$$

W przypadku sieci z więcej niż jedną warstwą ukrytą ta sama procedura jest stosowana rekurencyjnie w celu propagacji błędu z warstwy ukrytej n do warstwy ukrytej n - 1. Chociaż zapewnia rozwiązanie problemu uczenia się w sieciach wielowarstwowych, propagacja wsteczna nie jest pozbawiona własnej trudności. Podobnie jak w przypadku wspinaczki górskiej, może zbiegać się do lokalnych minimów, jak na rysunku 11.8. Wreszcie, obliczenia wstecznej propagacji mogą być kosztowne, zwłaszcza gdy sieć powoli się zbiega.

11.3.2 Wsteczna propagacja Przykład 1: NETtalk

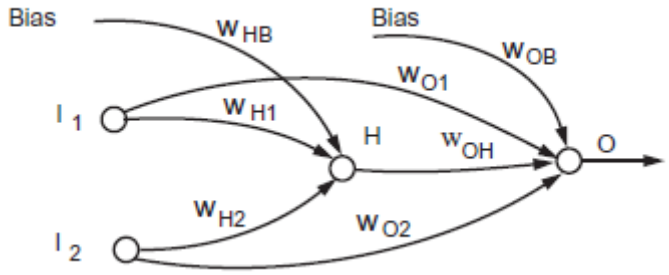
NETtalk jest interesującym przykładem rozwiązania trudnego problemu uczenia się poprzez sieć neuronową. NETtalk nauczył się wymawiać tekst w języku angielskim. Może to być trudne zadanie w przypadku podejścia opartego na jednoznacznych symbolach, na przykład systemu opartego na regułach, ponieważ wymowa angielska jest wysoce nieregularna. NETtalk nauczył się czytać ciąg tekstu i zwracać fonem oraz skojarzony z nim akcent dla każdej litery w ciągu. Fonem jest podstawową jednostką dźwiękową w języku; stres to względna głośność tego dźwięku. Ponieważ wymowa pojedynczej litery zależy od jej kontekstu i liter wokół niej, NETtalk otrzymał siedmioznakowe okno. Gdy tekst przechodzi przez to okno, NETtalk zwraca parę fonem / akcent dla każdej litery. Rysunek 11 przedstawia architekturę NETtalk.



Sieć składa się z trzech warstw jednostek. Jednostki wprowadzania odpowiadają siedmioznakowemu oknu w tekście. Każda pozycja w oknie jest reprezentowana przez 29 jednostek wejściowych, po jednej na każdą literę alfabetu i 3 na znaki interpunkcyjne i spacje. Litera na każdej pozycji aktywuje odpowiednią jednostkę. Jednostki wyjściowe kodują fonemy wykorzystując 21 różnych cech artykulacji człowieka. Pozostałe pięć jednostek zakodowało akcent i granice sylab. NETtalk ma 80 ukrytych jednostek, 26 wartości wyjściowych i 18 629 połączeń. NETtalk jest trenowany przez nadanie mu okna z siedmioma znakami i umożliwienie mu próby wymówienia środkowego znaku. Porównując swoją próbę wymowy z poprawną wymową, dostosowuje swoje wagi za pomocą wstecznej propagacji. Ten przykład ilustruje szereg interesujących właściwości sieci neuronowych, z których wiele odzwierciedla naturę uczenia się człowieka. Na przykład uczenie się, mierzone jako procent poprawnych odpowiedzi, na początku przebiega szybko i zwalnia wraz ze wzrostem odsetka poprawnych odpowiedzi. Podobnie jak w przypadku ludzi, im więcej słów nauczy się wymawiać sieć, tym lepiej będzie poprawnie wymawiać nowe słowa. Eksperymenty, w których niektóre z wag w pełni wyszkolona sieć została losowo zmieniona, co pokazało, że jest ona odporna na uszkodzenia, degradując się z wdziękiem wraz ze zmianą wagi. Badacze odkryli również, że ponowne uczenie się w uszkodzonej sieci było bardzo wydajne. Innym interesującym aspektem sieci wielowarstwowych jest rola warstw ukrytych. Każdy algorytm uczący się musi nauczyć się uogólnień, które mają zastosowanie do niewidocznych wystąpień w domenie problemu. Ukryte warstwy odgrywają ważną rolę w umożliwieniu sieci neuronowej uogólnienia. NETtalk, podobnie jak wiele sieci z propagacją wsteczną, ma mniej neuronów w warstwie ukrytej niż w warstwie wejściowej. Oznacza to, że ponieważ mniej węzłów w warstwie ukrytej jest używanych do kodowania informacji we wzorcach uczących, ma miejsce pewna forma abstrakcji. Krótsze kodowanie oznacza, że różne wzorce na warstwie wejściowej można odwzorować na identyczne wzorce w warstwie ukrytej. Ta redukcja jest uogólnieniem. NETtalk uczy się skutecznie, chociaż wymaga dużej liczby instancji szkoleniowych, a także wielokrotnych przejść przez dane szkoleniowe. W serii testów empirycznych porównujących propagację wsteczną i ID3 dotyczących tego problemu, Shavlik i inni stwierdzili, że algorytmy dają równoważne wyniki, chociaż ich uczenie i wykorzystanie danych było zupełnie inne. W badaniu oceniono algorytmy, dzieląc cały zestaw przykładów na oddzielne zestawy uczące i testowe. Zarówno ID3, jak i NETtalk były w stanie poprawnie wymówić około 60 procent danych testowych po przeszkoleniu na 500 przykładach. Ale tam, gdzie ID3 wymagało tylko jednego przejścia przez dane uczące, NETtalk wymagał wielu powtórzeń zestawu uczącego. W tym badaniu NETtalk miał 100 przejść przez dane treningowe. Jak pokazuje nasz przykład, związek między uczeniem się koneksjonizmu a uczeniem symbolicznym jest bardziej skomplikowany, niż mogłoby się wydawać na początku. W naszym następnym przykładzie zajmiemy się szczegółami rozwiązania wstecznej propagacji problemu wyłącznego.

11.3.3 Wsteczna propagacja Przykład 2: exclusive-or

Kończymy tę sekcję, przedstawiając proste rozwiązanie problemu z ukrytą warstwą. Rysunek 12 przedstawia sieć z dwoma węzłami wejściowymi, jednym ukrytym i jednym wyjściowym. Sieć ma również dwa węzły bias, pierwszy do ukrytego, a drugi do węzła wyjściowego



Wartości netto dla węzłów ukrytych i wyjściowych są obliczane w zwykły sposób, jako iloczyn wektorowy wartości wejściowych pomnożonych przez ich wyuczone wagi. Do tej sumy dodaje się odchylenie. Wagi są trenowane przez propagację wsteczną, a funkcja aktywacji jest sigmoidalna. Należy zauważyć, że węzły wejściowe są również bezpośrednio połączone, z wytrenowanymi wagami, z węzłem wyjściowym. To dodatkowe połączenie może często pozwolić projektantowi na uzyskanie sieci z mniejszą liczbą węzłów w warstwie ukrytej i szybszą konwergencją. W rzeczywistości nie ma nic wyjątkowego w sieci przedstawionej na rysunku 12; do obliczeń na zasadzie wyłączności lub. Wyszkoliliśmy naszą losowo zainicjowaną sieć za pomocą wielu wystąpień czterech wzorców, które reprezentują wartości prawdy wyłączności lub:

(0, 0) → 0; (1, 0) → 1; (0, 1) → 1; (1, 1) → 0

Łącznie 1400 cykli treningowych wykorzystujących te cztery instancje dało następujące wartości, zaokrąglone do najbliższej dziesiątej, dla parametrów wagi z rysunku 12:

W_{H1} = −7,0 W_{HB} = 2,6 W_{O1} = −5,0 W_{OH} = −11,0

W_{H2} = -7,0 W_{OB} = 7,0 W_{O2} = -4,0

Przy wartościach wejściowych (0, 0) wyjście ukrytego węzła to:

f (0 * (- 7,0) + 0 * (- 7,0) + 1 * 2,6) = f (2,6) → 1

Wyjście węzła wyjściowego dla (0,0) to:

f (0 * (- 5,0) + 0 * (- 4,0) + 1 * (- 11,0) + 1 * (7,0)) = f (- 4,0) → 0

Przy wartościach wejściowych (1, 0) wyjście ukrytego węzła to:

f (1 * (- 7,0) + 0 * (- 7,0) + 1 * 2,6) = f (−4,4) → 0

Wynik węzła wyjściowego dla (1,0) to:

f (1 * (- 5,0) + 0 * (- 4,0) + 0 * (- 11,0) + 1 * (7,0)) = f (2,0) → 1

Wartość wejściowa (0, 1) jest podobna. Na koniec sprawdźmy naszą wyłączną lub sieć z wartościami wejściowymi (1, 1). Dane wyjściowe ukrytego węzła to:

f (1 * (- 7,0) + 1 * (- 7,0) + 1 * 2,6) = f (−11,4) → 0

Wynik węzła wyjściowego dla (1,1) to:

f (1 * (- 5,0) + 1 * (- 4,0) + 0 * (- 11,0) + 1 * (7,0)) = f (−2,0) → 0

Czytelnik może zobaczyć, że ta sieć sprzężenia zwrotnego z uczeniem się z propagacją wsteczną spowodowała nieliniowe oddzielenie tych punktów danych. Funkcja progowa f jest sigmoidalną z

rysunku 7b, wyuczone odchylenia bardzo nieznacznie przetłumaczyły ją w dodatnim kierunku na osi x. Następnie rozważymy modele konkurencyjnego uczenia się.

11.4 Konkurencyjne uczenie się

11.4.1 Uczenie się typu „zwycięzca bierze wszystko" na potrzeby klasyfikacji

Algorytm zwycięzca bierze wszystko działa z pojedynczym węzłem w warstwie węzłów, która najsilniej reaguje na wzorzec wejściowy. Zwycięzca bierze wszystko może być postrzegany jako konkurencja między zestawem węzłów sieci, jak na rysunku 13. Na tym rysunku mamy wektor wartości wejściowych, $X = (x_1, x_2, ..., x_m)$, przekazanych do warstwy węzłów sieci, A, B, ..., N. Na diagramie węzeł B jest zwycięzcą konkurencja, z sygnałem wyjściowym 1. Uczenie się, aby zwycięzca bierze wszystko pod uwagę, ponieważ zwycięzca jest określany przez test „maksymalnej aktywacji". Wektor wagi zwycięzcy jest następnie nagradzany poprzez przybliżenie jego składowych do elementów wektora wejściowego. Dla wag W zwycięskiego węzła i składowych X wektora wejściowego przyrost jest następujący:

$\Delta W^t = c\ (X^t{}^-1 - W^t{}^-1)$,

gdzie c jest małą pozytywną stałą uczenia się, która zwykle maleje wraz z postępem uczenia się. Zwycięski wektor wagi jest następnie korygowany poprzez dodanie ΔW^t. Ta nagroda zwiększa lub zmniejsza każdy składnik wektora wagi zwycięzcy o ułamek różnicy $x_i - w_i$. Efektem jest oczywiście lepsze dopasowanie zwycięskiego węzła do wektora wejściowego. Algorytm zwycięzca bierze wszystko nie musi bezpośrednio obliczać poziomów aktywacji, aby znaleźć węzeł z najsilniejszą odpowiedzią. Poziom aktywacji węzła jest bezpośrednio związany z bliskością jego wektora wagi do wektora wejściowego. W przypadku węzła i ze znormalizowanym wektorem wagi Wi, poziom aktywacji WiX jest funkcją odległości euklidesowej między W_i a wzorcem wejściowym X. Można to zobaczyć obliczając odległość euklidesową, ze znormalizowanym W_i:

$$\|X - W_i\| = \sqrt{(X - W_i)^2} = \sqrt{X^2 - 2XW_i + W_i^2}$$

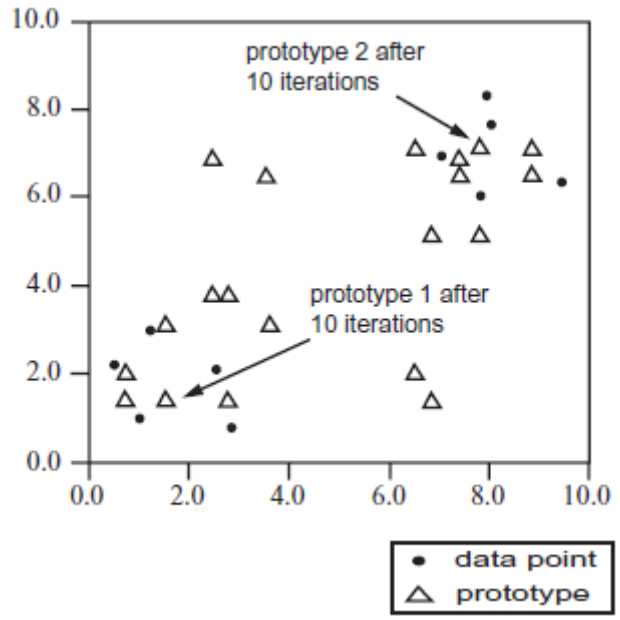
Z tego równania można zauważyć, że dla zbioru znormalizowanych wektorów wag, wektor wagi z najmniejszą odległością euklidesową, || X - W ||, będzie wektorem wagi z maksymalną wartością aktywacji, WX. W wielu przypadkach skuteczniejsze jest określenie zwycięzcy, obliczając odległości euklidesowe, zamiast porównywać poziomy aktywacji na znormalizowanych wektorach wagi. Rozważamy zasadę uczenia się Kohonena „zwycięzca bierze wszystko" z kilku powodów. Po pierwsze, traktujemy to jako metodę klasyfikacji i porównujemy z klasyfikacją perceptronów. Po drugie, można go łączyć z innymi architekturami sieciowymi, aby oferować bardziej wyrafinowane modele uczenia się. Patrzymy na połączenie prototypowego uczenia się Kohonena z zewnętrzną, nadzorowaną siecią uczenia się. Ta hybryda, po raz pierwszy zaproponowana przez Roberta Hecht-Nielsena , nazywana jest siecią kontrpropagacji. W sekcji 11.4.3 widzimy, jak możemy opisać uczenie warunkowe za pomocą kontrpropagacji. Zanim opuścimy to wprowadzenie, jest kilka kwestii ważnych dla algorytmów „winnertake-all". Czasami parametr „sumienia" jest ustawiany i aktualizowany przy każdej iteracji, aby zapobiec zbyt częstemu wygrywaniu poszczególnych węzłów. Zapewnia to, że wszystkie węzły sieci ostatecznie uczestniczą w reprezentowaniu przestrzeni wzorców. W niektórych algorytmach zamiast identyfikować zwycięzcę, który bierze wszystko, wybierany jest zbiór najbliższych węzłów, a waga każdego jest inkrementowana. Innym podejściem jest zróżnicowane wynagrodzenie sąsiednich węzłów zwycięzcy. Wagi są zwykle inicjowane w wartościach losowych, a następnie normalizowane podczas tej metody uczenia .Hecht-Nielsen pokazuje, jak algorytmy „zwycięzca bierze wszystko" mogą być postrzegane jako równoważne analizie k-średnich zbioru danych. W następnej sekcji

przedstawiamy metodę uczenia się klastrów metodą Kohonena, w której zwycięzca bierze wszystko bez nadzoru.

11.4.2 Sieć Kohonena do nauki prototypów

Klasyfikacja danych i rola prototypów w uczeniu się są stałym przedmiotem zainteresowania psychologów, lingwistów, informatyków i kognitywistów. Przedstawiliśmy algorytmy klasyfikacji oparte na symbolach i probabilistyczne grupowanie za pomocą COBWEB i CLUSTER / 2 w Części 9/ W modelach koneksjonistycznych zademonstrowaliśmy klasyfikację opartą na perceptronach w sekcji 11.2, a teraz pokazujemy algorytm grupowania Kohonena , zwycięzca bierze wszystko.

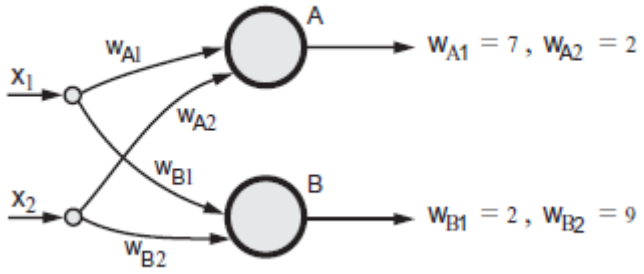
Rysunek 14 ponownie przedstawia punkty danych z tabeli 3.



Na te punkty nałożony jest szereg prototypów utworzonych podczas szkolenia sieci. Algorytm uczący perceptronu zbiegał się po wielu iteracjach, co skutkowało konfiguracją wagi sieci definiującą liniową separację między dwiema klasami. Jak widzieliśmy, linia określona przez te wagi została uzyskana przez niejawne obliczenie „centrum" euklidesowego każdego skupienia. To centrum klastra służy w klasyfikacji perceptronów jako prototyp tej klasy.

Z drugiej strony uczenie się Kohonena odbywa się bez nadzoru, z zestawem prototypów tworzonych losowo, a następnie udoskonalanych, aż zaczną jawnie reprezentować skupiska danych. W miarę kontynuowania działania algorytmu stała uczenia się jest stopniowo redukowana, tak aby każdy nowy wektor wejściowy powodował mniej zakłóceń w prototypach. Uczenie Kohonena, podobnie jak CLUSTER / 2, ma silny błąd indukcyjny polegający na tym, że liczba pożądanych prototypów jest wyraźnie identyfikowana na początku algorytmu, a następnie stale udoskonalana. Pozwala to projektantowi algorytmu sieci na zidentyfikowanie określonej liczby prototypów reprezentujących klastry danych. Kontrpropagacja umożliwia dalsze manipulowanie wybraną liczbą prototypów.

Rysunek 15 przedstawia sieć uczenia Kohonena do klasyfikacji danych z tabeli 3.



Dane są reprezentowane w dwuwymiarowej przestrzeni kartezjańskiej, więc prototypy reprezentujące klastry danych będą również parami uporządkowanymi. Wybieramy dwa prototypy, po jednym reprezentującym każdy klaster danych. Losowo zainicjowaliśmy węzeł A w (7, 2), a węzeł B w (2, 9). Losowa inicjalizacja działa tylko w prostych problemach, takich jak nasz; alternatywą jest ustawienie wektorów wag równych reprezentantom każdego z klastrów. Zwycięski węzeł będzie miał wektor wagi najbliższy wektorowi wejściowemu. Ten wektor wagi dla węzła wygrywającego zostanie nagrodzony jeszcze bardziej zbliżeniem do danych wejściowych, podczas gdy wagi węzłów przegrywających pozostaną niezmienione. Ponieważ jawnie obliczamy odległość euklidesową wektora wejściowego z każdego z prototypów, nie będziemy musieli normalizować wektorów, jak opisano w sekcji 11.4.1.

Uczenie Kohonena odbywa się bez nadzoru, ponieważ prosta miara odległości między każdym prototypem a punktem danych umożliwia wyłonienie zwycięzcy. Klasyfikacja zostanie „odkryta” w kontekście tej samoorganizującej się sieci. Chociaż nauka Kohonena wybiera punkty danych do analizy w kolejności losowej, punkty z tabeli 11.3 bierzemy w kolejności od góry do dołu. Dla punktu (1, 1) mierzymy odległość od każdego prototypu:

|| (1, 1) - (7, 2) || = (1 - 7) 2 + (1 - 2) 2 = 37 i

|| (1, 1) - (2, 9) || = (1 - 2) 2 + (1 - 9) 2 = 65.

Węzeł A (7, 2) jest zwycięzcą, ponieważ jest najbliżej (1, 1). || (1, 1) - (7, 2) || reprezentuje odległość między tymi dwoma punktami; nie musimy stosować funkcji pierwiastka kwadratowego w euklidesowej miary odległości, ponieważ relacja wielkości jest niezmienna. Teraz nagradzamy zwycięski węzeł, używając stałej uczenia c ustawionej na 0,5. W drugiej iteracji:

$W^2 = W^1 + c\,(X^1 - W^1)$

= (7, 2) + .5 ((1, 1) - (7, 2)) = (7, 2) + .5 ((1 - 7), (1 - 2))

= (7, 2) + (−3, −,5) = (4, 1,5)

W drugiej iteracji algorytmu uczącego się mamy dla punktu danych (9.4, 6.4):

|| (9,4; 6,4) - (4; 1,5) || = (9,4 - 4) 2 + (6,4 - 1,5) 2 = 53,17 i

|| (9,4; 6,4) - (2, 9) || = (9,4 - 2) 2 + (6,4 - 9) 2 = 60,15

Ponownie, węzeł A jest zwycięzcą. Waga dla trzeciej iteracji to:

$W^3 = W^2 + c\,(X^2 - W^2)$

= (4; 1,5) + .5 ((9,4; 6,4) - (4; 1,5))

= (4; 1,5) + (2,7; 2,5) = (6,7; 4)

W trzeciej iteracji mamy dla punktu danych (2.5, 2.1):

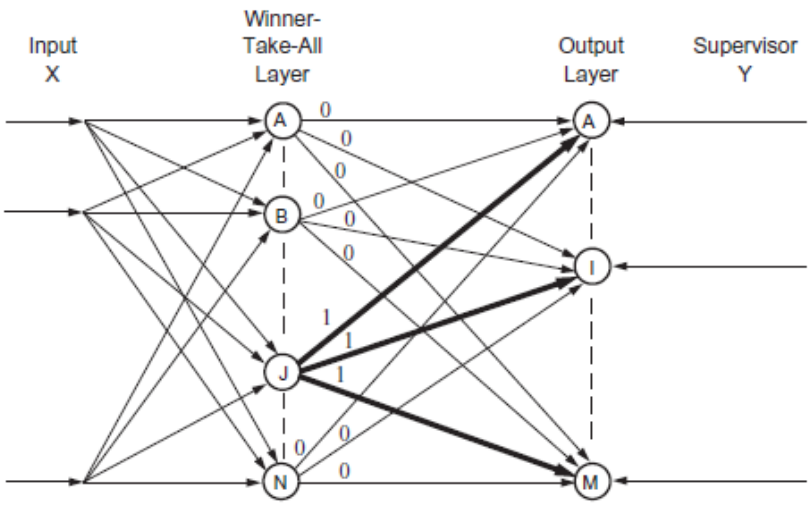
|| (2,5, 2,1) - (6,7, 4) || = (2,5 - 6,7) 2 + (2,1 - 4) 2 = 21,25 i

|| (2.5, 2.1) - (2, 9) || = (2,5 - 2) 2 + (2,1 - 9) 2 = 47,86.

Węzeł A ponownie wygrywa i przechodzimy do obliczenia nowego wektora wagi. Rysunek 14 przedstawia ewolucję prototypu po 10 iteracjach. Algorytm użyty do wygenerowania danych z rysunku 14 wybrał dane losowo z tabeli 3, więc pokazane prototypy będą różnić się od tych właśnie utworzonych. Można zauważyć postępującą poprawę prototypów w kierunku centrów klastrów danych. Ponownie, jest to nienadzorowany algorytm zbrojenia winnertake- all. Tworzy zestaw ewoluujących i wyraźnych prototypów reprezentujących klastry danych. Wielu badaczy, w tym Zurada (1992) i Hecht-Nielsen (1990), zwraca uwagę, że klasyfikacja danych bez nadzoru Kohonena jest zasadniczo taka sama jak analiza k-średnich. Następnie rozważymy, wraz z Grossbergiem lub gwiazdą zewnętrzną, rozszerzeniem analizy Kohonena zwycięzca bierze wszystko, algorytmem, który pozwoli nam rozszerzyć moc wyboru prototypów.

11.4.3 Sieci Outstar i kontrpropagacja

Do tego momentu rozważaliśmy nienadzorowane grupowanie danych wejściowych. Uczenie się tutaj wymaga niewielkiej wiedzy a priori o dziedzinie problemowej. Stopniowo wykrywane cechy danych, a także historia szkolenia, prowadzą do identyfikacji klas i odkrywania granic między nimi. Gdy punkty danych zostaną zgrupowane zgodnie z podobieństwami w ich reprezentacjach wektorowych, nauczyciel może pomóc w kalibracji lub nadawaniu nazw klasom danych. Odbywa się to w formie nadzorowanego szkolenia, w którym pobieramy węzły wyjściowe warstwy sieciowej „zwycięzca bierze wszystko" i wykorzystujemy je jako dane wejściowe do drugiej warstwy sieci. Następnie wyraźnie wzmocnimy decyzje w tej warstwie wyjściowej. To nadzorowane szkolenie, a następnie wzmocnione dane wyjściowe, pozwalają nam na przykład odwzorować wyniki sieci Kohonena na wzorzec lub klasę wyjściową. Pozwala nam na to warstwa Grossberga , implementująca algorytm zwany outstar. Połączona sieć, warstwa Kohonena połączona z warstwą Grossberga, nazywana jest kontrpropagacją i została po raz pierwszy zaproponowana przez Roberta Hecht-Nielsena . W sekcji 11.4.2 omówiliśmy bardziej szczegółowo warstwę Kohonena; tutaj rozważamy warstwę Grossberg. Rysunek 16 przedstawia warstwę węzłów A, B, ..., N, gdzie jeden węzeł, J, jest wybierany jako zwycięzca.

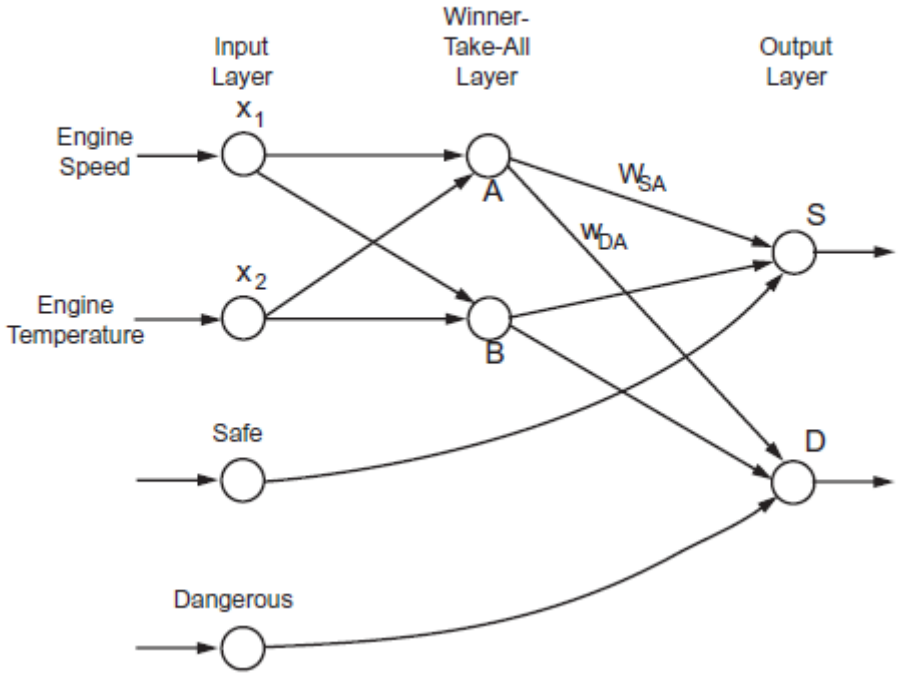


Uczenie Grossberga jest nadzorowane w ten sposób, że chcemy, aby wraz z informacją zwrotną nauczyciela, reprezentowaną przez wektor Y, wzmocnić wagę łączącą J z węzłem I w warstwie

wyjściowej, która ma odpalić. Dzięki uczeniu się od gwiazdy identyfikujemy i zwiększamy wagę WJI na wychodzącym łączu J do I. Aby wytrenować sieć kontrpropagacji, najpierw trenujemy warstwę Kohonena. Po znalezieniu zwycięzcy wartości wszystkich wychodzących z niego łączy będą wynosić 1, podczas gdy wszystkie wartości wyjściowe jego konkurentów pozostaną 0. Ten węzeł, wraz ze wszystkimi węzłami na warstwie wyjściowej, z którą jest połączony, tworzą tak zwana gwiazda zewnętrzna . Szkolenie dla warstwy Grossberg opiera się na komponentach outstar. Jeśli każdy klaster wektorów wejściowych reprezentuje jedną klasę i chcemy, aby wszyscy członkowie klasy odwzorowywali tę samą wartość w warstwie wyjściowej, nie jest nam potrzebne szkolenie iteracyjne. Musimy tylko określić, który węzeł w warstwie zwycięzca bierze wszystko jest powiązany z którą klasą, a następnie przypisać wagi z tych węzłów do węzłów wyjściowych na podstawie powiązania między klasami i pożądanymi wartościami wyjściowymi. Na przykład, jeśli J-ta jednostka zwycięzca bierze wszystko wygrywa dla wszystkich elementów klastra, dla których I = 1 jest pożądanym wyjściem sieci, ustawiamy $w_{JI} = 1$ i $w_{JK} = 0$ dla wszystkich innych wag na zewnątrz gwiazdy J . Jeśli pożądane dane wyjściowe dla elementów klastra są różne, istnieje procedura iteracyjna, wykorzystująca wektor nadzoru Y, do wyrównywania wag zewnętrznych. Wynikiem tej procedury uczącej jest uśrednienie pożądanych wartości wyjściowych dla elementów określonego klastra. Trenujemy wagi połączeń zewnętrznych od zwycięskiego węzła do węzłów wyjściowych zgodnie z równaniem:

$$W^{t+1} = W^t + c\,(Y - W^t)$$

gdzie c jest małą dodatnią stałą uczenia się, W^t jest wektorem wagi składowej zewnętrznej, a Y jest pożądanym wektorem wyjściowym. Zauważ, że ten algorytm uczenia się skutkuje zwiększeniem połączenia między węzłem J w warstwie Kohonena a węzłem I na warstwie wyjściowej dokładnie wtedy, gdy I jest zwycięskim węzłem z wyjściem 1 i pożądanym wyjściem J jest również 1. To sprawia, że Jest to przykład uczenia się Hebba, formy uczenia się, w której ścieżka neuronowa jest wzmacniana za każdym razem, gdy jeden węzeł przyczynia się do odpalenia innego. Bardziej szczegółowo omówimy naukę języka Hebba w sekcji 11.5. Następnie zastosujemy regułę do uczenia sieci kontrpropagacji w celu rozpoznawania klastrów danych z tabeli 11.3. Na tym przykładzie pokazujemy również, jak sieci przeciwpropagacyjne wdrażają uczenie warunkowe. Załóżmy, że parametr x_1 w tabeli 3 reprezentuje prędkość silnika w układzie napędowym. x_2 oznacza temperaturę silnika. Zarówno prędkość, jak i temperatura systemu są kalibrowane w celu uzyskania punktów danych w zakresie [0, 10]. Nasz system monitorowania regularnie pobiera próbki danych. Zawsze, gdy prędkość i temperatura są zbyt wysokie, chcemy nadać ostrzeżenie. Zmieńmy nazwy wartości wyjściowych z tabeli 3 z +1 na „bezpieczne” i z -1 na „niebezpieczne”. Nasza sieć kontrpropagacji będzie wyglądać jak na rysunku 17.



Ponieważ wiemy dokładnie, jakie wartości chcemy, aby każdy zwycięski węzeł sieci Kohonena odwzorował na warstwie wyjściowej sieci Grossberga, możemy bezpośrednio ustawić te wartości. Jednak aby zademonstrować uczenie się od gwiazd, będziemy szkolić sieć przy użyciu właśnie podanego wzoru. Jeśli podejmiemy (arbitralną) decyzję, węzeł S warstwy wyjściowej powinien sygnalizować sytuacje bezpieczne i węzeł D jest niebezpieczny, wówczas wagi zewnętrznej gwiazdy dla węzła A na warstwie wyjściowej sieci Kohonena powinny wynosić [1, 0], a wagi zewnętrznej gwiazdy dla B powinny wynosić [0, 1]. Ze względu na symetrię sytuacji pokazujemy trening gwiazdy zewnętrznej tylko dla węzła A. Siatka Kohonena musiała się ustabilizować, zanim można będzie trenować siatkę Grossberga. Zbieżność Kohonena tej samej sieci zademonstrowaliśmy w sekcji 11.4.2. Wektory wejściowe do uczenia węzła zewnętrznego A mają postać [x1, x2, 1, 0]. x1 i x2 to wartości z tabeli 11.3, które są skupione w węźle wyjściowym A Kohonena, a ostatnie dwie składowe wskazują, że gdy A jest zwycięzcą Kohonena, bezpieczne jest „prawda”, a niebezpieczne jest „fałsz”, jak na rysunku 11.15. Inicjalizujemy wagi gwiazdy zewnętrznej A na [0, 0] i używamy .2 jako stałej uczenia się:

$W^1 = [0, 0] + .2\ [[1, 0] - [0, 0]] = [0, 0] + [.2, 0] = [.2, 0]$

$W^2 = [.2, 0] + .2\ [[1, 0] - [.2, 0]] = [.2, 0] + [.16, 0] = [.36, 0]$

$W^3 = [.36, 0] + .2\ [[1, 0] - [.36, 0]] = [.36, 0] + [.13, 0] = [.49, 0]$

$W^4 = [.49, 0] + .2\ [[1, 0] - [.49, 0]] = [.49, 0] + [.10, 0] = [.59, 0]$

$W^5 = [.59, 0] + .2\ [[1, 0] - [.59, 0]] = [.59, 0] + [.08, 0] = [.67, 0].$

Jak widać, wraz z treningiem ciężary te zmierzają w kierunku [1, 0]. Oczywiście, ponieważ w tym przypadku elementy klastra powiązane z A zawsze mapują się do [1,0], mogliśmy raczej użyć prostego algorytmu przypisania niż algorytmu uśredniającego do uczenia. Pokażemy teraz, że to przypisanie daje odpowiednią odpowiedź z sieci kontrpropagacji. Kiedy pierwszy wektor wejściowy z tabeli 3 zostanie zastosowany do sieci na rysunku 17, otrzymamy aktywację [1, 1] dla wag outstar węzła A i [0, 0] dla

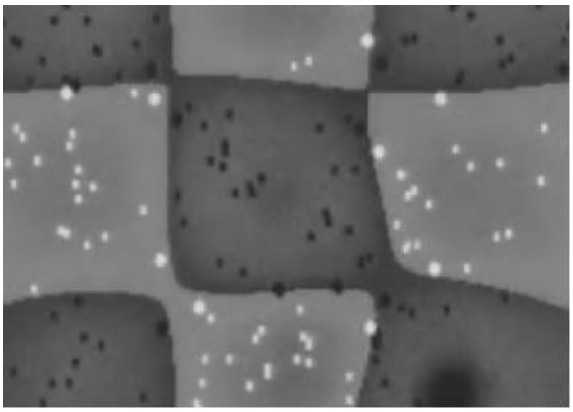
outstar z B. Iloczyn skalarny aktywacji a wagi dla węzła S warstwy wyjściowej to [1, 0] * [1, 0]; daje to aktywację 1 do węzła wyjściowego S. Przy wagach outstar B wytrenowanych do [0, 1], aktywacja dla węzła D wynosi [1, 0] * [0, 1]; to są wartości, których oczekujemy. Testując drugi rząd punktów danych w tabeli 11.3, otrzymujemy aktywację [0, 0] z węzła A i [1,1] z punktu B na poziomie zwycięzca bierze wszystko. Iloczyn skalarny tych wartości i wytrenowanych wag daje 0 do węzła S i 1 do D, znowu to, czego się oczekuje. Czytelnik może kontynuować testowanie innych danych z tabeli 3.

Z poznawczego punktu widzenia, sieci kontrpropagacji możemy nadać asocjalistyczną interpretację. Rozważ jeszcze raz Rysunek 17. Uczenie się na warstwie Kohonena można postrzegać jako zdobywanie warunkowego bodźca, ponieważ sieć uczą się wzorców w zdarzeniach. Z drugiej strony uczenie się na poziomie Grossberga jest skojarzeniem węzłów (bodźców bezwarunkowych) z jakąś reakcją. W naszej sytuacji system uczy się nadawać ostrzeżenie o niebezpieczeństwie, gdy dane pasują do określonego wzorca. Po nauczeniu się właściwej odpowiedzi, system reaguje odpowiednio na nowe dane, nawet bez dalszego coachingu nauczyciela. Drugą poznawczą interpretacją kontrpropagacji jest wzmocnienie powiązań pamięciowych dla wzorca zjawisk. Jest to podobne do tworzenia tabeli odnośników dla odpowiedzi na wzorce danych. W niektórych przypadkach kontrpropagacja ma znaczną przewagę nad propagacją wsteczną. Podobnie jak wsteczna propagacja, jest w stanie nauczyć się nieliniowo rozdzielnych klasyfikacji. Dzieje się to jednak dzięki wstępnemu przetwarzaniu, które odbywa się w warstwie Kohonena, gdzie zestaw danych jest podzielony na klastry jednorodnych danych. To partycjonowanie może dać znaczną przewagę nad propagacją wsteczną w szybkości uczenia się, ponieważ jawne partycjonowanie danych na oddzielne klastry zastępuje często szeroko zakrojone wyszukiwanie wymagane w ukrytych warstwach w sieciach z propagacją wsteczną.

11.4.4 Maszyny wektorów nośnych

Maszyny wektorów nośnych (SVM) stanowią kolejny przykład konkurencyjnego uczenia się. W podejściu opartym na wektorach nośnych miary statystyczne służą do określenia minimalnego zestawu punktów danych (wektorów nośnych), które maksymalnie oddzielają pozytywne i negatywne wystąpienia wyuczonej koncepcji. Te wektory pomocnicze, reprezentujące wybrane punkty danych zarówno z pozytywnych, jak i negatywnych przykładów koncepcji, domyślnie definiują hiperpłaszczyznę oddzielającą te dwa zbiory danych. Na przykład uruchomienie algorytmu SVM identyfikuje punkty (2.5, 2.1) i (1.2, 3.0) jako wektory pomocnicze dla instancji dodatnich oraz (7.0, 7.0) i (7.8. 6.1) jako wektory pomocnicze dla ujemnych instancji danych Tabela 3 i Rysunek 5. Po nauczeniu się wektorów nośnych inne punkty danych nie muszą już być zachowywane, same wektory nośne wystarczą do określenia oddzielającej hiperpłaszczyzny. Maszyna wektorów nośnych jest klasyfikatorem liniowym, w którym nadzorowane jest uczenie się wektorów nośnych. Zakłada się, że dane do uczenia SVM są tworzone niezależnie i identycznie na podstawie ustalonego, choć nieznanego, rozkładu danych. Hiperpłaszczyzna, pośrednio zdefiniowana przez same wektory nośne, oddziela instancje danych dodatnich od ujemnych. Punkty danych najbliżej hiperpłaszczyzny znajdują się na marginesie decyzyjnym. Każde dodanie lub usunięcie wektora nośnego zmienia granicę hiperpłaszczyzny. Jak wspomniano wcześniej, po zakończeniu uczenia można zrekonstruować hiperpłaszczyznę i sklasyfikować nowe zestawy danych z samych wektorów nośnych. Algorytm SVM klasyfikuje elementy danych, obliczając odległość punktu danych od oddzielającej hiperpłaszczyzny jako problem optymalizacji. Pomyślne kontrolowanie zwiększonej elastyczności przestrzeni cech, (często przekształcanych) parametrów instancji, których należy się nauczyć, wymaga wyrafinowanej teorii uogólnień. Teoria ta musi być w stanie precyzyjnie opisać cechy, które muszą być kontrolowane, aby uzyskać dobre uogólnienie. W statystyce zagadnienie to znane jest jako badanie wskaźników jednolitej konwergencji. Przykład tego widzieliśmy już w Części, gdzie przedstawiono prawdopodobnie w przybliżeniu poprawny model uczenia się, czyli PAC. Wyniki uczenia się PAC można postrzegać jako

ustalenie granic liczby przykładów wymaganych do zagwarantowania określonego ograniczenia błędu. W tym zadaniu uogólniającym stosuje się Bayesian lub inne techniki kompresji danych. W nauce SVM często wykorzystuje się teorię Vapnika i Czerwonenkisa. Wymiar Vapnik Chervonenkis (VC) definiuje się jako maksymalną liczbę punktów szkoleniowych, które można podzielić na dwie kategorie za pomocą zestawu funkcji. Tak więc teoria VC zapewnia dystrybucję wolną od uogólnienia spójnej hipotezy. Algorytm SVM wykorzystuje tę teorię VC do obliczenia hiperpłaszczyzny i kontroluje margines błędu dla dokładności uogólnień, czasami nazywanego pojemnością funkcji. Maszyny SVM używają miary podobieństwa iloczynu skalarnego do mapowania danych z przestrzeni cech. Wyniki iloczynu punktowego reprezentujące odwzorowane wektory są liniowo łączone przez wagi znalezione przez rozwiązanie programu kwadratowego. Funkcja jądra, taka jak wielomian, splajn lub Gaussian, jest używana do tworzenia odwzorowania wektora cech, w którym wybór jądra jest określany przez rozkład problemu. Maszyny SVM obliczają odległości w celu określenia klasyfikacji elementów danych. Te reguły decyzyjne stworzone przez SVM odzwierciedlają statystyczne prawidłowości w danych. Po przeszkoleniu SVM klasyfikacja nowych punktów danych jest po prostu kwestią porównania z wektorami nośnymi. W wektorach pomocniczych krytyczne cechy charakteryzujące wyuczoną koncepcję są skupione po jednej stronie hiperpłaszczyzny, te opisujące jej negację po drugiej, a cechy, które nie dyskryminują, nie są używane. Dla algorytmu perceptronowego z sekcji 11.2 ważna jest liniowa rozdzielność danych: jeśli danych nie da się rozdzielić, algorytm nie będzie zbieżny. SVM, alternatywnie, stara się zmaksymalizować margines decyzyjny i jest bardziej odporny na radzenie sobie ze słabym rozdziałem spowodowanym nakładaniem się punktów danych. Jest w stanie wykorzystać zmienne luzu do złagodzenia więzów liniowych w celu znalezienia miękkiego marginesu, z wartościami, które oznaczają poziom ufności granicy klasyfikacji (Cristianini i Shawe-Taylor 2000). W rezultacie niektóre wektory pomocnicze, które są wartościami odstającymi, mogą zostać błędnie sklasyfikowane w celu utworzenia hiperpłaszczyzny, w wyniku czego margines decyzyjny zostanie zawężony, gdy dane są zaszumione. SVM można uogólnić od dwóch problemów klasyfikacji kategorii do dyskryminacji wielu klas poprzez wielokrotne uruchamianie SVM dla każdej kategorii zainteresowania względem wszystkich innych kategorii . Maszyny SVM najlepiej nadają się do rozwiązywania problemów z danymi liczbowymi, a nie jakościowymi; w rezultacie ich zastosowanie w wielu klasycznych problemach kategoryzacji z granicami jakościowymi jest ograniczone. Ich siła tkwi w matematycznych podstawach: minimalizacji wypukłej funkcji kwadratowej przy liniowych ograniczeniach nierówności. Maszyny SVM są stosowane w wielu sytuacjach uczenia się, w tym w klasyfikacji stron internetowych. W kategoryzacji tekstu ważona jest obecność wyszukiwania lub innych powiązanych słów. Każdy dokument staje się następnie danymi wejściowymi dla SVM w celu kategoryzacji na podstawie informacji o częstotliwości słów. SVM są również używane do rozpoznawania obrazu, koncentrując się na wykrywaniu krawędzi i opisie kształtu przy użyciu skali szarości lub informacji o intensywności koloru . Na rysunku 18, zaadaptowanym na podstawie Cristianini i Shawe-Taylor , maszyna wektorów nośnych rozróżnia granice na szachownicy.



Więcej szczegółów na temat maszyn SVM i pełnego algorytmu uczenia SVM można znaleźć w.

11.5 Hebrajskie uczenie się przez przypadek

11.5.1 Wprowadzenie

Teoria uczenia się Hebba opiera się na obserwacji, że w systemach biologicznych, kiedy jeden neuron przyczynia się do odpalenia innego neuronu, wzmacnia się połączenie lub ścieżka między dwoma neuronami. Hebb stwierdził: Kiedy akson komórki A jest wystarczająco blisko, aby wzbudzić komórkę B i wielokrotnie lub trwale ma miejsce podczas jej odpalania, pewien proces wzrostu lub przemiana metaboliczna zachodzi w jednej lub obu komórkach, tak że wydajność A jako jednej z komórek odpalających B jest zwiększona. Nauka języka Hebba jest atrakcyjna, ponieważ tworzy oparte na zachowaniu koncepcje nagrody na poziomie neuronalnym. Fizjologiczne badania neuronalne potwierdziły, że idea Hebba, zgodnie z którą czasowa bliskość odpalenia połączonych neuronów może modyfikować siłę synaps, choć w znacznie bardziej złożony sposób, niż prosty „wzrost wydajności" Hebba, jest przynajmniej w przybliżeniu poprawny. Konkretne prawo dotyczące uczenia się przedstawione w tej sekcji jest obecnie nazywane uczeniem się języka Hebba, mimo że jego idee były nieco bardziej abstrakcyjne. To uczenie się należy do kategorii koincydencji praw uczenia się, które powodują zmiany wagi w odpowiedzi na zlokalizowane zdarzenia w przetwarzaniu neuronowym. Opisujemy prawa uczenia się z tej kategorii poprzez ich lokalne właściwości czasu i przestrzeni. Uczenie Hebba było używane w wielu architekturach sieciowych. Jest używany zarówno w nadzorowanych, jak i nienadzorowanych trybach uczenia się. Efekt wzmocnienia połączenia między dwoma neuronami, gdy jeden przyczynia się do odpalenia drugiego, można zasymulować matematycznie, dostosowując ciężar ich połączenia przez stałą pomnożenie znaku iloczynu ich wartości wyjściowych. Zobaczmy, jak to działa. Załóżmy, że neurony i i j są połączone w taki sposób, że wyjście i jest wejściem j. Możemy zdefiniować dopasowanie wagi na połączeniu między nimi, ΔW, jako znak c * (o_i * o_j), gdzie c jest stałą sterującą szybkością uczenia się. W tabeli 4, Oi jest znakiem wartości wyjściowej i oraz O_j wyjścia j. Z pierwszego wiersza tabeli widzimy, że gdy O_i i O_j są dodatnie, korekta wagi ΔW jest dodatnia. Ma to wpływ na wzmocnienie połączenia między i i j, kiedy i przyczyniło się do „wypalenia" j.

O_i	O_j	O_i*O_j
+	+	+
+	–	–
–	+	–
–	–	+

W drugim i trzecim rzędzie tabeli 4, i oraz j mają przeciwne znaki. Ponieważ ich znaki się różnią, chcemy zahamować udział i w wartości wyjściowej j. Dlatego dostosowujemy wagę połączenia o ujemny przyrost. Wreszcie, w czwartym rzędzie, i i j ponownie mają ten sam znak. Oznacza to, że zwiększamy siłę ich połączenia. To mechanizm regulacji wagi wzmacnia ścieżkę między neuronami, gdy mają podobne sygnały, a inaczej je hamuje. W następnych rozdziałach rozważymy dwa typy uczenia się języka hebrajskiego, nienadzorowane i nadzorowane. Rozpoczynamy od zbadania nienadzorowanego formularza.

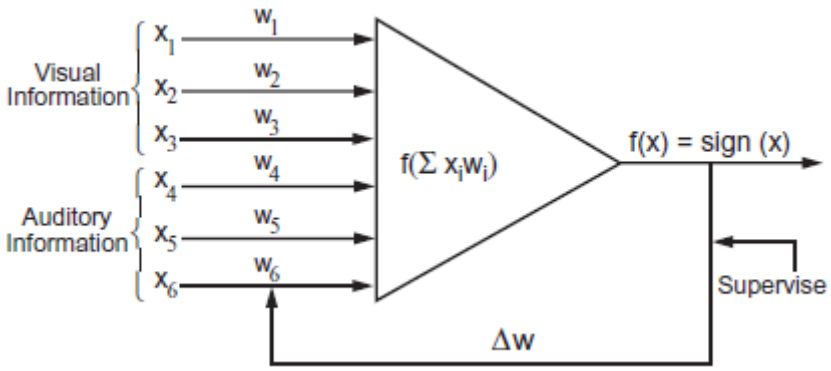
11.5.2 Przykład nienadzorowanej nauki języka Hebba

Przypomnij sobie, że w uczeniu się bez nadzoru krytyk nie jest dostępny, aby zapewnić „prawidłową" wartość wyjściową; w ten sposób wagi są modyfikowane wyłącznie jako funkcja wartości wejściowych i wyjściowych neuronu. Szkolenie tej sieci skutkuje wzmocnieniem reakcji sieci na wzorce, które już widziała. W następnym przykładzie pokazujemy, jak techniki Hebbowskie można wykorzystać do

modelowania uczenia się w odpowiedziach warunkowych, gdzie arbitralnie wybrany bodziec może być użyty jako warunek pożądanej odpowiedzi. Wagę można dostosować, ΔW, dla węzła i w nienadzorowanym uczeniu się języka Hebbian za pomocą:

$\Delta W = c * f(X, W) * X$

gdzie c to stała uczenia się, mała liczba dodatnia, f (X, W) to wyjście i, a X to wektor wejściowy do i. Pokażemy teraz, jak sieć może wykorzystać uczenie Hebba do przeniesienia odpowiedzi z bodźca pierwotnego lub bezwarunkowego na bodziec warunkowy. To pozwala nam modelować typ uczenia się badany w eksperymentach Pawłowa, w których poprzez jednoczesne dzwonienie dzwonkiem za każdym razem, gdy podawano pożywienie, przenoszono do dzwonka reakcję ślinienia psa na pokarm. Sieć pokazana na rysunku 19 ma dwie warstwy, warstwę wejściową z sześcioma węzłami i warstwę wyjściową z jednym węzłem.



Warstwa wyjściowa zwraca +1, co oznacza, że neuron wyjściowy został odpalony, lub -1, co oznacza, że jest w stanie spoczynku.

Niech stała uczenia się będzie małą dodatnią liczbą rzeczywistą 0,2. W tym przykładzie trenujemy sieć według wzorca [1, −1, 1, −1, 1 −1], który jest połączeniem dwóch wzorców, [1, −1, 1] i [−1, 1, - 1]. Wzór [1, −1, 1] reprezentuje bodziec bezwarunkowy, a [−1, 1, −1] reprezentuje nowy bodziec. Zakładamy, że sieć już reaguje pozytywnie na bezwarunkowy bodziec, ale jest neutralna w stosunku do nowego bodźca. Pozytywną reakcję sieci na bodziec bezwarunkowy symulujemy za pomocą wektora wagi [1, −1, 1], dokładnie dopasowanego do wzorca wejściowego, natomiast neutralną odpowiedź sieci na nowy bodziec symuluje wektor wag [0 , 0, 0]. Połączenie tych dwóch wektorów wag daje nam początkowy wektor wag dla sieci, [1, −1, 1, 0, 0, 0]. Teraz trenujemy sieć według wzorca wejściowego, mając nadzieję na wywołanie takiej konfiguracji wag, która da pozytywną odpowiedź sieci na nowy bodziec. Pierwsza iteracja sieci daje:

$W * X = (1 * 1) + (-1 * -1) + (1 * 1) + (0 * -1) + (0 * 1) + (0 * -1)$

$= (1) + (1) + (1) = 3$

$f(3) = \text{znak}(3) = 1.$

Teraz tworzymy nową wagę W^2:

$W^2 = [1, -1, 1, 0, 0, 0] + .2 * (1) * [1, -1, 1, -1, 1, -1]$

$= [1, -1, 1, 0, 0, 0] + [.2, -.2, .2, -.2, .2, -.2]$

$= [1,2, -1,2, 1,2, -,2, .2, -,2.]$

Wystawiamy dostosowaną sieć na oryginalny wzorzec wejściowy:

W * X = (1,2 * 1) + (−1,2 * −1) + (1,2 * 1) + (−.2 * −1) + (.2 * 1) + (−.2 * −1) = (1.2) + (1.2) + (1.2) + (+.2) + (.2) + (.2) = 4,2 i znak (4.2) = 1.

Teraz tworzymy nową wagę W^3:

W^3 = [1,2, −1,2, 1,2, −,2, .2, −,2] + .2 * (1) * [1, −1, 1, −1, 1 −1]

= [1.2, −1.2, 1.2, −.2, .2, −.2] + [.2, −.2, .2, −.2, .2, −.2]

= [1,4, −1,4, 1,4, −,4, .4, −,4.]

Można teraz zobaczyć, że iloczyn wektorowy W * X będzie nadal rósł w kierunku dodatnim, przy czym wartość bezwzględna każdego elementu wektora wagi będzie wzrastać o 0,2 w każdym cyklu treningowym. Po 10 kolejnych iteracjach treningu Hebba, wektor wagi będzie wyglądał następująco:

W^{13} = [3,4, −3,4, 3,4, −2,4, 2,4, −2,4].

Teraz używamy tego wytrenowanego wektora wagi do testowania odpowiedzi sieci na dwa częściowe wzorce. Chcielibyśmy zobaczyć, czy sieć nadal pozytywnie reaguje na bezwarunkowy bodziec i, co ważniejsze, czy sieć uzyskała teraz pozytywną odpowiedź na nowy, warunkowy bodziec. Najpierw testujemy sieć na bezwarunkowym bodźcu [1, –1, 1]. Wypełniamy ostatnie trzy argumenty wektora wejściowego losowymi przypisaniami 1 i –1. Na przykład, testujemy sieć na wektorze [1, –1, 1, 1, 1, –1]:

sign (W * X) = sign ((3,4 * 1) + (−3,4 * −1) + (3,4 * 1)

+ (−2,4 * 1) + (2,4 * 1) + (−2,4 * −1))

= sign (3,4 + 3,4 + 3,4 - 2,4 + 2,4 + 2,4)

= sign (12,6) = +1.

W ten sposób sieć nadal pozytywnie reaguje na pierwotny bezwarunkowy bodziec. Teraz wykonujemy drugi test, używając oryginalnego bezwarunkowego bodźca i innego losowego wektora w ostatnich trzech pozycjach [1, –1, 1, 1, –1, –1]:

sign (W * X) = znak ((3,4 * 1) + (−3,4 * −1) + (3,4 * 1)

+ (−2,4 * 1) + (2,4 * −1) + (−2,4 * −1))

= sign (3,4 + 3,4 + 3,4 - 2,4 - 2,4 + 2,4)

= sign (7,8) = +1.

Drugi wektor również daje pozytywną odpowiedź sieci. W rzeczywistości zauważamy w tych dwóch przykładach, że wrażliwość sieci na pierwotny bodziec, mierzona jego pierwotną aktywacją, została wzmocniona z powodu wielokrotnej ekspozycji na ten bodziec. Teraz testujemy odpowiedź sieci na nowy wzorzec bodźca, [–1, 1, –1], zakodowany w ostatnich trzech pozycjach wektora wejściowego. Wypełniamy pierwsze trzy pozycje wektora losowymi przypisaniami ze zbioru {1, –1} i testujemy sieć na wektorze [1, 1, 1, –1, 1, –1]:

sign (W * X) = znak ((3,4 * 1) + (−3,4 * −1) + (3,4 * 1)

+ (−2,4 * 1) + (2,4 * 1) + (−2,4 * −1))

= sign (3,4 - 3,4 + 3,4 + 2,4 + 2,4 + 2,4)

= sign (10,6) = +1.

Rozpoznawany jest również wzór bodźca wtórnego! Wykonujemy ostatni eksperyment z nieznacznie zdegradowanymi wzorcami wektorów. To mogłoby przedstawiać sytuację bodźca, w której sygnały wejściowe są nieco zmienione, być może z powodu użycia nowego pokarmu i innego dzwonka. Testujemy sieć na wektorze wejściowym [1, –1, –1, 1, 1, –1], w którym pierwsze trzy parametry są o jeden od oryginalnego bezwarunkowego bodźca, a ostatnie trzy o jeden od warunkowego bodźca:

sign (W * X) = znak ((3,4 * 1) + (–3,4 * –1) + (3,4 * 1)

+ (–2,4 * 1) + (2,4 * 1) + (–2,4 * –1))

= sign (3,4 + 3,4 - 3,4 - 2,4 + 2,4 + 2,4)

= sign (5,8) = +1.

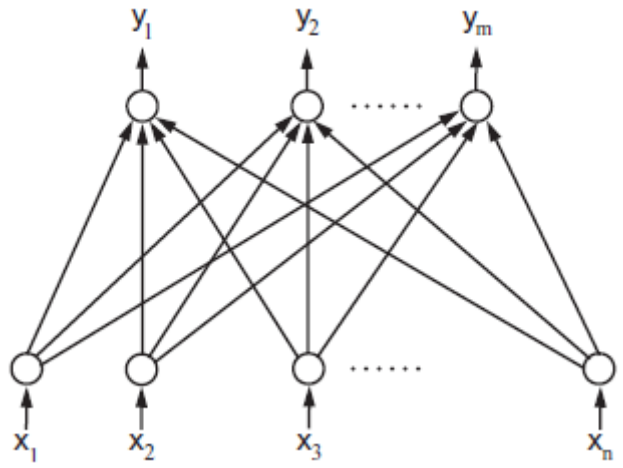
Nawet częściowo zdegradowany bodziec jest rozpoznawany!

Co stworzył model nauczania Hebba? Stworzyliśmy skojarzenie między nowym bodźcem a starą odpowiedzią, wielokrotnie przedstawiając razem stary i nowy bodziec. Sieć uczy się bez nadzoru przekazywać swoją odpowiedź na nowy bodziec. Ta wzmocniona czułość pozwala również sieci reagować w ten sam sposób na nieco zdegradowaną wersję bodźca. Osiągnięto to dzięki zastosowaniu metody Hebba, uczenia się przez przypadek, aby zwiększyć siłę reakcji sieci na ogólny wzorzec, co skutkuje zwiększeniem siły odpowiedzi sieci na każdy indywidualny składnik wzoru.

11.5.3 Nadzorowana nauka języka Hebba

Reguła uczenia się Hebba opiera się na zasadzie, że siła połączenia między neuronami wzrasta, gdy jeden neuron przyczynia się do odpalenia innego. Zasadę tę można dostosować do nadzorowanej sytuacji uczenia się, opierając regulację wagi połączenia na pożądanym wyjściu neuronu, a nie na rzeczywistym wyjściu. Na przykład, jeśli wejście neuronu A do neuronu B jest dodatnie, a pożądana odpowiedź neuronu B jest wyjściem dodatnim, wówczas waga połączenia z A do B jest zwiększona. Przeanalizujemy aplikację nadzorowanej nauki języka Hebba pokazującą, jak działa sieć można nauczyć rozpoznawać zestaw skojarzeń między wzorami. Powiązania są podane przez zbiór uporządkowanych par, {<X_1, Y_1>, <X_2, Y_2>, ..., <X_t, Y_t>}, gdzie X_i i Y_i to wzorce wektorowe, które mają być skojarzone. Załóżmy, że długość X_i wynosi n, a Y_i to m.

Projektujemy sieć tak, aby wyraźnie pasowała do tej sytuacji. Dlatego ma dwie warstwy, warstwę wejściową o rozmiarze n i warstwę wyjściową o rozmiarze m, jak na rysunku 20.



Formułę uczenia się dla tej sieci można wyprowadzić, zaczynając od formuły uczenia się języka Hebbian z poprzedniej sekcji:

$\Delta W = c * f(X, W) * X$

gdzie f(X, W) jest rzeczywistym wyjściem węzła sieci. W uczeniu nadzorowanym zastępujemy rzeczywiste dane wyjściowe węzła pożądanym wektorem wyjściowym D, otrzymując wzór:

$\Delta W = c * D * X$

Mając parę wektorów <X, Y> ze zbioru skojarzonych par, stosujemy tę regułę uczenia się do k-tego węzła w warstwie wyjściowej:

$\Delta W^{ik} = c * d_k * x_i,$

gdzie ΔW_{ik} jest korektą wagi na i-tym wejściu do k-tego węzła warstwy wyjściowej, d_k jest pożądanym wyjściem z k-tego węzła, a xi jest i-tym elementem X. Stosujemy tę formułę, aby dopasować wszystkie wagi na wszystkich węzły w warstwie wyjściowej. Wektor $<x_1, x_2, .., x_n>$ jest po prostu wektorem wejściowym X, a wektor $<d_1, d_2, .., d_m>$ jest wektorem wyjściowym Y. Zastosowanie wzoru do indywidualnych korekt wagi w całej warstwie wyjściowej i zbierając terminy, możemy zapisać wzór na korektę wagi na warstwie wyjściowej jako:

$\Delta W = c * Y * X,$

gdzie iloczyn wektorowy Y * X jest iloczynem zewnętrznym wektorowym. Zewnętrzny iloczyn wektorowy YX jest ogólnie definiowany jako macierz:

$$YX = \begin{bmatrix} y_1 \bullet x_1 & y_1 \bullet x_2 & \cdots & y_1 \bullet x_m \\ y_2 \bullet x_1 & y_2 \bullet x_2 & \cdots & y_2 \bullet x_m \\ \cdots & \cdots & \cdots & \cdots \\ y_n \bullet x_1 & y_n \bullet x_2 & \cdots & y_n \bullet x_m \end{bmatrix}$$

Aby wytrenować sieć na całym zestawie skojarzonych par, przechodzimy przez te pary, dostosowując wagę każdej pary $<X_i, Y_i>$ zgodnie ze wzorem:

$W^{t+1} = W^t + c * Y_i * X_i.$

Za cały zestaw treningowy otrzymujemy:

$W^1 = W^0 + c\ (Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t)$,

gdzie W^0 to początkowa konfiguracja wagi. Jeśli następnie zainicjujemy W^0 do wektora 0, <0,

0, ..., 0> i ustawiając stałą uczenia c na 1, otrzymujemy następujący wzór na przypisywanie wag sieci:

$W = Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t$.

Sieć, która odwzorowuje wektory wejściowe na wektory wyjściowe przy użyciu tego wzoru do przypisywania wagi, nazywana jest asocjatorem liniowym. Pokazaliśmy, że liniowe sieci asocjacyjne są oparte na zasadzie uczenia się Hebba. W praktyce ten wzór można zastosować bezpośrednio do zainicjowania wag sieci bez jawnego uczenia. Następnie przeanalizujemy właściwości asocjatora liniowego. Ten model, jak już mamy , przechowuje wiele asocjacji w macierzy wektorów wagi. Zwiększa to możliwość interakcji między przechowywanymi wzorcami. W następnych sekcjach przeanalizujemy problemy stworzone przez te interakcje.

11.5.4 Pamięć asocjacyjna i asocjator liniowy

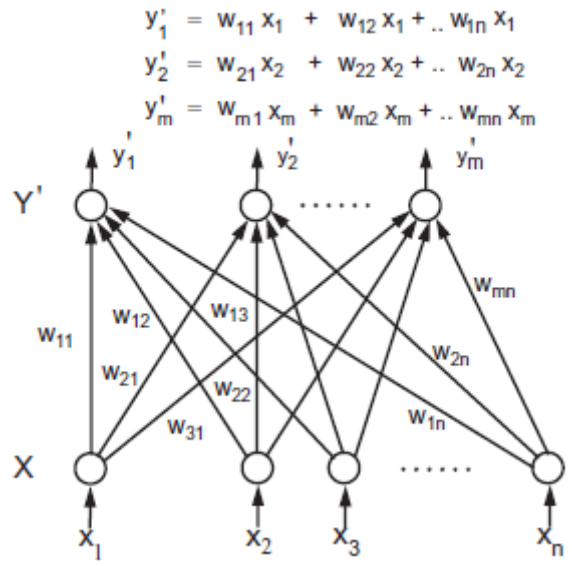
Liniowa sieć asocjacyjna została po raz pierwszy zaproponowana przez Tuevo Kohonena oraz Jamesa Andersona i inni. W tej sekcji przedstawiamy liniową sieć asocjacyjną jako metodę przechowywania i odtwarzania wzorców z pamięci. Badamy różne formy odzyskiwania pamięci, w tym modele heteroasocjacyjne, autoasocjacyjne i interpolacyjne. Analizujemy liniową sieć asocjatorów jako implementację pamięci interpolacyjnej opartej na uczeniu się języka Hebba Kończymy tę sekcję, rozważając problemy z zakłóceniami lub przesłuchami. Ten problem może wystąpić podczas kodowania wielu wzorców w pamięci. Rozpoczynamy badanie pamięci od kilku definicji. Wzorce i wartości pamięci są reprezentowane jako wektory. W redukowaniu reprezentacji problemu do zbioru wektorów cech zawsze występuje błąd indukcyjny. Asocjacje, które mają być przechowywane w pamięci, są reprezentowane jako zbiory par wektorów, $\{<X_1, Y_1>, <X_2, Y_2>, ..., <X_t, Y_t>\}$. Dla każdej pary wektorów $<X_i, Y_i>$, wzór X_i jest kluczem do wyszukiwania wzorca Y_i. Istnieją trzy typy pamięci asocjacyjnych:

1. Heteroasocjatywne: jest to odwzorowanie od X do Y takie, że jeśli dowolny wektor X jest bliżej wektora Xi niż jakikolwiek inny przykład, to zwracany jest powiązany wektor Y_i.

2. Autoasocjatywne: To mapowanie jest takie samo jak heteroasocjatywne, z wyjątkiem tego, że $X_i = Y_i$ dla wszystkich par wzorców. Ponieważ każdy wzorzec Xi jest powiązany z samym sobą, ta forma pamięci jest używana głównie wtedy, gdy częściowy lub zdegradowany wzorzec bodźca służy do przywołania pełnego wzorca.

3. Interpolacja: Jest to odwzorowanie Φ z X na Y takie, że gdy X różni się od przykładu, to znaczy $X = X_i + \Delta_i$, to wynik $\Phi(X) = \Phi(X_i + \Delta_i) = Y_i + E$, gdzie $E = \Phi(\Delta_i)$. Jeśli wektor wejściowy jest jednym z przykładów Xi, to pobierany jest powiązany z nim Yi. Jeśli różni się od przykładu wektorem Δ, to wektor wyjściowy różni się również różnicą wektorów E, gdzie $E = \Phi(\Delta)$.

Pamięci autoasocjacyjne i heteroasocjacyjne służą do wydobycia jednego z oryginalnych wzorców. Stanowią one pamięć w prawdziwym sensie, ponieważ pobrany wzór jest dosłowną kopią przechowywanego wzorca. Możemy również chcieć skonstruować wzór wyjściowy, który różni się od wzorców przechowywanych w pamięci w jakiś systematyczny sposób. To jest funkcja pamięci interpolacyjnej. Liniowa sieć asocjacyjna na rysunku 21 implementuje pewną formę pamięci interpolacyjnej.



Jak pokazano w sekcji 11.5.3, jest on oparty na modelu nauczania języka Hebba. Inicjalizacja wagi sieci jest opisana równaniem wyprowadzonym w sekcji 11.5.3:

$W = Y_1 * X_1 + Y_2 * X_2 + ... + Y_t * X_t$.

Biorąc pod uwagę to przypisanie wagi, sieć pobierze z dokładnym dopasowaniem jeden z przykładów; w przeciwnym razie tworzy mapowanie interpolacyjne. Następnie wprowadzimy kilka pojęć i notacji, aby pomóc nam przeanalizować zachowanie tej sieci. Najpierw chcemy wprowadzić metrykę, która pozwoli nam precyzyjnie określić odległość między wektorami. Wszystkie nasze wektory wzorcowe w przykładach są wektorami Hamminga, czyli wektorami złożonymi tylko z wartości +1 i –1. Do opisania odległości między dwoma wektorami Hamminga używamy odległości Hamminga. Formalnie definiujemy przestrzeń Hamminga:

$H^n = \{X = (x_1, x_2, ..., x_n)\}$, gdzie każdy xi pochodzi ze zbioru {+1, –1}.

Odległość Hamminga jest zdefiniowana dla dowolnych dwóch wektorów z przestrzeni Hamminga jako:

|| X, Y || = liczba składników, o które różnią się X i Y.

Na przykład odległość Hamminga w czterowymiarowej przestrzeni Hamminga między:

(1, –1, –1, 1) i (1, 1, –1, 1) wynosi 1

(–1, –1, –1, 1) i (1, 1, 1, –1) wynosi 4

(1, –1, 1, –1) i (1, –1, 1, –1) wynosi 0.

Potrzebujemy dwóch dalszych definicji. Po pierwsze, dopełnieniem wektora Hamminga jest ten wektor ze zmienionym każdym jego elementem: +1 do –1 i –1 do +1. Na przykład dopełnieniem (1, –1, –1, –1) jest (–1, 1, 1, 1). Po drugie, definiujemy ortonormalność wektorów. Wektory, które są ortonormalne, są ortogonalne lub prostopadłe i mają jednostkę długości. Dwa wektory ortonormalne, po pomnożeniu razem z iloczynem skalarnym, mają wszystkie składniki iloczynu krzyżowego do zera. Zatem w ortonormalnym zbiorze wektorów, gdy dowolne dwa wektory, X_i i X_j, są mnożone, iloczyn wynosi 0, chyba że są one tym samym wektorem:

$X_i X_j = \delta_{ij}$ gdzie $\delta_{ij} = 1$, gdy i = j, a 0 w innym przypadku.

Następnie pokazujemy, że zdefiniowana powyżej liniowa sieć asocjacyjna ma następujące dwie właściwości, przy czym ΦX) reprezentuje funkcję odwzorowania sieci. Po pierwsze, dla wejściowego wzorca X_i, który dokładnie pasuje do jednego z przykładów, wyjście sieciowe, Φ (Xi), to Y_i, powiązany przykład. Po drugie, dla wzorca wejściowego X_k, co niedokładnie pasuje do jednego z przykładów, wyjście sieciowe, $\Phi(X_k)$, to Y_k, czyli liniowa interpolacja X_k. Dokładniej, jeśli $X_k = X_i + \Delta_i$, gdzie X_i jest przykładem, sieć zwraca:

$Y_k = Y_i + E$, gdzie $E = \Phi(\Delta_i)$.

Najpierw pokazujemy, że gdy wejście sieciowe X_i jest jednym z przykładów, sieć zwraca skojarzony wzór. $\Phi(X_i) = WX_i$, zgodnie z definicją funkcji aktywacji sieci. Ponieważ $W = Y_1X_1 + Y_2X_2 + ... + YiX_i + ... + Y_nX_n$, otrzymujemy: $\Phi(X_i) = (Y_1X_1 + Y_2X_2 + ... + YiX_i + ... + Y_nX_n)\ X_i = Y_1X_1X_i + Y_2X_2X_i + ... + Y_iX_iX_i + ... + Y_nX_nX_i$, według podziału. Ponieważ, jak zdefiniowano powyżej, $X_iX_j = \delta_{ij}$:

$$\Phi(X_i) = Y_1\delta_{1i} + Y_2\delta_{2i} + ... + Y_i\delta_{ii} + ... + Y_n\delta_{ni}.$$

Zgodnie z warunkiem ortonormalności, $\delta_{ij} = 1$, gdy i = j i 0 w przeciwnym razie. W ten sposób otrzymujemy:

$$\Phi(X_i) = Y_1*0 + Y_2*0 + ... + Y_i*1 + ... + Y_n*0 = Y_i.$$

Można również wykazać, że dla X_k nie równego żadnemu z przykładów, sieć wykonuje mapowanie interpolacyjne. Oznacza to, że dla $X_k = Xi + \Delta_i$, gdzie X_i jest przykładem,

$\Phi(X_k) = \Phi(X_i + \Delta_i)$

$= Yi + E$,

gdzie Y_i jest wektorem powiązanym z X_i a

$$E = \Phi(\Delta_i) = (Y_1X_1 + Y_2X_2 + ... + Y_nX_n)\ \Delta_i.$$

Pomijamy szczegóły dowodu.

Podamy teraz przykład liniowego przetwarzania asocjacyjnego. Rysunek 22 przedstawia prostą liniową sieć asocjacyjną, która odwzorowuje czteroelementowy wektor X na trzyelementowy wektor Y. Ponieważ pracujemy w przestrzeni Hamminga, funkcja aktywacji sieci f jest wcześniej używaną funkcją znaku. Jeśli chcemy zapisać następujące dwie asocjacje wektorowe $<X_1, Y_1>$, $<X_2, Y_2>$ i:

$X_1 = [1, –1, –1, –1] \leftrightarrow Y1 = [–1, 1, 1]$,

$X_2 = [–1, –1, –1, 1] \leftrightarrow Y2 = [1, –1, 1]$.

Używając wzoru inicjalizacji wagi dla asocjatorów liniowych, z iloczynem wektorów zewnętrznych, jak zdefiniowano w poprzedniej sekcji:

$W = Y_1X_1 + Y_2X_2 + Y_3X_3 + ... + Y_nX_n$,

Możemy teraz obliczyć $Y_1X_1 + Y_2X_2$, macierz wag dla sieci:

$$W = \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 0 & 2 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix}$$

Uruchamiamy asocjatora liniowego na jednym z przykładów. Zaczynamy od X = [1, –1, –1, –1] z pierwszej pary przykładów, aby odzyskać powiązane Y:

y_1 = (–2 * 1) + (0 * –1) + (0 * –1) + (2 * –1) = –4 i sign(–4) = –1,

y_2 = (2 * 1) + (0 * –1) + (0 * –1) + (–2 * –1) = 4 i sign(4) = 1 oraz

y_3 = (0 * 1) + (–2 * –1) + (–2 * –1) + (0 * –1) = 4 i sign (4) = 1.

Zatem Y_1 = [–1, 1, 1], druga połowa pary wzorców, jest zwracana.

Następnie pokażemy przykład liniowej interpolacji wzorca. Rozważ wektor X

[1, –1, 1, –1]:

y_1 = (–2 * 1) + (0 * –1) + (0 * 1) + (2 * –1) = –4 i sign(–4) = –1,

y_2 = (2 * 1) + (0 * –1) + (0 * 1) + (–2 * –1) = 4 i sign(4) = 1 oraz

y_3 = (0 * 1) + (–2 * –1) + (–2 * 1) + (0 * –1) = 0 i sign(0) = 1.

Zauważ, że Y = [–1, 1, 1] nie jest jednym z oryginalnych przykładów Y. Zauważ, że mapowanie zachowuje wartości, które mają wspólne dwa przykłady Y. W rzeczywistości [1, –1, 1, –1] wektor X ma odległość Hamminga równą 1 od każdego z dwóch przykładów X; wektor wyjściowy [–1, 1, 1] również ma odległość Hamminga równą 1 od każdego z pozostałych przykładów Y. Podsumowujemy kilkoma obserwacjami dotyczącymi asocjatorów liniowych. Pożądane właściwości asocjatora liniowego zależą od wymogu, aby przykładowe wzorce zawierały zbiór ortonormalny. To ogranicza jego praktyczność na dwa sposoby. Po pierwsze, może nie być oczywistego odwzorowania sytuacji na świecie na ortonormalne wzorce wektorowe. Po drugie, liczba wzorów, które można przechowywać, jest ograniczona wymiarowością przestrzeni wektorowej. W przypadku naruszenia wymogu ortonormalności dochodzi do interferencji między zapisanymi wzorcami, powodując zjawisko zwane przesłuchami. Zwróć również uwagę, że asocjator liniowy pobiera skojarzony przykład Y tylko wtedy, gdy wektor wejściowy dokładnie pasuje do przykładu X. Jeśli nie ma dokładnego dopasowania we wzorcu wejściowym, wynikiem jest mapowanie interpolacyjne. Można argumentować, że interpolacja nie jest pamięcią w prawdziwym tego słowa znaczeniu. Często chcemy zaimplementować prawdziwą funkcję pobierania pamięci, w której przybliżenie do przykładu pobiera dokładny wzorzec, który jest z nim powiązany. Potrzebny jest basen przyciągający do przechwytywania wektorów w pobliskim regionie. W następnej sekcji przedstawiamy atraktorową wersję liniowej sieci asocjacyjnej.

11.6 Sieci przyciągające lub „wspomnienia”

11.6.1 Wprowadzenie

Sieci omówione do tej pory są z wyprzedzeniem. W sieciach z wyprzedzeniem informacje są prezentowane zestawowi węzłów wejściowych, a sygnał przechodzi do przodu przez węzły lub warstwy węzłów, aż pojawi się jakiś wynik. Inną ważną klasą sieci łącznikowych są sieci sprzężenia zwrotnego. Architektura tych sieci różni się tym, że sygnał wyjściowy węzła może być skierowany z powrotem, bezpośrednio lub pośrednio, jako wejście do tego węzła. Sieci sprzężenia zwrotnego różnią się od sieci sprzężenia zwrotnego na kilka ważnych sposobów:

1. obecność połączeń zwrotnych między węzłami,

2. opóźnienie czasowe, tj. Bezzwłoczna propagacja sygnału,

3. wyjściem z sieci jest stan sieci w momencie konwergencji,

4. użyteczność sieci zależy od właściwości konwergencji.

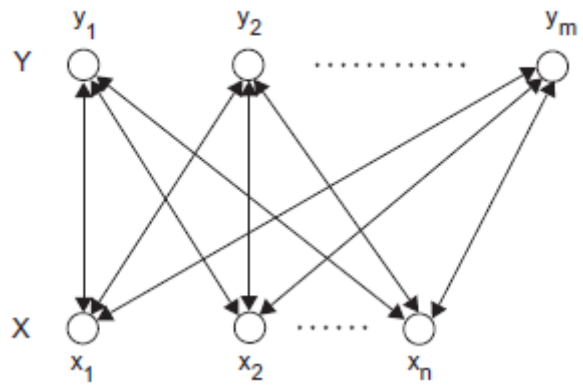
Kiedy sieć sprzężenia zwrotnego osiąga czas, w którym już się nie zmienia, mówi się, że jest w stanie równowagi. Stan, w którym sieć osiąga równowagę, jest uważany za wyjście sieci. W sieciach sprzężenia zwrotnego opisanego w sekcji11.6.2 stan sieci jest inicjowany za pomocą wzorca wejściowego. Sieć przetwarza ten wzór, przechodząc przez serię stanów, aż do osiągnięcia równowagi. Stan sieci w stanie równowagi to wzorzec pobrany z pamięci. W sekcji 11.6.3 rozważymy sieci, które implementują pamięć heteroasocjacyjną, a w sekji 11.6.4 pamięć autosocjatywną. Poznawcze aspekty tych wspomnień są zarówno interesujące, jak i ważne. Oferują nam model pamięci adresowalnej treści. Ten typ asocjatora może opisywać odzyskanie numeru telefonu, uczucie smutku ze starej pamięci, a nawet rozpoznanie osoby z częściowego widoku twarzy. Badacze podjęli próbę uchwycenia wielu asocjacyjnych aspektów tego typu pamięci w strukturach danych opartych na symbolach, w tym w sieciach semantycznych, ramkach i systemach obiektowych, jak pokazano w Części 6. Atraktor jest definiowany jako stan, do którego sąsiedni region ewoluuje w czasie. Każdy atraktor w sieci będzie miał region, w którym każdy stan sieci w tym regionie ewoluuje w kierunku tego atraktora. Region ten nazywany jest jego dorzeczem. Atraktor może składać się z pojedynczego stanu sieci lub szeregu stanów, przez które przechodzi sieć. Próby matematycznego zrozumienia atraktorów i ich basenów doprowadziły do powstania koncepcji funkcji energii sieci. Sieci sprzężenia zwrotnego z funkcją energetyczną, która ma tę właściwość, że każda zmiana sieci zmniejsza całkowitą energię sieci, co gwarantuje zbieżność. Opisujemy te sieci w sekcji 11.6.3. Sieci atraktorów można wykorzystać do implementacji pamięci adresowalnych treści poprzez zainstalowanie w pamięci pożądanych wzorców jako atraktorów. Mogą być również używane do rozwiązywania problemów optymalizacyjnych, takich jak problem podróżującego sprzedawcy, poprzez tworzenie mapowania między funkcją kosztu w problemie optymalizacji a energią sieciową. Rozwiązanie problemu polega więc na zmniejszeniu całkowitej energii sieci. Tego typu rozwiązywanie problemów odbywa się za pomocą tak zwanej sieci Hopfield.

11.6.2 BAM, dwukierunkowa pamięć asocjacyjna

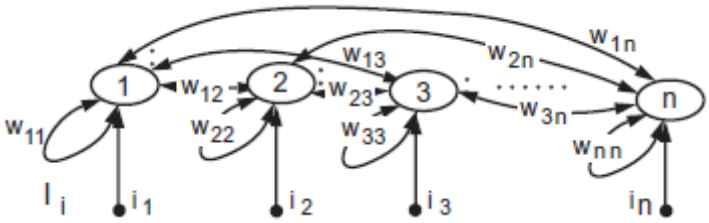
Sieć BAM, opisana po raz pierwszy przez Barta Kosko (1988), składa się z dwóch w pełni połączonych ze sobą warstw elementów przetwarzających. Może również istnieć łącze zwrotne łączące każdy węzeł ze sobą. Odwzorowanie BAM n-wymiarowego wektora wejściowego X_n na m-wymiarowy wektor wyjściowy Y_m przedstawiono na rysunku 22. Ponieważ każde łącze od X do Y jest dwukierunkowe, będą przypisane wagi związane z przepływem informacji w każdym kierunku. Podobnie jak wagi asocjatora liniowego, wagi w sieci BAM można obliczyć z wyprzedzeniem. W rzeczywistości używamy tej samej metody do obliczania wag sieci, jak w przypadku asocjatora liniowego. Wektory dla architektury BAM są pobierane z zestawu wektorów Hamminga. Biorąc pod uwagę N par wektorów, które tworzą zbiór przykładów, które chcemy przechowywać, budujemy macierz, tak jak to zrobiliśmy w sekcji 11.5.4:

$$W = Y_1 * X_1 + Y_2 * X_2 + \ldots + Y_t * X_t.$$

To równanie podaje wagi połączeń między warstwą X a warstwą Y, jak widać na rysunku 23.



Na przykład w_{32} jest ciężarem na połączeniu od drugiej jednostki na warstwie X do trzeciej jednostki na warstwie Y. Zakładamy, że dowolne dwa węzły mają między sobą tylko jedną ścieżkę. Dlatego wagi łączące węzły na warstwach X i Y są identyczne w obu kierunkach. Zatem macierz wag od Y do X jest transpozycją macierzy wag W. Sieć BAM można przekształcić w sieć autosocjatywną, używając tego samego wzoru inicjalizacji wagi na zbiorze asocjacji $<X_1, X_1>$, $<X_2, X_2>$, ... Ponieważ warstwy X i Y powstałe w wyniku tej procedury są identyczne, możemy wyeliminować warstwę Y, czego wynikiem jest sieć wyglądająca jak na rysunku 24. Przyjrzymy się przykładowi sieci autoasocjacyjnej w sekcji 11.6.4.



Sieć BAM jest używana do pobierania wzorców z pamięci poprzez inicjalizację warstwy X za pomocą wzorca wejściowego. Jeśli wzorzec wejściowy jest hałaśliwą lub niekompletną wersją jednego z przykładów, BAM może często uzupełnić wzorzec i pobrać skojarzony wzorzec. Aby przywołać dane za pomocą BAM, wykonujemy następujące czynności:

1. Zastosuj początkową parę wektorów (X, Y) do przetwarzanych elementów. X to wzór, dla którego chcemy pobrać przykład. Y jest inicjalizowany losowo.

2. Przenieś informacje z warstwy X do warstwy Y i zaktualizuj wartości w warstwie Y.

3. Wyślij zaktualizowane informacje Y z powrotem do warstwy X, aktualizując jednostki X.

4. Kontynuuj poprzednie dwa kroki, aż wektory się ustabilizują, czyli do momentu, gdy nie będzie już żadnych zmian wartości wektorów X i Y.

Przedstawiony właśnie algorytm zapewnia BAM przepływ informacji zwrotnej, dwukierunkowy ruch w kierunku równowagi. Poprzedni zestaw instrukcji mógł zaczynać się wzorem na poziomie Y, prowadzącym, po zbieżności, do wyboru przykładu wektora X. Jest w pełni dwukierunkowy: możemy wziąć wektor X jako dane wejściowe i uzyskać asocjację Y na zbieżności lub możemy wziąć wektor Y jako dane wejściowe i odzyskać asocjację X. Zobaczymy rozwiązanie tych problemów na przykładzie w następnej sekcji. Po osiągnięciu konwergencji, ostateczny stan równowagi zwraca jeden z przykładów użytych do zbudowania oryginalnej macierzy wag. Jeśli wszystko pójdzie zgodnie z oczekiwaniami,

pobieramy wektor o znanych właściwościach, identycznych lub nieco różniących się od jednej z przykładowych par wektorów. Używamy tego wektora do pobrania drugiego wektora z pary wzorców. Odległość jest odległością Hamminga mierzoną przez składowe porównania wektorów, licząc jeden dla każdej różnicy elementów. Ze względu na ograniczenia ortonormalności, gdy BAM zbiega się dla wektora, zbiega się również dla jego dopełnienia. Zatem zauważamy, że dopełnienie wektora również staje się atraktorem. Przykład tego podajemy w następnej sekcji. Jest kilka rzeczy, które mogą zakłócać konwergencję BAM. Jeśli zbyt wiele przykładów jest odwzorowanych na macierz wag, same przykłady mogą być zbyt blisko siebie i wytwarzać pseudostabilności w sieci. Zjawisko to nazywane jest przesłuchami i występuje jako lokalne minima w przestrzeni energetycznej sieci. Następnie rozważymy pokrótce przetwarzanie BAM. Mnożenie wektora wejściowego przez macierz wag oblicza sumy par produktów wektorów wektorów dla każdego elementu wektora wyjściowego. Prosta funkcja progowania tłumaczy następnie wynikowy wektor z powrotem do wektora w przestrzeni Hamminga. A zatem:

net (Y) = WX lub dla każdego składnika Y_i, net $(Y_i) = \Sigma w_{ij} * x_j$,

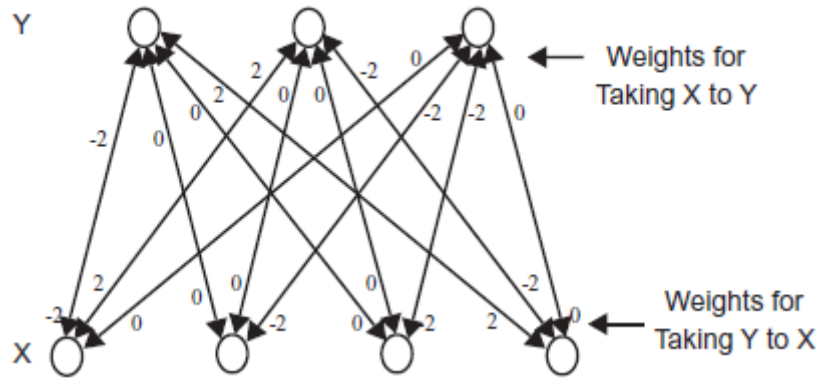
z podobnymi relacjami dla warstwy X. Funkcja progowa f dla netto (Y) w czasie t +1 jest również prosta:

$$f(net^{t+1}) = \begin{cases} +1 & \text{if net} > 0 \\ f(net^t) & \text{if net} = 0 \\ -1 & \text{if net} < 0 \end{cases}$$

W następnej sekcji zilustrujemy to dwukierunkowe asocjacyjne przetwarzanie pamięci kilkoma przykładami.

11.6.3 Przykłady przetwarzania BAM

Rysunek 25 przedstawia małą sieć BAM, prostą odmianę liniowego asocjatora przedstawioną w sekcji 11.5.4.



Ta sieć odwzorowuje czteroelementowy wektor X na trzyelementowy wektor Y i odwrotnie. Załóżmy, że chcemy stworzyć dwa przykłady par wektorów:

$x_1 = [1, -1, -1, -1] \leftrightarrow y_1 = [1, 1, 1]$ i

$x_2 = [-1, -1, -1, 1] \leftrightarrow y_2 = [1, -1, 1]$.

Teraz tworzymy macierz wag za pomocą wzoru przedstawionego w poprzedniej sekcji:

$$W = Y_1X_1^t + Y_2X_2^t + Y_3X_3^t + \ldots + Y_NX_N^t$$

$$W = \begin{bmatrix} 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & -2 & 0 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix}$$

Wektorem wagi dla mapowania z Y na X jest transpozycja W, lub:

$$\begin{bmatrix} 0 & 2 & 0 \\ -2 & 0 & -2 \\ -2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix}$$

Wybieramy teraz kilka wektorów i testujemy asocjator BAM. Zacznijmy od przykładowego powietrza, wybierając składową X i sprawdzając, czy otrzymamy Y. Niech X = [1, –1, –1, –1]:

Y_1 = (1 * 0) + (–1 * –2) + (–1 * –2) + (0 * –1) = 4, if (4) = 1,

Y_2 = (1 * 2) + (–1 * 0) + (–1 * 0) + (–1 * –2) = 4, if (4) = 1 oraz

Y_3 = (1 * 0) + (–1 * –2) + (–1 * –2) + (–1 * 0) = 4, if (4) = 1.

W ten sposób zwracana jest druga połowa przykładowej pary. Czytelnik może uczynić ten wektor Y wektorem wejściowym i sprawdzić, czy zwracany jest oryginalny wektor X [1, –1, –1, –1]. W naszym następnym przykładzie rozważmy wektor X [1, 1, 1, –1] z losowo zainicjowanym Y. Mapujemy X w naszej sieci BAM:

Y_1 = (1 * 0) + (1 * –2) + (1 * –2) + (–1 * 0) = –4, if (4) = –1,

Y_2 = (1 * 2) + (1 * 0) + (1 * 0) + (–1 * –2) = 4, if (4) = 1,

Y_3 = (1 * 0) + (1 * –2) + (1 * –2) + (–1 * 0) = –4, if (4) = –1.

Ten wynik, z funkcją progowania f zastosowaną do [–4, 4, –4], wynosi [–1, 1, –1]. Mapowanie z powrotem do X daje:

X_1 = (–1 * 0) + (1 * 2) + (–1 * 0) = 2,

X_2 = (–1 * –2) + (1 * 0) + (–1 * –2) = 4,

X_3 = (–1 * –2) + (1 * 0) + (–1 * –2) = 4,

X_4 = (–1 * 0) + (1 * –2) + (–1 * 0) = –2.

Zastosowana funkcja progowa, znowu jak powyżej, daje oryginalny wektor [1, 1, 1, –1]. Ponieważ wektor początkowy dał stabilny wynik przy pierwszej translacji, możemy pomyśleć, że właśnie odkryliśmy kolejną prototypową parę przykładową. W rzeczywistości wybrany przez nas przykład jest uzupełnieniem oryginalnego przykładu wektora <X_2, Y_2>! Okazuje się, że w sieci BAM, gdy para wektorów ustala się jako przykładowy prototyp, tak samo jest z jej uzupełnieniem. Dlatego nasza sieć BAM obejmuje jeszcze dwa prototypy:

X_3 = [–1, 1, 1, 1] ↔Y3 = [–1, –1, –1] i

$X_4 = [1, 1, 1, -1] \leftrightarrow Y4 = [-1, 1, -1]$.

Wybierzmy następnie wektor w pobliżu przykładu X, [1, –1, 1, –1]. Zauważ, że odległość Hamminga od najbliższego z czterech przykładów X wynosi 1. Następnie losowo inicjalizujemy wektor

Y do [–1, –1, –1]:

$Y_1^{t+1} = (1 * 0) + (-1 * -2) + (1 * -2) + (-1 * 0) = 0,$

$Y_2^{t+1} = (1 * 2) + (-1 * 0) + (1 * 0) + (-1 * -2) = 4,$

$Y_3^{t+1} = (1 * 0) + (-1 * -2) + (1 * -2) + (-1 * 0) = 0.$

Ocena funkcji netto $f(Y_i^{t+1}) = f(Y_i^t)$ kiedy $y_i^{t+1} = 0$, z równania progu na końcu sekcji 11.6.2. Zatem Y wynosi [–1, 1, –1] ze względu na losową inicjalizację pierwszego i trzeciego parametru Y^T na –1. Teraz przenosimy Y z powrotem do X:

$X_1 = (-1 * 0) + (1 * 2) + (-1 * 0) = 2,$

$X_2 = (-1 * -2) + (1 * 0) + (-1 * -2) = 4,$

$X_3 = (-1 * -2) + (1 * 0) + (-1 * -2) = 4,$

$X_4 = (-1 * 0) + (1 * -2) + (-1 * 0) = -2.$

Funkcja progowa odwzorowuje ten wynik na wektor X = [1, 1, 1, –1]. Powtarzamy proces, przenosząc ten wektor z powrotem do Y:

$Y_1 = (1 * 0) + (1 * -2) + (1 * -2) + (-1 * 0) = -4,$

$Y_2 = (1 * 2) + (1 * 0) + (1 * 0) + (-1 * -2) = 4,$

$Y_3 = (1 * 0) + (1 * -2) + (1 * -2) + (-1 * 0) = -4.$

Funkcja progowa zastosowana do [–4, 4, –4] ponownie daje Y = [–1, 1, –1]. Ten wektor jest identyczny z najnowszą wersją Y, więc sieć jest stabilna. To pokazuje, że po dwóch przejściach przez sieć BAM, wzór, który był bliski X_4, zbiegał się z zapisanym wzorcem. Byłoby to podobne do rozpoznawania twarzy lub innego zapisanego obrazu z brakiem lub zasłonięciem części informacji. Odległość Hamminga między pierwotnym wektorem X [1, –1, 1, –1] a prototypem X_4 [1, 1, 1, –1] wynosiła 1. Wektor osadzony w parze wzorców $<X_4, Y_4>$. W naszych przykładach BAM rozpoczęliśmy przetwarzanie z elementem X pary przykładów. Oczywiście mogliśmy zaprojektować przykłady z wektora Y, inicjując X w razie potrzeby. Hecht-Nielsen przedstawia interesującą analizę sieci BAM. Pokazuje, że właściwość ortonormalna dla obsługi liniowej sieci asocjacyjnej dla BAM jest zbyt restrykcyjna. Podaje argument wskazujący, że warunkiem budowy sieci jest to, aby wektory były liniowo niezależne, to znaczy, że żaden wektor nie może być utworzony z liniowej kombinacji innych wektorów w przestrzeni wzorców.

11.6.4 Pamięć autosocjacyjna i sieci Hopfielda

Badania Johna Hopfielda, fizyka z California Institute of Technology, są głównym powodem, dla którego architektury koneksjonistyczne są obecnie wiarygodne. Badał właściwości konwergencji sieci, wykorzystując koncepcję minimalizacji energii. Zaprojektował również rodzinę sieci opartych na tych zasadach. Jako fizyk Hopfield rozumiał stabilność zjawisk fizycznych jako punkty minimalizacji energii układu fizycznego. Przykładem takiego podejścia jest analiza symulowanego wyżarzania chłodzenia metali. Przyjrzyjmy się najpierw podstawowym cechom sieci asocjacyjnych ze sprzężeniem zwrotnym. Sieci te zaczynają się od stanu początkowego składającego się z wektora wejściowego. Następnie sieć

przetwarza ten sygnał poprzez ścieżki sprzężenia zwrotnego, aż osiągnie stabilny stan. Aby użyć tej architektury jako pamięci asocjacyjnej, chcielibyśmy, aby sieć miała dwie właściwości.

Po pierwsze, zaczynając od dowolnego stanu początkowego, chcielibyśmy mieć gwarancję, że sieć zbiegnie się w jakimś stabilnym stanie. Po drugie, chcielibyśmy, aby ten stan stabilny był najbliższy stanowi wejściowemu według jakiejś miary odległości. Najpierw przyjrzymy się sieci autoasocjacyjnej zbudowanej na tych samych zasadach, co sieć BAM. W poprzedniej sekcji zauważyliśmy, że sieci BAM można przekształcić w sieci autosocjacyjne, używając identycznych wektorów w pozycjach X i Y. Wynikiem tej transformacji, jak zobaczymy dalej, jest symetryczna macierz wag kwadratowych. Rysunek 23 w sekcji 11.6.2 zawiera przykład.

Macierz wag dla sieci autoasocjacyjnej, która przechowuje zbiór wzorców wektorów $\{X_1, X_2, ..., X_n\}$ jest tworzona przez:

$W = \Sigma X_i X_i^t$ dla i = 1, 2, ..., n.

Kiedy tworzymy pamięć autoasocjatywną z heteroasocjacji, waga od węzła x_i do x_j będzie identyczna jak ta z x_j do x_i, więc macierz wag będzie symetryczna. To założenie wymaga jedynie, aby dwa elementy przetwarzające były połączone jedną ścieżką mającą jeden ciężar. Możemy również mieć szczególny przypadek, ponownie z prawdopodobieństwem neuronowym, że żaden węzeł sieci nie jest bezpośrednio powiązany ze sobą, to znaczy nie ma połączeń x_i do x_i. W tej sytuacji główna przekątna macierzy wag wij, gdzie i = j, to same zera. Podobnie jak w przypadku BAM, obliczamy macierz wag na podstawie wzorców, które mają być zapisane w pamięci. Wyjaśnimy to na prostym przykładzie. Rozważmy zestaw przykładowych trzech wektorów:

$X_1 = [1, –1, 1, –1, 1]$,

$X_2 = [–1, 1, 1, –1, –1]$,

$X_3 = [1, 1, –1, 1, 1]$.

Następnie obliczamy macierz wag, używając $W = \Sigma X_i X_i^t$ dla i = 1, 2, 3:

$$W = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 & -1 & -1 & 1 & 3 \\ -1 & 3 & -1 & 1 & -1 \\ -1 & -1 & 3 & -3 & -1 \\ 1 & 1 & -3 & 3 & 1 \\ 3 & -1 & -1 & 1 & 3 \end{bmatrix}$$

Używamy funkcji progowania:

$$f(net^{t+1}) = \begin{cases} +1 & \text{if } net > 0 \\ f(net^t) & \text{if } net = 0 \\ -1 & \text{if } net < 0 \end{cases}$$

Najpierw testujemy sieć na przykładzie X_3 = [1, 1, –1, 1, 1] i otrzymujemy:

X_3 * W = [7, 3, –9, 9, 7],

oraz z funkcją progową, [1, 1, –1, 1, 1]. Widzimy, że ten wektor natychmiast się stabilizuje. To pokazuje, że same przykłady są stanami stabilnymi lub atraktorami. Następnie testujemy wektor, który jest odległością Hamminga 1 od przykładowego X3. Sieć powinna zwrócić ten przykład. Jest to równoważne z odzyskaniem wzorca pamięci z częściowo zdegradowanych danych. Wybieramy X = [1, 1, 1, 1, 1]:

X * W = [5, 1, –3, 3, 5].

Użycie funkcji progowej daje wektor X3 [1, –1, –1, 1, 1].

Następnie bierzemy trzeci przykład, tym razem wektor, którego odległość Hamminga jest oddalona o 2 od najbliższego prototypu, niech X = [1, –1, –1, 1, –1]. Można sprawdzić, że ten wektor jest oddalony o 2 od X_3, o 3 od X_1 i o 4 od X_2. Zaczynamy:

X * W = [3, –1, –5, 5, 3], co przy progu daje [1, –1, –1, 1, 1].

To nie wydaje się niczego przypominać, ani nie jest punktem stabilności, ponieważ:

[1, –1, –1, 1, 1] * W = [9, –3, –7, 7, 9], czyli [1, –1, –1, 1, 1].

Sieć jest teraz stabilna, ale nie ma jednej z oryginalnie zapisanych pamięci! Czy znaleźliśmy inne minimum energetyczne? Przy bliższym przyjrzeniu się zauważymy, że ten nowy wektor jest uzupełnieniem oryginalnego przykładu X_2 [–1, 1, 1, –1, –1]. Ponownie, podobnie jak w przypadku heteroasocjatywnej sieci BAM, nasza sieć autoasocjacyjna tworzy atraktory dla oryginalnych wzorców, jak również dla ich uzupełnień, w tym przypadku będziemy mieli w sumie sześć atraktorów. Do tego momentu w naszej prezentacji przyjrzeliśmy się sieciom autoasocjacyjnym opartym na liniowym modelu asocjacyjnym pamięci. Jednym z celów Johna Hopfielda było przedstawienie bardziej ogólnej teorii sieci autosocjatywnych, która miałaby zastosowanie do każdej jednowarstwowej sieci sprzężenia zwrotnego spełniającej pewien zestaw prostych ograniczeń. W przypadku tej klasy jednowarstwowych sieci sprzężenia zwrotnego Hopfield udowodnił, że zawsze będzie istniała funkcja sieciowa zapewniająca konwergencję. Kolejnym celem Hopfielda było zastąpienie modelu aktualizacji czasu dyskretnego, używanego wcześniej, modelem, który bardziej przypomina ciągłe przetwarzanie czasu rzeczywistych neuronów. Powszechnym sposobem symulacji ciągłej asynchronicznej aktualizacji w sieciach Hopfielda jest aktualizacja węzłów indywidualnie, a nie jako warstwa. Odbywa się to za pomocą procedury losowego wyboru w celu wybrania następnego no de do aktualizacji, przy jednoczesnym zastosowaniu metody zapewniającej, że średnio wszystkie węzły w sieci będą aktualizowane równie często. Struktura sieci Hopfielda jest identyczna jak powyżej sieci autosocjacyjnej: pojedyncza warstwa węzłów całkowicie połączonych. Aktywacja i progowanie również działają jak poprzednio. Dla węzła i,

$$x_i^{new} = \begin{cases} +1 & \text{if} \sum_j w_{ij} x_j^{old} > T_i, \\ x_i^{old} & \text{if} \sum_j w_{ij} x_j^{old} = T_i, \\ -1 & \text{if} \sum_j w_{ij} x_j^{old} < T_i, \end{cases}$$

Biorąc pod uwagę tę architekturę, do scharakteryzowania sieci Hopfielda wymagane jest tylko jedno dodatkowe ograniczenie. Jeśli wij jest wagą połączenia do węzła i z węzła j, definiujemy sieć Hopfielda jako taką, która przestrzega ograniczeń wagi:

$w_{ii} = 0$ dla wszystkich i,

$w_{ij} = w_{ji}$ dla wszystkich i, j.

Sieć Hopfield zazwyczaj nie ma powiązanej metody uczenia się. Podobnie jak BAM, jego wagi są zwykle obliczane z góry. Zachowanie sieci Hopfielda jest teraz lepiej poznane niż jakiejkolwiek innej klasy sieci z wyjątkiem perceptronów. Dzieje się tak, ponieważ jego zachowanie można scharakteryzować za pomocą zwięzłej funkcji energetycznej odkrytej przez Hopfielda:

$$H(X) = -\sum_i \sum_j w_{ij} x_i x_j + 2\sum_i T_i x_i$$

Pokażemy teraz, że ta funkcja energetyczna ma tę właściwość, że każda zmiana sieci zmniejsza całkowitą energię sieci. Biorąc pod uwagę fakt, że H ma z góry określone minimum i że za każdym razem, gdy H zmniejsza się, zmniejsza się o co najmniej ustaloną wartość minimalną, możemy wywnioskować, że z dowolnego stanu sieć jest zbieżna. Najpierw pokazujemy, że dla dowolnego elementu przetwarzającego k, który jest ostatnio aktualizowany, stan k zmienia się wtedy i tylko wtedy, gdy H maleje. Zmiana energii ΔH wynosi:

$$\Delta H = H(X^{new}) - H(X^{old}).$$

Rozszerzając to równanie za pomocą definicji H, otrzymujemy:

$$\Delta H = -\sum_i \sum_j w_{ij} x_i^{new} x_j^{new} - 2\sum_i T_i x_i^{new} + \sum_i \sum_j w_{ij} x_i^{old} x_j^{old} + 2\sum_i T_i x_i^{old}$$

Ponieważ x_k zmieniło się $x_i^{new} = x_i^{old}k$, dla i nie równe k. Oznacza to, że warunki sumy niezawierającej x_k znoszą się wzajemnie. Zmiana kolejności i zbieranie terminów:

$$\Delta H = -2x_k^{new}\sum_j w_{kj} x_j^{new} + 2T_k x_k^{new} + 2x_k^{old}\sum_j w_{kj} x_j^{old} - 2T_k x_k^{old}.$$

Korzystając z faktu, że $w_{ii} = 0$ i $w_{ij} = w_{ji}$ możemy ostatecznie przepisać to jako:

$$\Delta H = 2(x_k^{old} - x_k^{new})\left[\sum_j w_{kj} x_j^{old} - T_k\right].$$

Aby pokazać, że ΔH jest ujemne, rozważymy dwa przypadki. Najpierw załóżmy, że x_k zmieniło się z -1 na +1. Wtedy wyraz w nawiasach kwadratowych musiał być dodatni, aby dać x_k^{new} +1. Ponieważ x_k^{old} - x_k^{new} jest równe -2, ΔH musi być ujemne. Załóżmy, że x_k zmieniło się z 1 na -1. Zgodnie z tym samym rozumowaniem, ΔH ponownie musi być ujemne. Jeśli x_k nie zmienił stanu, x_k^{old} - x_k^{new} = 0 i ΔH = 0.

Biorąc pod uwagę ten wynik, z dowolnego stanu początkowego sieć musi się zbiegać. Ponadto stan sieci w zakresie konwergencji musi być lokalnym minimum energetycznym. Gdyby tak nie było, istniałoby przejście, które jeszcze bardziej zmniejszyłoby całkowitą energię sieci, a algorytm wyboru aktualizacji ostatecznie wybrałby ten węzeł do aktualizacji. Pokazaliśmy teraz, że sieci Hopfielda mają

jedną z dwóch właściwości, których potrzebujemy w sieci implementującej pamięć asocjacyjną. Można jednak wykazać, że sieci Hopfielda na ogół nie mają drugiej pożądanej właściwości: nie zawsze zbiegają się w stanie stabilnym najbliższym stanowi początkowemu. Nie ma ogólnie znanej metody rozwiązania tego problemu. Sieci Hopfielda można również zastosować do rozwiązywania problemów optymalizacyjnych, takich jak problem wędrownego sprzedawcy. W tym celu projektant musi znaleźć sposób na odwzorowanie funkcji kosztu problemu na funkcję energii Hopfielda. Przechodząc do minimum energetycznego, sieć będzie wtedy również minimalizować koszt w odniesieniu do danego stanu problemu. Chociaż takie odwzorowanie zostało znalezione dla niektórych interesujących problemów, w tym problemu podróżującego sprzedawcy, ogólnie to mapowanie ze stanów problemowych na stany energetyczne jest bardzo trudne do odkrycia. W tej sekcji przedstawiliśmy heteroasocjatywne i autoasocjatywne sieci sprzężenia zwrotnego. Przeanalizowaliśmy dynamiczne właściwości tych sieci i przedstawiliśmy bardzo proste przykłady pokazujące ewolucję tych systemów w kierunku ich atraktorów. Pokazaliśmy, jak liniową sieć asocjacyjną można zmienić w sieć atraktorów zwaną BAM. W naszej dyskusji o sieciach Hopfielda w czasie ciągłym widzieliśmy, jak zachowanie sieci można opisać w kategoriach funkcji energii. Klasa sieci Hopfielda gwarantuje konwergencję, ponieważ można wykazać, że każda zmiana sieci zmniejsza całkowitą energię sieci. Nadal istnieją pewne problemy z podejściem opartym na energii do sieci koneksjonistycznych. Po pierwsze, osiągnięty stan energetyczny nie musi być globalnym minimum systemu. Po drugie, sieci Hopfielda nie muszą zbiegać się do atraktora znajdującego się najbliżej wektora wejściowego. To sprawia, że nie nadają się do implementacji pamięci adresowalnych treści. Po trzecie, używając sieci Hopfielda do optymalizacji, nie ma ogólnej metody tworzenia mapowania ograniczeń na funkcję energii Hopfielda. Wreszcie istnieje ograniczenie całkowitej liczby minimów energii, które można przechowywać i pobierać z sieci, a co ważniejsze, tej liczby nie można precyzyjnie ustawić. Testy empiryczne tych sieci pokazują, że liczba atraktorów to niewielki ułamek liczby węzłów w sieci. Te i inne tematy są bieżącymi problemami badawczymi (Hecht-Nielsen 1990, Zurada 1992, Freeman i Skapura 1991). Podejścia oparte na biologii, takie jak algorytmy genetyczne i automaty komórkowe, próbują naśladować proces uczenia się związany z ewolucją form życia. Przetwarzanie w tych modelach jest również równoległe i rozproszone. Na przykład w modelu algorytmu genetycznego populacja wzorców reprezentuje potencjalne rozwiązania problemu. Wraz z cyklem algorytmu ta populacja wzorców „ewoluuje” poprzez operacje naśladujące reprodukcję, mutację i dobór naturalny.

Sztuczna inteligencja : Uczenie maszynowe : Genetyczne i awaryjne

(XII / XVI)

ĆWICZENIA

1. Algorytm genetyczny ma wspierać poszukiwanie różnorodności genetycznej wraz z przetrwaniem ważnych umiejętności (reprezentowanych przez wzorce genetyczne) dla domeny problemowej. Opisz, w jaki sposób różne operatory genetyczne mogą jednocześnie wspierać oba te cele.

2. Omów problem projektowania reprezentacji dla operatorów genetycznych w celu poszukiwania rozwiązań w różnych dziedzinach? Jaka jest tutaj rola błędu indukcyjnego?

3. Rozważ problem satysfakcji CNF w sekcji 12.1.3. W jaki sposób rola liczby dysjunkcji w wyrażeniu CNF wpływa na przestrzeń rozwiązań? Rozważ inne możliwe reprezentacje i operatory genetyczne dla problemu satysfakcji CNF. Czy możesz zaprojektować inny środek fitness?

4. Zbuduj algorytm genetyczny w języku do rozwiązania problemu satysfakcji CNF.

5. Rozważmy problem podróżującego sprzedawcy w Sekcji 12.1.3. Omów problem doboru odpowiedniej reprezentacji dla tego problemu. Zaprojektuj inne odpowiednie operatory genetyczne i środki przystosowania do tego problemu.

6. Zbuduj algorytm genetyczny do poszukiwania rozwiązania problemu podróżującego sprzedawcy.

7. Omów rolę technik, takich jak szare kodowanie, w kształtowaniu przestrzeni poszukiwań algorytmu genetycznego. Opisz dwa inne obszary, w których podobne techniki mogą być ważne.

8. Przeczytaj twierdzenie o schemacie Hollanda. W jaki sposób teoria schematów Hollanda opisuje ewolucję przestrzeni rozwiązań GA? Co to ma do powiedzenia na temat problemów, które nie są zakodowane jako ciągi bitów?

9. Jak ma się algorytm Bucket Brigade do algorytmu wstecznej propagacji?

10. Napisz program do rozwiązania problemu trzeciego prawa ruchu Keplera, opisanego ze wstępnym opisem przedstawionym w sekcji 12.2.2.

11. Omów ograniczenia (przedstawione w rozdziale 12.2.2) w stosowaniu technik programowania genetycznego do rozwiązywania problemów. Na przykład, które elementy rozwiązania nie mogą ewoluować w ramach paradygmatu programowania genetycznego?

12. Przeczytaj wczesną dyskusję na temat Gry w życie w kolumnie Gardnera w Scientific American . Omów inne konstrukcje ratunkowe, podobne do szybowca, w sekcji 12.3.1.

13. Napisz program a-life, który implementuje funkcjonalność przedstawioną na rysunkach 10-13.

14. Obszar badań agentowych został wprowadzony w sekcji 12.3. Zalecamy dalsze zapoznanie się z każdym z wymienionych projektów, a zwłaszcza badaniami Brooksa, Nilssona i Bensona lub Crutchfielda i Mitchella. Napisz krótki artykuł na jeden z tych tematów.

15. Omów rolę błędu indukcyjnego w reprezentacjach, strategiach wyszukiwania i operatorach stosowanych w modelach uczenia się przedstawionych w tej Części 12. Czy można rozwiązać ten problem? To znaczy, czy genetyczny model uczenia się działa wyłącznie ze względu na swoje założenia reprezentacyjne, czy też można go przełożyć na szersze dziedziny?

UCZENIE MASZYNOWE: GENETYCZNE I AWARYJNE

12.0 Społeczne i powstające modele uczenia się

Tak jak sieci koneksjonistyczne otrzymały wiele wczesnego wsparcia i inspiracji z celu stworzenia sztucznego systemu neuronowego, tak i szereg innych biologicznych analogii wpłynęło na projektowanie algorytmów uczenia maszynowego. W tym rozdziale omówiono algorytmy uczenia się wzorowane na procesach leżących u podstaw ewolucji: kształtowaniu populacji osobników poprzez przetrwanie jej najlepiej przystosowanych członków. Siła selekcji w populacji różnych osobników została wykazana w pojawieniu się gatunków w naturalnej ewolucji, a także w procesach społecznych leżących u podstaw zmiany kulturowej. Zostało również sformalizowane poprzez badania nad automatami komórkowymi, algorytmami genetycznymi, programowaniem genetycznym, sztucznym życiem i innymi formami nowych obliczeń.

Pojawiające się modele uczenia się symulują najbardziej elegancką i potężną formę adaptacji natury: ewolucję form życia roślin i zwierząt. Karol Darwin widział „… nie ma ograniczeń dla tej mocy powolnego i pięknego dostosowywania każdej formy do najbardziej złożonych relacji życiowych…". Dzięki temu prostemu procesowi wprowadzania odmian do kolejnych pokoleń i selektywnej eliminacji osób mniej sprawnych, w populacji pojawiają się adaptacje zwiększające możliwości i różnorodność. Ewolucja i wyłanianie się mają miejsce w populacjach osób wcielonych, których działania wpływają na innych, a na które z kolei wpływają inni. Zatem presje wybiórcze pochodzą nie tylko ze środowiska zewnętrznego, ale także z interakcji między członkami populacji. Ekosystem składa się z wielu członków, z których każdy ma role i umiejętności odpowiednie do własnego przetrwania, ale co ważniejsze, których skumulowane zachowanie kształtuje i jest kształtowane przez resztę populacji. Ze względu na swoją prostotę procesy leżące u podstaw ewolucji okazały się niezwykle ogólne. Ewolucja biologiczna tworzy gatunki, wybierając spośród zmian w genomie. Podobnie ewolucja kulturowa tworzy wiedzę, działając na przekazywanych społecznie i zmodyfikowanych jednostkach informacji. Algorytmy genetyczne i inne formalne analogi ewolucyjne tworzą coraz bardziej wydajne rozwiązania problemów, działając na populacjach potencjalnych rozwiązań problemów. Kiedy algorytm genetyczny jest używany do rozwiązywania problemów, ma on trzy odrębne etapy: po pierwsze, indywidualne potencjalne rozwiązania domeny problemowej są kodowane w reprezentacje, które obsługują niezbędne operacje zmiany i selekcji; często te reprezentacje są tak proste, jak ciągi bitów. W drugim etapie algorytmy kojarzenia i mutacji, analogiczne do aktywności seksualnej biologicznych form życia, tworzą nowe pokolenie osobników, które rekombinują cechy swoich rodziców. Wreszcie funkcja sprawności ocenia, które osoby są „najlepszymi" formami życia, to znaczy są najbardziej odpowiednie dla ostatecznego rozwiązania problemu. Osobom tym preferuje się przetrwanie i rozmnażanie, kształtując następną generację potencjalnych rozwiązań. Ostatecznie pokolenie osób zostanie zinterpretowane z powrotem do pierwotnej domeny problemu jako rozwiązania problemu. Algorytmy genetyczne są również stosowane do bardziej złożonych reprezentacji, w tym reguł produkcyjnych, w celu opracowania zestawów reguł przystosowanych do interakcji ze środowiskiem. Na przykład

programowanie genetyczne łączy i mutuje fragmenty kodu komputerowego, próbując rozwinąć program do rozwiązywania problemów, takich jak wychwytywanie niezmienników w zestawach danych. Przykład uczenia się jako interakcji społecznej prowadzącej do przetrwania można znaleźć w grach takich jak The Game of Life, pierwotnie stworzonych przez matematyka Johna Hortona Conwaya i przedstawionych szerszej społeczności przez Martina Gardnera w Scientific American. W tej grze narodziny, przetrwanie lub śmierć jednostek są funkcją ich własnego stanu i stanu ich bliskich sąsiadów. Zwykle do zdefiniowania gry wystarcza niewielka liczba reguł, zwykle trzy lub cztery. Pomimo tej prostoty, eksperymenty z grą w życie wykazały, że jest ona zdolna do ewolucji struktur o niezwykłej złożoności i zdolności, w tym samoreplikujących się wielokomórkowych „organizmów". Ważnym podejściem do sztucznego życia lub życia jest symulacja warunków ewolucji biologicznej poprzez interakcje maszyn skończonych, wraz z zestawami stanów i regułami przejścia. Te automaty są w stanie przyjmować informacje spoza nich samych, w szczególności od najbliższych sąsiadów. Ich zasady przejścia obejmują instrukcje dotyczące narodzin, kontynuowania życia i umierania. Kiedy populacja takich automatów zostaje uwolniona w domenie i pozwala działać jako równolegle asynchroniczni współpracujący agenci, czasami jesteśmy świadkami ewolucji pozornie niezależnych „form życia". Jako kolejny przykład, Rodney Brooks i jego uczniowie zaprojektowali i zbudowali proste roboty, które współdziałają jako autonomiczni agenci rozwiązujący problemy w warunkach laboratoryjnych. Nie ma centralnego algorytmu sterowania; raczej współpraca jawi się jako artefakt rozproszonych i autonomicznych interakcji jednostek. Społeczność a-life organizuje regularne konferencje i czasopisma odzwierciedlające ich pracę. W części 12.1 przedstawiamy modele ewolucyjne lub oparte na biologii z algorytmami genetycznymi , podejście do uczenia się, które wykorzystuje równoległość, wzajemne interakcje i często reprezentację na poziomie bitowym. W sekcji 12.2 przedstawiamy systemy klasyfikatorów i programowanie genetyczne, stosunkowo nowe obszary badawcze, w których techniki z algorytmów genetycznych są stosowane do bardziej złożonych reprezentacji, takich jak tworzenie i udoskonalanie zestawów reguł produkcji oraz tworzenie i dostosowywanie komputerów programy. W sekcji 12.3 przedstawiamy sztuczne życie. Rozpoczynamy 12.3 od wprowadzenia do „Gry w życie". Kończymy przykładem wyłaniającego się zachowania z badań w Santa Fe Institute.

12.1 Algorytm genetyczny

Podobnie jak sieci neuronowe, algorytmy genetyczne opierają się na metaforze biologicznej: postrzegają uczenie się jako konkurencję między populacją ewoluujących kandydatów do rozwiązań problemów. Funkcja „sprawności" ocenia każde rozwiązanie, aby zdecydować, czy przyczyni się ono do następnej generacji rozwiązań. Następnie, poprzez operacje analogiczne do transferu genów w rozmnażaniu płciowym, algorytm tworzy nową populację kandydujących rozwiązań. Niech P(t) zdefiniuje populację rozwiązań kandydujących x^t_i w czasie t:

$$P(t) = \{x^t_1, x^t_2, \ldots x^t_n\}$$

Przedstawiamy teraz ogólną postać algorytmu genetycznego:

procedure genetic algorithm;

begin

set time t:= 0;

initialize the population P(t);

while the termination condition is not met do

begin

evaluate fitness of each member of the population P(t);

select members from population P(t) based on fitness;

produce the offspring of these pairs using genetic operators;

replace, based on fitness, candidates of P(t), with these offspring;

set time t := t +1

end

end.

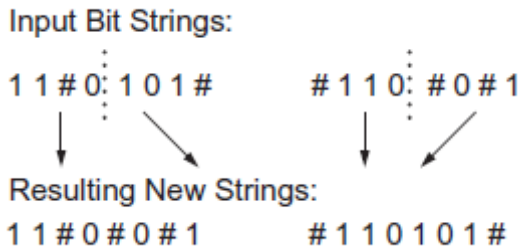
Ten algorytm określa podstawowe ramy uczenia się genetycznego; konkretne implementacje algorytmu tworzą instancję tego frameworka na różne sposoby. Jaki procent populacji zostaje zachowany? Jaki procent kojarzy się i rodzi potomstwo? Jak często i do kogo stosuje się operatory genetyczne? Procedura „zastąpienie najsłabszych kandydatów z P (t)" może zostać wdrożona w prosty sposób, eliminując określony procent najsłabszych kandydatów. Bardziej wyrafinowane podejścia mogą uporządkować populację według przystosowania, a następnie powiązać miarę prawdopodobieństwa eliminacji z każdym składnikiem, gdzie prawdopodobieństwo eliminacji jest funkcją odwrotną do jego przystosowania. Następnie algorytm zamiany wykorzystuje tę miarę jako czynnik przy wyborze kandydatów do wyeliminowania. Chociaż prawdopodobieństwo eliminacji byłoby bardzo niskie dla najsilniejszych członków społeczeństwa, istnieje szansa, że nawet najlepsze osobniki mogłyby zostać usunięte. Zaletą tego schematu jest to, że może on uratować niektóre osoby, których ogólna sprawność jest słaba, ale które zawierają jakiś element, który może przyczynić się do lepszego rozwiązania. Ten zastępczy algorytm ma wiele nazw, w tym Monte Carlo, proporcjonalny wybór sprawności i koło ruletki. Chociaż przykłady z sekcji 12.1.3 wprowadzają bardziej złożone reprezentacje, wprowadzimy kwestie reprezentacji związane z algorytmami genetycznymi przy użyciu prostego bitu ciągi reprezentujące rozwiązania problemów. Na przykład załóżmy, że chcemy, aby algorytm genetyczny nauczył się klasyfikować ciągi jedynek i zer. Możemy przedstawić populację ciągów bitowych jako wzorzec 1, 0 i #, gdzie # to „nie obchodzi", które może pasować do 0 lub 1. Zatem wzór 1 ## 00 ## 1 reprezentuje wszystkie ciągi ośmiu bitów, które zaczynają się i kończą 1 i które mają dwa 0 w środku. Algorytm genetyczny inicjalizuje P(0) do populacji wzorców kandydatów. Zazwyczaj początkowe populacje są wybierane losowo. Ocena rozwiązań kandydujących zakłada funkcję dopasowania $f(x_i^t)$, która zwraca miarę dopasowania kandydata w czasie t. Wspólna miara sprawności kandydata sprawdza go na zbiorze instancji szkoleniowych i zwraca procent poprawnych klasyfikacji. Korzystając z takiej funkcji przydatności, ocena nadaje każdemu kandydatowi rozwiązanie wartość:

$$f(x_i^t)/m(P, t)$$

gdzie m (P, t) jest średnią sprawnością wszystkich członków populacji. Często miara przydatności zmienia się w czasie, w związku z czym przydatność może być funkcją etapu ogólnego rozwiązania problemu, lub $f(x_i^t)$

Po ocenie każdego kandydata algorytm wybiera pary do rekombinacji. Rekombinacja wykorzystuje operatory genetyczne do tworzenia nowych rozwiązań, które łączą elementy ich rodziców. Podobnie jak w przypadku naturalnej ewolucji, sprawność kandydata określa zakres do którego się rozmnaża, przy czym kandydaci o najwyższych ocenach mają większe prawdopodobieństwo reprodukcji. Jak już wspomniano, selekcja jest często probabilistyczna, gdzie słabszym członkom daje się mniejsze

prawdopodobieństwo reprodukcji, ale nie są one całkowicie eliminowane. To, że niektórzy mniej sprawni kandydaci przeżyją, jest ważne, ponieważ nadal mogą zawierać jakiś istotny składnik rozwiązania, na przykład część wzoru bitowego, a reprodukcja może wydobyć ten składnik. Istnieje wiele operatorów genetycznych, które produkują potomstwo z cechami rodziców; najczęstszym z nich jest crossover. Crossover bierze dwa kandydujące rozwiązania i dzieli je, zamieniając komponenty, aby stworzyć dwóch nowych kandydatów. Rysunek 1 ilustruje skrzyżowanie na ciągach bitowych o długości 8.



Operator dzieli je w środku i tworzy dwoje dzieci, których początkowy segment pochodzi od jednego rodzica, a którego ogon pochodzi od drugiego. Zauważ, że podzielenie rozwiązania kandydata w środku to arbitralny wybór. Ten podział może występować w dowolnym punkcie reprezentacji i rzeczywiście, ten punkt podziału może być losowo dostosowywany lub zmieniany podczas procesu rozwiązywania. Na przykład załóżmy, że klasa docelowa jest zbiorem wszystkich ciągów zaczynających się i kończących na 1. Oba ciągi nadrzędne na rysunku 1 poradziłyby sobie stosunkowo dobrze w tym zadaniu. Jednak pierwsze potomstwo byłoby znacznie lepsze niż którekolwiek z rodziców: nie miałoby żadnych fałszywych alarmów i nie rozpoznałoby mniejszej liczby ciągów, które faktycznie znajdowały się w klasie rozwiązania. Zauważ również, że jego rodzeństwo jest gorsze niż którekolwiek z rodziców i prawdopodobnie tak będzie wyeliminowane w ciągu następnych kilku pokoleń. Mutacja to kolejny ważny operator genetyczny. Mutacja bierze jednego kandydata i losowo zmienia pewien jej aspekt. Na przykład mutacja może losowo wybrać bit we wzorze i zmienić go, przełączając 1 na 0 lub #. Mutacja jest ważna, ponieważ populacja początkowa może wykluczyć istotny składnik rozwiązania. W naszym przykładzie, jeśli żaden członek populacji początkowej nie ma 1 na pierwszej pozycji, to krzyżowanie, ponieważ zachowuje pierwsze cztery bity rodzica jako pierwsze cztery bity dziecka, nie może stworzyć potomstwa, które to robi. Mutacja byłaby potrzebna do zmiany wartości tych bitów. Inne operatory genetyczne, na przykład inwersja, również mogą wykonać to zadanie, i opisano w sekcji 12.1.3. Algorytm genetyczny działa aż do spełnienia pewnych wymagań dotyczących zakończenia, takich jak posiadanie jednego lub więcej kandydatów, których przydatność przekracza pewien próg. W następnej sekcji podajemy przykłady kodowania algorytmów genetycznych, operatorów i ocen sprawności dla dwóch sytuacji: spełnienia ograniczenia CNF i problemów podróżującego sprzedawcy.

12.1.3 Dwa przykłady: satysfakcja CNF i podróżujący sprzedawca

Następnie wybieramy dwa problemy i omawiamy kwestie reprezentacji oraz funkcje sprawności odpowiednie do ich rozwiązania. Należy zwrócić uwagę na trzy rzeczy: po pierwsze, wszystkie problemy nie są łatwo lub naturalnie zakodowane jako reprezentacje na poziomie bitowym. Po drugie, operatorzy genetyczni muszą zachować kluczowe relacje w populacji, na przykład obecność i wyjątkowość wszystkich miast w ramach wycieczki dla podróżującego sprzedawcy. Na koniec omówimy ważny związek między funkcją (funkcjami) przystosowania dla stanów problemu a kodowaniem tego problemu. Problem spełnialności formy normalnej koniunkcji (CNF) jest prosty: wyrażenie zdań występuje w postaci normalnej koniunkcji, gdy jest to ciąg zdań połączonych relacją i ($\wedge$). Każda z tych klauzul ma postać dysjunkcji, lub ($\vee$), literałów. Na przykład, jeśli literały to a, b, c, d,

e i f, to wyrażenie (¬a ∨ c) ∧ (¬a ∨ c ∨ ¬e) ∧ (¬b ∨ c ∨ d ∨ ¬ e) ∧ (a ∨ ¬b ∨ c) ∧ (¬e ∨ f) jest w CNF. To wyrażenie jest koniunkcją pięciu klauzul, każda klauzula jest rozłączeniem dwóch lub więcej literałów. Propozycje i ich satysfakcję przedstawiliśmy w rozdziale 2. Omówiliśmy formę CNF wyrażeń zdaniowych i zaproponowaliśmy metodę redukcji wyrażeń do CNF, kiedy przedstawiliśmy wnioskowanie o rozdzielczości. Spełnialność CNF oznacza, że musimy znaleźć przypisanie prawdy lub fałszu (1 lub 0) do każdego z sześciu literałów, tak aby wyrażenie CNF miało wartość true. Czytelnik powinien potwierdzić, że jednym rozwiązaniem dla wyrażenia CNF jest przypisanie fałszu każdemu z a, b i e. Inne rozwiązanie ma e false i c true.

Naturalną reprezentacją problemu satysfakcji CNF jest ciąg sześciu bitów, każdy bit w kolejności, reprezentujący prawdę (1) lub fałsz (0) dla każdego z sześciu literałów, ponownie w kolejności a, b, c, d , e i f. Zatem: 1 0 1 0 1 0 wskazuje, że a, c i e są prawdą, a b, d i f są fałszem, a zatem przykładowe wyrażenie CNF jest fałszywe. Czytelnik może zbadać wyniki innych przypisań prawdy do literałów wyrażenia. Wymagamy, aby działania każdego operatora genetycznego dały potomstwo, które jest przypisaniem prawdy dla wyrażenia CNF, a zatem każdy operator musi wytworzyć sześciobitowy wzór przypisań prawdy. Ważnym rezultatem naszego wyboru reprezentacji wzoru bitowego dla wartości prawdy literałów wyrażenia CNF jest to, że każdy z omówionych do tej pory operatorów genetycznych pozostawi wynikowy wzorzec bitowy jako uzasadnione możliwe rozwiązanie. Oznacza to, że crossover i mutacja pozostawiają wynikowy ciąg bitów jako możliwe rozwiązanie problemu. Nawet inne, rzadziej używane operatory genetyczne, takie jak inwersja (odwrócenie kolejności bitów we wzorze sześciobitowym) lub wymiana (zamiana dwóch różnych bitów we wzorze), pozostawiają wynikowy wzór bitowy jako uzasadnione możliwe rozwiązanie problemu CNF. W rzeczywistości, z tego punktu widzenia, trudno wyobrazić sobie lepiej dopasowaną reprezentację niż wzorzec bitowy dla problemu satysfakcji CNF. Wybór funkcji dopasowania dla tej populacji ciągów bitów nie jest tak prosty. Z jednego punktu widzenia albo przypisanie wartości prawdy do literałów sprawi, że wyrażenie będzie prawdziwe lub też będzie fałszywe. Jeśli określone przypisanie powoduje, że wyrażenie jest prawdziwe, wówczas znajduje się rozwiązanie; w przeciwnym razie tak nie jest. Na pierwszy rzut oka trudno jest określić funkcję dopasowania, która może oceniać „jakość" ciągów bitowych jako potencjalne rozwiązania. Istnieje jednak kilka alternatyw. Należy zauważyć, że pełne wyrażenie CNF składa się z koniunkcji pięciu klauzul. W ten sposób możemy stworzyć system oceny, który pozwoli nam uszeregować potencjalne rozwiązania wzorców bitowych w zakresie od 0 do 5, w zależności od liczby klauzul, które spełnia wzorzec. Tak więc wzór:

1 1 0 0 1 0 ma sprawność fizyczną 1,

0 1 0 0 1 0 ma sprawność fizyczną 2,

0 1 0 0 1 1 ma sprawność fizyczną 3, i

1 0 1 0 1 1 ma kondycję 5 i jest rozwiązaniem.

Ten algorytm genetyczny oferuje rozsądne podejście do problemu satysfakcji CNF. Jedną z jego najważniejszych właściwości jest użycie ukrytego paralelizmu zapewnianego przez populację rozwiązań. Operatory genetyczne mają naturalne dopasowanie do tej reprezentacji. Wreszcie, poszukiwanie rozwiązań wydaje się w naturalny sposób pasować do równoległej strategii „dziel i rządź", ponieważ przydatność jest oceniana na podstawie liczby elementów problemu, które są spełnione. W ćwiczeniach zachęca się czytelnika do rozważenia innych aspektów tego problemu.

PRZYKŁAD 12.2.2: PROBLEM Z PODRÓŻĄCYM SPRZEDAWCĄ

Problem podróżującego sprzedawcy (TSP) jest klasyczny dla sztucznej inteligencji i informatyki. Wprowadziliśmy to wraz z omówieniem wykresów w Części 3. Jego pełna przestrzeń stanów wymaga uwzględnienia N! podaje, gdzie N to liczba miast do odwiedzenia. Okazało się, że jest NP-trudny, a wielu badaczy proponuje heurystyczne podejście do jego rozwiązania. Sformułowanie problemu jest proste: sprzedawca musi odwiedzić N miast w ramach trasy sprzedaży. Z każdą parą miast na trasie związany jest koszt (np. Przebieg, opłata za przelot). Znajdź najmniej kosztowną ścieżkę dla sprzedawcy, aby rozpocząć w jednym mieście, odwiedzić wszystkie inne miasta dokładnie raz i wrócić do domu. TSP ma kilka bardzo przydatnych zastosowań, w tym wiercenie płytek drukowanych, krystalografię rentgenowską i trasowanie w produkcji VLSI. Niektóre z tych problemów wymagają odwiedzenia dziesiątek tysięcy punktów (miast) przy minimalnej ścieżce kosztów. Bardzo interesującym pytaniem w analizie klasy problemów TSP jest to, czy warto przez wiele godzin uruchamiać kosztowną stację roboczą, aby uzyskać prawie optymalne rozwiązanie, czy też uruchomić tani komputer na kilka minut, aby uzyskać „wystarczająco dobre" wyniki dla tych aplikacji . TSP to interesujący i trudny problem z wieloma konsekwencjami dotyczącymi strategii wyszukiwania. Jak możemy użyć algorytmu genetycznego do rozwiązania tego problemu? Po pierwsze, wybór reprezentacji ścieżki odwiedzanych miast, a także stworzenie zestawu operatorów genetycznych dla tej ścieżki nie jest trywialne. Projekt funkcji fitness jest jednak bardzo proste: wszystko, co musimy zrobić, to oszacować koszt długości ścieżki. Moglibyśmy wtedy uporządkować ścieżki według ich kosztu, im tańsze tym lepsze. Rozważmy kilka oczywistych reprezentacji, które okazują się skomplikowane konsekwencje. Załóżmy, że mamy dziewięć miast do odwiedzenia, 1, 2, ..., 9, więc reprezentację ścieżki tworzymy jako uporządkowaną listę tych dziewięciu liczb całkowitych. Załóżmy, że po prostu uczynimy każde miasto czterobitowym wzorem 0001, 0010,. . . 1001. Zatem wzorzec: 0001 0010 0011 0100 0101 0110 0111 1000 1001 reprezentuje wizytę w każdym mieście w kolejności jego numeracji. Dodaliśmy spacje do ciągu tylko po to, aby był łatwiejszy do odczytania. A co z operatorami genetycznymi? Crossover jest zdecydowanie niedostępny, ponieważ nowy sznurek wyprodukowany przez dwoje różnych rodziców najprawdopodobniej nie reprezentowałby ścieżki, która odwiedza każde miasto dokładnie raz. W rzeczywistości, dzięki skrzyżowaniu, niektóre miasta mogą zostać usunięte, podczas gdy inne są odwiedzane więcej niż raz. A co z mutacją? Załóżmy, że skrajny lewy fragment szóstego miasta, 0110, jest zmieniony na 1? 1110 lub 14 nie jest już legalnym miastem. Odwrócenie i zamiana miast (cztery bity we wzorze miasta) w obrębie wyrażenia ścieżki byłyby dopuszczalnymi operatorami genetycznymi, ale czy byłyby one wystarczająco potężne, aby uzyskać zadowalające rozwiązanie? W rzeczywistości jednym ze sposobów spojrzenia na poszukiwanie minimalnej ścieżki byłoby wygenerowanie i ocena wszystkich możliwych permutacji N elementów listy miast. Operatorzy genetyczni muszą być w stanie wyprodukować wszystkie permutacje. Innym podejściem do TSP byłoby zignorowanie reprezentacji wzoru bitowego i nadanie każdemu miastu alfabetycznej lub numerycznej nazwy, np. 1, 2, ..., 9; ułóż ścieżkę przez miasta w kolejności tych dziewięciu cyfr, a następnie wybierz odpowiednie operatory genetyczne do tworzenia nowych ścieżek. Mutacja, o ile byłaby przypadkową wymianą dwóch miast na ścieżce, byłaby w porządku, ale operator skrzyżowania między dwiema ścieżkami byłby bezużyteczny. Wymiana fragmentów ścieżki z innymi fragmentami tej samej ścieżki lub dowolny operator, który tasował litery ścieżki (bez usuwania, dodawania lub powielania jakichkolwiek miast) działałby. Takie podejście utrudnia jednak łączenie w potomstwo „lepszych" elementów wzorców na drogach miast dwojga różnych rodziców. Wielu badaczy stworzyło operatory krzyżowania, które rozwiązują te problemy i pozwalają nam pracować z uporządkowaną listą odwiedzonych miast. Na przykład Davis zdefiniował operator o nazwie crossover zamówienia. Załóżmy, że mamy dziewięć miast, 1, 2, ..., 9, a kolejność liczb całkowitych reprezentuje kolejność odwiedzonych miast. Order crossover buduje potomstwo, wybierając podciąg miast na drodze jednego z rodziców. Zachowuje również względne uporządkowanie miast od drugiego rodzica. Najpierw wybierz dwa punkty cięcia oznaczone „|", które są losowo wstawiane w to samo miejsce

każdego z rodziców. Lokalizacje punktów cięcia są losowe, ale po wybraniu te same lokalizacje są używane dla obojga rodziców. Na przykład dla dwojga rodziców p1 i p2, z punktami odcięcia po trzecim i siódmym mieście:

p1 = (1 9 2 | 4 6 5 7 | 8 3)

p2 = (4 5 9 | 1 8 7 6 | 2 3)

dwoje dzieci c1 i c2 jest tworzone w następujący sposób. Najpierw segmenty między punktami cięcia są kopiowane do potomstwa:

c1 = (x x x | 4 6 5 7 | x x)

c2 = (x x x | 1 8 7 6 | x x)

Następnie, zaczynając od drugiego punktu odcięcia jednego z rodziców, miasta drugiego rodzica są kopiowane w tej samej kolejności, pomijając miasta już obecne. Po osiągnięciu końca sznurka kontynuuj od początku. Zatem sekwencja miast z p2 jest następująca:

2 3 4 5 9 1 8 7 6

Po usunięciu miast 4, 6, 5 i 7, ponieważ są one już częścią pierwszego dziecka, otrzymujemy skróconą listę 2, 3, 9, 1 i 8, która następnie uzupełnia, zachowując kolejność znalezioną w p2 pozostałe miasta do odwiedzenia przez c1:

c1 = (2 3 9 | 4 6 5 7 | 1 8)

W podobny sposób możemy stworzyć drugie dziecko c2:

c2 = (3 9 2 | 1 8 7 6 | 4 5)

Podsumowując, w kolejności krzyżowania się fragmenty ścieżki są przekazywane od jednego rodzica p1 do dziecka c1, podczas gdy kolejność pozostałych miast dziecka c1 jest dziedziczona po drugim rodzicu p2. Potwierdza to oczywistą intuicję, że kolejność miast będzie ważna w generowaniu najmniej kosztownej ścieżki, dlatego też bardzo ważne jest, aby informacje dotyczące zamawiania były przekazywane dzieciom przez sprawnych rodziców. Algorytm krzyżowania zamówień gwarantuje również, że dzieci będą miały legalne wycieczki, odwiedzając wszystkie miasta dokładnie raz. Gdybyśmy chcieli dodać operator mutacji do tego wyniku, musielibyśmy, jak wspomniano wcześniej, uważać, aby była to wymiana miast w obrębie ścieżka. Operator inwersji, po prostu odwracający kolejność wszystkich miast w wycieczce, nie zadziała (nie ma nowej ścieżki, gdy wszystkie miasta są odwrócone). Jeśli jednak kawałek w ścieżce zostanie wycięty i odwrócony, a następnie zastąpiony, byłoby to dopuszczalne użycie inwersji. Na przykład, używając cięcia | wskaźnik jak poprzednio, ścieżka:

c1 = (2 3 9 | 4 6 5 7 | 1 8),

ulega odwróceniu środkowej sekcji,

c1 = (2 3 9 | 7 5 6 4 | 1 8)

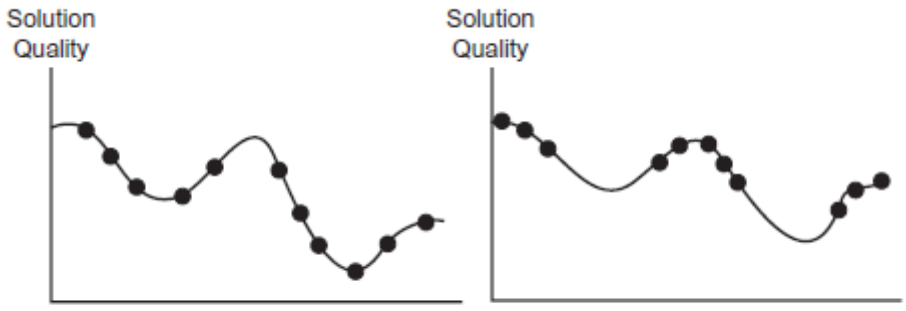
Można zdefiniować nowy operator mutacji, który losowo wybierze miasto i umieści je w nowej, losowo wybranej lokalizacji na ścieżce. Ten operator mutacji mógłby również działać na fragmencie ścieżki, na przykład, aby zająć podścieżkę trzech miast i umieścić je w tej samej kolejności w nowym miejscu na ścieżce. Inne sugestie są w ćwiczeniach.

12.1.4 Ocena algorytmu genetycznego

Powyższe przykłady podkreślają unikalne problemy algorytmu genetycznego dotyczące reprezentacji wiedzy, wyboru operatora i projektowania funkcji przystosowania. Wybrana reprezentacja musi wspierać operatory genetyczne. Czasami, tak jak w przypadku problemu satysfakcji CNF, reprezentacja poziomu bitów jest naturalna. W tej sytuacji tradycyjne genetyczne operatory krzyżowania i mutacji mogłyby zostać bezpośrednio wykorzystane do stworzenia potencjalnych rozwiązań. Zupełnie inną sprawą był problem podróżującego sprzedawcy. Po pierwsze, wydaje się, że nie ma żadnych naturalnych reprezentacji poziomu bitów dla tego problemu. Po drugie, należało wymyślić nowe operatory mutacji i krzyżowania, które zachowałyby własność potomstwa, która miała być legalną ścieżką przez wszystkie miasta, odwiedzając każde tylko raz. Wreszcie operatorzy genetyczni muszą przekazać „znaczące" informacje o potencjalnym rozwiązaniu następnemu pokoleniu. Jeśli ta informacja, podobnie jak w przypadku spełnialności CNF, jest przypisaniem wartości prawdy, operatorzy genetyczni muszą ją zachować w następnym pokoleniu. W przypadku TSP organizacja ścieżki była krytyczna, więc jak omówiliśmy, składniki tej ścieżki muszą być przekazywane potomkom. Ten udany transfer opiera się zarówno na wybranej reprezentacji, jak i na operatorach genetycznych zaprojektowanych dla każdego problemu. Przedstawienie pozostawiamy jeszcze jedną ostatnią kwestią, problemem „naturalności" wybranej reprezentacji. Załóżmy, że jako prosty, choć nieco sztuczny przykład, chcemy, aby nasze operatory genetyczne rozróżniały liczby 6, 7, 8 i 9. Reprezentacja liczb całkowitych daje bardzo naturalny i równomiernie rozłożony porządek, ponieważ w ramach dziesięciu liczb całkowitych o podstawie następny element jest po prostu o jeden więcej niż poprzedni. Jednak wraz ze zmianą na binarność ta naturalność znika. Rozważ wzory bitów dla 6, 7, 8 i 9: 0110 0111 1000 1001 Zwróć uwagę, że między 6 a 7 oraz między 8 a 9 występuje zmiana 1 bitu. Jednak między 7 a 8 wszystkie cztery bity się zmieniają! Ta reprezentacyjna anomalia może być ogromna w przypadku próby wygenerowania rozwiązania wymagającego jakiejkolwiek organizacji tych czterech wzorców bitowych. Szereg technik, zwykle w ramach ogólnego pojęcia szarego kodowania, rozwiązuje ten problem niejednolitej reprezentacji. Na przykład zakodowaną na szaro wersję pierwszych szesnastu liczb dwójkowych można znaleźć w tabeli 1. Zwróć uwagę, że każda liczba różni się dokładnie o jeden bit od swoich sąsiadów. Używając szarego kodowania zamiast standardowych liczb binarnych, przejścia operatora genetycznego między stanami bliskich sąsiadów są naturalne i płynne.

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Ważną zaletą algorytmu genetycznego jest równoległy charakter jego poszukiwań. Algorytmy genetyczne wdrażają potężną formę wspinaczki górskiej, która utrzymuje wiele rozwiązań, eliminuje mało obiecujące i ulepsza dobre rozwiązania. Rysunek 2, dostosowany od Hollanda , pokazuje wiele rozwiązań zbiegających się w kierunku optymalnych punktów w przestrzeni poszukiwań.



a. The beginning search space

b. The search space after n generations

Na tym rysunku oś pozioma przedstawia możliwe punkty w przestrzeni rozwiązań, podczas gdy oś pionowa odzwierciedla jakość tych rozwiązań. Kropki na krzywej należą do aktualnej populacji potencjalnych rozwiązań algorytmu genetycznego. Początkowo rozwiązania są rozproszone w przestrzeni możliwych rozwiązań. Po kilku pokoleniach mają tendencję do skupiania się wokół obszarów o wyższej jakości rozwiązań. Kiedy opisujemy nasze poszukiwania genetyczne jako „wspinanie się po wzgórzach", pośrednio uznajemy poruszanie się po „krajobrazie fitness". Ten krajobraz będzie miał swoje doliny, szczyty, z lokalnymi maksimami i minimami. W rzeczywistości niektóre z nieciągłości w przestrzeni będą artefaktami reprezentacji i operatorów genetycznych wybranych dla problemu. Ta nieciągłość może być na przykład spowodowana brakiem szarego kodowania, jak właśnie omówiono. Zwróć też uwagę na to ,że algorytmy genetyczne, w przeciwieństwie do sekwencyjnych form wspinaczki górskiej, nie odrzucają od razu mało obiecujących rozwiązań. Dzięki operatorom genetycznym nawet słabe rozwiązania mogą nadal przyczyniać się do tworzenia przyszłych kandydatów. Inną różnicą między algorytmami genetycznymi a heurystykami przestrzeni stanów przedstawionymi w rozdziale 4 jest analiza różnicy stanu obecnego / stanu celu. Treść informacji wspierająca algorytm A*, wymagała oszacowania „wysiłku", aby przejść między stanem obecnym a stanem celu. Żadna taka miara nie jest wymagana w przypadku algorytmów genetycznych, po prostu pewna miara przydatności każdego z nich bieżące generowanie potencjalnych rozwiązań. Nie jest również wymagane ścisłe porządkowanie kolejnych stanów na liście otwartej, jak widzieliśmy podczas przeszukiwania przestrzeni stanów; istnieje raczej populacja pasujących rozwiązań do problemu, każdy potencjalnie dostępny, aby pomóc w tworzeniu nowych możliwych rozwiązań w ramach paradygmatu wyszukiwania równoległego. Ważnym źródłem mocy algorytmu genetycznego jest ukryta równoległość nieodłączna od operatorów ewolucyjnych. W porównaniu z przeszukiwaniem w przestrzeni stanów i uporządkowaną listą otwartą, wyszukiwanie przebiega równolegle, operując na całych rodzinach potencjalnych rozwiązań. Ograniczając reprodukcję słabszych kandydatów, algorytmy genetyczne mogą nie tylko wyeliminować to rozwiązanie, ale wszystkich jego potomków. Na przykład ciąg 101 # 0 ## 1, jeśli zostanie przerwany w środku, może być rodzicem całej rodziny ciągów w postaci 101 #____. Jeśli rodzic zostanie uznany za niezdolnego do życia, jego eliminacja może również usunąć całe potencjalne potomstwo i być może także możliwość rozwiązania problemu. Ponieważ algorytmy genetyczne są coraz szerzej wykorzystywane w rozwiązywaniu problemów stosowanych, a także w modelowaniu naukowym,

rośnie zainteresowanie próbami zrozumienia ich podstaw teoretycznych. Kilka pytań, które pojawiają się naturalnie, to:

1. Czy możemy scharakteryzować typy problemów, w przypadku których GA będą dobrze działać?

2. W przypadku jakich typów problemów radzą sobie słabo?

3. Co to w ogóle „oznacza" dla AH, aby działał dobrze lub źle dla typu problemu?

4. Czy są jakieś prawa, które mogą opisać makropoziom zachowania GA? W szczególności, czy istnieją jakieś przewidywania dotyczące zmian przystosowania podgrup populacji w czasie?

5. Czy istnieje sposób, aby opisać zróżnicowane skutki różnych operatorów genetycznych, krzyżowania się, mutacji, inwersji itp. W czasie?

6. W jakich okolicznościach (jakie problemy i jakie operatory genetyczne) GA będą działać lepiej niż tradycyjne metody wyszukiwania AI?

Zajęcie się wieloma z tych problemów znacznie wykracza poza zakres naszej książki. W rzeczywistości, jak wskazuje Mitchell, u podstaw algorytmów genetycznych wciąż istnieje więcej otwartych pytań niż ogólnie przyjętych odpowiedzi. Niemniej od początku pracy w GA próbowali to zrobić badacze, w tym Holland zrozumieć, jak działają GA. Chociaż zajmują się kwestiami na poziomie makro, takimi jak sześć właśnie zadanych pytań, ich analiza rozpoczyna się od przedstawienia na poziomie mikro lub bitowym. Holland (1975) wprowadził pojęcie schematu jako ogólnego wzorca i „budulca" rozwiązań. Schemat to wzorzec ciągów bitów opisany przez szablon składający się z 1, 0 i # (nie obchodzi mnie to). Na przykład schemat 1 0 # # 0 1 przedstawia rodzinę ciągów sześciobitowych rozpoczynających się od 1 0 i kończących się na 0 1. Ponieważ środkowy wzorzec # # opisuje cztery wzorce bitowe, 0 0, 0 1, 1 0, 1 1, cały schemat reprezentuje cztery wzorce po sześć jedynek i zer. Tradycyjnie mówi się, że każdy schemat opisuje hiperpłaszczyznę (Goldberg 1989); w tym przykładzie hiperpłaszczyzna przecina zestaw wszystkich możliwych reprezentacji sześciobitowych. Głównym założeniem tradycyjnej teorii AH jest to, że schematy są elementami składowymi rodzin rozwiązań. Mówi się, że genetyczni operatorzy krzyżowania i mutacji manipulują tymi schematami w kierunku potencjalnych rozwiązań. Specyfikacja opisująca tę manipulację nosi nazwę twierdzenia o schemacie (Holland 1975, Goldberg 1989). Według Holland, system adaptacyjny musi identyfikować, testować i uwzględniać właściwości strukturalne, zgodnie z hipotezą, aby zapewnić lepszą wydajność w pewnym środowisku. Schematy mają być formalizacją tych właściwości strukturalnych. Analiza schematu Hollanda sugeruje, że algorytm doboru przystosowania w coraz większym stopniu koncentruje się na podzbiorach przestrzeni poszukiwań o szacowanej najlepszej sprawności; to są podzbiory opisywane są schematami sprawności ponadprzeciętnej. Crossover operatora genetycznego łączy razem bloki budulcowe o wysokiej sprawności w tej samej strunie, próbując stworzyć coraz bardziej dopasowane struny. Mutacja pomaga zagwarantować, że różnorodność (genetyczna) nigdy nie zostanie usunięta z poszukiwań; to znaczy, że nadal badamy nowe elementy krajobrazu fitness. Algorytm genetyczny można zatem postrzegać jako napięcie między otwarciem ogólnego procesu wyszukiwania a uchwyceniem i zachowaniem ważnych (genetycznych) cech w tej przestrzeni poszukiwań. Chociaż oryginalna analiza Hollanda wyszukiwania GA skupiała się na poziomie bitowym, nowsze prace rozszerzyły tę analizę na alternatywne schematy reprezentacji (Goldberg 1989). W następnej sekcji zastosujemy techniki AH do bardziej złożonych reprezentacji,

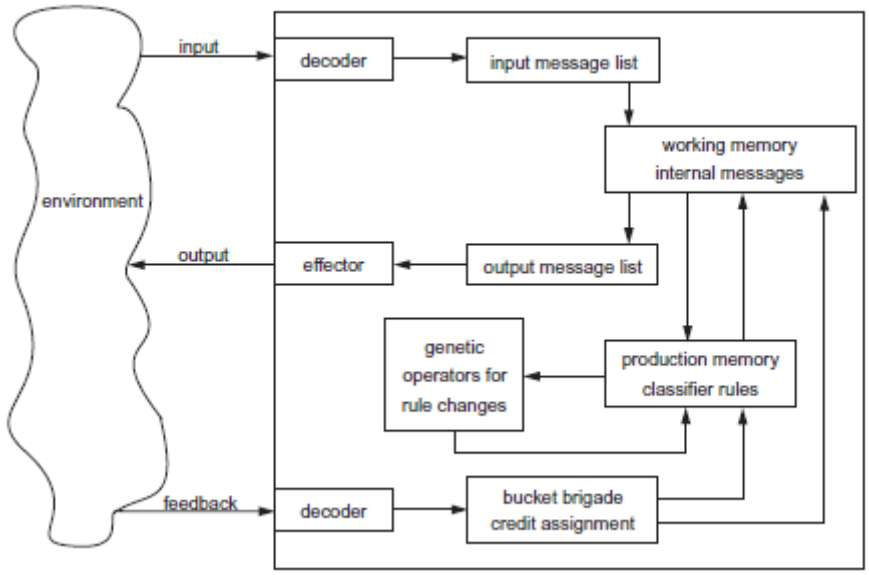
12.2 Systemy klasyfikacyjne i programowanie genetyczne

Wczesne badania algorytmów genetycznych skupiały się prawie wyłącznie na reprezentacjach niskiego poziomu, takich jak ciągi {0, 1, #}. Oprócz obsługi prostych instancji operatorów genetycznych, ciągi

bitów i podobne reprezentacje dają algorytmom genetycznym dużą moc innych podejść podsymbolicznych, takich jak sieci koneksjonistyczne. Istnieją jednak problemy, takie jak wędrowny sprzedawca, który ma bardziej naturalne kodowanie na bardziej złożonym poziomie reprezentacji. Możemy dalej zapytać, czy algorytmy genetyczne można zdefiniować dla jeszcze bogatszych reprezentacji, takich jak gdyby… wtedy… reguły lub fragmenty kodu komputerowego. Ważnym aspektem takich reprezentacji jest ich zdolność do łączenia odrębnych źródeł wiedzy wyższego poziomu poprzez łączenie reguł lub wywołania funkcji w celu spełnienia wymagania dotyczące konkretnego wystąpienia problemu. Niestety, trudno jest zdefiniować operatory genetyczne, które uchwycą składniową i semantyczną strukturę relacji logicznych, jednocześnie umożliwiając efektywne zastosowanie operatorów, takich jak crossover czy mutacja. Jednym z możliwych sposobów połączenia zdolności rozumowania reguł z uczeniem się genetyki jest przetłumaczenie zdań logicznych na ciągi bitowe i użycie standardowego operatora przecięcia. Niestety, w przypadku wielu tłumaczeń większość ciągów bitów wytwarzanych przez krzyżowanie i mutację nie będzie odpowiadać znaczącym zdaniom logicznym. Jako alternatywę dla przedstawiania rozwiązań problemów jako ciągów bitów, możemy zdefiniować odmiany crossovera, które można zastosować bezpośrednio do reprezentacji wyższego poziomu, na przykład jeśli ... to ... reguły lub fragmenty kodu w języku programowania wyższego poziomu. W tej sekcji omówiono przykłady każdego podejścia do rozszerzania mocy algorytmów genetycznych.

12.2.1 Systemy klasyfikatorów

Holland opracował architekturę rozwiązywania problemów zwaną systemami klasyfikatorów, która stosuje uczenie genetyczne do reguł w systemie produkcyjnym. System klasyfikatora zawiera znane elementy systemu produkcyjnego: reguły produkcji (zwane tutaj klasyfikatorami), pamięć roboczą, czujniki wejściowe (lub dekodery) i wyjścia (lub efektory).



Niezwykłe cechy systemu klasyfikatora obejmują wykorzystanie konkurencyjnego przetargu do rozwiązywania konfliktów, algorytmów genetycznych do uczenia się i algorytmu brygady wiaderkowej do przypisania zasługi i winy za reguły podczas nauki. Informacje zwrotne ze środowiska zewnętrznego umożliwiają ocenę sprawności kandydatów do klasyfikatorów, zgodnie z wymogami uczenia się genetyki. System klasyfikatora z rysunku 3 składa się z następujących głównych elementów:

1. Detektory komunikatów wejściowych z otoczenia.

2. Detektory komunikatów zwrotnych z otoczenia.

3. Efektory przenoszące wyniki aplikacji reguł z powrotem do środowiska.

4. Zestaw reguł produkcji składający się z populacji klasyfikatorów. Każdy klasyfikator ma przypisaną miarę sprawności.

5. Pamięć robocza dla reguł klasyfikatora. Ta pamięć integruje wyniki odpalania reguły produkcyjnej z informacjami wejściowymi.

6. Zbiór operatorów genetycznych do modyfikacji reguł produkcji.

7. System przypisywania uznania regułom tworzenia udanych działań

W rozwiązywaniu problemów klasyfikator działa jak tradycyjny system produkcyjny. Środowisko wysyła wiadomość, być może ruch w grze, do detektorów systemu klasyfikatora. To zdarzenie jest dekodowane i umieszczane jako wzorzec na wewnętrznej liście komunikatów, pamięci roboczej systemu produkcyjnego. Te komunikaty, w normalnym działaniu systemu produkcyjnego opartego na danych, są zgodne ze wzorcami warunków reguł klasyfikatora. O wyborze „najsilniej aktywowanych klasyfikatorów" decyduje schemat licytacji, w którym oferta jest funkcją zarówno skumulowanej sprawności klasyfikatora, jak i jakości dopasowania między bodźcem wejściowym a jego wzorcem stanu. Klasyfikatory z najbliższym dopasowaniem dodają komunikaty (działanie uruchomionych reguł) do pamięci roboczej. Zmieniona lista komunikatów może wysyłać komunikaty do efektorów, które działają na środowisko lub aktywują nowe reguły klasyfikatora w miarę kontynuacji przetwarzania systemu produkcyjnego. Systemy klasyfikatorów wdrażają formę uczenia się ze wzmocnieniem. Na podstawie informacji zwrotnej od nauczyciela lub funkcji oceny sprawności, uczący się oblicza przydatność populacji reguł kandydata i dostosowuje tę populację przy użyciu odmian uczenia się genetycznego. Systemy klasyfikatorów uczą się na dwa sposoby. Po pierwsze, istnieje system nagród, który dostosowuje mierniki sprawności reguł klasyfikatora, nagradzając pomyślne odpalenia reguł i karając za błędy. Algorytm przypisywania kredytów przekazuje część nagrody lub kary z powrotem do wszelkich reguł klasyfikatora, które przyczyniły się do uruchomienia ostatecznej reguły. To rozłożenie zróżnicowanych nagród między współpracującymi klasyfikatorami, a także tymi, które umożliwiły ich odpalenie, jest często realizowane w algorytmie brygady wiaderkowej. Algorytm brygady kubełkowej rozwiązuje problem przypisywania kredytu lub winy w sytuacjach, w których wyjście systemu może być iloczynem sekwencji odpalania reguły. W przypadku błędu, skąd mamy wiedzieć, którą regułę należy obwiniać? Czy odpowiedzialność za strzelanie ponosi ostatnia reguła, czy też jakaś poprzednia reguła, która dostarczyła błędnych informacji? Algorytm brygady wiaderkowej przypisuje zarówno zasługi, jak i winę do sekwencji zastosowań reguł zgodnie z miarami wkładu każdej reguły w ostateczny wniosek. Druga forma uczenia się modyfikuje same reguły za pomocą operatorów genetycznych, takich jak mutacja i krzyżowanie. Pozwala to na przetrwanie reguł odnoszących największe sukcesy i łączenie ich w celu tworzenia nowych klasyfikatorów, podczas gdy nieudane klasyfikatory reguł znikają. Każda reguła klasyfikatora składa się z trzech komponentów: warunek reguły pasuje do danych w pamięci roboczej w typowym sensie systemu produkcyjnego. Ucząc się, operatorzy genetyczni mogą modyfikować zarówno warunki, jak i działania reguł produkcji. Drugi składnik reguły, akcja, może skutkować zmianą wewnętrznej listy komunikatów (pamięci produkcyjnej). Wreszcie, każda reguła ma miarę sprawności. Ten parametr zmienia się, jak już wspomniano, zarówno przez udane, jak i nieudane działania. Środek ten jest pierwotnie przypisany do każdej reguły dotyczącej jej tworzenia przez operatorów genetycznych; na przykład można go ustawić jako średnią sprawność fizyczną jego dwojga rodziców. Prosty przykład ilustruje interakcje między tymi elementami systemu klasyfikatora. Załóżmy,

że zbiór obiektów do sklasyfikowania jest zdefiniowany przez sześć atrybutów (warunki c1, c2, ..., c6), a następnie przypuśćmy, że każdy z tych atrybutów może mieć pięć różnych wartości. Chociaż możliwe wartości każdego atrybutu są oczywiście różne (na przykład wartość c3 może być kolorem, podczas gdy c5 może opisywać pogodę), bez utraty ogólności nadamy każdemu atrybutowi wartość całkowitą z {1, 2, ..., 5}. Załóżmy, że warunki tych reguł umieszczają odpowiadający im obiekt w jednej z czterech klas: A1, A2, A3, A4. Bazując na tych ograniczeniach, każdy klasyfikator będzie miał postać: (c1 c2 c3 c4 c5 c6) → Ai, gdzie i = 1, 2, 3, 4. gdzie każdy ci we wzorze warunku oznacza wartość {1, 2,. .., 5} i-tego atrybutu warunku. Zwykle warunki mogą również przypisywać atrybutowi wartość # lub „nie obchodzi mnie to". Ai oznacza klasyfikację, A1, A2, A3 lub A4. Tabela 2 przedstawia zestaw klasyfikatorów. Należy zauważyć, że różne wzorce warunków mogą mieć tę samą klasyfikację, jak w regułach 1 i 2, lub te same wzorce, jak w regułach 3 i 5, mogą prowadzić do różnych klasyfikacji.

Condition (Attributes)	Action (Classification)	Rule Number
(1 # # # 1 #)	→ A1	1
(2 # # 3 # #)	→ A1	2
(# # 4 3 # #)	→ A2	3
(1 # # # # #)	→ A2	4
(# # 4 3 # #)	→ A3	5
etc.		

Jak opisano do tej pory, system klasyfikatorów jest po prostu kolejną formą wszechobecnego systemu produkcyjnego. Jedyną naprawdę nowatorską cechą reguł klasyfikatora w tym przykładzie jest użycie ciągów cyfr i # do reprezentowania wzorców warunków. To właśnie ta reprezentacja warunków ogranicza zastosowanie algorytmów genetycznych do reguł. Pozostała część dyskusji dotyczy uczenia się genów w systemach klasyfikatorów. Aby uprościć pozostałą część przykładu, weźmiemy pod uwagę tylko wydajność systemu klasyfikatora w uczeniu się klasyfikacji A1. Oznacza to, że zignorujemy inne klasyfikacje i przypiszemy wzorcom warunków wartość 1 lub 0 w zależności od tego, czy obsługują one klasyfikację A1, czy nie. Należy zauważyć, że w tym uproszczeniu nie ma utraty ogólności; można go rozszerzyć na problemy związane z uczeniem się więcej niż jednej klasyfikacji poprzez użycie wektora do wskazania klasyfikacji, które pasują do określonego wzorca warunkowego. Na przykład klasyfikatory z tabeli 12.2 można podsumować następująco:

(1 # # # 1 #) → (1 0 0 0)
(2 # # 3 # #) → (1 0 0 0)
(1 # # # # #) → (0 1 0 0)
(# # 4 3 # #) → (0 1 1 0)

W tym przykładzie ostatnie z tych podsumowań wskazuje, że atrybuty warunku obsługują reguły klasyfikacji A2 i A3, a nie A1 lub A4. Zastępując przypisanie 0 lub 1 tymi wektorami, algorytm uczący się może ocenić wydajność reguły w wielu klasyfikacjach. W tym przykładzie użyjemy reguł z tabeli 12.2 do wskazania prawidłowych klasyfikacji; zasadniczo będą funkcjonować jako nauczyciele lub osoby oceniające zgodność reguł w systemie uczenia się. Podobnie jak w przypadku większości uczących się genetyki, zaczynamy od przypadkowej populacji reguł. Każdy wzorzec stanu ma również przypisany parametr siły lub sprawności (liczba rzeczywista między 0,0 brak siły a 1,0 pełna siła. Ten parametr siły, s, jest obliczany na podstawie sprawności rodziców każdej reguły i mierzy jej przydatność historyczną . W każdym cyklu uczenia się reguły próbują klasyfikować dane wejściowe, a następnie są uszeregowane według nauczyciela lub miernika sprawności. Załóżmy na przykład, że w pewnym cyklu klasyfikator ma

następującą populację reguł klasyfikacji kandydatów, gdzie wynik 1 wskazuje że wzorzec prowadził do poprawnej klasyfikacji i 0, że nie:

(# # # 2 1 #) → 1 s = 0,6

(# # 3 # # 5) → 0 s = 0,5

(2 1 # # # #) → 1 s = 0,4

(# 4 # # # 2) → 0 s = 0,23

Załóżmy, że ze środowiska pochodzi nowa wiadomość wejściowa: (1 4 3 2 1 5), a nauczyciel (korzystając z pierwszej reguły z tabeli 12.2) klasyfikuje ten wektor wejściowy jako pozytywny przykład A1. Zastanówmy się, co się stanie, gdy pamięć robocza otrzyma ten wzorzec, a cztery kandydujące reguły klasyfikujące spróbują go dopasować. Dopasowanie zasad 1 i 2. Rozwiązywanie konfliktów odbywa się w drodze przetargu konkurencyjnego między dopasowanymi regułami. W naszym przykładzie cena ofertowa jest funkcją sumy dopasowań wartości atrybutów pomnożonych przez miarę siły reguły. Dopasowania typu „bez znaczenia" mają wartość 0,5, a dopasowania ścisłe mają wartość 1,0. Aby znormalizować, dzielimy ten wynik przez długość wektora wejściowego. Ponieważ wektor wejściowy pasuje do pierwszego klasyfikatora z dwoma dokładnymi i czterema „nie obchodzi", jego stawka wynosi ((4 * 0,5 + 2 * 1) * 0,6) / 6 lub 0,4. Drugi klasyfikator również pasuje do dwóch atrybutów i ma cztery „nie obchodzi", więc jego stawka wynosi 0,33. W naszym przykładzie uruchamia się tylko klasyfikator składający najwyższą ofertę, ale w bardziej skomplikowanych sytuacjach może być pożądane, aby procent ofert został zaakceptowany. Pierwsza reguła wygrywa i ogłasza swoje działanie, 1, wskazując, że ten wzorzec jest przykładem A1. Ponieważ ta czynność jest prawidłowa, miara przydatności przepisu 1 zostaje zwiększona do wartości od aktualnej do 1,0. Gdyby działanie tej zasady było nieprawidłowe, środek sprawności zostałby obniżony. Gdyby system wymagał wielokrotnego odpalenia zestawu reguł, aby wywołać jakiś skutek w środowisku, wszystkie reguły odpowiedzialne za ten wynik otrzymałyby pewną część nagrody. Dokładna procedura obliczania sprawności reguły różni się w zależności od systemu i może być dość złożona, obejmując użycie algorytmu brygady kubełkowej lub podobnej techniki przypisywania punktów. Szczegóły można znaleźć w Holland. Po obliczeniu przydatności reguł kandydata algorytm uczący się stosuje operatory genetyczne w celu utworzenia reguł następnej generacji. Najpierw algorytm wyboru określi najbardziej pasujących członków zestawu reguł. Ten wybór opiera się na mierniku sprawności, ale może również obejmować dodatkową wartość losową. Wartość losowa daje regułom o słabej przydatności możliwość reprodukcji, pomagając uniknąć zbyt pośpiesznej eliminacji reguł, które, choć ogólnie są słabe, mogą zawierać jakiś element pożądanego rozwiązania. Załóżmy, że pierwsze dwie reguły klasyfikacyjne w przykładzie są wybrane, aby przetrwać i rozmnażać się. Losowe wybieranie położenia zwrotnicy między czwartym i piątym elementem,

(# # # 2 | 1 #) → 1 s = 0,6

(# # 3 # | # 5) → 0 s = 0,5

produkuje potomstwo:

(# # 3 # | 1 #) → 0 s = 0,53

(# # # 2 | # 5) → 1 s = 0,57

Miara sprawności dzieci jest ważoną funkcją sprawności rodziców. Ważenie jest funkcją położenia punktu przecięcia. Pierwsze potomstwo ma 1/3 pierwotnego klasyfikatora 0,6 i 2/3 pierwotnego klasyfikatora 0,5. Zatem pierwsze potomstwo ma siłę (1/3 * 0,6) + (2/3 * 0,5) = 0,53. Przy podobnych

obliczeniach sprawność drugiego dziecka wynosi 0,57. Wynik odpalenia reguły klasyfikatora, zawsze 0 lub 1, idzie w parze z większością atrybutów, zachowując w ten sposób intuicję, że te wzorce są ważne w wynikach reguł. W typowym systemie klasyfikatorów te dwie nowe reguły, wraz z ich rodzicami, stanowiłyby podzbiór klasyfikatorów do działania systemu w następnym kroku czasowym. Można również zdefiniować operator mutacji. Prosta reguła mutacji polegałaby na losowej zmianie dowolnego wzorca atrybutu na inny prawidłowy wzorzec atrybutu; na przykład 5 może zostać zmutowane na 1, 2, 3, 4 lub #. Ponownie, jak zauważono w naszej dyskusji o GA, operatory mutacji są postrzegane jako wymuszające różnorodność w poszukiwaniu klasyfikatorów, podczas gdy crossover próbuje zachować i zbudować nowe dzieci z udanych fragmentów wzorców rodzicielskich. Nasz przykład był prosty i przeznaczony przede wszystkim do zilustrowania głównych elementów systemu klasyfikatora. W rzeczywistym systemie może zostać uruchomionych więcej niż jedna reguła, a każda z nich przekazuje wyniki do pamięci produkcyjnej. Często istnieje schemat podatkowy, który uniemożliwia klasyfikatorowi zajęcie zbyt dużego miejsca w procesie rozwiązywania problemu, obniżając jego przydatność za każdym razem, gdy wygrywa ofertę. Nie zilustrowaliśmy również algorytmu brygady wiaderkowej, który w różny sposób nagradza reguły wspierające pomyślne komunikaty wyjściowe do środowiska. Ponadto operatorzy genetyczni zwykle nie przerabiają klasyfikatorów przy każdej operacji systemu. Zamiast tego istnieje pewien ogólny parametr dla każdej aplikacji, który decyduje, być może, o analizie informacji zwrotnej ze środowiska, kiedy klasyfikatory powinny być ocenione i zastosowane operatory genetyczne. Wreszcie, nasz przykład pochodzi z systemów klasyfikacyjnych, które zaproponował Holland z University of Michigan. Podejście Michigan można postrzegać jako obliczeniowy model poznania, w którym wiedza (klasyfikatory) jednostki poznawczej jest wystawiona na działanie reagującego środowiska iw rezultacie podlega modyfikacjom w czasie. Oceniamy sukces całego systemu w czasie, przy czym znaczenie poszczególnych klasyfikatorów jest minimalne. Badano również alternatywne systemy klasyfikatorów, w tym prace na Uniwersytecie w Pittsburghu. Klasyfikator Pittsburgh koncentruje się na roli poszczególnych reguł w tworzeniu nowych generacji klasyfikatorów. Podejście to realizuje model uczenia indukcyjnego zaproponowany przez Michalskiego. W następnej sekcji rozważymy inną i szczególnie ekscytującą aplikację dla GA, ewolucję programów komputerowych.

12.2.2 Programowanie z operatorami genetycznymi

W kilku ostatnich podsekcjach widzieliśmy, jak GA są stosowane do coraz większych struktur reprezentacyjnych. To, co zaczęło się jako transformacja genetyczna na ciągach bitowych, przekształciło się w operacje na if. . . następnie . . . zasady. Można w naturalny sposób zapytać, czy można zastosować techniki genetyczne i ewolucyjne do produkcji innych narzędzi obliczeniowych na większą skalę. Były tego dwa główne przykłady: generowanie programów komputerowych i ewolucja systemów maszyn skończonych. Koza zasugerował, że udany program komputerowy może ewoluować poprzez kolejne zastosowania operatorów genetycznych. W programowaniu genetycznym adaptowane struktury są hierarchicznie zorganizowanymi segmentami programów komputerowych. Algorytm uczenia się utrzymuje populację programów kandydujących. Dopasowanie programu będzie mierzone jego zdolnością do rozwiązywania zestawu zadań, a programy są modyfikowane przez zastosowanie krzyżowania i mutacji w poddrzewach programu. Programowanie genetyczne przeszukuje przestrzeń programów komputerowych o różnej wielkości i złożoności; w rzeczywistości przestrzeń poszukiwań jest przestrzenią wszystkich możliwych programów komputerowych złożoną z funkcji i symboli terminali odpowiednich dla dziedziny problemu. Podobnie jak w przypadku wszystkich uczących się genetyki, to wyszukiwanie jest przypadkowe, w dużej mierze ślepe, a jednocześnie zaskakująco skuteczne. Programowanie genetyczne rozpoczyna się od początkowej populacji losowo generowanych programów składających się z odpowiednich fragmentów programu. Elementy te, odpowiednie dla dziedziny problemowej, mogą składać się ze standardowych operacji arytmetycznych,

innych powiązanych operacji programistycznych i funkcji matematycznych, a także funkcji logicznych i specyficznych dla dziedziny. Komponenty programu zawierają elementy danych typowych typów: logiczne, całkowite, zmiennoprzecinkowe, wektorowe, symboliczne lub wielowartościowe. Po inicjalizacji tysiące programów komputerowych jest hodowanych genetycznie. Produkcja nowych programów wiąże się z zastosowaniem operatorów genetycznych. Do produkcji programów komputerowych należy dostosować algorytmy krzyżowania, mutacji i innych algorytmów hodowlanych. Wkrótce zobaczymy kilka przykładów. Adekwatność każdego nowego programu jest następnie określana poprzez sprawdzenie, jak dobrze działa on w określonym środowisku problemowym. Charakter środka sprawności różni się w zależności od dziedziny problemu. Każdy program, który dobrze sobie radzi w tym zadaniu fitness, przetrwa, pomagając w wychowaniu dzieci następnego pokolenia. Podsumowując, programowanie genetyczne obejmuje sześć komponentów, z których wiele jest bardzo podobnych do wymagań dla GA:

1. Zbiór struktur podlegających transformacji przez operatory genetyczne.

2. Zestaw początkowych struktur dopasowanych do dziedziny problemowej.

3. Miara przydatności, ponownie zależna od dziedziny, do oceny struktur.

4. Zbiór operatorów genetycznych do transformacji struktur.

5. Parametry i opisy stanów, które opisują członków każdego pokolenia.

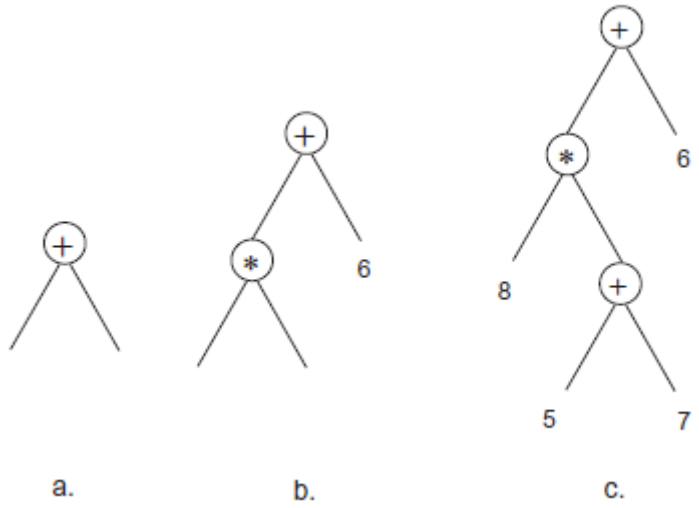
6. Zestaw warunków zakończenia.

W kolejnych akapitach szczegółowo omówimy każdy z tych tematów. Programowanie genetyczne manipuluje hierarchicznie zorganizowanymi modułami programu. Lisp był (i nadal pozostaje) podstawową reprezentacją komponentów języka programowania: Koza reprezentuje segmenty programu jako wyrażenia symboli Lispa lub wyrażenia s. Operatory genetyczne manipulują wyrażeniami s. W szczególności operatorzy odwzorowują struktury drzewiaste wyrażeń s (segmenty programu Lisp) na nowe drzewa (nowe segmenty programu Lisp). Chociaż to wyrażenie s jest podstawą wczesnych prac Kozy, inni badacze w większym stopniu zastosowali to podejście do różnych języków / paradygmatów programowania. Programowanie genetyczne stworzy użyteczne programy, biorąc pod uwagę, że dostępne są fragmenty atomowe i możliwe do oceny predykaty domeny problemowej. Kiedy ustawiamy domenę do generowania programów wystarczających do rozwiązania zestawu problemów, musimy najpierw przeanalizować, jakie terminale są wymagane dla jednostek w jej rozwiązaniu, a także jakie funkcje są niezbędne do wytworzenia tych terminali. Jak zauważa Koza „… użytkownik programowania genetycznego powinien wiedzieć… że pewien zestaw funkcji i terminali, które dostarcza, może przynieść rozwiązanie problemu”. Aby zainicjować struktury do adaptacji przez operatory genetyczne, musimy utworzyć dwa zbiory: F, zbiór funkcji i T, zbiór wartości końcowych wymaganych dla domeny. F może być tak proste, jak {+, *, -, /} lub może wymagać bardziej złożonych funkcji, takich jak sin (X), cos (X) lub funkcje dla operacji macierzowych. T może być liczbami całkowitymi, liczbami rzeczywistymi, macierzami lub bardziej złożonymi wyrażeniami. Symbole w T muszą być zamknięte pod funkcjami określonymi w F.

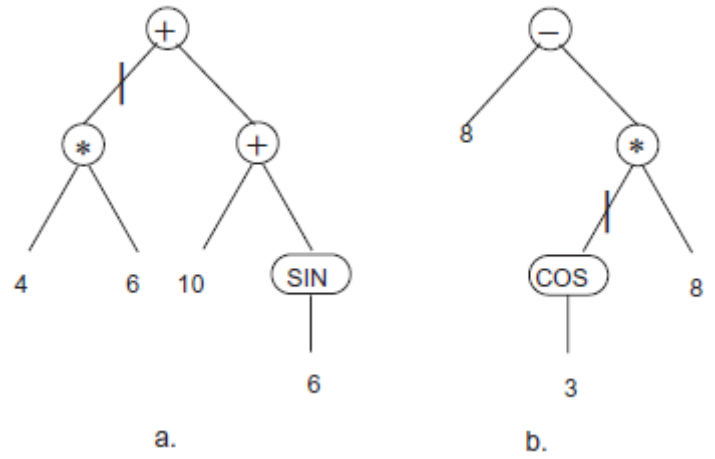
Następnie generowana jest populacja początkowych „programów” poprzez losowe wybieranie elementów ze związku zbiorów F i T. Na przykład, jeśli zaczniemy od wybrania elementu T, mamy zdegenerowane drzewo pojedynczego węzła głównego. Co ciekawsze, kiedy zaczniemy od elementu z F, powiedzmy +, otrzymamy węzeł główny drzewa z dwojgiem potencjalnych dzieci. Załóżmy, że inicjator następnie wybiera * (z dwojgiem potencjalnych dzieci) z F jako pierwsze dziecko, a następnie

terminal 6 z T jako drugie dziecko. Kolejny losowy wybór może dać terminal 8, a następnie funkcję + z F. Załóżmy, że kończy się wybierając 5 i 7 z T.

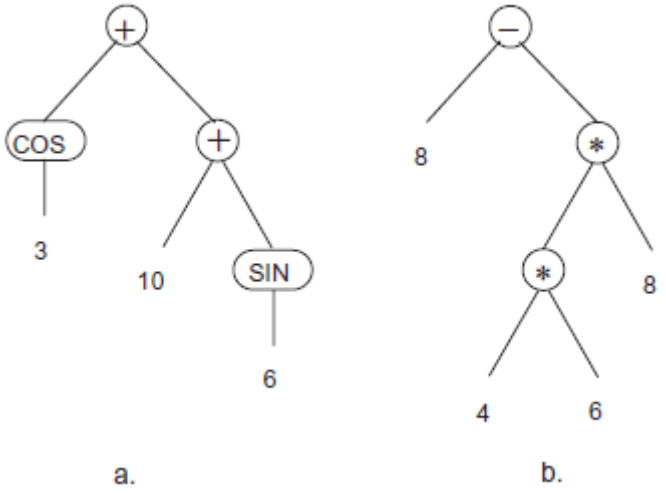
Program, który stworzyliśmy losowo, jest przedstawiony na rysunku 4.



Rysunek 4a przedstawia drzewo po pierwszym wybraniu +, 15.4b po wybraniu terminala 6 i 15.4c programu końcowego. Tworzona jest populacja podobnych programów w celu zainicjowania procesu programowania genetycznego. Zestawy ograniczeń, takie jak maksymalna głębokość rozwoju programów, mogą pomóc w przycinaniu tej populacji. Opis tych ograniczeń, a także różnych metod generowania populacji początkowych, można znaleźć u Kozy. Dyskusja do tej pory dotyczy kwestii reprezentacji (wyrażeń s) i zestawu struktur drzewiastych niezbędnych do zainicjowania sytuacji ewolucji programu. Następnie wymagamy miernika sprawności dla populacji programów. Miara sprawności zależy od dziedziny problemu i zwykle składa się z zestawu zadań, do których muszą się zmierzyć wyewoluowane programy. Sam miernik sprawności jest funkcją tego, jak dobrze każdy program radzi sobie z tymi zadaniami. Prosty surowy wynik sprawności dodałby różnice między tym, co wygenerował program, a wynikami wymaganymi przez rzeczywiste zadanie z domeny problemowej. W związku z tym surową przydatność można postrzegać jako sumę błędów w zestawie zadań. Oczywiście możliwe są inne środki sprawności. Dopasowanie znormalizowane dzieli surową przydatność przez całkowitą sumę możliwych błędów, a tym samym umieszcza wszystkie miary przystosowania w zakresie od 0 do 1. Normalizacja może mieć przewagę, gdy próbuje się dokonać wyboru z dużej populacji programów. Środek sprawności może również obejmować dostosowanie do rozmiaru programu, na przykład w celu nagradzania mniejszych, bardziej oszczędnych programów. Operatory genetyczne w programach obejmują zarówno transformacje samego drzewa, jak i wymianę struktur między drzewami. Koza (1992) opisuje pierwotne przemiany jako reprodukcję i krzyżowanie. Reprodukcja po prostu wybiera programy z obecnej generacji i kopiuje je (niezmienione) do następnej generacji. Crossover wymienia poddrzewa między drzewami reprezentującymi dwa programy. Na przykład, załóżmy, że pracujemy z dwoma programami nadrzędnymi z rysunku 5 i że losowe punkty są oznaczone | w rodzicu a i rodzicu b są wybierane do krzyżowania.



Wynikowe elementy potomne pokazano na rysunku 6.



Crossover może być również użyty do przekształcenia samotnego rodzica poprzez zamianę dwóch poddrzew od tego rodzica. Dwóch identycznych rodziców może stworzyć różne dzieci z losowo wybranymi punktami krzyżowania. Rdzeń programu można również wybrać jako punkt przecięcia. Istnieje wiele wtórnych i znacznie rzadziej używanych transformacji genetycznych drzew programu. Należą do nich mutacja, która po prostu wprowadza losowe zmiany w strukturach programu. Na przykład zastąpienie wartości końcowej inną wartością lub poddrzewem funkcji. Transformacja permutacyjna, podobnie jak operator inwersji na łańcuchach, działa również na pojedynczych programach, na przykład wymieniając symbole terminali lub poddrzewa. Stan rozwiązania odzwierciedla aktualna generacja programów. Nie ma zapisów dotyczących cofania się ani żadnej innej metody omijania krajobrazu fitness. Pod tym względem programowanie genetyczne jest bardzo podobne do algorytmu wspinania się po wzgórzu. Paradygmat programowania genetycznego odpowiada naturze, ponieważ ewolucja nowych programów jest procesem ciągłym. Niemniej jednak, z powodu braku nieskończonego czasu i obliczeń, ustalane są warunki zakończenia. Zwykle są one funkcją zarówno sprawności programu, jak i zasobów obliczeniowych. Fakt, że programowanie genetyczne jest techniką obliczeniowego generowania programów komputerowych, umieszcza ją w tradycji badań nad programowaniem automatycznym. Od najwcześniejszych dni sztucznej inteligencji naukowcy pracowali nad automatycznym tworzeniem programów komputerowych na podstawie fragmentarycznych informacji . Programowanie genetyczne można postrzegać jako kolejne narzędzie

w tej ważnej dziedzinie badawczej. Kończymy tę sekcję prostym przykładem programowania genetycznego zaczerpniętym z Mitchella.

PRZYKŁAD 3.2.1: EWOLUCJA PROGRAMU TRZECIEGO PRAWA RUCHU PLANETARNEGO KEPLERA

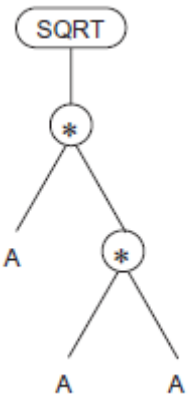
Koza opisuje wiele zastosowań programowania genetycznego do rozwiązywania interesujących problemów, ale większość z tych przykładów jest obszerna i zbyt złożona dla naszych obecnych celów. Mitchell stworzył jednak prosty przykład ilustrujący wiele koncepcji programowania genetycznego. Trzecie prawo ruchu planetarnego Keplera opisuje związek funkcjonalny między okresem orbitalnym planety P a jej średnią odległością A od Słońca. Funkcja trzeciego prawa Keplera, przy czym c jest stałą, to:

$P^2 = cA^3$

Jeśli przyjmiemy, że P jest wyrażone w latach ziemskich, a A w jednostkach średniej odległości Ziemi od Słońca, to c = 1. Wyrażenie s tej zależności to:

P = (sqrt (* A (* A A)))

Tak więc program, który chcemy rozwijać, jest reprezentowany przez strukturę drzewa na rysunku 7.



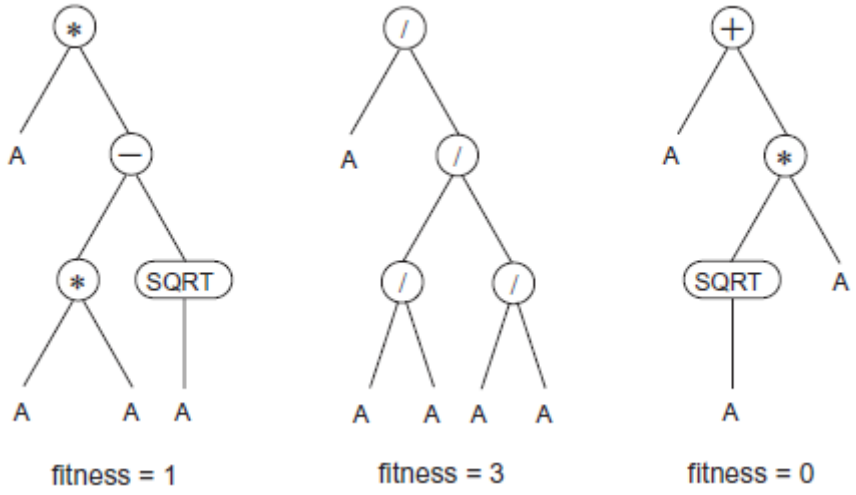
Wybór zestawu symboli terminala w tym przykładzie jest prosty; jest to pojedyncza wartość rzeczywista podana przez A. Zbiór funkcji mógłby być równie prosty, powiedzmy {+, -, *, /, sq, sqrt}. Następnie utworzymy początkową losową populację programów. Początkowa populacja może obejmować:

(* A (- (* A A) (sqrt A))) przydatność: 1

(/ A (/ (/ A A) (/ A A))) sprawność: 3

(+ A (* (sqrt A) A)) przydatność: 0

(Wkrótce wyjaśnimy załączone środki sprawności). Jak zauważono wcześniej w tej sekcji, ta inicjująca populacja często ma a priori ograniczenia zarówno rozmiaru, jak i głębokości wyszukiwania, biorąc pod uwagę wiedzę o problemie. Te trzy przykłady są opisane przez drzewa programów na rysunku 8



fitness = 1 fitness = 3 fitness = 0

Następnie określamy zestaw testów dla populacji programów. Załóżmy, że znamy pewne dane planetarne, które chcemy wyjaśnić w naszym wyewoluowanym programie. Na przykład, mamy dane planetarne w tabeli 3, zaczerpnięte z Urey, która zawiera zestaw punktów danych, które ewoluujące programy muszą wyjaśnić.

Planet	A (input)	P (output)
Venus	0.72	0.61
Earth	1.0	1.0
Mars	1.52	1.87
Jupiter	5.2	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

Ponieważ miara sprawności jest funkcją punktów danych, które chcemy wyjaśnić, definiujemy dopasowanie jako liczbę wyników programu, które mieszczą się w granicach 20% prawidłowych wartości wyjściowych. Używamy tej definicji do tworzenia miar sprawności trzech programów przedstawionych na rysunku 12.8. Czytelnikowi pozostaje stworzenie większej liczby członków tej początkowej populacji, zbudowanie operatorów krzyżowania i mutacji, które mogą tworzyć kolejne generacje programów, oraz określenie warunków zakończenia.

12.3 Sztuczne życie i uczenie się oparte na społeczeństwie

Wcześniej w tym rozdziale opisaliśmy uproszczoną wersję „Gry w życie”. Ta gra, najskuteczniej pokazana w komputerowych symulacjach wizualnych, w których kolejne generacje szybko się zmieniają i ewoluują na ekranie, ma bardzo prostą specyfikację. Po raz pierwszy została zaproponowana jako gra planszowa przez matematyka Johna Hortona Conwaya i rozsławiona dzięki omówieniu jej przez Martina Gardnera w Scientific American. Gra w życie to prosty przykład modelu obliczeniowego zwanego automatami komórkowymi (CA). Automaty komórkowe to rodziny prostych maszyn o skończonych stanach, które wykazują interesujące, wyłaniające się zachowania poprzez interakcje w populacji.

DEFINICJA

MASZYNA SKOŃCZONA (FSM) LUB AUTOMAT KOMÓRKOWY

1. Zestaw, który nazwałem alfabetem wejściowym.

2. Zbiór S stanów, w których może znajdować się automat.

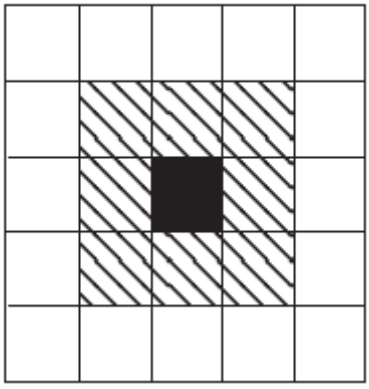
3. Wyznaczony stan s0, stan początkowy.

4. Następna funkcja stanu N: S x I → S, która przypisuje każdemu kolejny stan uporządkowaną parą składająca się z aktualnego stanu i wejścia prądowego.

Dane wyjściowe maszyny skończonej, jak przedstawiono wcześniej, są funkcją jej obecnego stanu i wartości wejściowych. Automaty komórkowe sprawiają, że wejście do obecnego stanu jest funkcją jego „sąsiednich" stanów. Zatem stan w czasie (t + 1) jest funkcją jego obecnego stanu i stanu jego sąsiadów w czasie t. To właśnie dzięki tym interakcjom z sąsiadami zbiory automatów komórkowych osiągają znacznie bogatsze zachowania niż proste maszyny skończone. Ponieważ wynik wszystkich stanów systemu jest funkcją ich sąsiednich stanów, możemy opisać ewolucję zbioru sąsiednich FSM jako adaptację i uczenie się oparte na społeczeństwie. W przypadku towarzystw opisanych w tym rozdziale nie ma jednoznacznej oceny sprawności poszczególnych członków. Sprawność jest wynikiem interakcji w populacji, interakcji, które mogą doprowadzić do „śmierci" poszczególnych automatów. Sprawność fizyczna jest nieodzowna w przetrwaniu jednostek z pokolenia na pokolenie. Uczenie się między automatami komórkowymi odbywa się zazwyczaj bez nadzoru; jak to ma miejsce w naturalnej ewolucji, adaptacja jest kształtowana przez działania innych, współewoluujących członków populacji. Globalny lub zorientowany na społeczeństwo punkt widzenia pozwala również spojrzeć na naukę z ważnej perspektywy. Nie musimy już skupiać się wyłącznie na jednostce, ale raczej dostrzegamy niezmienności i regularności pojawiające się w społeczeństwie jako całości. Jest to ważny aspekt badań Crutchfielda-Mitchella przedstawionych w sekcji 12.3.2. Wreszcie, w przeciwieństwie do nadzorowanego uczenia się, ewolucja nie musi być „zamierzona". Oznacza to, że społeczeństwo agentów nie musi być postrzegane jako „idące gdzieś", powiedzmy do jakiegoś punktu „omega". Mieliśmy tendencję do zbieżności, kiedy używaliśmy wyraźnych miar sprawności we wcześniejszych sekcjach tego rozdziału. Ale jak podkreśla Stephen Jay Gould, ewolucja nie musi być postrzegana jako „ulepszanie" rzeczy, a raczej sprzyja ona przetrwaniu. Jedynym sukcesem jest dalsze istnienie, a wzorce, które się wyłaniają, są wzorami społeczeństwa.

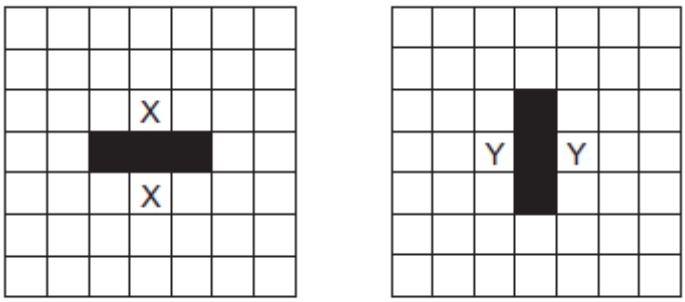
12.3.1 „Gra w życie"

Rozważmy prostą dwuwymiarową siatkę lub planszę do gry z rysunku 9.



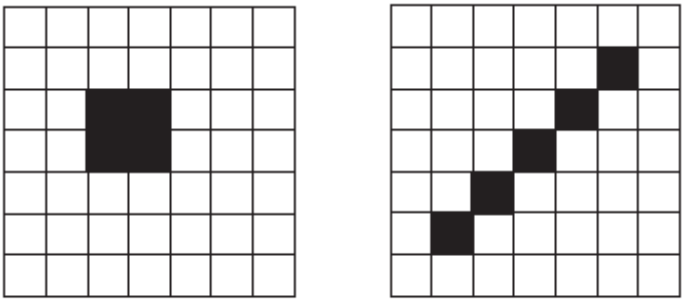
Tutaj mamy jeden kwadrat „zajęty" - o wartości bitu 1 - w kolorze czarnym, z ośmioma najbliższymi sąsiadami zaznaczonymi na szaro. Tablica jest przekształcana w okresach czasu, gdzie stan każdego kwadratu w czasie t + 1 jest funkcją jego stanu i stanu wskazanych najbliższych sąsiadów w czasie t.

Trzy proste zasady mogą napędzać ewolucję w grze: Po pierwsze, jeśli na jakimkolwiek polu, zajętym lub nie, są zajęte dokładnie trzy najbliższe sąsiadki, zostanie ono zajęte w następnym okresie. Po drugie, jeśli na jakimkolwiek zajętym polu są zajęte dokładnie dwóch najbliższych sąsiadów, zostanie ono zajęte w następnym okresie. Wreszcie we wszystkich innych sytuacjach plac nie będzie zajęty w następnym okresie. Jedną z interpretacji tych reguł jest to, że dla każdego pokolenia lub okresu życie w dowolnym miejscu, tj. To, czy kwadrat jest zajęty (ma wartość stanu 1), jest wynikiem życia własnego, jak również jego sąsiadów w okresie Poprzednia generacja. W szczególności zbyt gęsta populacja sąsiadujących sąsiadów (więcej niż trzech) lub zbyt mała populacja sąsiednich (mniej niż dwie) w dowolnym okresie nie pozwoli na życie następnemu pokoleniu. Rozważmy na przykład stan życia przedstawiony na rysunku 10a. Tutaj dokładnie dwa kwadraty, oznaczone x, mają dokładnie trzech zajętych sąsiadów. W następnym cyklu życia zostanie utworzony rysunek 10b. Tutaj znowu są dokładnie dwa kwadraty, wskazane przez y, z dokładnie trzema zajętymi sąsiadami. Można zauważyć, że stan świata będzie się zmieniać między rysunkami 10a i 10b.



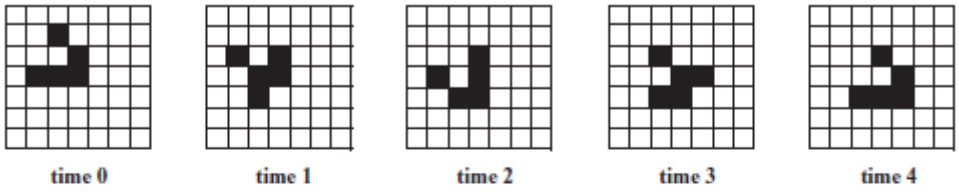
a. b.

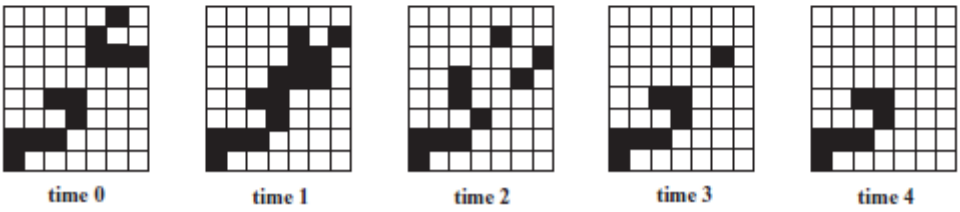
Czytelnik może określić, jaki będzie następny stan dla rysunków 11a i 11b oraz zbadać inne możliwe konfiguracje „świata”.



a. b.

Poundstone opisuje niezwykłą różnorodność i bogactwo struktur, które mogą pojawić się w grze życia, takich jak szybowce, wzory komórek poruszających się po świecie poprzez powtarzające się cykle zmian kształtu, jak pokazano na rysunku 12.





Ze względu na ich zdolność do tworzenia bogatych zachowań zbiorowych poprzez interakcje prostych komórek, automaty komórkowe okazały się potężnym narzędziem do badania matematyki wyłaniania się życia z prostych, nieożywionych składników. Życie sztuczne definiuje się jako życie stworzone raczej przez ludzki wysiłek niż przez naturę. Jak widać w poprzednim przykładzie, sztuczne życie ma silny smak „oddolny"; to znaczy, atomy systemu życia są definiowane i składane, a ich fizyczne interakcje „wyłaniają się". Regularności tej formy życia są uchwycone przez reguły skończonej maszyny stanowej. Ale jak można wykorzystać konstrukcje a-life? Na przykład w biologii zbiór żywych istot tworzonych przez naturę, tak złożony i różnorodny, jak tylko bywa, jest zdominowany przez przypadek i historyczną przygodność. Ufamy, że istnieją logiczne prawidłowości w tworzeniu tego zbioru, ale nie ma takiej potrzeby i jest mało prawdopodobne, że odkryjemy wiele z całkowitych możliwych prawidłowości, gdy ograniczymy nasz pogląd do zbioru bytów biologicznych, które faktycznie występują w naturze. zapewnia. Konieczne jest zbadanie pełnego zestawu możliwych prawidłowości biologicznych, z których część mogła zostać wyeliminowana przez przypadek historyczny. Zawsze możemy się zastanawiać, jak wyglądałby obecny świat, gdyby istnienie dinozaurów nie zostało definitywnie zakończone. Aby mieć teorię tego, co rzeczywiste, konieczne jest zrozumienie granic tego, co możliwe. Oprócz zdecydowanych wysiłków antropologów i innych naukowców, aby wypełnić luki w wiedzy o naszej rzeczywistej ewolucji, trwają spekulacje na temat ponownego przedstawienia historii samej ewolucji. Co by się stało, gdyby ewolucja rozpoczęła się w innych warunkach początkowych? A co by było, gdyby w naszym fizycznym i biologicznym otoczeniu wystąpiły alternatywne „wypadki" interwencji? Co może się pojawić? Co pozostanie niezmienne? Ścieżka ewolucyjna, która faktycznie miała miejsce na Ziemi, jest tylko jedną z wielu możliwych trajektorii. Na niektóre z tych pytań można by odpowiedzieć, gdybyśmy mogli wygenerować kilka z wielu biologiczne, które są możliwe. Technologia A-life nie jest tylko artefaktem dziedzin obliczeniowych lub biologicznych. Naukowcy z tak różnych dziedzin, jak chemia i farmakologia, zbudowali syntetyczne artefakty, wiele z nich związanych ze znajomością rzeczywistych bytów istniejących w naszym świecie. Na przykład w dziedzinie chemii badania nad budową materii i wieloma związkami dostarczanymi przez naturę doprowadziły do analizy tych związków, ich części składowych i wiązań. Ta analiza i rekombinacja doprowadziły do powstania wielu związków, które nie istnieją naturalnie. Nasza wiedza o elementach budulcowych przyrody doprowadziła nas do naszych własnych syntetycznych wersji, łączących elementy rzeczywistości w nowe i odmienne wzory. Dzięki tej dokładnej analizie naturalnych związków chemicznych dochodzimy do pewnego zrozumienia zestawu możliwych związków. Jednym z narzędzi do zrozumienia możliwych światów jest symulacja i analiza efektów ruchu i interakcji opartych na społeczeństwie. Mamy na to proste przykłady w Game of Life. Sekwencja cykli czasowych pokazana na rysunku 12 realizuje

wspomniany wcześniej szybowiec. Szybowiec przemierza przestrzeń gry, poruszając się po niewielkiej liczbie wzorów. Jego działanie jest proste, ponieważ w czterech przedziałach czasowych przesuwa się do nowej lokalizacji o jeden rząd dalej w prawo i jeden wiersz bliżej dolnej części siatki. Interesującym aspektem gry w życie jest to, że istoty takie jak szybowiec trwają do czasu interakcji z innymi członkami społeczeństwa; co się wtedy stanie, może być trudne do zrozumienia i przewidzenia. Na przykład na rysunku 13 widzimy sytuację, w której dwa szybowce wynurzają się i atakują. Po czterech okresach szybowiec poruszający się w dół i w lewo jest „konsumowany" przez drugą jednostkę. Warto zauważyć, że nasze opisy ontologiczne, to znaczy użycie przez nas terminów takich jak „byt", „migające światło", „szybowiec", „pochłonięty", odzwierciedla nasze własne antropocentryczne uprzedzenia w postrzeganiu form życia i interakcji, czy to sztucznych albo nie. Nadawanie nazw prawidłowościom pojawiającym się w naszych strukturach społecznych jest bardzo ludzkie.

12.3.2 Programowanie ewolucyjne

„Gra w życie" to intuicyjny, bardzo opisowy przykład automatów komórkowych. Możemy uogólnić naszą dyskusję o automatach komórkowych, charakteryzując je jako maszyny skończone. Omawiamy teraz społeczeństwa powiązanych FSM i analizujemy je jako powstające jednostki. To badanie jest czasami nazywane programowaniem ewolucyjnym. Historia programowania ewolucyjnego sięga początków samych komputerów. John von Neumann w serii wykładów w 1949 r. Zbadał, jaki poziom złożoności organizacyjnej jest wymagany do wystąpienia samoreplikacji (Burks 1970). Burks cytuje cel von Neumanna jako „… nie próbowanie symulowania autoreprodukcji naturalnego systemu na poziomie genetyki i biochemii. Chciał abstrahować od naturalnego problemu reprodukcji jego logicznej formy ". Usuwając szczegóły chemiczne, biologiczne i mechaniczne, von Neumann był w stanie przedstawić podstawowe wymagania dla samoreplikacji. von Neumann zaprojektował (jak nigdy dotąd) samoodtwarzający się automat składający się z dwuwymiarowego układu komórkowego zawierającego dużą liczbę indywidualnych 29-stanowych automatów, gdzie następny stan każdego automatu był funkcją jego obecnego stanu oraz stany jego czterech bezpośrednich sąsiadów. Co ciekawe, von Neumann zaprojektował swój samoreplikujący się automat, który, jak się szacuje, zawiera co najmniej 40000 komórek, miał funkcjonalność Uniwersalnej Maszyny Turinga. To uniwersalne urządzenie obliczeniowe miało również konstrukcję uniwersalną, w tym sensie, że było w stanie odczytać taśmę wejściową, zinterpretować dane na taśmie i za pomocą ramienia konstrukcyjnego zbudować konfigurację opisaną na taśmie w niezajętej części przestrzeń komórkowa. Umieszczając na taśmie opis samego automatu konstruującego, von Neumann stworzył samoodtwarzający się automat (Arbib 1966). Później Codd (1968) zredukował liczbę stanów wymaganych dla obliczeniowo uniwersalnego, samoodtwarzającego się automatu z 29 do 8, ale wymagał szacunkowo 100 000 000 komórek dla pełnego projektu. Później Devore uprościł maszynę Codda, aby zajmowała tylko około 87 500 komórek. W dzisiejszych czasach Langton stworzył samoreplikujący się automat, bez obliczeniowej uniwersalności, w którym każda komórka miała tylko osiem stanów i zajmowała tylko 100 komórek. Aktualne opisy tych wysiłków badawczych można znaleźć w materiałach z konferencji a-life. Tak więc formalna analiza samoreplikujących się maszyn ma głębokie korzenie w teorii obliczeń. Być może nawet bardziej ekscytujące wyniki są ukryte w badaniach empirycznych nad formami życia. O sukcesie tych programów nie świadczy jakaś funkcja sprawności a priori, ale raczej prosty fakt, że mogą one przetrwać i powielać się. Ich znakiem sukcesu jest przetrwanie. Po ciemnej stronie doświadczyliśmy spuścizny wirusów komputerowych i robaków, które są w stanie przedostać się do obcych hostów, replikować się (zwykle niszcząc wszelkie informacje w pamięci wymaganej do replikacji) i przenosić się do kolejnych obcych hostów. . Kończymy tę sekcję, omawiając kilka projektów badawczych, które można traktować jako przykłady obliczeń a-life. Ostatnia część rozdziału 12 to szczegółowy przykład wyłaniających się obliczeń Melanie Mitchell i jej współpracowników z Instytutu Sante Fe. Najpierw przedstawiamy dwa projekty omówione już w naszych wcześniejszych rozdziałach: Rodney Brooks z

MIT oraz Nils Nilsson i jego studenci ze Stanford. We wcześniejszych prezentacjach praca Brooksa znajdowała się pod ogólnym tytułem reprezentacji alternatywnych, a praca Nilssona, w temacie planowania. W kontekście teraźniejszości , przeformułowujemy ich prace w kontekście sztucznego życia i pojawiających się zjawisk. Rodney Brooks z MIT zbudował program badawczy oparty na założeniu a-life, a mianowicie, że inteligencja wyłania się poprzez interakcje wielu prostych autonomicznych agentów Brook opisuje swoje podejście jako „inteligencję bez reprezentacji", budując serię robotów zdolnych do wykrywania przeszkód i poruszania się po biurach i korytarzach MIT. Opierając się na architekturze subsumpcji, agenci mogą wędrować, badać i unikać innych obiektów. Inteligencja tego systemu jest artefaktem prostej organizacji i ucieleśnionych interakcji z otoczeniem. Brooks stwierdza: „Łączymy maszyny o skończonych stanach w warstwy kontroli. Każda warstwa jest zbudowana na wierzchu istniejących warstw. Warstwy niższego poziomu nigdy nie opierają się na istnieniu warstw wyższego poziomu ". Nils Nilsson i jego uczniowie opracowali program teleoreaktywny (TR) do kontroli agentów, program, który kieruje agenta w kierunku celu w sposób, który stale uwzględnia zmieniające się okoliczności środowiskowe . Ten program działa bardzo podobnie do systemu produkcyjnego, ale obsługuje również działanie trwające lub działanie, które ma miejsce w dowolnych okresach czasu, na przykład do ... Tak więc, w przeciwieństwie do zwykłych systemów produkcyjnych, warunki muszą być stale oceniane, a akcja związana z aktualnie najwyższym prawdziwym warunkiem jest zawsze wykonywana. W celu uzyskania dalszych szczegółów, zainteresowany czytelnik jest skierowany do Nilsson, Benson , Benson i Nilsson , Klein i inni. Komisja Europejska wspiera projekt PACE, Programmable Artificial Cell Evolution. Dzięki tym badaniom powstała nowa generacja syntetycznych ogniw chemicznych. Skupiamy się na tym, że życie ewoluuje wokół żywych systemów obliczeniowych, które same się organizują i samonaprawiają. Całe sztuczne komórki również rozmnażają się samoczynnie, są zdolne do ewolucji, mogą zachować swoją złożoną strukturę przy użyciu prostszych zasobów środowiskowych. Więcej szczegółów można znaleźć w witrynie internetowej PACE http://www.istpace.org/research_overview/artificial_cells_in_pace.html.

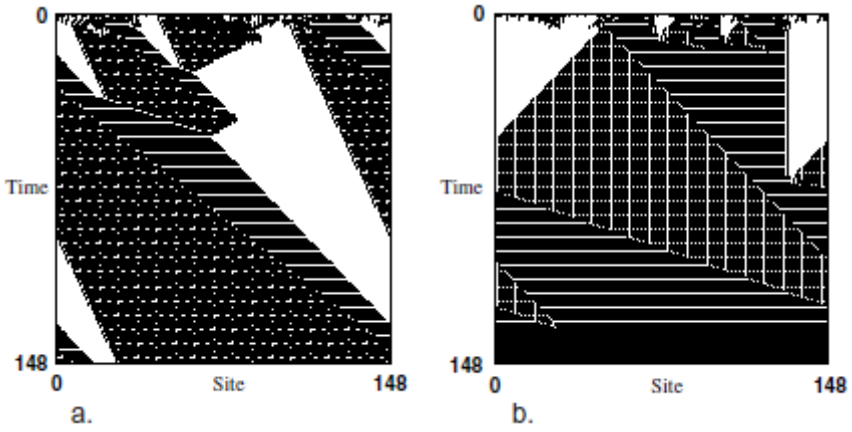
Wysiłki badawcze Johna Koza w zakresie programowania genetycznego na Uniwersytecie Stanforda nadal publikują wiele sukcesów. Ostatnie postępy obejmują automatyczny rozwój obwodów i sterowników, syntezę topologii obwodów, wymiarowanie, rozmieszczenie i routing, a także inne artefakty inżynieryjne. Odniesienia można znaleźć u Kozy.

Te cztery wysiłki badawcze są próbkami z bardzo dużej populacji projektów badawczych opartych na automatach. Te projekty są zasadniczo eksperymentalne. Zadają pytania dotyczące świata przyrody. Świat przyrody reaguje przetrwaniem i rozwojem udanych algorytmów, a także unicestwieniem systemu niezdolnego do adaptacji. Na koniec rozważymy badania przeprowadzone przez Santa Fe Institute: studium przypadku w wyłonieniu.

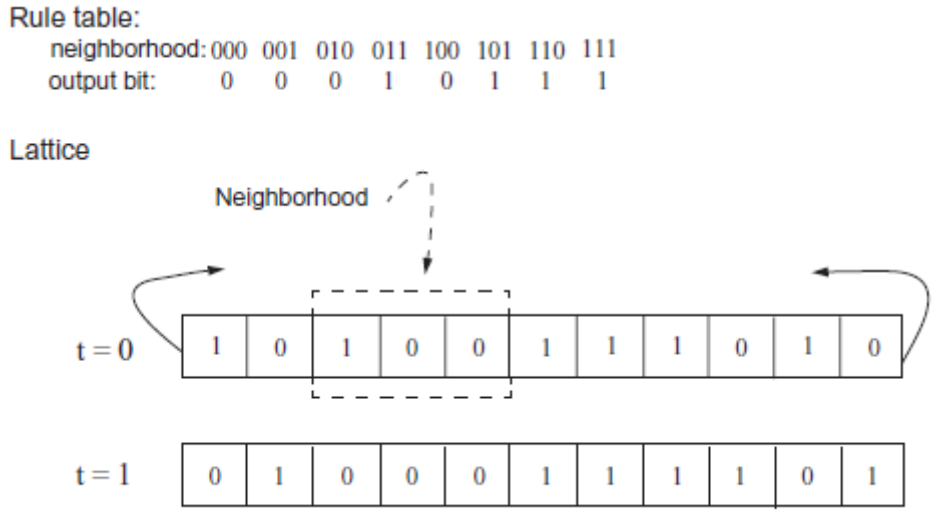
12.3.3 Studium przypadku w Emergence

Crutchfield i Mitchell badają zdolność ewolucji i interakcji w ramach prostych systemów do tworzenia relacji zbiorowego przetwarzania informacji wyższego poziomu. Ich badania stanowią przykład pojawienia się globalnych obliczeń w systemie przestrzennym składającym się z rozproszonych i oddziałujących lokalnie procesorów. Termin „emergent computation" opisuje pojawienie się globalnych struktur przetwarzania informacji w tych systemach. Celem badań Crutchfielda i Mitchella jest opisanie architektury i mechanizmów wystarczających do ewolucji i wspierania metod obliczeń wyłaniających się. Mówiąc konkretnie, automat komórkowy (CA) składa się z wielu pojedynczych komórek; w rzeczywistości w każdym automacie z przykładów, które prezentujemy, znajduje się 149 komórek. Te komórki stanu binarnego są rozmieszczone w jednowymiarowej przestrzeni bez globalnej koordynacji. Każda komórka zmienia stan w zależności od własnego stanu i stanów jej dwóch bezpośrednich sąsiadów. CA tworzy dwuwymiarową siatkę, gdy ewoluuje w różnych okresach czasu.

Sieć zaczyna się od początkowego losowo wygenerowanego zestawu N komórek. Na przykładzie z rysunku 12.14 istnieje 149 komórek i 149 etapów ich ewolucji. (Istnieje zerowy okres czasu, a komórki są ponumerowane od 0, 1, ..., 148). Dwa przykłady zachowania tych automatów komórkowych można zobaczyć na diagramach czasoprzestrzennych na rysunku 14.



Na tych diagramach jedynki są podane jako czarne pola, a 0 jako białe pola. Oczywiście różne reguły dotyczące sąsiedztwa komórek spowodują powstanie różnych wzorców na diagramie czasoprzestrzennym CA. Następnie opiszemy zestaw reguł, który określa aktywność komórek tworzących każdy ośrodek certyfikacji. Rysunek 15 przedstawia jednowymiarowy stan binarny najbliższego sąsiada CA, z N = 11 komórkami.



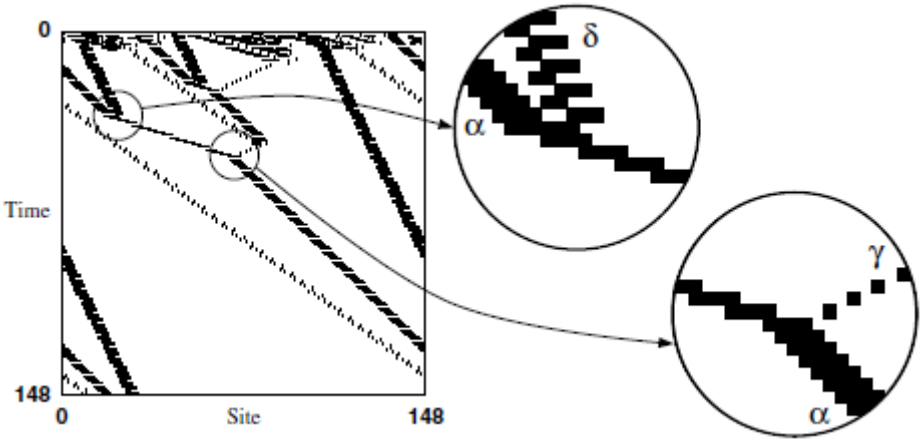
Przedstawiono zarówno siatkę, jak i tabelę reguł do aktualizacji sieci. Krata jest pokazana zmieniając się w jednym kroku czasowym. Krata jest właściwie walcem, z lewym i prawym końcem kraty w każdym przedziale czasowym sąsiadującymi (jest to ważne przy stosowaniu zestawu reguł). Tabela reguł obsługuje zasadę głosowania przez lokalną większość: jeśli lokalne sąsiedztwo trzech komórek ma większość jedynek, to w następnym kroku środkowa komórka staje się jedną; w przeciwnym razie staje się zerem w następnym kroku czasowym. Crutchfield i Mitchell chcą znaleźć ośrodek CA, który wykonuje następujące zbiorcze obliczenia, zwane tutaj większością wygrywa: jeśli początkowa sieć zawiera większość jedynek, CA powinien z czasem ewoluować do wszystkich; w przeciwnym razie powinna ewoluować do samych zer. Używają CA z sąsiedztwami zawierającymi siedem komórek, centralną komórkę z trzema sąsiadami po każdej stronie. Ciekawym aspektem tych badań jest to, że trudno jest zaprojektować regułę CA, która wykonuje obliczenia większości wygranych. W

rzeczywistości w Mitchell i inni wykazali, że prosta reguła „większości głosów” siedmiu sąsiadów nie wykonuje obliczenia „większość wygrywa”. GA służy do wyszukiwania reguły, która to zrobi. Algorytm genetyczny (GA), sekcja 12.1, służy do tworzenia tabel reguł dla różnych eksperymentów z CA. W szczególności GA służy do ewolucji reguł dla jednowymiarowej populacji komórek stanu binarnego, która tworzy każdy CA. Funkcja przystosowania ma na celu nagradzanie tych reguł, które wspierają wynik większościowy dla samego CA. Tak więc z biegiem czasu GA zbudowało zestaw reguł, którego przydatność była funkcją jej ostatecznego sukcesu w egzekwowaniu globalnych reguł większości. Najlepiej przystosowane reguły w populacji zostały wybrane, aby przetrwać i losowo łączone przez krzyżowanie w celu wytworzenia potomstwa, przy czym każde potomstwo podlega niewielkiemu prawdopodobieństwu mutacji. Proces ten był powtarzany przez 100 pokoleń, a przydatność szacowano dla nowego zestawu początkowych komórek w każdym pokoleniu. Pełne szczegóły można znaleźć w Crutchfield i Mitchell. W jaki sposób możemy określić ilościowo wyłaniające się obliczenia wspierane przez bardziej skuteczne CA? Podobnie jak wiele rozszerzonych przestrzennie procesów naturalnych, konfiguracje komórek często organizują się w czasie w regiony przestrzenne, które są dynamicznie jednorodne. W idealnym przypadku analiza i określenie podstawowych prawidłowości powinno być procesem zautomatyzowanym. W rzeczywistości Hanson i Crutchfield stworzyli język dla minimalnego deterministycznego automatu skończonego i używają go do opisu atraktorów-basenów w każdym automacie komórkowym. W tym języku można opisać nasz przykład. Czasami, jak na rysunku 14a, regiony te są oczywiste dla człowieka jako domeny niezmienne, to znaczy regiony, w których powtarza się ten sam wzór. Oznaczymy te domeny jako wartości Λ, a następnie odfiltrujemy niezmienne elementy, aby lepiej opisać interakcje (obliczenia), na które wpływają przecięcia tych domen. Tabela 4

Regular Domains		
$\Lambda^0 = 0^*$	$\Lambda^1 = 1^*$	$\Lambda^2 = (10001)^*$
Particles (Velocities)		
$\alpha \sim \Lambda^1\Lambda^0$ (1)	$\beta \sim \Lambda^0\Lambda^1$ (0)	
$\gamma \sim \Lambda^2\Lambda^0$ (–2)	$\delta \sim \Lambda^0\Lambda^2$ (1/2)	
$\eta \sim \Lambda^2\Lambda^1$ (4/3)	$\mu \sim \Lambda^1\Lambda^2$ (3)	
Interactions		
decay	$\alpha \rightarrow \gamma + \mu$	
react	$\alpha + \delta \rightarrow \mu,\ \eta + \alpha \rightarrow \gamma,\ \mu + \gamma \rightarrow \alpha$	
annihilate	$\eta + \mu \rightarrow \emptyset_1\ ,\ \gamma + \delta \rightarrow \emptyset_0$	

opisuje trzy regiony Λ: Λ0, powtarzające się 0; Λ1, powtarzające się jedynki; i Λ2, powtarzający się wzór 10001. Na rysunku 14a są inne regiony Λ, ale omówimy tylko ten podzbiór. Po odfiltrowaniu niezmiennych elementów domen Λ możemy zobaczyć interakcje między tymi domenami. W tabeli 4 opisujemy oddziaływanie sześciu obszarów Λ, na przykład cząstek na granicach domen Λ1 i Λ0. Granica, na której wszystkie domeny 1 spotyka się ze wszystkimi domenami 0, nazywana jest osadzoną cząstką α. Crutchfield i Mitchell twierdzi, że gromadzenie osadzonych cząstek jest głównym mechanizmem

przenoszenia informacji (lub sygnałów) w długich kontinuach czasoprzestrzennych. Logiczne operacje na tych cząstkach lub sygnałach zachodzą, gdy zderzają się. Zatem zbiór domen, ścian domenowych, cząstek i interakcji cząstek dla CA reprezentuje podstawowe elementy przetwarzania informacji, które są osadzone w zachowaniu CA, czyli składają się na wewnętrzne obliczenia CA. Jako przykład, rysunek 16 przedstawia logikę emergentną z rysunku 14a.



Obszary domeny Λ zostały odfiltrowane pod kątem ich niezmiennej zawartości, aby umożliwić łatwą obserwację cząstek ściany domeny. Każdy z powiększonych obszarów na rysunku 16 przedstawia logikę dwóch oddziałujących ze sobą osadzonych cząstek. Oddziaływanie cząstek $\alpha + \delta \rightarrow \mu$, pokazane w prawym górnym rogu, realizuje logikę odwzorowania konfiguracji przestrzennej reprezentującej sygnały α i δ na sygnał μ. Podobne szczegóły przedstawiono dla interakcji cząstek $\mu + \gamma \rightarrow \alpha$, która odwzorowuje konfigurację reprezentującą μ i γ na sygnał α. Bardziej kompletną listę interakcji cząstek z rysunku 12.16 można znaleźć w tabeli 4. Podsumowując, ważnym wynikiem badań Crutchfielda-Mitchella jest odkrycie metod opisywania nowych obliczeń w ramach przestrzennie rozproszonego systemu składającego się z lokalnie oddziałujących procesorów komórkowych. Lokalność komunikacji w komórkach narzuca ograniczenie globalnej komunikacji. Rolą GA jest odkrywanie lokalnych reguł komórkowe, których efektem jest przetwarzanie informacji na „duże" odległości czasoprzestrzenne. Crutchfield i Mitchell wykorzystali pomysły zaczerpnięte z formalnej teorii języka, aby scharakteryzować te wzorce czasoprzestrzenne. Dla Crutchfielda i Mitchella wynik ewoluującego automatu odzwierciedla całkowicie nowy poziom zachowania, różniący się od interakcji rozproszonych komórek na niższym poziomie. Globalne interakcje oparte na cząsteczkach pokazują, jak złożona koordynacja może wyłonić się w zbiorze prostych pojedynczych działań. Wynik GA działającego na lokalnych regułach komórkowych pokazał, jak proces ewolucyjny, wykorzystując pewne nieliniowe działania komórek tworzące wzorce, wytworzył nowy poziom zachowania i delikatną równowagę niezbędną do skutecznego wyłaniania się obliczeń. Wyniki badań Crutchfielda-Mitchella są ważne, ponieważ, przy wsparciu GA, wykazały pojawienie się niezmienności wyższego poziomu w świecie automatów komórkowych. Ponadto przedstawiają narzędzia obliczeniowe, zaadaptowane z formalnej teorii języka, które można wykorzystać do opisu tych niezmienności. Dalsze badania w tej dziedzinie mogą wyjaśnić pojawienie się złożoności: definiującej cechy żywych istot i fundamentalnej dla zrozumienia pochodzenia umysłów, gatunków i ekosystemów.

Sztuczna inteligencja : Uczenie maszynowe : Probabilistyczne

(XIII / XVI)

ĆWICZENIA

1. Utwórz ukryty model Markowa, aby przedstawić sekwencję punktacji w meczu futbolu amerykańskiego, w którym przyłożenia wynoszą 6 punktów, po których można wykonać próbę 0, 1 lub 2 dodatkowych punktów. Oczywiście są dwie drużyny i każda może strzelić gola. Jeśli drużyna ma piłkę i nie zdobywa punktów, emituje 0. Załóżmy więc, że emitowany strumień wyników to 6,1,6,0,3,0,6,1,0,3,0; jak wyglądałby Twój HMM?

a. Omów ten problem i jak może wyglądać najlepszy HMM.

b. Przetestuj swój HMM na dwóch dodatkowych strumieniach wyników.

c. Śledź aktualną grę i zobacz, jak dobrze Twój system przewiduje strumień punktacji.

2. Utwórz ukryty model Markowa, aby przewidzieć sekwencję punktów w połowie zmiany w amerykańskim meczu baseballowym. Załóżmy, że sekwencja połowy zmiany jest równa 0, 0, 0, 1, 0, 1, 1, 2, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0. Dla uproszczenia ogranicz punktację dla każdego drużyna do 0, 1 lub 2 runów podczas ich połowy .

a. Jak zmieniłbyś model, aby zezwolić na dowolną liczbę przebiegów na połowę zmiany?

b. Jak możesz wytrenować swój system, aby uzyskać bardziej realistyczne wartości punktacji?

3. Utwórz figurę reprezentującą hierarchiczny, ukryty model Markowa z sekcji 13.1.2. Do jakiego typu sytuacji problemowej może być odpowiedni Twój HHMM? Omów kwestię dopasowania struktur HMM do domen aplikacji.

4. Biorąc pod uwagę przykład algorytmu Viterbiego przetwarzającego probabilistyczną maszynę skończoną z rysunków 7 i 8:

a. Dlaczego nowy jest postrzegany jako lepsza interpretacja niż kolano dla obserwowanych telefonów?

b. W jaki sposób algorytm Viterbiego obsługuje stany alternatywne w probabilistycznej maszynie skończonej, na przykład wybór telefonów uw i iy w słowie nowy?

5. Biorąc pod uwagę ukryty model Markowa i algorytm Viterbiego z Części 9, przeprowadź pełne śledzenie, w tym ustawienie odpowiednich wskaźników wstecznych, które pokażą, jak obserwacja #, n, iy, t, # byłaby przetwarzana.

6. Uruchom ręcznie robota opisanego w przykładzie procesu decyzyjnego Markowa w Rozdziale 13.3.3. Użyj tego samego mechanizmu nagrody i wybierz wartości probabilistyczne dla a i b do przetwarzania decyzji.

a. Uruchom robota ponownie z różnymi wartościami a i b. Jakie zasady dają robotowi największe szanse na nagrodę?

7. Zaprogramuj robota z sekcji 13.3.3 obsługiwanego przez MDP w wybranym przez siebie języku. Eksperymentuj z różnymi wartościami a i b, które mogą zoptymalizować nagrodę. Istnieje kilka interesujących możliwych polityk: Jeśli doładowanie jest zasadą A (wysokie), czy Twój robot nauczy się, że ta zasada jest nieoptymalna? W jakich okolicznościach robot zawsze szukałby pustych puszek, tj. Zasada A (niski) = ładowanie jest nieoptymalna?

8. Jack prowadzi salon samochodowy i szuka sposobu na maksymalizację swoich zysków. Co tydzień Jack zamawia zapasy samochodów kosztem d dolarów za samochód. Te samochody są dostarczane natychmiast. Nowe samochody zostaną dodane do jego ekwipunku. Następnie w ciągu tygodnia sprzedaje losową liczbę samochodów k po cenie c każdy. Jack ponosi również koszty u każdego niesprzedanego samochodu, który musi przechowywać w ekwipunku. Sformułuj ten problem jako proces decyzyjny Markowa. Jakie są stany i działania? Jakie są nagrody? Jakie są prawdopodobieństwa przejścia?

Opisz długoterminowy zwrot.

9. Rozważ ogólną dziedzinę zadań nawigacji w świecie siatki, w której występuje stan celu, przeszkody i współczynnik dyskontowy $\gamma < 1$. Działania są stochastyczne, więc agent może wślizgnąć się do innej komórki podczas próby ruchu. Istnieje pięć możliwych działań: idź na północ, południe, wschód, zachód lub pozostań w tym samym miejscu. Rozważmy sytuację, w której podczas uderzania o ściany ponoszone są ujemne koszty. Czy potrafisz narysować przykładowe środowisko 3x3, w którym najlepszą akcją w przynajmniej jednym stanie jest pozostanie? Jeśli tak, określ działania, nagrody i prawdopodobieństwa przejścia. Jeśli nie, wyjaśnij dlaczego.

10. Kiedy wychodzimy na kolację, zawsze lubimy parkować jak najbliżej restauracji. Załóżmy, że restauracja znajduje się na bardzo długiej ulicy biegnącej ze wschodu na zachód, która umożliwia parkowanie tylko z jednej strony. Ulica podzielona jest na odcinki o długości jednego samochodu. Do restauracji zbliżamy się od wschodu, zaczynając od jednostek D. Prawdopodobieństwo, że miejsce parkingowe w odległości s od restauracji jest wolne, wynosi ps, niezależnie od wszystkich innych miejsc. Sformułuj ten problem jako proces decyzyjny Markowa. Pamiętaj, aby określić wszystkie elementy swojego MDP!

11. Jak zmieniłbyś reprezentację MDP w sekcji 13.3 na POMDP? Weźmy prosty problem z robotem i jego macierz przejść Markowa utworzoną w sekcji 13.3.3 i zmieńcie to na POMDP. Wskazówka: pomyśl o zastosowaniu macierzy prawdopodobieństwa dla stanów częściowo obserwowalnych.

12. Omówiliśmy pokrótce grę karcianą Poker w sekcji 13.3. Biorąc pod uwagę, że obecny (probabilistyczny) stan gracza to albo dobry zakład, wątpliwy zakład, albo zły zakład, opracuj POMDP, aby przedstawić tę sytuację. Możesz omówić to, korzystając z prawdopodobieństwa rozdania pewnych możliwych układów pokerowych. Oblicz koszt złożoności znalezienia optymalnej polityki dla problemu POMDP

UCZENIE MASZYNOWE: PROBABILISTYCZNE

13.0 Stochastyczne i dynamiczne modele uczenia się

Jak zauważono we wcześniejszych częściach , istnieją dwa główne powody stosowania narzędzi probabilistycznych do zrozumienia i przewidywania zjawisk na świecie: po pierwsze, zdarzenia mogą być ze sobą rzeczywiście probabilistycznie powiązane, a po drugie, deterministyczne związki przyczynowe między sytuacjami w zmieniającym się świecie mogą być tak złożone, że ich interakcje najlepiej oddają modele stochastyczne. Społeczność AI przyjęła i wdrożyła modele probabilistyczne z obu tych powodów, a te stochastyczne technologie miały bardzo ważny wpływ na projekt, moc i elastyczność algorytmów uczenia maszynowego. Reguła Bayesa, przedstawiona po raz pierwszy w sekcji 5.3, jest podstawą probabilistycznych modeli uczenia maszynowego. Podejścia bayesowskie wspierają interpretację nowych doświadczeń w oparciu o wcześniej wyuczone relacje: zrozumienie obecnych wydarzeń jest funkcją wyuczonych oczekiwań z poprzednich wydarzeń. Podobnie modele Markowa i ich warianty stanowią podstawę stochastycznego podejścia do uczenia się. W łańcuchu Markowa prawdopodobieństwo wystąpienia zdarzenia w dowolnym momencie jest funkcją prawdopodobieństwa, z jakim zdarzenia wystąpiły w poprzednich okresach. W łańcuchu Markowa pierwszego rzędu, prawdopodobieństwo wystąpienia zdarzenia o godzinie t jest jedynie funkcją prawdopodobieństwa jego poprzednika w czasie t-1. Część 13 składa się z trzech sekcji, pierwsza sekcja 13.1 kontynuuje prezentację modeli Markowa wprowadzających ukryte modele Markowa (lub HMM) oraz kilka ważnych wariantów i rozszerzeń. Sekcja 13.2 rozszerza prezentację sieci bayesowskich, skupiając się zwłaszcza na dynamicznych sieciach bayesowskich (lub DBN) i kilku rozszerzeniach. Sekcja 13.3 kontynuuje prezentację uczenia się przez wzmacnianie, z dodaniem probabilistycznych miar wzmocnienia.

13.1 Ukryte modele Markowa (HMM)

Na początku XX wieku rosyjski matematyk Andrey Markov zaproponował matematykę do modelowania probabilistycznie powiązanych dyskretnych zdarzeń schodkowych. W porównaniu z deterministyczną sekwencją zdarzeń, jak można by się spodziewać, gdy komputer wykonuje ustalony zestaw instrukcji, formalizmy Markowa wspierały ideę, że każde zdarzenie może być probabilistycznie powiązane z wydarzeniami, które je poprzedzały, jak we wzorcach fonemów lub słów w wypowiedziane zdanie. W naszych próbach zaprogramowania komputerów do uczenia się zjawisk w zmieniającym się świecie spostrzeżenia Markowa okazały się niezwykle ważne.

13.1.1 Wprowadzenie i definicja ukrytych modeli Markowa

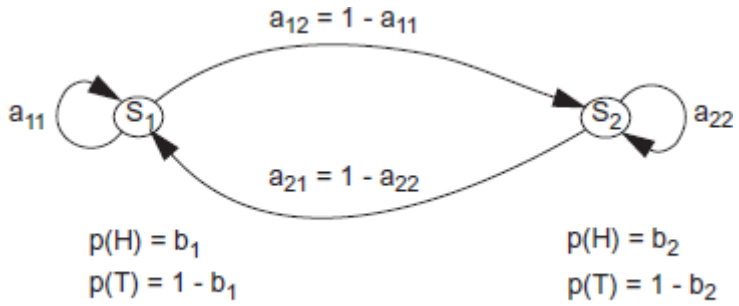
Ukryty model Markowa (lub HMM) jest uogólnieniem tradycyjnego łańcucha (lub procesu) Markowa. W łańcuchach Markowa widzianych do tej pory każdy stan odpowiadał dyskretnemu fizycznemu - i możliwemu do zaobserwowania - wydarzeniu, jak na przykład pogoda o określonej porze dnia. Ta klasa modeli jest dość ograniczona i teraz uogólniamy ją na szerszą klasę problemów. W tej sekcji rozszerzamy model Markowa na sytuacje, w których obserwacje same w sobie są probabilistycznymi funkcjami aktualnych stanów ukrytych. Zatem wynikowy model, zwany ukrytym modelem Markowa, jest podwójnie osadzonym procesem stochastycznym. HMM można opisać jako obserwowalny proces stochastyczny wspierany przez dalszy nieobserwowalny lub ukryty proces stochastyczny. Przykładowym zastosowaniem HMM może być obsługa komputerowego rozpoznawania słów poprzez interpretację hałaśliwych sygnałów akustycznych. Same wzorce telefoniczne, czyli fonemy, których mówca zamierza użyć w ramach tworzenia określonych słów w języku, składają się na ukryty poziom modelu Markowa. Obserwacje, czyli słyszalne szumne sygnały akustyczne, są stochastyczną funkcją tych zamierzonych fonemów. Fonemy, które zamierzał mówca, można zobaczyć tylko probabilistycznie przez najwyższy poziom (obserwowalny) strumień zaszumionych sygnałów akustycznych. Definiujemy HMM:

DEFINICJA

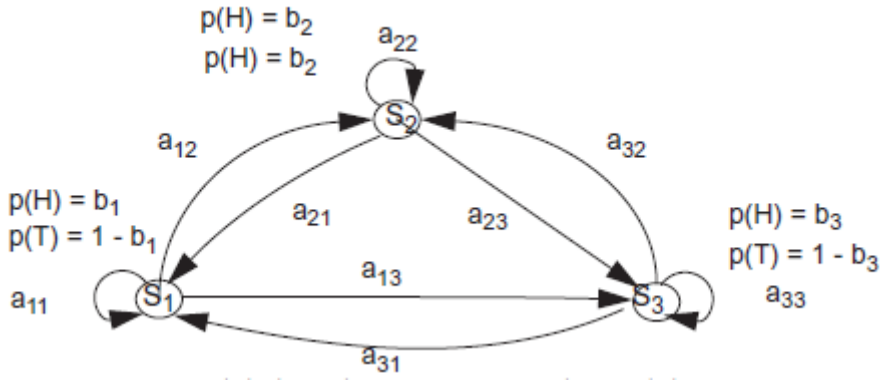
UKRYTY MODEL MARKOWA

Model graficzny nazywany jest ukrytym modelem Markowa (HMM), jeśli jest to model Markowa, którego stany nie są bezpośrednio obserwowalne, ale są ukryte przez dalszy system stochastyczny interpretujący ich wyniki. Bardziej formalnie, biorąc pod uwagę zbiór stanów $S = s_1, s_2, ..., s_n$ oraz zbiór prawdopodobieństw przejścia stanów $A = a_{11}, a_{12}, ..., a_{1n}, a_{21}, a_{22}, ..., .. ., a_{nn}$, istnieje zbiór prawdopodobieństw obserwacji $O = p_i(o_t)$, z których każde wyraża prawdopodobieństwo wygenerowania obserwacji ot (w czasie t) przez stan s_t.

Podajemy teraz dwa przykłady HMM, zaadaptowane z Rabinera. Najpierw rozważ sytuację, w której rzuca się monetą. Załóżmy, że w pokoju jest osoba, która rzuca monetami pojedynczo. Nie mamy pojęcia, co się dzieje w pokoju, w rzeczywistości może być wiele monet, losowo wybranych do przewrócenia, a każda z nich może mieć własne tendencyjne wyniki, tj. mogą nie być uczciwymi monetami. Wszystko, co obserwujemy na zewnątrz pomieszczenia, to seria wyników z odwracania, np. Obserwowalne O = H, H, T, H, T, H, ... Naszym zadaniem jest zaprojektowanie modelu monet przerzuconych wewnątrz pomieszczenia, który tworzy otrzymany obserwowalny ciąg. W tej sytuacji spróbujemy teraz zamodelować wynikowy zestaw obserwacji pochodzących z pomieszczenia. Zaczynamy od prostego modelu; załóżmy, że rzuca się tylko jedną monetą. W tym przypadku musimy tylko określić odchylenie monety, ponieważ jest ona odwracana w czasie, tworząc zestaw obserwacji głowy / ogona. To podejście daje w wyniku bezpośrednio obserwowalny (zerowego rzędu) model Markowa, który jest po prostu zbiorem prób Bernoulliego. Ten model może w rzeczywistości być zbyt prosty, aby wiarygodnie uchwycić to, co dzieje się w problemie z rzucaniem monetą. Następnie rozważymy model z dwiema monetami, z dwoma stanami ukrytymi s_1 i s_2, jak widać na rysunku 1.



Probabilistyczna macierz przejścia, A, kontroluje, w jakim stanie jest system, tj. Która moneta jest rzucana w dowolnym momencie. Każda moneta ma inne odchylenie H / T, b_1 i b_2. Załóżmy, że zaobserwowaliśmy ciąg rzutów monetą: O = H, T, T, H, T, H, H, H, T, T, H. Ten zestaw obserwacji można wyjaśnić następującą ścieżką przez diagram przejść stanów z rysunku 1: $s_2, s_1, s_1, s_2, s_2, s_2, s_1, s_2, s_2, s_1, s_2$. Trzeci model przedstawiono na Rysunku 2, gdzie trzy stany (trzy monety) są używane do przechwytywania wyniku rzutu monetą.



Ponownie, każda moneta / stan, si, ma swoje własne odchylenie, b_i, a stan systemu (która moneta jest odwrócona) jest funkcją pewnego zdarzenia probabilistycznego, takiego jak rzut kostką i macierz przejścia, A. maszyna trójstanowa może odpowiadać za wyjście H / T z sekwencją stanów: s_3, s_1, s_2, s_3, s_3, s_1, s_1, s_2, s_3, s_1, s_3. Trudną kwestią w przypadku rzutu monetą jest wybór optymalnego modelu. Najprostszy model ma jeden parametr, odchylenie jednej monety. Model z dwoma stanami ma cztery parametry, przejścia stanów i odchylenia każdej monety. Model trójstanowy z rysunku 2 ma dziewięć parametrów. Im więcej stopni swobody, tym większy model ma większe możliwości, aby uchwycić (prawdopodobnie) bardziej złożoną sytuację. Jednak trudności związane z określeniem parametrów dla większego modelu mogą zepsuć to ćwiczenie. Na przykład rzeczywiste obserwowalne mogą powstać w prostszej sytuacji, w którym to przypadku większy model byłby nieprawidłowy, niedokreślony i wymagałby znacznie więcej danych do ustalenia dowolnego stopnia pewności. Podobne problemy, takie jak nadmierne dopasowanie, również prześladują mniejsze modele. Wreszcie, w każdym modelu istnieje niejawne założenie stacjonarności, to znaczy, że prawdopodobieństwa przejścia i odchylenia systemu nie zmieniają się w czasie. Jako drugi przykład rozważmy problem N urn, gdzie każda urna zawiera zbiór M różnokolorowych kulek. Fizyczny proces uzyskiwania obserwacji polega, zgodnie z pewnym przypadkowym procesem, na wybraniu jednej z N urn. Po wybraniu urny kula jest usuwana, a jej kolor jest rejestrowany w strumieniu wyjściowym. Kula jest następnie zastępowana, a losowy proces powiązany z bieżącą urną wybiera następną (która może być taka sama), aby kontynuować proces. Ten proces generuje sekwencję obserwacji składającą się z kolorów kulek. Oczywiste jest, że najprostszym HMM odpowiadającym temu procesowi jest model, w którym każdy stan odpowiada określonej urnie, wartości macierzy przejścia dla tego stanu tworzą następny wybór stanu, a na koniec model, w którym prawdopodobieństwo koloru piłek jest określane przez liczbę i kolor piłek w każdym stanie (urna). Jednak gdyby modelarz nie miał wcześniejszych informacji, określenie wszystkich parametrów wymaganych do wybrania optymalnego HMM byłoby rzeczywiście bardzo trudne. W następnej sekcji rozważymy kilka odmian HMM.

13.1.2 Ważne warianty ukrytych modeli Markowa

W poprzedniej sekcji zauważyliśmy, że HMM są niezwykle potężnymi, ale prostymi modelami, które mogą reprezentować niepewne dziedziny. Dotychczasowa prostota obliczeniowa HMM wynika z markowej zasady systemu pierwszego rzędu: stan obecny zależy tylko od poprzedniego stanu. W tej sekcji zbadamy, jak możemy zmodyfikować podstawowe zasady HMM, aby uzyskać jeszcze bogatsze możliwości reprezentacji. Zwykłe HMM z sekcji 13.1.1 mają ograniczenia w określonych domenach, a następnie pokażemy warianty HMM, które mogą rozwiązać wiele z tych ograniczeń. Szeroko stosowane rozszerzenie HMM nazywa się autoregresywnym HMM, gdzie poprzednia obserwacja może również wpływać na wartość bieżącej obserwacji. To rozszerzenie służy do przechwytywania większej ilości informacji z przeszłości i uwzględniania ich w interpretacji stanu obecnego. W celu modelowania złożonych procesów składających się ze skojarzeń zestawów prostszych podprocesów często stosuje

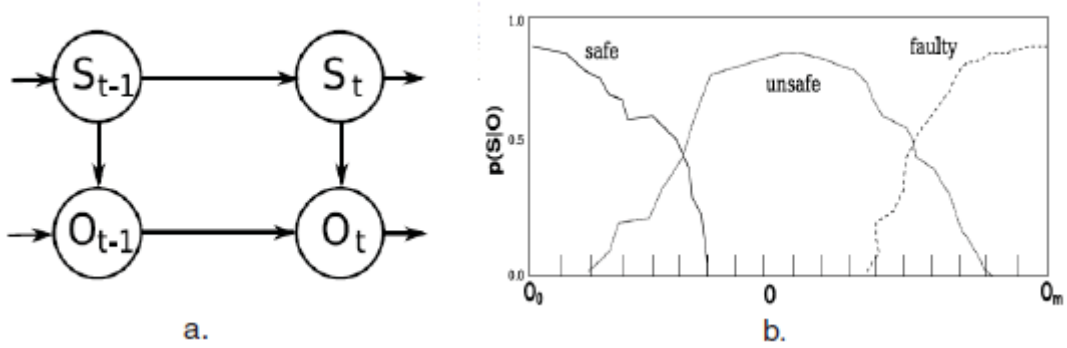
się czynnikowe HMM. Jedno rozszerzenie HMM, hierarchiczne HMM lub (HHMS), zakłada, że każdy stan systemu sam jest HMM. Następnie rozważymy bardziej szczegółowo te i inne rozszerzenia HMM.

HMM z autoregresją (AR-HMM)

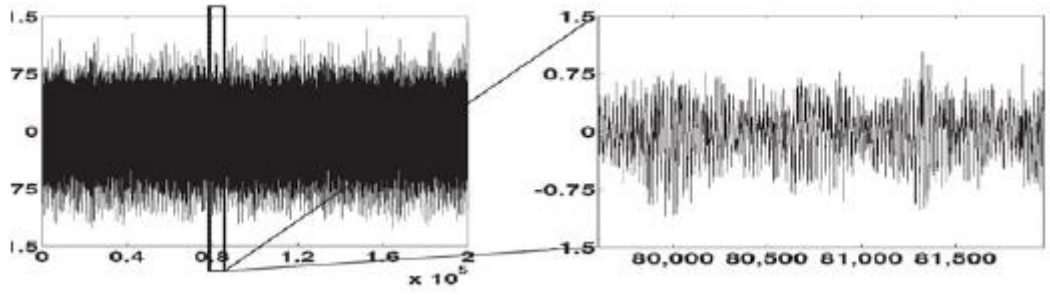
W ukrytych modelach Markowa właściwość pierwszego rzędu zakłada, że stan obecny jest zależny tylko od bezpośredniego poprzedniego stanu i że obserwowane zmienne nie są korelowane. Może to być ograniczenie, w którym obecny stan nie pozwala odpowiednio uchwycić informacji dostępnych w poprzednich stanach. Na przykład w rozumowaniu diagnostycznym może to prowadzić do gorszych uogólnień, zwłaszcza podczas modelowania danych szeregów czasowych. Faktem jest, że w złożonych środowiskach obserwowalne wartości w jednym okresie często korelują z kolejnymi obserwacjami. Należy to uwzględnić w strukturze HMM. Przezwyciężamy ten aspekt ograniczonego uogólnienia, pozwalając poprzedniej obserwacji wpływać na interpretację bieżącej obserwacji wraz ze zwykłymi wpływami HMM poprzedniego stanu. Ten model HMM nazywa się Model Markowa z ukrytą autoregresją, którego model emisji jest określony przez niezerowy model prawdopodobieństwa:

$p(O_t \mid S_t, O_{t-1})$.

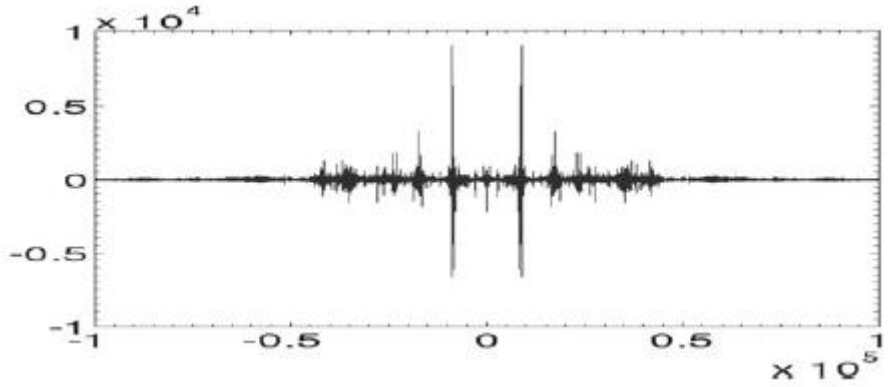
gdzie O to stan emisji lub obserwacji, a S to warstwa ukryta, jak pokazano na rysunku 3a.



Różnica między tradycyjnym HMM a AR-HMM polega na istnieniu powiązania (wpływu) między O_{t-1} i Ot. Motywacja dla tego wpływu jest prosta: stan obserwacji w jednym okresie będzie wskaźnikiem tego, jaki prawdopodobnie nastąpi następna obserwacja. Innymi słowy, obserwowane wartości wyjściowe analizy szeregów czasowych zmienią się zgodnie z pewnym podstawowym rozkładem. AR-HMM często prowadzi do modeli o wyższym prawdopodobieństwie zbieżności niż zwykłe HMM, szczególnie podczas modelowania niezwykle złożonych i zaszumionych danych szeregów czasowych. Przedstawiamy AR-HMM na przykładzie zaczerpniętym z pracy Chakrabarti i inych. Zadanie polega na monitorowaniu stanu pracy układu wirnika helikoptera. Rysunek 4 przedstawia próbkę danych obserwowanych z uruchomionego systemu.



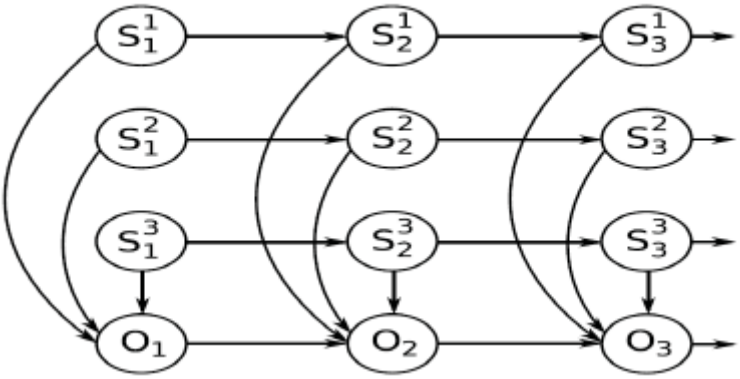
Dane pochodzą z wielu czujników, w tym informacje z pomiarów ciepła i wibracji wielu elementów. Rysunek 5 przedstawia wyniki przetwarzania danych z rysunku 4 przy użyciu szybkiej transformaty Fouriera (FFT) w celu uzyskania równoważnych danych w dziedzinie częstotliwości.



Te punkty danych są następnie wykorzystywane do szkolenia AR-HMM. Trzy stany ukrytego węzła, S_t, na rysunku 3, są bezpieczne, niebezpieczne i wadliwe. Celem tego projektu było zbadanie technik wykrywania i diagnostyki usterek w układach mechanicznych. Wykorzystując zestaw danych szeregów czasowych z czujników monitorujących procesy mechaniczne, zadaniem było zbudowanie ilościowego modelu całego procesu i przeszkolenie go na zestawach danych (usiane błędy, takie jak pęknięty wał) do późniejszego wykorzystania w diagnostyce w czasie rzeczywistym i przewidywanie przyszłych usterek. Po przeszkoleniu AR-HMM został zwolniony do pracy w czasie rzeczywistym. Chociaż raportowana dokładność wynosiła nieco ponad 75%, ważne jest, aby zobaczyć, jak można zaprojektować takie narzędzia prognostyczne.

Silnia HMMs

W naszych poprzednich dyskusjach na temat HMM i ich wariantów opisaliśmy systemy, których przestrzeń stanów była częściowo ukryta. Wskazówki dotyczące przestrzeni stanów mogliśmy uzyskać tylko poprzez serię obserwacji. Próbowaliśmy wywnioskować sekwencję ukrytych stanów ukrytych, wykorzystując informacje z obserwowanej sekwencji danych emitowanych przez system. Ale co się stanie, jeśli podstawowy proces jest bardzo złożony i nie można go skutecznie przedstawić jako prostego zestawu stanów? Co się stanie, jeśli podstawowy proces jest w rzeczywistości połączeniem kilku podprocesów? Nasz regularny HMM nie jest w stanie w wystarczającym stopniu uchwycić informacji o procesach w takich sytuacjach. Potrzebujemy bogatszego modelu, aby przedstawić bardziej złożone sytuacje. Możliwym rozwiązaniem są czynnikowe HMM. Rysunek 6 przedstawia silnię autoregresyjną



HMM. Widzimy, że podstawowy proces można podzielić na n podprocesów, gdzie każdy z tych n procesów wpływa na bieżącą obserwację, Ot. Ponadto każdy podproces jest zgodny z właściwością Markowa pierwszego rzędu, to znaczy jego bieżący stan zależy tylko od poprzedniego stanu (stanów). Ten związek z obserwowalną zmienną losową można zdefiniować za pomocą CPD:

$p(O_t|O_{t-1},S^1_t, S^2_t, ..., S^i_t)$

Hierarchiczne HMM (HHMM)

Hierarchiczne moduły HMM są używane do modelowania niezwykle złożonych systemów. W hierarchicznym ukrytym modelu Markowa (lub HHMM) każdy stan jest uważany za samodzielny model probabilistyczny. Dokładniej, każdy stan HHMM może sam być HHMM. Oznacza to, że stany HHMM emitują sekwencje obserwacji, a nie tylko pojedyncze obserwacje, jak ma to miejsce w przypadku standardowych HMM i wariantów widzianych do tej pory. Kiedy stan w HHMM zostanie osiągnięty, aktywuje swój własny model probabilistyczny, tj. Aktywuje jeden ze stanów bazowego HHMM, który z kolei może aktywować jego bazowy HHMM itd., Podstawowym przypadkiem jest tradycyjny HMM. Proces jest powtarzany do momentu aktywacji stanu zwanego stanem produkcyjnym. Tylko stany produkcyjne emitują symbole obserwacji w znaczeniu zwykłego ukrytego modelu Markowa. Gdy stan produkcji wyemituje symbol, sterowanie powraca do stanu, który aktywował stan produkcji. Stany, które nie emitują bezpośrednio symboli obserwacji, nazywane są stanami wewnętrznymi. Aktywacja stanu w HHMM w stanie wewnętrznym nazywana jest przejściem pionowym. Po zakończeniu przejścia pionowego następuje przejście poziome do stanu na tym samym poziomie. Kiedy przejście poziome prowadzi do stanu końcowego, sterowanie jest przywracane do stanu w HHMM wyżej w hierarchii, który spowodował ostatnie przejście pionowe. Należy zauważyć, że przejście wertykalne może skutkować bardziej pionowymi przejściami przed osiągnięciem sekwencji stanów produkcyjnych i ostatecznym powrotem na najwyższy poziom. W ten sposób odwiedzane stany produkcji dają początek sekwencji symboli obserwacji, które są wytwarzane przez państwo na najwyższym szczeblu.

HMM N-Gram

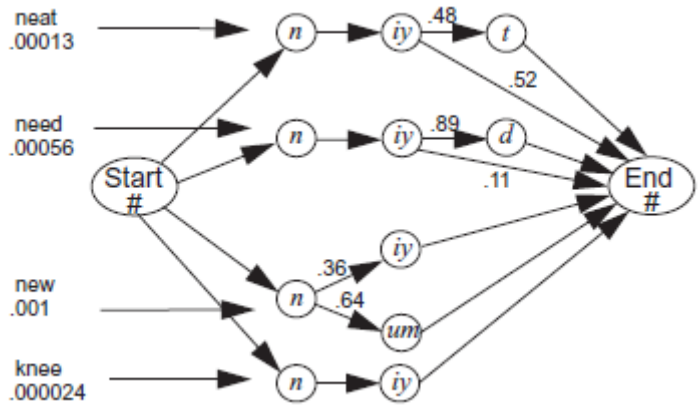
Jak wspomniano wcześniej, w tradycyjnych modelach Markowa pierwszego rzędu stan obecny, biorąc pod uwagę stan bezpośrednio poprzedni, jest niezależny od wszystkich wcześniejszych stanów. Chociaż ta reguła zmniejsza złożoność obliczeniową przy korzystaniu z procesów Markowa, może się zdarzyć, że sam poprzedni stan nie może całkowicie uchwycić ważnych informacji dostępnych w domenie problemu. Wyraźnie rozwiązujemy tę sytuację za pomocą n-gramowego ukrytego modelu Markowa (lub N-gramowego HMM). Modele te, zwłaszcza w ich postaci bi-gramowej i trigramowej, są szczególnie ważne w stochastycznych podejściach do rozumienia języka naturalnego. W bi-gramowych modelach Markowa interpretacja obserwowalnego stanu prądu zależy tylko od interpretacji stanu

poprzedniego, jako $p(O_t|O_{t-1})$; jest to odpowiednik tradycyjnego łańcucha Markowa pierwszego rzędu. Chociaż założenie Markowa pierwszego rzędu jest oczywiste, warto zastanowić się, co oznacza przetwarzanie bi-gramowe, na przykład w rozumieniu języka naturalnego. Przyjmuje hipotezę, że w przypadku rozpoznawania wyrazów lub fonemów, biorąc pod uwagę sekwencję poprzednich słów, prawdopodobieństwo wystąpienia określonego słowa jest najlepiej oszacowane, gdy używamy tylko interpretacji bezpośrednio poprzedniego słowa. Zakłada się zatem, że bigram daje lepsze wyniki niż wiedza o braku informacji o poprzednim słowie lub znajomość większej sekwencji poprzednich słów. Na przykład, biorąc pod uwagę zbiór danych językowych, p (lamb | little) jest lepszym predyktorem bieżącego obserwowanego słowa będącego jagnięciem, niż albo p (lamb | a little), p (lamb | had a little) lub p (jagnięcina | Mary miała trochę). Podobnie modele trigramowe to modele Markowa, w których interpretacja obserwowalnego stanu prądu zależy od dwóch poprzednich stanów. Model trigramowy jest reprezentowany przez $P (O_t|O_{t-1}, O_{t-2})$. Kontynuując przykład z języka naturalnego, użycie modeli tri-gramowych oznacza, że p (lamb | a little) jest lepszym predyktorem obecnego słowa będącego jagnięciem niż p (lamb | little) lub większej liczby poprzednich słów, w tym p (lamb | Mary miała trochę). Możemy rozszerzyć n-gramowy model Markowa do dowolnej długości n, która naszym zdaniem obejmuje niezmienności obecne w danych, które próbujemy modelować. N-gramowy model Markowa to łańcuch Markowa, w którym prawdopodobieństwo bieżącego stanu zależy dokładnie od n-1 poprzednich stanów. Jak zobaczymy bardziej szczegółowo w Części 15, modele n-gramowe są szczególnie przydatne do przechwytywania sekwencji głosek, sylab lub słów podczas rozumienia mowy lub innych zadań związanych z rozpoznawaniem języka. Możemy ustawić wartość n w zależności od pożądanej złożoności domeny, którą chcemy reprezentować. Jeśli n jest zbyt małe, możemy mieć słabszą siłę reprezentacji i przegapić zdolność uchwycenia mocy predykcyjnej dłuższych strun. Z drugiej strony, jeśli wartość n jest zbyt duża, wówczas koszt obliczeniowy walidacji modelu może być bardzo wysoki lub nawet niemożliwy. Głównym tego powodem jest fakt, że wcześniejsze oczekiwania dotyczące połączeń telefonicznych lub słów są pobierane z dużych baz danych (często połączonych baz danych) zgromadzonych zjawisk językowych zwanych korpusem. Po prostu możemy nie mieć wystarczających danych, aby zweryfikować n-gram z bardzo dużym n. Na przykład, możemy nie mieć żadnych informacji na temat p (baranek | Mary miała trochę), podczas gdy mamy dużo informacji o p (jagnię | mało), p (jagnię | trochę) lub p (jagnię | trochę). Zwykle w zadaniach rozpoznawania mowy lub tekstu używamy n-gramów o wartości n nie większej niż 3. Istnieje wiele innych wariantów HMM, których tutaj nie opisaliśmy, w tym HMM z pamięcią mieszaną, które używają kilku niższego rzędu Modele Markowa w celu przezwyciężenia problemów złożoności. Dalsze rozszerzenie autoregresywnych HMM, zwanych zakopanymi modelami Markowa, pozwala na nieliniowe zależności między obserwowalnymi węzłami. Moduły HMM wejściowe i wyjściowe służą do mapowania sekwencji wejść na sekwencję wyjść. W sytuacjach, w których interakcje między podprocesami złożonego procesu są bardziej skomplikowane niż tylko kompozycja, często stosuje się sprzężony HMM. Ukryty model semi-Markowa uogólnia HMM, aby uniknąć efektu rozpadu geometrycznego HMM poprzez uzależnienie prawdopodobieństwa przejść między stanami ukrytymi od ilości czasu, jaki upłynął od ostatniej zmiany stanu. Odsyłamy do sekcji 13.4 w celu uzyskania odniesień. W kolejnej części do problemu tłumaczenia ustnego zastosujemy technologię bi-gramowego HMM. Angielskie kombinacje fonemów i wykorzystanie programowania dynamicznego z algorytmem Viterbiego do implementacji wyszukiwania HMM

13.1.3 Używanie HMM i Viterbi do dekodowania ciągów fonemów

Nasza ostatnia sekcja o ukrytych modelach Markowa pokazuje ważne zastosowanie technologii HMM: identyfikowanie wzorców w mówionym języku naturalnym. Zakładamy obecność dużego korpusu językowego, który wspiera wcześniejsze oczekiwania dotyczące kombinacji telefonów. Na koniec używamy algorytmów programowania dynamicznego, do implementacji wnioskowania HMM. Kiedy

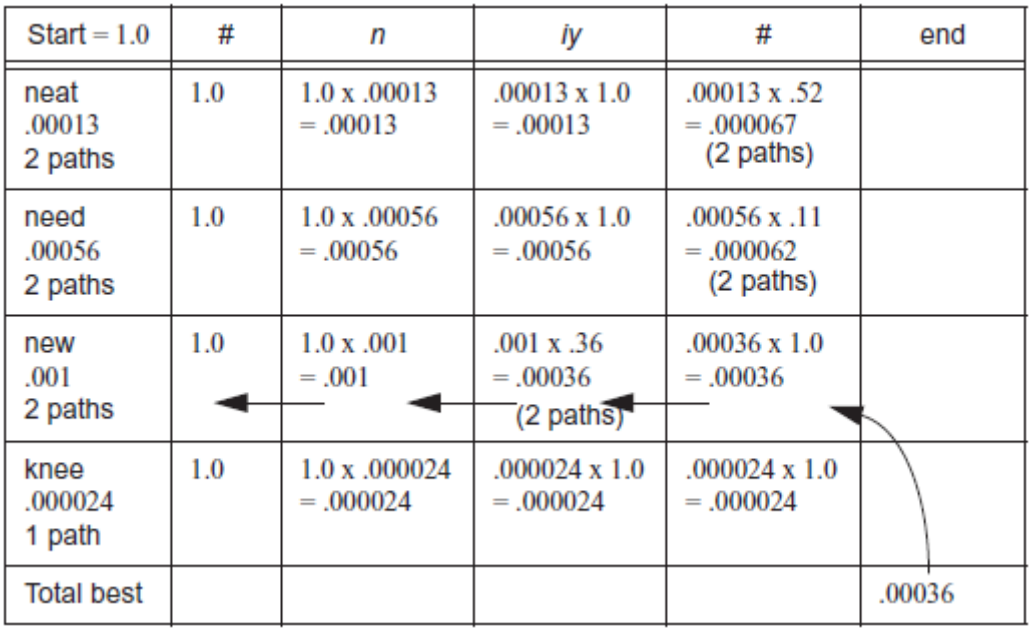
programowanie dynamiczne jest używane do znalezienia maksymalnego prawdopodobieństwa wystąpienia ciągu a posteriori, często nazywa się to algorytmem Viterbiego. Poniższy przykład obliczeniowej analizy wzorców mowy ludzkiej i wykorzystanie algorytmu Viterbiego do interpretacji ciągów fonemów jest zaadaptowane z Jurafsky i Martin. Dane wejściowe do tej probabilistycznej maszyny to ciąg telefonów lub podstawowe dźwięki mowy. Wynikają one z rozkładu sygnału akustycznego wytwarzanego podczas używania języka mówionego. W automatycznym rozumieniu mowy niezwykłe jest to, że sygnały akustyczne byłyby jednoznacznie rejestrowane jako ciąg fonemów. Sygnały mowy są raczej interpretowane jako określone telefony probabilistycznie. Zakładamy jednoznaczną interpretację sygnałów, aby uprościć naszą prezentację algorytmu Viterbiego przetwarzania HMM. Rysunek 7 przedstawia wycinek bazy danych słów, powiązanych pod względem bliskości zbiorów fonemów tworzących ich składowe akustyczne.



Chociaż ten zestaw słów, schludne, nowe, potrzeba i kolano, to tylko niewielka część pełnego słownictwa angielskiego, można sobie wyobrazić, że bardzo duża liczba tych powiązanych klastrów mogłaby wspierać system rozumienia mowy. Rysunek 13.7 jest przykładem probabilistycznej maszyny skończonej, przedstawionej po raz pierwszy w sekcji 5.3. Zakładamy, że nasz korpus językowy jest zbiorem podobnych probabilistycznych maszyn skończonych, które mogą dawać miary prawdopodobieństwa różnych możliwych kombinacji telefonów, i jak ostatecznie te telefony można postrzegać jako części słów.

Celem analizy jest określenie, które słowo angielskie z naszej bazy danych najlepiej reprezentuje wejście sygnału akustycznego. Wymaga to użycia technologii HMM, ponieważ przedstawienie możliwych słów samo w sobie jest stochastyczne. W niedeterministycznej maszynie skończonej z rysunku 7, ciąg znaków daje nam zbiór obserwacje do zinterpretowania. Załóżmy, że ciąg obserwacji składa się z telefonów #, n, iy, #; gdzie # oznacza przerwę między dźwiękami. Używamy algorytmu Viterbiego, aby zobaczyć, która ścieżka przez probabilistyczną maszynę skończoną najlepiej oddaje te obserwacje. W trybie do przodu Viterbi iteracyjnie znajduje następny najlepszy stan i jego wartość, a następnie ustawia na niego wskaźnik. W trybie wstecznym śledzimy te wskaźniki, aby uzyskać najlepszą ścieżkę. Zatem wyjście Viterbiego jest jedną z optymalnych ścieżek stanów na wykresie związanym z prawdopodobieństwem tej ścieżki. Używając Viterbi, każdy stan wyszukiwania jest powiązany z wartością. Wartość bycia w stanie si w czasie t to viterbi [s_i, t]. Wartość związana z następnym stanem s_j w automacie stanu w czasie t + 1 to viterbi [s_j, t + 1]. Wartość następnego stanu jest obliczana jako wynik wyniku obecnego stanu, viterbi [s_i, t], razy prawdopodobieństwo przejścia ze stanu obecnego do następnego, ścieżka [s_i, s_j], razy prawdopodobieństwo obserwacji s_j dane si, p (s_j | s_i). Prawdopodobieństwo przejścia, ścieżka [si, sj], jest pobierane z niedeterministycznej maszyny skończonej, a p (s_j | s_i) jest pobierane ze znanych prawdopodobieństw obserwacji par telefonicznych

występujących w języku angielskim. Następnie przedstawiamy pseudokod algorytmu Viterbiego. Zwróć uwagę na podobieństwo do programowania dynamicznego. Tablica do przechowywania i koordynowania wartości musi obsługiwać iterację w R wierszach - równą liczbie telefonów w probabilistycznym automacie skończonym (PFSM) plus dwa, aby obsłużyć każdy stan oraz stany początkowe i końcowe. Musi również iterować po kolumnach C - równej liczbie obserwacji plus dwa, aby obsłużyć użycie pustego numeru telefonu #. Kolumny wskazują również sekwencję czasową obserwacji, a każdy stan musi być powiązany z odpowiednim obserwowanym telefonem, jak pokazano na rysunku 8.



function Viterbi(Observations of length T, Probabilistic FSM)

begin

number := number of states in FSM

create probability matrix viterbi[R = N + 2, C = T + 2];

viterbi[0, 0] := 1.0;

for each time step (observation) t from 0 to T do

for each state si from i = 0 to number do

for each transition from si to sj in the Probabilistic FSM do

begin

new-count := viterbi[si, t] x path[si, sj] x p(sj | si);

if ((viterbi[sj, t + 1] = 0) or (new-count > viterbi[sj, t + 1]))

then

begin

viterbi[si, t + 1] := new-count

append back-pointer [sj , t + 1] to back-pointer list

end;

end;

return viterbi[R, C];

return back-pointer list

end.

Rysunek 8, zaadaptowany na podstawie Jurafsky i Martina, przedstawia ślad algorytmu Viterbiego przetwarzającego składnik probabilistycznej maszyny skończonej z rysunku 7 oraz sekwencję telefoniczną #, n, iy, # (obserwacje o długości T = 4) . Linki śledzenia wstecznego wskazują optymalną ścieżkę przez probabilistyczną skończoną maszynę stanów. Ta ścieżka wskazuje, że najbardziej prawdopodobną interpretacją ciągu obserwowanych telefonów jest słowo nowy. W następnej sekcji przedstawiamy inny model graficzny, uogólnienie modeli Markowa, zwany dynamiczną siecią bayesowską (lub DBN).

13.2 Dynamiczne sieci bayesowskie i uczenie się

Często napotykamy problemy w modelowaniu, w których podstawowy proces, który ma być scharakteryzowany, najlepiej opisać jako sekwencję lub progresję stanów. Dane zebrane z procesów sekwencyjnych są zwykle indeksowane za pomocą parametru, takiego jak znacznik czasu pozyskania danych lub pozycja w ciągu tekstowym. Takie zależności sekwencyjne mogą być zarówno deterministyczne, jak i stochastyczne. Dynamiczne procesy, które są z natury stochastycznymi modelami popytu, które są w stanie modelować ich niedeterminizm. Chociaż większość deterministycznych sekwencji można dobrze scharakteryzować za pomocą maszyn skończonych, niektóre ważne sekwencje mogą być dowolnie długie. Dlatego modelowanie każdego wystąpienia tych sekwencji jako zbiorów poszczególnych stanów staje się problematyczne, ponieważ liczba przejść między stanami rośnie wykładniczo w stosunku do liczby stanów w danym momencie. Dlatego też probabilistyczne modelowanie takich sekwencji pozwala nam na utrzymanie modeli wykonalnych, jednocześnie rejestrując informacje o sekwencjach istotne dla danego problemu. W sekcji 13.2 opisujemy dwa różne podejścia do uczenia się, uczenia się struktury i uczenia się parametrów w kontekście modeli graficznych. Pokrótce opisujemy techniki uczenia się struktur, w tym wykorzystanie próbkowania metodą Monte Carlo (lub MCMC) w łańcuchu Markowa.

Opisujemy również popularny meta-algorytm zwany Maksymalizacją oczekiwań (lub EM) do uczenia parametrów. Pokazujemy, jak rekurencyjny algorytm programowania dynamicznego zwany algorytmem Bauma-Welcha dostosowuje meta-algorytm EM do użytku zarówno z modułami DBM, jak i HMM. W sekcji 13.3 wyjaśniamy, w jaki sposób modele Markowa mogą być wykorzystywane do wspomagania decyzji poprzez wprowadzenie procesu decyzyjnego Markowa (lub MDP) i częściowo obserwowanego MDP (lub POMDP). Następnie bierzemy naukę ze wzmocnieniem i uzupełniamy ją o probabilistyczne modele stanu i sprzężenia zwrotnego. Wreszcie, kończymy Część 13, wykorzystując uczenie ze wzmocnieniem przy użyciu MDP do zaprojektowania systemu nawigacji robota.

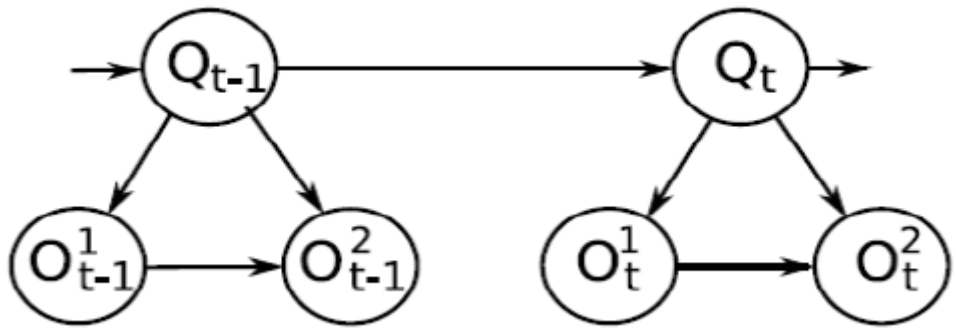
13.2.1 Dynamiczne sieci bayesowskie

Przedstawiliśmy sieć przekonań bayesowskich, bardzo popularny i z powodzeniem stosowany formalizm do modelowania probabilistycznego. Jednak w przypadku modelowania procesów dynamicznych lub sekwencyjnych BBN są dość nieelastyczne. Nie mają możliwości modelowania danych szeregów czasowych, w tym procesów przyczynowych lub sekwencyjnych korelacji danych,

które często występują w złożonych zadaniach, w tym w zrozumieniu języka naturalnego i diagnostyce w czasie rzeczywistym. Problem polega na tym, że zakłada się, że wszystkie warunkowe zależności BBN są aktywne w tym samym momencie. Krótko mówiąc, dynamiczna, czasami nazywana czasową, sieć bayesowska to sekwencja identycznych sieci bayesowskich, których węzły są połączone w (ukierunkowanym) wymiarze czasu. dynamiczne sieci bayesowskie to sekwencja ukierunkowanych modeli graficznych procesów stochastycznych. Są to uogólnienia popularnych narzędzi stochastycznych, takich jak ukryte modele Markowa (HMM) i liniowe systemy dynamiczne (LDS), które reprezentują stany ukryte (i obserwowane) jako zmienne stanu, które mogą mieć złożone współzależności w różnych okresach czasu. Struktura graficzna umożliwia określenie zależności warunkowych, a tym samym oferuje kompaktową parametryzację modelu. Poszczególne BN, które tworzą DBN, nie są dynamiczne w tym sensie, że same nie zmieniają się w czasie, przy założeniu stacjonarności, ale nadal wspierają modelowanie procesów dynamicznych. Zatem DBN działają przy założeniu, że ich parametry definiujące są niezmienne w czasie. Jednak, jak zobaczymy, ukryte węzły mogą być dodawane do reprezentowania bieżących warunków pracy, tworząc w ten sposób mieszanki modeli w celu uchwycenia okresowych niezmienności w wymiarze czasowym. W każdym przedziale czasowym (wycinku) DBN jest podobny do statycznego BBN, który ma zestaw Q_t, z N_h zmiennych losowych reprezentujących stany ukryte i zbiór zmiennych losowych O_t z N_o reprezentujących stany, które są obserwowalne. Każda zmienna losowa może być dyskretna lub ciągła. Musimy zdefiniować model reprezentujący rozkłady prawdopodobieństwa warunkowego między stanami w pojedynczym wycinku czasu i modelem przejścia, który reprezentuje relacje między BBN dla kolejnych wycinków czasu. Dlatego też, jeśli $Z_t = \{Q_t \cup O_t\}$ jest sumą obu zbiorów zmiennych, model przejścia i obserwacji można zdefiniować jako iloczyn warunkowych rozkładów prawdopodobieństwa w dwóch okresach czasu BBN bez cykli, nazwijmy to B_t. Notacja (1: N) oznacza dla wszystkich zmiennych od 1 do N:

$$p(Z_t(1:N) \mid Z_{t-1}(1:N)) = \prod_{i=1}^{N} p(Z_t(i) \mid Pa(Z_t(i)))$$

gdzie $Z_t(i)$ jest i-tym węzłem w wycinku czasu t, $Pa(Z_t(i))$ są rodzicami $Z_t(i)$ i $N = N_h + N_o$. Należy zauważyć, że tutaj, dla uproszczenia, ograniczyliśmy definicję do pierwszego -porządek procesów Markowa, w których na zmienne w czasie t wpływa tylko ich poprzednik w czasie t - 1. Możemy przedstawić bezwarunkowy rozkład stanu początkowego $p(Z_1(1: N))$, jako bezwarunkową sieć Bayesa B_1. B_1 i B_t, dla wszystkich t, definiują DBN. Na rysunku 9 pokazujemy dwa wycinki czasu prostego DBN. Jak powinno być oczywiste, dynamiczna sieć bayesowska jest uogólnieniem ukrytych modeli Markowa przedstawionych w rozdziale 13.1. Na rysunku 9 sieci bayesowskie są połączone między czasami t i t + 1.



Co najmniej jeden węzeł z dwóch sieci musi być połączony w czasie.

13.2.2 Nauka sieci bayesowskich

Uczenie się ma kluczowe znaczenie dla inteligentnego działania. Inteligentni agenci często rejestrują zdarzenia zaobserwowane w przeszłości w oczekiwaniu, że te informacje mogą w przyszłości posłużyć do jakiejś użyteczności. W ciągu swojego życia inteligentny agent napotyka wiele zdarzeń, być może znacznie więcej, niż ma zasoby do przechowywania. Dlatego konieczne staje się kodowanie informacji o zdarzeniach za pomocą zwięzłej abstrakcji, a nie pełnej dokumentacji każdej obserwacji. Ponadto zagęszczenie informacji pomaga w szybszym wyszukiwaniu. Dlatego uczenie się można postrzegać jako połączenie kompresji danych, indeksowania i przechowywania w celu wydajnego odzyskiwania. Każda nowa obserwacja agenta może nieznacznie różnić się od poprzednich obserwacji, ale nadal może należeć do ogólnej klasy wydarzeń wcześniejszych. W takim przypadku pożądane jest nauczenie się tego zdarzenia jako wystąpienia poprzedniej klasy, ignorując mniej ważne szczegóły konkretnego zdarzenia. Nowe zdarzenie może również bardzo różnić się od wszystkich wcześniejszych, a wtedy agent będzie musiał nauczyć się go jako nowej klasy lub określonej anomalii bezklasowej. Agent musi również utworzyć wystarczającą liczbę klas, które w pełni charakteryzują również przestrzeń parametrów jak być w stanie skutecznie obsługiwać nowe anomalne zdarzenia. Zatem dobre uczenie się to równowaga zarówno ogólności, jak i specyfiki. W kontekście probabilistycznych modeli graficznych napotykamy dwa ogólne problemy związane z uczeniem się, które pasują do naszej poprzedniej dyskusji: uczenie się struktury, w którym uczone są całe nowe relacje (modele graficzne), oraz uczenie parametrów, w przypadku którego komponenty istniejącego modelu są rekalibrowane.

Uczenie się i wyszukiwanie struktur

Uczenie się struktur dotyczy ogólnego problemu określania istnienia zależności statystycznych między zmiennymi. Na przykład odpowiedzi na poniższe pytania pomogłyby w określeniu struktury probabilistycznego modelu graficznego pogody. Czy ciemne chmury są związane z szansą na deszcz? Czy istnieje korelacja między położeniem Saturna a liczbą huraganów w danym dniu? Czy globalne ocieplenie jest związane z obecnością tajfuny? Czy pogoda w Arktyce jest związana ze zmieniającymi się granicami Sahary? Jak widać, w modelu probabilistycznym pozytywna odpowiedź na takie pytania uzasadniałaby warunkową zależność między danymi zmiennymi, a odpowiedź negatywna pozwoliłaby nam traktować te zmienne jako niezależne. Projektując probabilistyczny model graficzny, każda zmienna może być zależna od każdej innej zmiennej, przy jednoczesnym zachowaniu ograniczenia, że wynikowa sieć jest skierowanym grafem acyklicznym. W najgorszym przypadku otrzymujemy wtedy w pełni podłączony DAG jako model. Jest to jednak bardzo rzadkie w większości praktycznych zastosowań dowolnej wielkości, ponieważ tworzenie tabel prawdopodobieństwa warunkowego staje się trudne do wykonania. Ważnym spostrzeżeniem wspierającym projektowanie modeli graficznych jest to, że bardzo często możemy stwierdzić, że kilka zmiennych wykazuje niezależność względem siebie w kontekście modelowanego problemu. Ponieważ złożoność wnioskowania w sieciach przekonań bayesowskich zależy od liczby zależności między zmiennymi w modelu, chcielibyśmy mieć jak najmniej zależności, jednocześnie zachowując ważność modelu. Innymi słowy, chcemy usunąć jak najwięcej (niepotrzebnych) zależności z w pełni połączonego grafu. Uczenie się struktur można więc traktować jako problem poszukiwania optymalnej struktury w przestrzeni wszystkich możliwych struktur dla danego zbioru zmiennych reprezentujących jakąś dziedzinę aplikacji. Jak zauważyli Chickering i inni optymalnym rozwiązaniem tego wyszukiwania w istniejących zbiorach danych jest NP-trudne. Optymalna struktura dobrze opisuje dane, pozostając tak prostymi, jak to tylko możliwe. Jednak ta wyuczona struktura musi opisywać zarówno wcześniej zaobserwowane dane, jak i pasować do nowych, prawdopodobnie istotnych informacji. Tak więc, kiedy tworzymy (uczymy się) złożoną strukturę, która dokładnie charakteryzuje zgromadzone dane, możemy otrzymać strukturę, która jest zbyt specyficzna, w wyniku czego nie da się dobrze uogólnić na nowych danych. W literaturze

dotyczącej optymalizacji sytuacja ta jest czasami nazywana nadmiernym dopasowaniem. Chociaż przeszukiwanie przestrzeni wykładniczej możliwych ukierunkowanych struktur acyklicznych grafów metodami brutalnej siły jest raczej nierealistyczne dla praktycznych zastosowań z wieloma zmiennymi, istnieją pewne zasady, które pomagają rozwiązać ten problem. Pierwsza praktyczna zasada to brzytwa Ockhama, czyli zasada oszczędności. Spośród dwóch konkurencyjnych struktur, które działają porównywalnie, należy preferować prostszą konstrukcję nad bardziej złożoną. Brzytwa Ockhama podpowiadałaby, że w niektórych przypadkach dobrym pomysłem może być poszukiwanie możliwych struktur przechodzących od prostych do złożonych, kończąc poszukiwania, gdy znajdziemy strukturę, która jest wystarczająco prosta, ale udaje się być odpowiednim modelem do zadania na dłoni. Użycie tej heurystyki prostoty narzuca Bayesian przed przeszukiwaniem struktury. Jednak często trudno jest zdefiniować a priori miarę złożoności konstrukcji. Wybór rozsądnej miary złożoności, która narzuca całkowity porządek na możliwe konstrukcje, nie jest oczywisty. Mogą istnieć klasy struktur, które są równie złożone, w takim przypadku moglibyśmy narzucić częściowy porządek w przestrzeni struktury. Jednak nawet wśród struktur tej samej klasy wyczerpujące przeszukiwanie wszystkich możliwości jest często niepraktyczne. Podobnie jak w przypadku wielu trudnych problemów związanych ze sztuczną inteligencją, w ogólnym przypadku indukowanie struktur jest po prostu niemożliwe w rozsądnie złożonych sytuacjach. W rezultacie stosujemy heurystykę w próbaie ułatwienia przeszukiwania struktury. Mając działający, choć niezadowalający model sytuacji, jedną z metod heurystycznych jest przeprowadzenie chciwego wyszukiwania lokalnego w pobliżu uszkodzonych komponentów modelu. Biorąc pod uwagę, że celem jest znalezienie tylko lokalnej optymalizacji w strukturze graficznej, może to stanowić wystarczająco dobre rozwiązanie problemu modelowania. Inną heurystyką byłoby poleganie na ekspertach zajmujących się problematyką w celu uzyskania porady, która pomoże przemyśleć strukturę modelu. Znowu ekspert może skupić się na punktach załamania obecnego modelu. Bardziej ogólne podejście do problemu indukcji modeli polega na próbkowaniu przestrzeni możliwych modeli. Przedstawiamy następną popularną technikę pobierania próbek tego typu, łańcuch Markowa Monte Carlo.

Łańcuch Markowa Monte Carlo (MCMC)

Łańcuch Markowa Monte Carlo (lub MCMC) to klasa algorytmów do próbkowania z rozkładów prawdopodobieństwa. MCMC opiera się na konstrukcji łańcucha Markowa, który ma pożądany rozkład jako rozkład równowagi. Stan łańcucha po dużej liczbie kroków, czasami nazywane wypalaniem w czasie, są następnie używane jako próbka dla pożądanych rozkładów możliwej struktury DAG dla domeny aplikacji. Oczywiście jakość próbki łańcucha Markowa poprawia się wraz z liczbą pobranych próbek. W kontekście przeszukiwania struktur zakładamy, że możliwe struktury dla danego zbioru zmiennych tworzą przestrzeń stanów łańcucha Markowa. Macierz przejść dla tych stanów jest wypełniona jakąś techniką opartą na wadze. Wagi mogą zaczynać się jako jednolite, gdzie losowy spacer Monte Carlo między tymi strukturami ważyłby równie prawdopodobne wszystkie modyfikacje konstrukcyjne. Z biegiem czasu, oczywiście, większe wagi są używane do przejść między blisko spokrewnionymi strukturami. Jeśli struktura A różni się od struktury B tylko jednym połączeniem, to spacer Monte Carlo w tej przestrzeni jest poszukiwaniem w heurystycznie zorganizowanych dzielnicach tych struktur. Każda struktura, gdy próbkowana, jest powiązana z wynikiem, zwykle maksymalnym wynikiem opartym na a posteriori (lub MAP), p (struktura | dane). Technika próbkowania Monte Carlo przemierza przestrzeń struktury, wybierając z większym prawdopodobieństwem przejścia poprawiające prawdopodobieństwo danych, a nie przejścia, które pogarszają wynik. Po wielu iteracjach podejście MCMC zbiega się w kierunku lepszych struktur, poprawiając późniejszy rozkład p (struktura | dane) w porównaniu z punktem początkowym. Znowu pojawia się problem odkrycia lokalnych maksimów w próbkowanej przestrzeni; problem ten można rozwiązać za pomocą technik, takich jak losowy restart lub symulowane wyżarzanie.

Uczenie się parametrów i maksymalizacja oczekiwań (EM)

Uczenie parametryczne można skontrastować z uczeniem się struktur w przestrzeni modeli graficznych. Biorąc pod uwagę istniejący model dziedziny problemowej, uczenie parametrów próbuje wywnioskować optymalny wspólny rozkład dla zbioru zmiennych przy danym zbiorze obserwacji. Uczenie się parametrów jest często używane jako podprogram w ramach ogólnego wyszukiwania struktury. Gdy istnieje kompletny zestaw danych, uczenie się parametrów dla dystrybucji ogranicza się do zliczania. W wielu interesujących zastosowaniach modeli graficznych może jednak wystąpić problem z niepełnym zestawem danych dla zmiennej. Kolejnym - i pokrewnym - problemem jest istnienie w modelu ewentualnych ukrytych lub utajonych zmiennych. W takich przypadkach algorytm maksymalizacji oczekiwań (lub EM) służy jako potężne narzędzie do wnioskowania o prawdopodobnych szacunkach dla wspólnych rozkładów. Algorytm EM został po raz pierwszy wyjaśniony i nadano mu nazwę w klasycznej pracy A. Dempstera, N. Lairda i D. Rubina. W artykule uogólniono metodologię i rozwinięto stojącą za nią teorię. EM okazała się jedną z najpopularniejszych metod uczenia się w modelowaniu statystycznym i probabilistycznym. Algorytm EM jest używany w statystykach do znajdowania oszacowań parametrów maksymalnego prawdopodobieństwa w modelach probabilistycznych, gdzie model zależy od możliwych nieobserwowanych zmiennych latentnych. EM to iteracyjny algorytm, który naprzemiennie wykonuje krok oczekiwania (E), który oblicza oczekiwanie prawdopodobieństwa zmiennej poprzez uwzględnienie zmiennych latentnych tak, jakby były obserwowane, oraz krok maksymalizacji (M), który oblicza oszacowania maksymalnego prawdopodobieństwa parametrów z oczekiwanego prawdopodobieństwa znalezionego w kroku E. Parametry znalezione na kroku M są następnie używane do rozpoczęcia kolejnego kroku E, a proces jest powtarzany do osiągnięcia zbieżności, to znaczy do momentu aż pozostaje bardzo mała oscylacja między wartościami w krokach E i M. W sekcji 13.2.3 przedstawiamy przykład algorytmu maksymalizacji oczekiwań.

Maksymalizacja oczekiwań dla HMM (Baum-Welch)

Ważnym narzędziem do uczenia parametrów jest Baum-Welch, wariant algorytmu EM. Oblicza oszacowania największej wiarygodności i oszacowania trybu późniejszego dla parametrów (prawdopodobieństwa przejścia i emisji) HMM lub innego czasowego modelu graficznego, gdy podane są tylko zestawy danych treningowych emisji (obserwowalnych). Algorytm Baum-Welch ma dwa kroki:

1. Oblicz prawdopodobieństwa w przód i wstecz dla każdego stanu czasowego;

2. Na podstawie 1 wyznaczyć częstość wartości par przejścia-emisja i podzielić przez prawdopodobieństwo całej sekwencji.

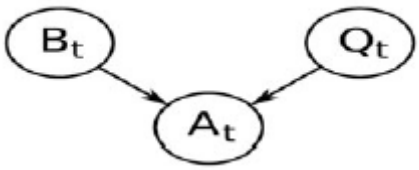
Krok 1 Baum-Welch wykorzystuje algorytm do przodu-do tyłu. Jest to algorytm obsługujący programowanie dynamiczne.

Krok 2 algorytmu Bauma-Welcha służy do obliczenia prawdopodobieństwa wystąpienia określonej sekwencji wyjściowej, przy danych parametrach modelu, w kontekście ukrytych modeli Markowa lub innych modeli czasowych. Krok 2 sprowadza się do obliczenia spodziewanej liczby konkretów para przejście-emisja. Za każdym razem, gdy zostanie znalezione przejście, wartość stosunku przejścia do prawdopodobieństwa całej sekwencji wzrasta i ta wartość może być następnie zmieniona na nową wartość przejścia. Brute force procedura rozwiązywania problemu uczenia parametrów HMM polega na wygenerowaniu wszystkich możliwych sekwencji obserwowanych zdarzeń i stanów ukrytych wraz z ich prawdopodobieństwami przy użyciu dwóch macierzy przejść. Łączne prawdopodobieństwo dwóch sekwencji, biorąc pod uwagę model oblicza się poprzez pomnożenie odpowiednich prawdopodobieństw w macierzach. Procedura ta ma złożoność czasową O ($2TN^T$), gdzie T jest długością

sekwencji obserwowanych zdarzeń, a N jest całkowitą liczbą symboli w alfabecie stanu. Tym razem koszt złożoności jest nie do pokonania w przypadku realistycznych problemów, dlatego wybrana metoda wdrażania Baum-Welch wykorzystuje programowanie dynamiczne. Następnie przedstawiamy prosty przykład uczenia parametrów przy użyciu propagacji zapętlonych przekonań z algorytmem maksymalizacji oczekiwań .

13.2.3 Maksymalizacja oczekiwań: przykład

Zilustrujemy maksymalizację oczekiwań (EM) na przykładzie i używamy algorytmu propagacji zapętlonych przekonań Pearl'a , aby pokazać kroki wnioskowania. Ten przykład został zaimplementowany w programie, a reprezentacja pola losowego Markowa jest zaczerpnięta z logiki pętli ,stochastycznego języka modelowania pierwszego rzędu. Chociaż dostępne są prostsze metody rozwiązywania ograniczeń przedstawionego przez nas przykładu alarmu włamaniowego, naszym celem jest zademonstrowanie ważnego schematu wnioskowania dla BBN, propagacji zapętlonych przekonań i algorytmu EM w zrozumiałym otoczeniu. Rozważmy sieć przekonań bayesowskich przedstawioną na rysunku 13.10, BBN z 3 węzłami, alarmem A, włamaniem B i trzęsieniem ziemi Q.



Model ten opisuje system antywłamaniowy At, aktywny w czasie obserwacji t, zależny przyczynowo od możliwości włamania B_t lub trzęsienie ziemi Qt. Model ten, z pewnymi założonymi miarami prawdopodobieństwa, można opisać w logice pętli przez program:

W | B_t, Q_t = [[[.999, .001], [. 7, .3]], [[. 8, .2], [. 1, .9]]]

Uprościliśmy rzeczywistą składnię logiki pętli, aby prezentacja była wyraźniejsza. W tym programie zakładamy, że wszystkie zmienne są logiczne. Zakładamy również, że BBN z rysunku 10 zawiera określony rozkład prawdopodobieństwa warunkowego (CPD) powyżej. Na przykład tabela podaje, że prawdopodobieństwo nie zadziałania alarmu, biorąc pod uwagę włamanie i trzęsienie ziemi, wynosi 0,001; prawdopodobieństwo uruchomienia alarmu, biorąc pod uwagę ani włamanie, ani trzęsienie ziemi, wynosi 0,1. Zauważ również, że ten program określa ogólną klasę BBN, a raczej BBN dla każdego czasu t. Jest to nieco podobne do podejścia reprezentacyjnego przyjętego przez Kersting i DeRaedt (2000). Jednak logika pętli rozszerza podejście przyjęte przez Kerstinga i DeRaedta na kilka sposobów. Na przykład logika loopy przekształca zdania logiki probabilistycznej, takie jak to powyżej, w pole losowe Markowa, aby zastosować algorytm wnioskowania, propagację zapętlonych przekonań (Pearl 1988). Pole losowe Markowa lub sieć Markowa to probabilistyczny model graficzny, którego struktura jest grafem nieukierunkowanym (w porównaniu z ukierunkowanymi modelami graficznymi sieci przekonań bayesowskich). Węzły grafu struktury pola losowego Markowa odpowiadają zmiennym losowym, a powiązania między węzłami odpowiadają zależnościom między połączonymi zmiennymi losowymi. W logice loopy pole losowe Markowa jest używane jako pośrednia i równoważna reprezentacja oryginalnego modelu graficznego (tak jak drzewo kliki zostało użyte w rozdziale 9.3), pole losowe Markowa wspiera wnioskowanie probabilistyczne, w naszym przypadku propagacja przekonań o pętli . Ilościowe składowe pola losowego Markowa są określone przez zbiór funkcji potencjalnych, których dziedziną jest klika struktury grafowej pola losowego Markowa, i które definiują zgodność między zbiorem wszystkich możliwych wspólnych przypisań do zmiennej z kliki i zbioru

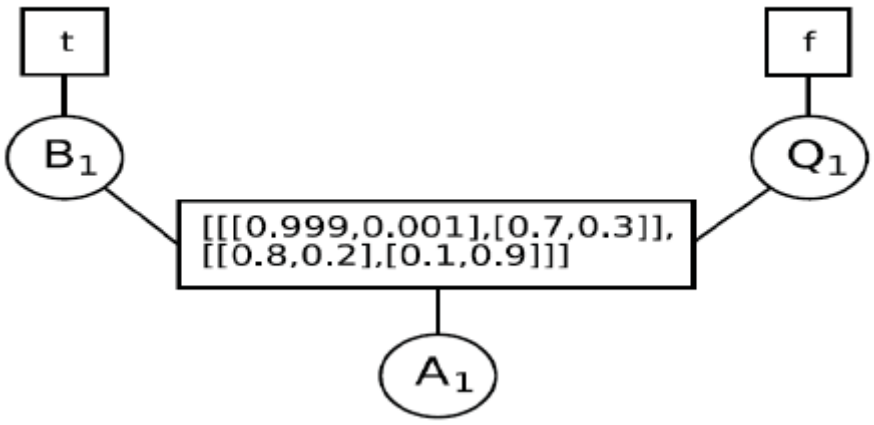
nieujemnych liczb rzeczywistych. W systemie logiki pętli używana jest specjalna wersja pola losowego Markowa. Do każdej zależności między zmiennymi losowymi przypisana jest funkcja potencjalna. W konsekwencji, losowe pole Markowa może być reprezentowane przez dwudzielny graf z dwoma typami węzłów, zmiennymi węzłami i węzłami klastra. Węzły zmienne odpowiadają zmiennym losowym, podczas gdy węzły klastrów odpowiadają potencjalnym funkcjom zmiennych losowych określonych przez sąsiadów węzła klastra. Rysunek 11 ilustruje dwudzielne pole losowe Markowa, które obejmuje potencjalne funkcje BBN z rysunku 10. W logice loopy możemy określić fakty dotyczące modelu i zadawać pytania; na przykład możemy stwierdzić:

B_1 = t.

Q_1 = f.

A_1 =?

Interpretacja tych trzech stwierdzeń jest taka, że wiemy, że w czasie 1 doszło do włamania, nie było trzęsienia ziemi i chcielibyśmy poznać prawdopodobieństwo włączenia alarmu. Używając tych trzech instrukcji wraz z $A_t \mid B_t, Q_t$ = [[[.999, .001], [.7, .3]], [[.8, .2], [.1, .9]]], pętlowy system logiczny konstruuje losowe pole z faktami i potencjalnymi funkcjami przedstawionymi na Rysunku 11.



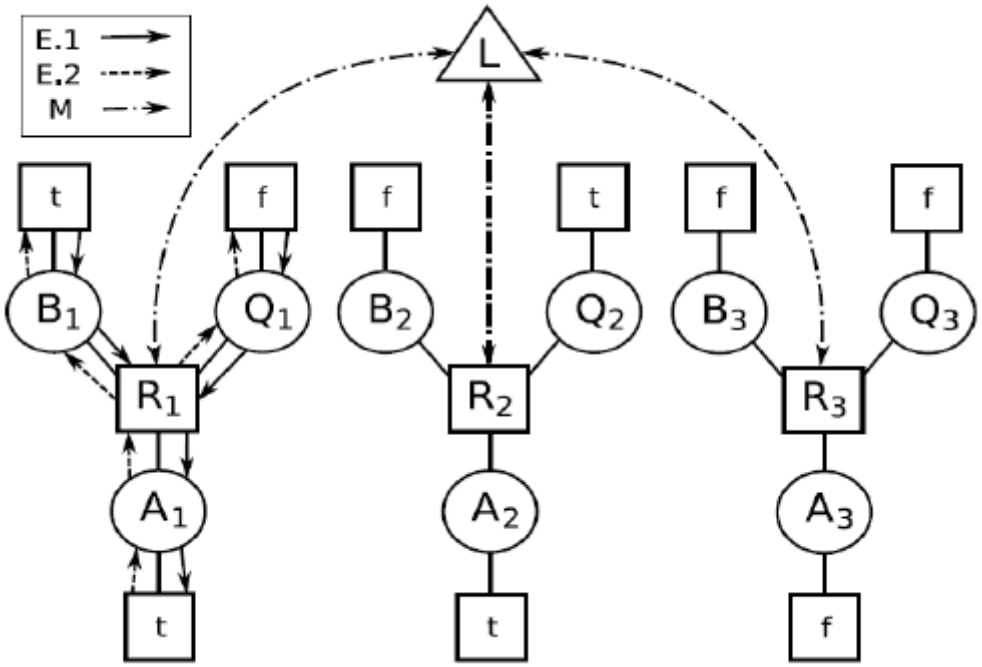
Zauważ, że na rysunku 11 fakty z programu są włączone do pola losowego Markowa jako węzły klastra-liścia, a tabela warunkowego rozkładu prawdopodobieństwa (CPD) jest zakodowana w węźle klastra łączącym trzy zmienne węzły. Aby wywnioskować odpowiedź na to zapytanie, system stosuje algorytm propagacji zapętlonych przekonań do pola losowego Markowa. Ponieważ określiliśmy proste deterministyczne fakty, a oryginalny model graficzny nie ma pętli, algorytm wnioskowania jest dokładny (Pearl 1988) i zbiega się w jednej iteracji, zwracając rozkład dla alarmu, [0.8,0.2]. Ponadto logika pętli wspomaga uczenie się, umożliwiając użytkownikom przypisywanie możliwych do nauczenia dystrybucji do wybranych reguł programu logicznego w pętli. Parametry te można oszacować za pomocą algorytmu przekazywania wiadomości wyprowadzonego z maksymalizacji oczekiwań. Aby to zademonstrować, zakładamy następnie, że nie znamy warunkowego rozkładu prawdopodobieństwa BBN z rysunku 10. W logice pętli, aby uzyskać wartości dla tego rozkładu, użytkownik ustawia CPD jako rozkład, którego można się nauczyć w dowolnym czasie t. Aby to oznaczyć, używamy w programie zmiennej L:

$W \mid B_t, Q_t = L$

Możemy teraz przedstawić zebrany zestaw obserwacji interpreterowi logiki pętli w zbiorze okresów czasu, używając reprezentacji faktów poniżej. Dla uproszczenia zbieramy tylko trzy punkty danych w każdym z trzech przedziałów czasowych:

$Q_1 = f$
$B_1 = t$
$A_1 = t$
$Q_2 = t$
$B_2 = f$
$A_2 = t$
$Q_3 = f$
$B_3 = f$
$A_3 = f$

Logika pętli wykorzystuje kombinację schematu wnioskowania propagacji przekonań pętli w połączeniu z maksymalizacją oczekiwań w celu oszacowania parametrów modelu, których można się nauczyć. Aby to osiągnąć, interpreter konstruuje, jak poprzednio, pole losowe Markowa dla modelu. Dla alarmu BBN pokazanego na rysunku 13.10, odpowiadające mu pole losowe Markowa, z możliwym do nauczenia parametrem L, powiązanym z każdym węzłem klastra, którego reguła jest ujednolicona z regułą możliwą do nauczenia przedstawioną wcześniej (A_t | B_t, Q_t = L), przedstawiono na rysunku 12.



Następnie wypełniamy znane rozkłady BBN i dokonujemy losowej inicjalizacji w węzłach klastra R_1, R_2 i R_3, a także w węźle możliwym do nauczenia L. Na koniec system ten jest przedstawiany z danymi reprezentującymi trzy okresy, w których znane fakty są włączane do węzłów klastra typu liść, jak poprzednio. Rysunek 12 przedstawia wynik. Algorytm maksymalizacji oczekiwań jest procesem iteracyjnym realizowanym poprzez przekazywanie wiadomości. Po każdej iteracji bieżące przybliżenie tabeli prawdopodobieństwa warunkowego jest wiadomością możliwego do nauczenia się węzła do każdego z jego połączonych węzłów klastra. Kiedy węzeł, którego można się nauczyć, jest aktualizowany, każdy sąsiedni węzeł klastra wysyła wiadomość do L (strzałki-kropki oznaczone M na

rysunku 12). Ta wiadomość jest wynikiem wszystkich wiadomości przychodzących do tego węzła klastra z sąsiednich węzłów zmiennych. Te (nie znormalizowane) tabele są oszacowaniem wspólnego prawdopodobieństwa w każdym węźle klastra. Jest to rozkład na wszystkie stany zmiennych warunkowych i warunkujących. Węzeł, którego można się nauczyć, pobiera sumę wszystkich tych komunikatów klastra i konwertuje je na znormalizowaną tabelę prawdopodobieństwa warunkowego. Dokonując wnioskowania (propagacja zapętlonych przekonań) na klastrze i węzłach zmiennych, obliczamy komunikat dla możliwych do nauczenia się węzłów. Odzwierciedlając dwudzielną strukturę podstawowego pola losowego Markowa, propagacja zapętlonych przekonań odbywa się z dwoma etapami przekazywania wiadomości: wysyłaniem wiadomości ze wszystkich węzłów klastra do ich sąsiednich węzłów zmiennych, pełnymi strzałkami oznaczonymi E.1 na rysunku 13.12, a następnie z powrotem , przerywane strzałki oznaczone E.2 na rysunku 13.12. Zastosowanie tego algorytmu propagacji przekonań do momentu, gdy zbieżność da przybliżenie oczekiwanych wartości. Jest to równoważne z krokiem oczekiwania w algorytmie EM. Uśrednianie, które ma miejsce we wszystkich możliwych do nauczenia węzłach klastra, odpowiada krokowi maksymalizacji zwracającemu oszacowanie maksymalnego prawdopodobieństwa parametrów w możliwym do nauczenia się węźle. Iteracja, która obsługuje konwergencję w węzłach zmiennych i klastrach, po której następuje aktualizacja węzłów, których można się nauczyć, a następnie kontynuowanie iteracji jest równoważne z pełnym algorytmem EM.

Podsumowując algorytm, jak pokazano na rysunku 12:

W oczekiwaniu lub kroku E:

1. Wszystkie węzły klastra wysyłają swoje komunikaty do węzłów zmiennych.

2. Wszystkie węzły zmienne wysyłają swoje komunikaty z powrotem do węzłów klastra, aktualizując informacje o węzłach klastra.

W kroku maksymalizacji lub M:

1. Węzły klastra wysyłają komunikaty zaktualizowane w kroku E. do możliwych do nauczenia się węzłów, gdzie są uśrednione.

W opisanym algorytmie węzły mogą być zmieniane w dowolnej kolejności, a aktualizacje klastra i węzłów zmiennych mogą pokrywać się z aktualizacjami węzłów uczących się. Ten iteracyjny proces aktualizacji reprezentuje rodzinę algorytmów w stylu EM, z których niektóre mogą być bardziej wydajne niż standardowe EM dla niektórych domen. Rozszerzeniem algorytmicznym, które również obsługuje ta struktura, jest uogólniony algorytm propagacji przekonań zaproponowany przez Yedidia i innych.

13.3 Stochastyczne rozszerzenia uczenia się ze wzmocnieniem

W sekcji z Części 10 po raz pierwszy wprowadziliśmy uczenie się ze wzmocnieniem, gdzie wraz ze wzmocnieniem nagrody agent uczy się podejmować różne zestawy działań w środowisku. Celem agenta jest maksymalizacja długoterminowej nagrody. Ogólnie rzecz biorąc, agent chce nauczyć się polityki lub mapowania między stanami nagrody na świecie a jego własnymi działaniami na świecie. Aby zilustrować koncepcje uczenia się ze wzmocnieniem, rozważmy przykład robota recyklingowego. Rozważmy robota mobilnego, którego zadaniem jest zbieranie pustych puszek po napojach z biur. Robot posiada akumulator. Wyposażony jest w czujniki do wyszukiwania puszek oraz w ramię do ich zbierania. Zakładamy, że robot posiada system sterowania do interpretacji informacji sensorycznych, nawigacji i poruszania ramieniem w celu zbierania puszek. Robot wykorzystuje uczenie się przez

wzmacnianie oparte na aktualnym poziomie naładowania swojej baterii, aby wyszukać puszki. Agent musi podjąć jedną z trzech decyzji:

1. Robot może aktywnie szukać puszki po napojach w określonym przedziale czasu;

2. Robot może się zatrzymać i czekać, aż ktoś przyniesie mu puszkę; lub

3. Robot może wrócić do bazy domowej, aby naładować swoje baterie.

Robot podejmuje decyzje w określonym przedziale czasowym, po znalezieniu pustej puszki lub w przypadku wystąpienia innego zdarzenia. W konsekwencji robot wykonuje trzy akcje, a jego stan jest określany przez poziom naładowania akumulatora. Nagroda jest ustawiona na zero przez większość czasu staje się dodatnia po zebraniu puszki, a nagroda jest ujemna, jeśli bateria wyczerpie się. Należy zauważyć, że w tym przykładzie agent uczący się oparty na wzmocnieniach jest częścią robota, która monitoruje zarówno stan fizyczny robota, jak i sytuację (stan) jego otoczenia zewnętrznego: agent robot monitoruje zarówno stan robota a także otoczenie zewnętrzne. Istnieje poważne ograniczenie tradycyjnej struktury uczenia się przez wzmacnianie, którą właśnie opisaliśmy: ma ona charakter deterministyczny. Aby przezwyciężyć to ograniczenie i móc wykorzystać uczenie się przez wzmacnianie w szerszym zakresie złożonych zadań, rozszerzamy ramy deterministyczne o składnik stochastyczny. W kolejnych rozdziałach rozważymy dwa przykłady stochastycznego uczenia się ze wzmocnieniem: proces decyzyjny Markowa i częściowo obserwowalny proces decyzyjny Markowa.

13.3.1 Proces decyzyjny Markowa (lub MDP)

Przykłady uczenia się przez wzmacnianie przedstawione do tej pory, zarówno w sekcji 10.7, jak i we wprowadzeniu do sekcji 13.3, miały charakter deterministyczny. W efekcie, gdy agent wykonuje akcję z określonego stanu w czasie t, zawsze skutkuje to deterministycznie określony nowy stan w czasie t +1: system jest całkowicie deterministyczny. Jednak w wielu rzeczywistych zastosowaniach uczenia się przez wzmacnianie może tak nie być: agent nie ma pełnej wiedzy ani kontroli nad następnym stanem świata. W szczególności, podjęcie działania ze stanu st może prowadzić do więcej niż jednego możliwego stanu wynikowego w czasie t + 1. Na przykład w grze w szachy, kiedy wykonujemy określony ruch, często nie wiemy, jaki ruch wykona nasz przeciwnik. Nie mamy pełnej wiedzy, w jakim stanie będzie świat w wyniku podjęcia naszego ruchu. W rzeczywistości ta niepewność dotyczy wielu gier wieloosobowych. Drugi przykład to loteria lub inne gry losowe. Wybieramy ruch niepewny co do prawdopodobnej nagrody. Trzeci przykład dotyczy podejmowania decyzji przez jednego agenta (a nie rywalizacji z jednym lub większą liczbą innych agentów), kiedy świat jest tak złożony, że deterministyczna odpowiedź na wybór stanu nie jest obliczalna. Ponownie, w tej sytuacji odpowiedź oparta na prawdopodobieństwach może być najlepszą, jaką możemy uzasadnić. We wszystkich tych sytuacjach potrzebujemy mocniejszych ram, które pozwolą rozwiązać problem uczenia się przez wzmacnianie. Jedną z takich ram jest proces decyzyjny Markowa (lub MDP). MDP są oparte na własności Markowa pierwszego rzędu, która stwierdza, że przejście do następnego stanu jest reprezentowane przez rozkład prawdopodobieństwa, który zależy tylko od aktualnego stanu i możliwych działań. Jest niezależny od wszystkich stanów poprzedzających stan obecny. MDP to stochastyczne procesy dyskretne składające się z zestawu stanów, zestawu działań, które mogą być wykonane z każdego stanu oraz funkcji nagrody związanej z każdym stanem. W rezultacie MDP zapewniają bardzo potężne ramy, które można wykorzystać do modelowania szerokiej klasy sytuacji uczenia się przez wzmacnianie. Następnie definiujemy MDP:

DEFINICJA

PROCES DECYZJI MARKOV, czyli MDP

Proces decyzyjny Markowa to krotka <S, A, P, R>, gdzie:

S jest zbiorem stanów i

A to zbiór działań.

$p_a(s_t, s_{t+1}) = p(s_{t+1} \mid st, a_t = a)$ jest prawdopodobieństwem, że jeśli agent wykona akcję $a \in A$ ze stanu s_t w czasie t, skutkuje to stanem s_{t+1} w czasie t +1. Ponieważ prawdopodobieństwo $p_a \in P$ jest zdefiniowane w całej przestrzeni stanów działań, często jest przedstawiane za pomocą macierzy przejść.

R(s) to nagroda otrzymana przez agenta w stanie s.

Powinniśmy zauważyć, że jedyną różnicą między schematem MDP dla uczenia się ze wzmocnieniem a prezentacją uczenia się ze wzmocnieniem z sekcji z Części 10 jest to, że funkcja przejścia jest teraz zastąpiona funkcją rozkładu prawdopodobieństwa. Jest to ważna modyfikacja naszej teorii, która pozwala nam uchwycić niepewność w świecie, gdy świat jest albo nieobliczalnie złożony, albo w rzeczywistości jest stochastyczny. W takich przypadkach reakcja nagrody na działanie agenta jest najlepiej przedstawiana jako rozkład prawdopodobieństwa. Zauważyliśmy, że MDP można wykorzystać do modelowania gry takiej jak szachy, w której następny stan świata może znajdować się poza deterministyczną kontrolą agenta. Kiedy wykonujemy ruch w szachach, wynikowy stan gry zależy od ruchu naszego przeciwnika. Dzięki temu jesteśmy świadomi obecnego stanu, w jakim się znajdujemy, jesteśmy świadomi, jakie działanie (ruch) podejmujemy, ale wypadkowy stan nie jest dla nas natychmiast dostępny. Mamy tylko rozkład prawdopodobieństwa na zestaw możliwych stanów, w których prawdopodobnie się znajdziemy, w zależności od ruchu naszego przeciwnika. Rozważmy teraz grę w pokera. Tutaj nie jesteśmy nawet pewni naszego obecnego stanu! Znamy karty, które posiadamy, ale nie mamy doskonałej wiedzy o kartach naszych przeciwników. Możemy się tylko domyślać, być może dzięki wiedzy z licytacji, jakie karty ma nasz przeciwnik. Świat jest jeszcze bardziej niepewny niż świat MDP. Nie tylko nie wiemy, w jakim stanie się znajdujemy, ale musimy również wykonać akcję w oparciu o nasze przypuszczenia co do rzeczywistego stanu świata. W rezultacie nasze przypuszczenia co do wywołanego działaniem nowego stanu świata będą jeszcze bardziej niepewne. Dlatego potrzebujemy bogatszych ram niż MDP, aby modelować tę podwójnie niepewną sytuację. Jedną z ram realizacji tego zadania jest częściowo obserwowalny proces decyzyjny Markowa (lub POMDP). Nazywa się to POMDP, ponieważ możemy tylko częściowo obserwować stan, w którym się znajdujemy, oraz reakcję przeciwnika. Zamiast doskonałej wiedzy o naszym obecnym stanie, mamy tylko rozkład prawdopodobieństwa na możliwy zbiór stanów, w których moglibyśmy się ewentualnie znaleźć. Definiujemy POMDP:

DEFINICJA

CZĘŚCIOWO OBSERWOWALNY PROCES DECYZJI MARKOV lub POMDP

Częściowo obserwowalny proces decyzyjny Markowa to krotka <S, A, O, P, R>, gdzie:

S jest zbiorem stanów i

A to zbiór działań.

O to zbiór obserwacji oznaczających, co agent może zobaczyć w swoim świecie. Ponieważ agent nie może bezpośrednio obserwować swojego aktualnego stanu, obserwacje są prawdopodobnie powiązane z podstawowym faktycznym stanem świata. $p_a(s_t, o, s_{t+1}) = p(s_{t+1}, o_t = o \mid s_t, a_t = a)$ to prawdopodobieństwo, że gdy agent wykona akcję a ze stanu s_t w czasie t, skutkuje to obserwacją o co

prowadzi do stanu podstawowego s_{t+1} w czasie t + 1. R (s_t, a, s_{t+1}) to nagroda otrzymana przez agenta, gdy wykonuje akcję a w stanie s_t i przechodzi do stanu s_{t+1}.

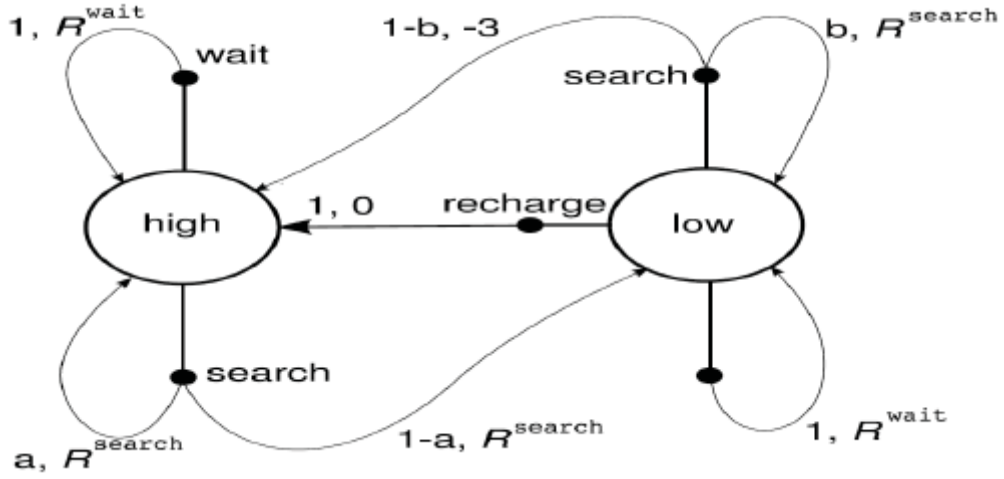
Podsumowując, MDP można uznać za szczególny przypadek POMDP, gdy obserwacja o daje nam dokładne wskazanie stanu aktualnego. W zrozumieniu tego rozróżnienia pomocna może być analogia: MDP jest powiązany z łańcuchem Markowa, tak jak POMDP jest powiązany z ukrytym modelem Markowa. Istnieje kilka algorytmów tworzenia MDP i POMDP. Złożoność tych algorytmów jest oczywiście większa niż w przypadku czysto deterministycznych przypadków przedstawionych wcześniej. W rzeczywistości algorytmy rozwiązywania POMDP są obliczeniowo niewykonalne. Oznacza to, że znalezienie optymalnego rozwiązania problemu przy użyciu POMDP w czasie wielomianowym może być niemożliwe, a rozwiązanie może być rzeczywiście nieobliczalne. W rezultacie rozwiązujemy te problemy za pomocą algorytmów, które przybliżają optymalne rozwiązanie.

13.3.3 Przykładowa implementacja procesu decyzyjnego Markowa

Aby zilustrować prosty MDP, ponownie zajmiemy się przykładem robota recyklingowego przedstawionego na początku sekcji 13.3. Przypomnij sobie, że za każdym razem, gdy ten agent uczący się wzmacniania napotyka zdarzenie zewnętrzne, podejmuje decyzję o aktywnym poszukiwaniu puszki po napojach do recyklingu, nazywamy to wyszukiwaniem, czekamy, aż ktoś przyniesie puszkę, nazywamy to czekaniem lub powrotem do bazę domową, aby naładować baterię, naładuj. Te decyzje są wykonywane przez robota wyłącznie na podstawie stanu energii poziom naładowania baterii. Dla uproszczenia rozróżnia się dwa poziomy baterii, niski i wysoki: przestrzeń stanów S = {niski, wysoki}. Jeśli aktualny stan akumulatora jest wysoki, akumulator jest ładowany, w przeciwnym razie jest niski. Dlatego zestawy działań agenta to A (niski) = {szukaj, czekaj, doładuj} i A (wysoki) = {szukaj, czekaj}. Jeśli A (wysoka) obejmuje ładowanie, spodziewalibyśmy się, że agent nauczy się, że polityka wykorzystująca to działanie byłaby nieoptymalna! Stan baterii jest określany niezależnie od podjętych działań. Jeśli jest wysoka, spodziewamy się, że robot będzie w stanie ukończyć okres aktywnego poszukiwania bez ryzyka wyczerpania baterii. Stąd, jeśli stan jest wysoki, to agent pozostaje w tym stanie z prawdopodobieństwem a i zmienia swój stan na niski z prawdopodobieństwem 1-a. Jeśli agent zdecyduje się przeprowadzić aktywne wyszukiwanie będąc w stanie niskim, poziom energii baterii pozostanie to samo z prawdopodobieństwem b, a bateria wyczerpie się z prawdopodobieństwem 1–b. Następnie tworzymy system nagród: gdy bateria jest wyczerpana, S = niski, robot jest uratowany, jego bateria jest ładowana do wysokiego poziomu i otrzymuje nagrodę w wysokości -3. Za każdym razem, gdy robot znajdzie puszkę, otrzymuje nagrodę w wysokości 1. Oczekiwaną ilość puszek oznaczamy robot zbierze (oczekiwaną nagrodę, którą agent otrzyma) podczas aktywnego wyszukiwania i podczas oczekiwania przez R_search i R_wait, i przyjmie, że R_search> R_wait. Zakładamy również, że robot nie może odebrać żadnych puszek po napojach wracając do naładowania oraz że nie może zebrać żadnych puszek podczas kroku, na którym bateria jest słaba. Opisany właśnie system może być reprezentowany przez skończone MDP, którego prawdopodobieństwa przejścia (pa (s_t, s_{t+1})) i oczekiwane nagrody są podane w tabeli 1.

s_t	s_{t+1}	a_t	$p_a(s_t,s_{t+1})$	$R_a(s_t,s_{t+1})$
high	high	search	a	R_search
high	low	search	1 - a	R_search
low	high	search	1 - b	-3
low	low	search	b	R_search
high	high	wait	1	R_wait
high	low	wait	0	R_wait
low	high	wait	0	R_wait
low	low	wait	1	R_wait
low	high	recharge	1	0
low	low	recharge	0	0

Aby przedstawić dynamikę tego skończonego MDP, możemy użyć wykresu, takiego jak Rysunek 13, wykres przejścia MDP dla robota recyklingowego.



Graf przejścia ma dwa typy węzłów: węzły stanu i węzły działań. Każdy stan agenta jest reprezentowany przez węzeł stanu przedstawiony jako duży okrąg z nazwą stanu, s, wewnątrz. Każda para stan-akcja jest reprezentowana przez węzeł akcji połączony z węzłem stanu oznaczonym jako małe czarne kółko. Jeśli agent uruchomi się w stanie s i podejmie akcję a, porusza się wzdłuż linii od węzła stanu s do węzła akcji (s_t, a). W konsekwencji środowisko reaguje przejściem do węzła następnego stanu przez jedną ze strzałek wychodzących z węzła akcji (st, a), gdzie każda strzałka odpowiada trójce (s_t, a, s_{t+1}), gdzie s_{t+1} to następny stan. Strzałka jest oznaczona prawdopodobieństwem przejścia, pa (s_t, s_{t+1}), a oczekiwana nagroda za przejście, $R_a(s_t, s_{t+1})$, jest reprezentowana przez strzałkę. Zwróć uwagę, że prawdopodobieństwa przejścia związane ze strzałkami opuszczającymi węzeł akcji zawsze sumują się do 1.

Sztuczna inteligencja : Automatyczne wnioskowanie

(XIV / XVI)

ĆWICZENIA

1. Weź doradcę finansowego opartego na logice z sekcji Części 2.4, umieść predykaty opisujące problem w formie klauzuli i użyj odrzucenia rozwiązania, aby odpowiedzieć na pytania, takie jak czy konkretny inwestor powinien dokonać inwestycji (kombinacji).

2. Użyj rozdzielczości, aby udowodnić twierdzenie Wirtha w ćwiczeniu 12, Część 2.

3. Użyj rozdzielczości, aby odpowiedzieć na pytanie w przykładzie 3.3.4.

4. W Części 5 przedstawiliśmy uproszczoną formę oprowadzania rycerskiego. Weź regułę path3, umieść ją w formie klauzuli i użyj rozdzielczości, aby odpowiedzieć na zapytania, takie jak path3 (3,6). Następnie użyj rekurencyjnego wywołania ścieżki w formie klauzuli, aby odpowiedzieć na zapytania.

5. W jaki sposób możesz użyć rozdzielczości do zaimplementowania wyszukiwania „systemu produkcyjnego”?

6. Jak przeprowadziłbyś rozumowanie oparte na danych z rozwiązaniem? Skorzystaj z tego, aby zająć się przestrzenią poszukiwań Ćwiczenia 1. Jakie problemy mogą pojawić się w dużej przestrzeni problemów?

7. Użyj rozwiązania dla zapytań w problemie rolnika, wilka, kozy i kapusty.

8. Użyj rozdzielczości, aby rozwiązać następującą zagadkę z Wos . Cztery osoby: Roberta, Thelma, Steve i Pete zajmują osiem różnych stanowisk, a każda osoba ma dokładnie dwie prace. Zawody to kucharz, strażnik, pielęgniarka, telefonista, policjant, nauczyciel, aktor i bokser. Pielęgniarka jest mężczyzną. Mąż szefa kuchni jest telefonistą. Roberta nie jest bokserem. Pete nie ma wykształcenia powyżej dziewiątej klasy. Roberta, szef kuchni i policjant grali razem w golfa. Kto piastuje jakie stanowiska? Pokaż, jak dodanie uprzedzeń dotyczących płci może zmienić problem.

9. Opracuj dwa przykłady hiper-rozdzielczości, w których jądro ma co najmniej cztery literały.

10. Napisz demodulator dla sumy, który powodowałby redukcję klauzul o postaci równej (ans, sum (5, sum (6, minus (6)))) do równej (ans, sum (5, 0)). Napisz kolejny demodulator, aby zredukować ten ostatni wynik do równości (ans, 5).

11. Wybierz „kanoniczny zestaw” sześciu relacji rodzinnych. Napisz demodulatory, aby zredukować alternatywne formy relacji do zbioru. Na przykład „brat Twojej matki” to „wujek”.

12. Weź szczęśliwy problem studenta z rysunku 5 i zastosuj do jego rozwiązania trzy strategie obalenia z rozdziału 14.2.4.

13. Umieść następujące wyrażenie rachunku predykatów w postaci klauzuli:

∀ (X) (p (X) → {∀ (Y) [p (Y) → p (f (X, Y))] ∧ ¬ ∀ (Y) [q (X, Y) → p (Y)] })

14. Utwórz wykres i / lub wykres dla poniższej dedukcji rachunku predykatów opartego na danych.

Fakt: ¬ d (f) ∨ [b (f) ∧ c (f)].

Reguły: ¬ d (X) → ¬a (X) ib (Y) → e (Y) ig (W) ← c (W).

Udowodnić: ¬ a (Z) ∨ e (Z).

15. Udowodnij, że strategia liniowego formularza wejściowego nie jest zakończona.

16. Utwórz wykres i / lub wykres dla następującego problemu. Dlaczego nie można osiągnąć celu:

r (Z) ∨ s (Z)?

Fakt: p (X) ∨ q (X).

Reguły: p (a) → r (a) iq (b) → s (b).

17. Skorzystaj z faktoringu i rozdzielczości, aby zaprzeczyć następującym klauzulom:

p (X) ∨ p (f (Y)) i ¬ p (W) ∨ ¬ p (f (Z)). Spróbuj obalić bez faktoringu.

18. Wyprowadź dowód rozdzielczości twierdzenia z rysunku 14.1.

19. Alternatywnym modelem semantycznym dla programowania logicznego jest model Flat Concurrent Prolog. Porównaj Prolog, jak pokazano w sekcji 14.3, z Flat Concurrent Prolog.

AUTOMATYCZNE WNIOSKOWANIE

14.0 Wprowadzenie do słabych metod dowodzenia twierdzeń

Wos i inni opisują automatyczny program rozumowania jako taki, który „wykorzystuje jednoznaczną i dokładną notację do przedstawiania informacji, precyzyjne reguły wnioskowania do wyciągania wniosków oraz starannie nakreślone strategie kontrolowania tych reguł wnioskowania". Dodają, że stosowanie strategii do reguł wnioskowania w celu wydedukowania nowych informacji to sztuka: „Dobry wybór reprezentacji obejmuje zapis, który zwiększa szansę rozwiązania problemu i zawiera informacje, które choć nie są potrzebne, są pomocne. Dobry wybór reguł wnioskowania to taki, który dobrze pasuje do wybranej reprezentacji. Dobry wybór strategii to taki, który kontroluje reguły wnioskowania w sposób, który znacznie zwiększa skuteczność programu rozumowania ". Zautomatyzowane rozumowanie, jak właśnie opisano, wykorzystuje słabe metody rozwiązywania problemów. Używa jednolitej reprezentacji, takiej jak rachunek predykatów pierwszego rzędu (Część 2), rachunek z klauzulą Horna (sekcja 14.3) lub formularz klauzuli używany do rozstrzygania (sekcja 14.2). Jego reguły wnioskowania są rozsądne i, o ile to możliwe, kompletne. Wykorzystuje ogólne strategie, takie jak przeszukiwanie najpierw wszerz, najpierw w głąb lub najpierw najlepszy, oraz, jak widzimy w tym rozdziale, heurystyki, takie jak zestaw wsparcia i preferencje jednostek w celu zwalczania kombinatoryki wyszukiwania wyczerpującego. Projektowanie strategii wyszukiwania, a zwłaszcza heurystycznych strategii wyszukiwania, jest w dużej mierze sztuką; nie możemy zagwarantować, że znajdą użyteczne rozwiązanie problemu przy użyciu rozsądnej ilości czasu i pamięci.

Rozwiązywanie problemów metodami słabymi jest samo w sobie ważnym narzędziem, a także podstawową podstawą do rozwiązywania problemów z silnymi metodami. Systemy produkcyjne i powłoki systemów eksperckich oparte na regułach są przykładami słabych metod rozwiązywania problemów. Chociaż reguły systemu produkcyjnego lub systemu eksperckiego opartego na regułach kodują silne heurystyki rozwiązywania problemów, ich zastosowanie jest zwykle wspierane przez ogólne (słabe metody) strategie wnioskowania.

Techniki rozwiązywania problemów metodą słabych metod były od samego początku przedmiotem zainteresowania badań nad sztuczną inteligencją. Często te techniki wchodzą w zakres dowodzenia twierdzeń, chociaż wolimy bardziej ogólne automatyczne rozumowanie tytułu. Rozpoczynamy ten rozdział (Sekcja 14.1) od wczesnego przykładu automatycznego rozumowania, problemu ogólnego

Solver i jego użycie metod średnich kończy analizę i tabele różnic w celu sterowania wyszukiwaniem. W sekcji 14.2 przedstawiamy ważny produkt badań nad automatycznym rozumowaniem, system obalania rozdzielczości. Omówimy język reprezentacji, regułę wnioskowania o rozdzielczości, strategie wyszukiwania i procesy ekstrakcji odpowiedzi stosowane w dowodzeniu twierdzeń o rozdzielczości. Jako przykład rozumowania klauzuli Horn, w sekcji 14.3 opisujemy mechanizm wnioskowania dla Prologu i pokazujemy, jak ten język przyczynia się do filozofii programowania deklaratywnego z użyciem interpretera opartego na twierdzeniu o rozwiązaniu. Kończymy ten rozdział (sekcja 14.4) krótkimi komentarzami na temat naturalnej dedukcji, traktowania równości i bardziej wyrafinowanych reguł wnioskowania.

14.1 Ogólne rozwiązywanie problemów i tabele różnic

The General Problem Solver (GPS) wyszedł z badań Allena Newella i Herberta Simona na Carnegie Mellon University, a następnie Carnegie Institute of Technology. Jego korzenie sięgają wcześniejszego programu komputerowego zwanego Teoretykiem Logiki (LT) Newella, Shawa i Simona. Program LT dowiódł wielu twierdzeń zawartych w Principia Mathematica Russella i Whiteheada (Whitehead i Russell 1950). Podobnie jak w przypadku wszystkich słabych metod rozwiązywania problemów, teoretyk logiki zastosował jednolity środek reprezentacji i solidne reguły wnioskowania oraz przyjął kilka strategii lub metod heurystycznych, aby pokierować procesem rozwiązywania. Teoretyk logiki użył rachunku zdań jako medium reprezentacji. Reguły wnioskowania to podstawianie, zastępowanie i odłączanie. Podstawienie umożliwia zastąpienie dowolnego wyrażenia w każdym wystąpieniu symbolu w zdaniu, które jest aksjomatem lub twierdzeniem, o którym już wiadomo, że jest prawdziwe. Na przykład, (B ∨ B) → B może mieć wyrażenie ¬ A podstawione za B, aby wytworzyć (¬ A ∨ ¬ A) → ¬ A. Zastąpienie umożliwia zastąpienie łącznika jego definicją lub równoważną formą. Na przykład logiczna równoważność ¬ A ∨ B i A → B mogą prowadzić do zamiany (¬ A ∨ ¬ A) na (A → ¬ A). Oderwanie jest zasadą wnioskowania, którą nazwaliśmy modus ponens (rozdział 2). LT stosuje te reguły wnioskowania w sposób wnikliwy i ukierunkowany na cel do twierdzenia, które ma zostać udowodnione, próbując znaleźć serię operacji, które prowadzą do aksjomatów lub twierdzeń, o których wiadomo, że są prawdziwe. Strategia LT składa się z czterech metod zorganizowanych w ramach rutyny wykonawczej:

Po pierwsze, metoda podstawiania jest bezpośrednio stosowana do obecnego celu, próbując dopasować go do wszystkich znanych aksjomatów i twierdzeń.

Po drugie, jeśli nie doprowadzi to do udowodnienia, wszystkie możliwe oderwania i zastąpienia są stosowane do celu i każdy z tych wyników jest testowany pod kątem sukcesu za pomocą zastępowania. Jeśli podstawienie nie pasuje do żadnego z nich z celem, są one dodawane do listy podproblemów.

Po trzecie, metoda łańcuchowa, wykorzystująca przechodniość implikacji, jest używana do znalezienia nowego podproblemu, którego rozwiązanie dostarczyłoby dowodu. Zatem, jeśli a → c jest problemem i b → c zostanie znalezione, to a → b jest ustawiany jako nowy podproblem.

Po czwarte, jeśli pierwsze trzy metody zawiodą w pierwotnym problemie, przejdź do listy podproblemów i wybierz następny nierozpróbowany podproblem.

Procedura wykonawcza kontynuuje stosowanie tych czterech metod, dopóki nie zostanie znalezione rozwiązanie, nie pozostanie więcej problemów na liście podproblemów lub nie wyczerpią się pamięć i czas poświęcony na znalezienie dowodu. W ten sposób teoretyk logiki przeprowadza ukierunkowane na cel, przeszukiwanie wszerz w przestrzeni problemowej. Częścią rutyny wykonawczej, która umożliwia podstawianie, zastępowanie i wnioskowanie o oderwaniu, jest proces dopasowywania. Załóżmy, że chcemy udowodnić p → (q → p). Proces dopasowywania najpierw identyfikuje jeden z aksjomatów, p → (q ∨ p), jako bardziej odpowiedni niż inne - to znaczy bardziej zbliżony pod względem różnicy zdefiniowanej w dziedzinie - ponieważ główny łącznik, tutaj →, jest taki sam w oba wyrażenia. Po drugie, proces dopasowywania potwierdza, że wyrażenia po lewej stronie głównego łącznika są identyczne. Wreszcie, dopasowanie identyfikuje różnicę między wyrażeniami na prawo od głównego łącznika. Ta końcowa różnica między → a ∨ sugeruje oczywistą alternatywę dla udowodnienia twierdzenia. Proces dopasowywania pomaga kontrolować (wyczerpujące) wyszukiwanie, które byłoby konieczne do zastosowania wszystkich podstawień, zastąpień i odłączeń. W rzeczywistości dopasowanie wyeliminowało wystarczającą liczbę prób i błędów, aby LT stał się skutecznym narzędziem do rozwiązywania problemów. Przykładowy dowód LT pokazuje moc procesu dopasowywania. Twierdzenie 2.02 z Principia Mathematica to p → (q → p). Dopasowanie znajduje aksjomat p → (q ∨ p) jako odpowiedni do zamiany. Podstawienie ¬ q przez q dowodzi twierdzenia. Reguły dopasowywania, kontrolowania podstawiania i zastępowania potwierdziły to twierdzenie bezpośrednio, bez przeszukiwania innych aksjomatów lub twierdzeń. W innym przykładzie załóżmy, że chcemy, aby LT udowodnił:

(p → ¬ p) → ¬ p.

1. (A ∨ A) → A Dopasowanie identyfikuje najlepszy z pięciu dostępnych aksjomatów.

2. (¬ A∨¬ A) → ¬A Zastąpienie ¬ A A, aby złożyć wniosek

3. (A → ¬A) → ¬ Zastąpienie → dla ∨ i ¬, po którym następuje

4. (p → ¬ p) → ¬ p podstawienie p za A.

CO BYŁO DO OKAZANIA

Oryginalny LT udowodnił to twierdzenie w około 10 sekund, używając pięciu aksjomatów. Rzeczywisty dowód wymagał dwóch kroków i nie wymagał przeszukiwania. Dopasowanie wybrało odpowiedni aksjomat dla pierwszego kroku, ponieważ jego forma była bardzo podobna do wniosku, który próbowano ustalić: (wyrażenie) → zdanie. Następnie ¬ A zostało zastąpione przez A. Pozwoliło to na zastąpienie drugiego i ostatniego kroku, który sam w sobie był motywowany celem wymagającym → zamiast a ∨. Teoretyk logiki był nie tylko pierwszym przykładem zautomatyzowanego systemu rozumowania, ale także wykazał znaczenie strategii wyszukiwania i heurystyki w programie rozumowania. W wielu przypadkach LT znalazł rozwiązania w kilku krokach, których wyczerpujące wyszukiwanie może nigdy nie znaleźć. Niektóre twierdzenia nie zostały rozwiązane przez LT, a Newell i inni wskazali usprawnienia, które mogą umożliwić ich rozwiązanie. Mniej więcej w tym czasie badacze z Carnegie i inni z Yale zaczęli badać protokoły myślenia na głos ludzi rozwiązujących problemy logiczne. Chociaż ich głównym celem było zidentyfikowanie ludzkich procesów, które mogłyby

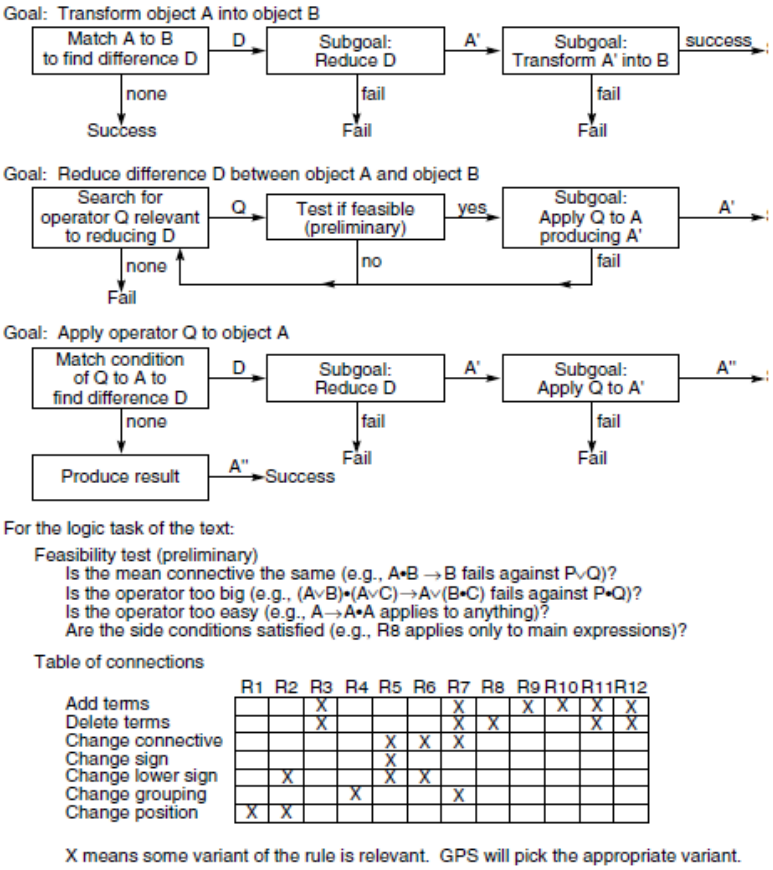
rozwiązać tę klasę problemów, naukowcy zaczęli porównywać rozwiązywanie ludzkich problemów z programami komputerowymi, takimi jak teoretyk logiki. Miało to stać się pierwszym przykładem tego, co obecnie nazywa się psychologią przetwarzania informacji, w której wyjaśnienia obserwowanego zachowania organizmu dostarcza program prymitywnych procesów informacyjnych, które generują to zachowanie. Badania te były również jednymi z pierwszych prac, które stworzyły współczesną dyscyplinę kognitywistyki. Bliższa analiza tych pierwszych protokołów wykazała pod wieloma względami, że rozwiązania LT różniły się od rozwiązań stosowanych przez ludzi. Ludzkie zachowanie dowiodło mocnych dowodów na mechanizm dopasowywania i redukcji różnic nazywany analizą średnich i celów. W analizie średnich i końcowych metody redukcji różnicy (średnie) były silnie powiązane z konkretnymi różnicami, które miały być redukowane (końcami): operatory redukcji różnicy były indeksowane o różnice, które mogły zmniejszyć. W bardzo prostym przykładzie, jeśli instrukcja początkowa byłaby p → q, a celem było ¬ p ∨ q, różnice obejmowałyby symbol → na początku i ∨ w bramce, a także różnicę p na początku i ¬ p w bramce. Tabela różnic zawierałaby różne sposoby zastąpienia → przez ∨ i usunięcia ¬. Te transformacje będą podejmowane pojedynczo, aż różnice zostaną usunięte i twierdzenie zostanie udowodnione. W najciekawszych problemach różnice między startem a bramką nie mogły być bezpośrednio zmniejszone. W tym przypadku szukano operatora (z tabeli), aby częściowo zmniejszyć różnicę. Cała procedura była stosowana rekurencyjnie do tych wyników, dopóki nie było żadnych różnic. Może to również wymagać podążania różnymi ścieżkami wyszukiwania, reprezentowanymi przez różne zastosowania redukcji. Rysunek 1a, autorstwa Newella i Simona, przedstawia dwanaście reguł transformacji, środkową kolumnę, do rozwiązywania problemów logicznych. Prawa kolumna zawiera wytyczne dotyczące tego, kiedy mają zostać użyte przekształcenia.

R 1.	$A \cdot B \rightarrow B \cdot A$ $A \vee B \rightarrow B \vee A$	Applies to main expression only.
R 2.	$A \supset B \rightarrow \sim B \supset \sim A$	Applies to main expression only.
R 3.	$A \cdot A \leftrightarrow A$ $A \vee A \leftrightarrow A$	A and B are two main expressions.
R 4.	$A \cdot (B \cdot C) \leftrightarrow (A \cdot B) \cdot C$ $A \vee (B \vee C) \leftrightarrow (A \vee B) \vee C$	A and $A \supset B$ are two main expressions.
R 5.	$A \vee B \leftrightarrow \sim(\sim A \cdot \sim B)$	$A \supset B$ and $B \supset C$ are two main expressions.
R 6.	$A \supset B \leftrightarrow \sim A \vee B$	
R 7.	$A \cdot (B \vee C) \leftrightarrow (A \cdot B) \vee (A \cdot C)$ $A \vee (B \cdot C) \leftrightarrow (A \vee B) \cdot (A \vee C)$	
R 8.	$A \cdot B \rightarrow A$ $A \cdot B \rightarrow B$	Applies to main expression only.
R 9.	$A \rightarrow A \vee X$	Applies to main expression only.
R 10.	$\left.\begin{matrix} A \\ B \end{matrix}\right\} \rightarrow A \cdot B$	A and B are two main expressions.
R 11.	$\left.\begin{matrix} A \\ A \supset B \end{matrix}\right\} \rightarrow B$	A and $A \supset B$ are two main expressions.
R 12.	$\left.\begin{matrix} A \supset B \\ B \supset C \end{matrix}\right\} \rightarrow A \supset C$	$A \supset B$ and $B \supset C$ are two main expressions.

Rysunek 1b przedstawia dowód od Newella i Simona , wygenerowany przez człowieka.

1.	$(R \supset \sim P) \cdot (\sim R \supset Q)$	$\sim (\sim Q \cdot P)$
2.	$(\sim R \vee \sim P) \cdot (R \vee Q)$	Rule *6* applied to left and right of 1.
3.	$(\sim R \vee \sim P) \cdot (\sim R \supset Q)$	Rule *6* applied to left of 1.
4.	$R \supset \sim P$	Rule *8* applied to 1.
5.	$\sim R \vee \sim P$	Rule *6* applied to 4.
6.	$\sim R \supset Q$	Rule *8* applied to 1.
7.	$R \vee Q$	Rule *6* applied to 6.
8.	$(\sim R \vee \sim P) \cdot (R \vee Q)$	Rule *10* applied to 5. and 7.
9.	$P \supset \sim R$	Rule *2* applied to 4.
10.	$\sim Q \supset R$	Rule *2* applied to 6.
11.	$P \supset Q$	Rule *12* applied to 6. and 9.
12.	$\sim P \vee Q$	Rule *6* applied to 11.
13.	$\sim (P \cdot \sim Q)$	Rule *5* applied to 12.
14.	$\sim (\sim Q \cdot P)$	Rule *1* applied to 13. QED.

Przed dowodem, podmiotowi dostępne są reguły transformacji z rysunku 1a, który bez doświadczenia w logice formalnej proszony jest o zmianę wyrażenia (R ⊃ ¬ P) • (¬ R ⊃ Q) na ¬ (¬ Q • P). W zapisie rozdziału 2 ~ to ¬, • to ∧, a ⊃ to →. → lub ↔ na rysunku 14.1a oznacza legalną wymianę. Skrajna prawa kolumna na rysunku 14.1b wskazuje regułę z rysunku 14.1a, która jest stosowana na każdym etapie dowodu. Newell i Simon nazwali strategie rozwiązywania problemów, polegające na zmniejszaniu różnic między podmiotami ludzkimi, oraz ogólny proces stosowania przekształceń odpowiednich do zmniejszania specyficznych różnic problemowych analizą środków i celów. Algorytmem stosowania analizy średnich i celów za pomocą redukcji różnic jest General Problem Solver (GPS). Rysunek 2 przedstawia schemat sterowania oraz tabelę połączeń dla GPS.



Celem jest przekształcenie wyrażenia A w wyrażenie B. Pierwszym krokiem jest zlokalizowanie różnicy D między A i B. Cel podrzędny, redukcja D, jest zidentyfikowany w drugiej ramce pierwszej linii; trzecia ramka wskazuje, że redukcja różnicy jest rekurencyjna. „Redukcja" to druga linia, w której dla różnicy D. zidentyfikowany jest operator Q. W rzeczywistości lista operatorów jest identyfikowana z tabeli połączeń. Ta lista zawiera uporządkowane alternatywy dla zmniejszenia różnicy, gdyby wybrany operator nie był akceptowalny, na przykład przez nieprzekazanie testu wykonalności. W trzecim wierszu na rysunku 2 stosuje się operator i zmniejsza się D. Model rozwiązywania problemów oparty na GPS wymaga dwóch elementów. Pierwsza to opisana właśnie ogólna procedura porównywania dwóch opisów stanów i zmniejszania ich różnic. Drugim składnikiem GPS jest tablica połączeń, podająca powiązania między różnicami problemowymi a konkretnymi przekształceniami, które je redukują, stosownie do obszaru zastosowania. Rysunek 2 przedstawia różnice i ich redukcje (dwanaście przekształceń z rysunku 1a) dla wyrażeń rachunku zdań. Można by zbudować inne tabele połączeń w celu zmniejszenia różnic w formach algebraicznych lub do zadań takich jak Wieże Hanoi lub do bardziej złożonych gier, takich jak szachy. Z powodu tej modułowości tabeli różnic, tj. Tabele są zmieniane dla różnych zastosowań, rozwiązanie problemu nazwano generałem. Szereg różnych obszarów zastosowań GPS to opisane przez Ernsta i Newella. Rzeczywista struktura redukcji różnic w konkretnej domenie problemowej pomaga uporządkować poszukiwania tej domeny. Kolejność heurystyczna lub priorytetowa dla redukcji różnych klas różnic jest niejawna w kolejności przekształceń w tabeli redukcji różnic. Taka kolejność priorytetów może przedłożyć bardziej ogólnie stosowane transformacje przed wyspecjalizowanymi lub zlecić ekspertowi dziedzinowemu które uznają za najbardziej odpowiednie. Szereg kierunków badawczych wyewoluował z pracy w General Problem Solver. Jednym z nich jest wykorzystanie technik sztucznej inteligencji do analizy ludzkich zachowań związanych z rozwiązywaniem problemów. W szczególności system produkcyjny zastąpił metody środków i celów GPS jako preferowaną formę modelowania przetwarzania informacji przez ludzi.

Reguły produkcji w nowoczesnych systemach ekspertowych opartych na regułach zastąpiły określone wpisy w tabeli różnic GPS. W kolejnej interesującej ewolucji GPS, sama tabela różnic ewoluowała w dalszy sposób, stając się stołem operatora do planowania, takiego jak STRIPS i ABSTRIPS. Planowanie jest ważne w rozwiązywaniu problemów z robotami. Aby wykonać zadanie, takie jak przejście do następnego pomieszczenia i przyniesienie przedmiotu, komputer musi opracować plan. Ten plan koordynuje działania robota: odłóż wszystko, co teraz trzyma, przejdź do drzwi obecnego pokoju, przejdź przez drzwi, znajdź wymagane pomieszczenie, przejdź przez drzwi, przejdź do obiektu i tak dalej . Tworzenie planu dla STRIPS, Stanford Research Institute Problem Solver używa tabeli operatora, podobnie jak tabela różnic GPS. Każdy operator (prymitywne działanie robota) w tej tabeli ma dołączony zestaw warunków wstępnych, które są podobne do testów wykonalności przedstawionych na rysunku 2. Tabela operatorów zawiera również listy dodawania i usuwania, które służą do aktualizacji modelu „świata" po zastosowaniu samego operatora. W sekcji 7.4 przedstawiliśmy planer podobny do STRIPS. a następnie zbuduj go w Prologu w pomocniczych materiałach programowych. Podsumowując, pierwsze modele automatycznego rozumowania w sztucznej inteligencji można znaleźć w Logic Theorist and General Problem Solver opracowanym w Carnegie Institute. Programy te oferowały już wszystkie warunki wstępne do rozwiązywania problemów metodą słabych metod: jednolity środek reprezentacji, zestaw prawidłowych reguł wnioskowania oraz zestaw metod lub strategii stosowania tych reguł. Te same komponenty składają się na procedury sprawdzające rozdzielczość, nowoczesną i potężniejszą podstawę automatycznego rozumowania.

14.2 Dowodzenie twierdzenia o rozdzielczości

14.2.1 Wprowadzenie

Rozdzielczość to technika dowodzenia twierdzeń w rachunku zdań lub predykatów, która była częścią badań nad rozwiązywaniem problemów AI od połowy lat sześćdziesiątych. Rozdzielczość to rozsądna reguła wnioskowania, która stosowana do obalenia jest również kompletna. W ważnym praktycznym zastosowaniu dowodzenie twierdzenia o rozdzielczości, w szczególności system obalania rozdzielczości, umożliwiło obecną generację interpretatorów Prologu. Zasada rozdzielczości, wprowadzona w ważnym artykule Robinsona , opisuje sposób znajdowania sprzeczności w bazie danych klauzul przy minimalnym użyciu substytucji. Odrzucenie rezolucji udowadnia twierdzenie, negując twierdzenie, które ma być udowodnione, i dodając ten zanegowany cel do zbioru aksjomatów, które są znane (zostały przyjęte) jako prawdziwe. Następnie wykorzystuje regułę rozstrzygania wnioskowania, aby wykazać, że prowadzi to do sprzeczności. Gdy twierdzenie twierdzenia pokazuje, że zanegowany cel jest niezgodny z danym zbiorem aksjomatów, wynika z tego, że pierwotny cel musi być spójny. To potwierdza twierdzenie. Dowody obalenia rezolucji obejmują następujące kroki:

1. Umieść przesłanki lub aksjomaty w formie klauzuli.

2. Dodaj negację tego, co ma być udowodnione, w formie klauzuli, do zbioru aksjomatów.

3. Rozwiąż te klauzule razem, tworząc nowe klauzule, które logicznie z nich wynikają.

4. Stwórz sprzeczność, generując pustą klauzulę.

5. Podstawienia użyte do wytworzenia klauzuli pustej to takie, w których odwrotność zanegowanego celu (tego, co pierwotnie miało być udowodnione) jest prawdą.

Rozdzielczość jest zasadą wnioskowania dźwiękowego w rozumieniu Części 2. Jednak nie jest ona kompletna. Rezolucja to obalenie kompletne; to znaczy, klauzula pusta lub pusta może być zawsze generowana zawsze, gdy istnieje sprzeczność w zbiorze klauzul. Więcej powiemy na ten temat, gdy przedstawimy strategie obalenia w sekcji 14.2.4. Dowody obalenia rezolucji wymagają, aby aksjomaty

i negacja celu były umieszczone w normalnej formie zwanej formą klauzuli. Forma klauzuli reprezentuje logiczną bazę danych jako zbiór rozłączeń literałów. Dosłowny to wyrażenie atomowe lub negacja wyrażenia atomowego. Najpopularniejsza forma rozdzielczości, zwana rozdzielczością binarną, jest stosowana do dwóch klauzul, gdy jedna zawiera literał, a druga jego negację. Jeśli te literały zawierają zmienne, należy je ujednolicić, aby były równoważne. Następnie tworzona jest nowa klauzula składająca się z rozłączników wszystkich predykatów w dwóch klauzulach bez literału i jego negatywnej instancji, o których mówi się, że zostały „rozwiązane". Wynikowa klauzula otrzymuje podstawienie unifikacyjne, w ramach którego predykat i jego negacja są uznawane za „równoważne". Zanim wyjaśnimy to bardziej szczegółowo w kolejnych podrozdziałach, posłużymy się prostym przykładem. Rozdzielczość daje dowód podobny do tego, który został już wyprodukowany z modus ponens. Nie ma to na celu pokazania, że te reguły wnioskowania są równoważne (rozdzielczość jest w rzeczywistości bardziej ogólna niż modus ponens), ale ma dać czytelnikowi wyczucie procesu. Chcemy udowodnić, że „Fido umrze" stwierdzeniami, że „Fido to pies" i „wszystkie psy są zwierzętami" i „wszystkie zwierzęta umrą". Zmiana tych trzech przesłanek na predykaty a zastosowanie modus ponens daje:

1. Wszystkie psy są zwierzętami: ∀ (X) (pies (X) → zwierzę (X)).

2. Fido to pies: pies (fido).

3. Modus ponens i {fido / X} daje: zwierzę (fido).

4. Wszystkie zwierzęta umrą: ∀ (Y) (zwierzę (Y) → umrą (Y)).

5. Modus ponens i {fido / Y} daje: die (fido).

Równoważne rozumowanie przez rezolucję przekształca te predykaty w postać klauzuli:

FORMULARZ PROGNOZY: FORMA KLAUZUL

1. ∀ (X) (pies) (X) → zwierzę (X)): ¬ pies (X) ∨ zwierzę (X)

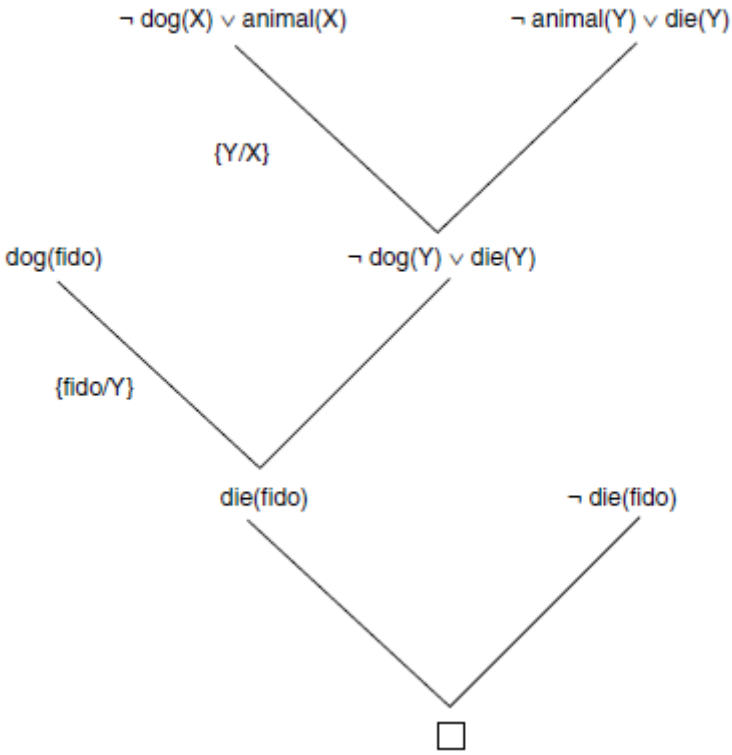
2. pies (fido): pies (fido)

3. ∀ (Y) (zwierzę (Y) → kość (Y)): ¬ zwierzę (Y) ∨ kość (Y)

Odrzuć wniosek, że Fido umrze:

4. ¬ die (fido): ¬ die (fido)

Rozwiązuj klauzule mające przeciwne literały, tworząc nowe klauzule według rozdzielczości, jak na rysunku 3. Ten proces jest często nazywany konfliktem.



Symbol □ na rysunku 3 wskazuje, że została utworzona pusta klauzula i znaleziono sprzeczność. □ symbolizuje zderzenie orzeczenia i jego zaprzeczenie: sytuację, w której w przestrzeni klauzul występują dwa wzajemnie sprzeczne zdania. Są one zderzane, aby uzyskać pustą klauzulę. Sekwencja podstawień (unifikacji) używana do równoważenia predykatów daje nam również wartość zmiennych, przy których cel jest prawdziwy. Na przykład, gdybyśmy zapytali, czy coś umrze, naszym zanegowanym celem byłoby ¬ (∃ (Z) die (Z)), a nie ¬ die (fido). Podstawienie {fido / Z} na rysunku 3 określiłoby, że fido jest przykładem zwierzęcia, które umrze. Kwestie ukryte w tym przykładzie wyjaśniono w dalszej części sekcji 14.2.

14.2.2 Opracowanie formularza klauzuli dla odrzucenia rezolucji

Procedura sprawdzająca rozwiązanie wymaga, aby wszystkie instrukcje w bazie danych opisujące sytuację zostały przekonwertowane na standardowy formularz zwany klauzulą. Jest to motywowane faktem, że rozdzielczość jest operatorem na parach rozłączników w celu wytworzenia nowych dysjunkcji. Forma, jaką przyjmuje baza danych, nazywana jest koniunkcją rozłączników. Jest to koniunkcja, ponieważ zakłada się, że wszystkie klauzule tworzące bazę danych są prawdziwe w tym samym czasie. Jest to dysjunkcja w tym sensie, że każda z poszczególnych klauzul jest wyrażona za pomocą dysjunkcji (lub ∨) jako łącznika. Zatem cała baza danych z rysunku 3 może być przedstawiona w postaci klauzuli jako:

(¬ pies (X) ∨ zwierzę (X)) ∧ (¬ zwierzę (Y) ∨ umrzeć (Y)) ∧ (pies (fido)).

Do tego wyrażenia dodajemy (przez koniunkcję) negację tego, co chcemy udowodnić, w tym przypadku ¬ die (fido). Zasadniczo baza danych jest zapisywana jako zbiór rozłączeń, a operatory ∧ są pomijane. Przedstawiamy teraz algorytm, składający się z sekwencji przekształceń, służący do redukcji dowolnego zestawu instrukcji rachunku predykatów do postaci klauzuli. Wykazano, że transformacje te mogą być użyte do zredukowania dowolnego zestawu wyrażeń rachunku predykatów do zbioru klauzul, które są

niespójne wtedy i tylko wtedy, gdy oryginalny zbiór wyrażeń jest niespójny. Forma klauzuli nie będzie ściśle równoważna z oryginalnym zestawem wyrażeń rachunku predykatów, ponieważ może dojść do utraty pewnych interpretacji. Dzieje się tak, ponieważ skolemizacja ogranicza możliwe substytucje dla zmiennych egzystencjalnie określonych ilościowo . Zachowa jednak niezaspokojenie. Oznacza to, że jeśli istniała sprzeczność (obalenie) w pierwotnym zestawie wyrażeń rachunku predykatów, istnieje sprzeczność w formie klauzuli. Transformacje nie poświęcają kompletności dla dowodów obalenia. Pokazujemy ten proces łącznej redukcji postaci normalnej na przykładzie i podajemy krótki opis racjonalizujący każdy krok. Nie mają one być dowodem równoważności tych przekształceń we wszystkich wyrażeniach rachunku predykatów. W poniższym wyrażeniu, zgodnie z konwencjami zawartymi w rozdziale 2, wielkie litery oznaczają zmienne (W, X, Y i Z); małe litery w środku alfabetu oznaczają stałe lub zmienne powiązane (l, m i n); a wczesne małe litery alfabetu wskazują nazwy predykatów (a, b, c, d i e). Aby poprawić czytelność wyrażeń, używamy dwóch typów nawiasów: () i [] oraz usuwamy zbędne nawiasy. Jako przykład, rozważ następujące wyrażenie, w którym X, Y i Z są zmiennymi, a l jest stałą:

(i) (∀X) ([a (X) ∧ b (X)] → [c (X, l) ∧ (∃Y) ((∃Z) [c (Y, Z)] → d (X, Y))]) ∨ (∀X) (e (X))

1. Najpierw eliminujemy → stosując równoważną formę udowodnioną w rozdziale 2: a → b ≡ ¬ a ∨ b. Ta transformacja redukuje wyrażenie w (i) powyżej:

(ii) (∀X) (¬ [a (X) ∧ b (X)] ∨ [c (X, l) ∧ (∃Y) ((∃Z) [¬ c (Y, Z)] ∨ d (X, Y))] ∨ (∀X) (e (X))

2. Następnie zmniejszamy zakres negacji. Można to osiągnąć za pomocą szeregu przekształceń opisanych w Części 2. Należą do nich:

¬ (¬ a) ≡ a

¬ (∃X) a (X) ≡ (∀X) ¬ a (X)

¬ (∀X) b (X) ≡ (∃X) ¬ b (X)

¬ (a ∧ b) ≡ ¬ a ∨ ¬ b

¬ (a ∨ b) ≡ ¬ a ∧ ¬ b

Korzystanie z czwartych równoważników (ii) staje się:

(iii) (∀X) ([¬ a (X) ∨ ¬ b (X)] ∨ [c (X, l) ∧ (∃Y) ((∃Z) [¬ c (Y, Z)] ∨ d (X, Y))]) ∨ (∀X) (e (X))

3. Następnie dokonujemy standaryzacji, zmieniając nazwy wszystkich zmiennych, tak aby zmienne powiązane różnymi kwantyfikatorami miały unikalne nazwy. Jak wskazano w rozdziale 2, ponieważ nazwy zmiennych są „manekinami" lub „znacznikami miejsca", konkretna nazwa wybrana dla zmiennej nie wpływa ani na wartość prawdziwości, ani na ogólność klauzuli. Transformacje użyte w tym kroku mają postać:

((∀X) a (X) ∨ (∀X) b (X)) ≡ (∀X) a (X) ∨ (∀Y) b (Y)

Ponieważ (iii) ma dwa wystąpienia zmiennej X, zmieniamy nazwę:

(iv) (∀X) ([¬ a (X) ∨ ¬ b (X)] ∨ [c (X, l) ∧ (∃Y) ((∃Z) [¬ c (Y, Z)] ∨ d (X, Y))]) ∨ (∀W) (e (W))

4. Przenieś wszystkie kwantyfikatory w lewo bez zmiany ich kolejności. Jest to możliwe, ponieważ krok 3 usunął możliwość jakiegokolwiek konfliktu między nazwami zmiennych. (iv) teraz staje się:

(v) (∀X) (∃Y) (∃Z) (∀W) ([¬ a (X) ∨ ¬ b (X)] ∨ [c (X, l) ∧ (¬ c (Y, Z)) ∨ d (X, Y))] ∨ e (W))

Po kroku 4 mówi się, że klauzula ma postać normalną przedrostka, ponieważ wszystkie kwantyfikatory znajdują się na początku jako przedrostek, a wyrażenie lub macierz następuje po.

5. W tym momencie wszystkie egzystencjalne kwantyfikatory są eliminowane przez proces zwany skolemizacją. Wyrażenie (v) ma egzystencjalny kwantyfikator dla Y. Gdy wyrażenie zawiera zmienną kwantyfikowaną egzystencjalnie, na przykład (∃Z) (foo (…, Z,…)), można wywnioskować, że istnieje przypisanie do Z w ramach które foo jest prawdziwe. Skolemizacja identyfikuje taką wartość. Skolemizacja niekoniecznie pokazuje, jak wytworzyć taką wartość; jest to tylko metoda nadania nazwy przypisaniu, które musi istnieć. Jeśli k reprezentuje to przypisanie, to mamy foo (…, k,…). Zatem: (∃X) (pies (X)) można zastąpić psem (fido) gdzie nazwa fido jest wybierana z dziedziny definicji X, aby reprezentować to indywidualne X. fido nazywane jest stałą skolema. Jeśli predykat ma więcej niż jeden argument, a zmienna kwantyfikowana egzystencjalnie mieści się w zakresie zmiennych kwantyfikowanych uniwersalnie, zmienna egzystencjalna musi być funkcją tych innych zmiennych. Znajduje to odzwierciedlenie w procesie skolemizacji:

(∀X) (∃Y) (matka (X, Y))

To wyrażenie wskazuje, że każda osoba ma matkę. Każda osoba jest X, a istniejąca matka będzie funkcją konkretnej wybranej osoby X. Zatem skolemizacja daje:

(∀X) matka (X, m (X))

co oznacza, że każdy X ma matkę (m tego X). W innym przykładzie:

(∀X) (∀Y) (∃Z) (∀W) (foo (X, Y, Z, W))

jest skolemizowany do:

(∀X) (∀Y) (∀W) (foo (X, Y, f (X, Y), W)).

Egzystencjalnie skwantyfikowane Y i Z mieszczą się w zakresie (na prawo od) powszechnie określanego ilościowo X, ale nie w zakresie W. W ten sposób każdy z nich zostanie zastąpiony funkcją skolema X. Zastąpienie Y funkcją skolema f (X) i Z z g (X), (v) staje się:

(vi) (∀X) (∀W) ([¬ a (X) ∨ ¬ b (X)] ∨ [c (X, l) ∧ (¬ c (f (X), g (X)) ∨ d (X, f (X)))]) ∨ e (W))

Po skolemizacji można wykonać krok 6, który po prostu usuwa prefiks.

6. Porzuć wszystkie uniwersalne kwantyfikacje. W tym momencie istnieją tylko zmienne kwantyfikowane uniwersalnie (krok 5) bez konfliktów zmiennych (krok 3). W ten sposób można odrzucić wszystkie kwantyfikatory, a każda zastosowana procedura dowodzenia zakłada, że wszystkie zmienne są określane ilościowo uniwersalnie. Formuła (vi) staje się teraz:

(vii) [¬ a (X) ∨ ¬ b (X)] ∨ [c (X, l) ∧ (¬ c (f (X), g (X)) ∨ d (X, f (X)))] ∨ e (W)

7. Następnie konwertujemy wyrażenie na formę koniunkcji rozłączników. Wymaga to użycia asocjacyjnych i dystrybucyjnych właściwości ∧ i ∨. Przypomnij sobie z rozdziału Części 2, że

a ∨ (b ∨ c) = (a ∨ b) ∨ c

a ∧ (b ∧ c) = (a ∧ b) ∧ c

co wskazuje, że ∧ lub ∨ mogą być grupowane w dowolny pożądany sposób. W razie potrzeby stosuje się również rozdzielczą właściwość rozdziału 2. Ponieważ a ∧ (b ∨ c) jest już w formie klauzuli, ∧ nie jest rozpowszechniane. Jednak ∨ musi być rozdzielone między ∧ przy użyciu:

a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c)

Ostateczna forma (vii) to:

(viii) [¬ a (X) ∨ ¬ b (X) ∨ c (X, l) ∨ e (W)] ∧

[¬ a (X) ∨ ¬ b (X) ∨ ¬ c (f (X), g (X)) ∨ d (X, f (X)) ∨ e (W)]

8. Teraz nazwij każdy spójnik oddzielną klauzulą. W powyższym przykładzie (viii) są dwie klauzule:

(ixa) ¬ a (X) ∨ ¬ b (X) ∨ c (X, l) ∨ e (W)

(ixb) ¬ a (X) ∨ ¬ b (X) ∨ ¬ c (f (X), g (X)) ∨ d (X, f (X)) ∨ e (W)

9. Ostatnim krokiem jest ponowna standaryzacja zmiennych. Wymaga to nadania zmiennej w każdej klauzuli wygenerowanej w kroku 8 różnych nazw. Ta procedura wynika z równoważności ustalonej w rozdziale 2, że

(∀X) (a (X) ∧ b (X)) ≡ (∀X) a (X) ∧ (∀Y) b (Y)

co wynika z natury nazw zmiennych jako posiadaczy miejsca. (ixa) i (ixb) stają się teraz, używając nowych nazw zmiennych U i V:

(xa) ¬ a (X) ∨ ¬ b (X) ∨ c (X, l) ∨ e (W)

(xb) ¬ a (U) ∨ ¬ b (U) ∨ ¬ c (f (U), g (U)) ∨ d (U, f (U)) ∨ e (V)

Znaczenie tej ostatecznej standaryzacji staje się widoczne dopiero wtedy, gdy przedstawimy etapy unifikacji rozwiązania. Znajdujemy najbardziej ogólne ujednolicenie, aby uczynić dwa predykaty w dwóch klauzulach równoważnymi, a następnie to podstawienie jest wykonywane dla wszystkich zmiennych o tej samej nazwie w każdej klauzuli. Tak więc, jeśli niektóre zmienne (niepotrzebnie) mają wspólne nazwy z innymi, to w procesie unifikacji można zmienić ich nazwy, co może skutkować utratą ogólności w rozwiązaniu. Ten dziewięciostopniowy proces służy do zmiany dowolnego zestawu wyrażeń rachunku predykatów na formę klauzuli. Właściwość kompletności obaleń rezolucji nie zostaje utracona. Następnie przedstawiamy procedurę rozwiązywania generowania dowodów z tych klauzul.

14.2.3 Procedura potwierdzania rozdzielczości plików binarnych

Procedura dowodu na odrzucenie rezolucji odpowiada na zapytanie lub wyprowadza nowy wynik, redukując zestaw klauzul do sprzeczności reprezentowanej przez klauzulę null (). Sprzeczność jest tworzona przez rozwiązywanie par klauzul z bazy danych. Jeśli rozstrzygnięcie nie prowadzi bezpośrednio do sprzeczności, to klauzula utworzona przez rezolucję, czyli rozwiązanie, jest dodawana do bazy danych klauzul i proces jest kontynuowany. Zanim pokażemy, jak działa proces rozwiązywania w rachunku predykatów, podajemy przykład z rachunku zdań lub rachunku bez zmiennych. Rozważmy dwie klauzule nadrzędne p1 i p2 z rachunku zdań:

p1: $a1 \vee a2 \vee \cdots \vee a_n$

p2: $b1 \vee b2 \vee \cdots \vee b_m$

mając dwa literały a_i i b_j, gdzie $1 < i \leq n$ i $1 \leq j \leq m$, takie, że $\neg a_i = b_j$. Rozdzielczość binarna daje klauzulę:

$a_1 \vee \cdots \vee a_{i-1} \vee a_{i+1} \vee \cdots \vee a_n \vee b_1 \vee \cdots \vee b_{j-1} \vee b_{j+1} \vee \cdots \vee b_m$. Powyższy zapis wskazuje, że resolvent składa się z dysjunkcji wszystkich literałów dwóch klauzul nadrzędnych, z wyjątkiem literałów a_i i b_j. Prosty argument może dać intuicję stojącą za zasadą rozdzielczości. Przypuśćmy ,e

a ∨ ¬ b i b ∨ c

są prawdziwymi stwierdzeniami. Zauważ, że jedno z b i ¬ b zawsze musi być prawdziwe, a jedno zawsze fałszywe (b ∨ ¬ b jest tautologią). Dlatego jeden z a ∨ c musi zawsze być prawdziwe. a ∨ c jest rozwiązaniem dwóch klauzul macierzystych a ∨ ¬ b i b ∨ c. Rozważmy teraz przykład z rachunku zdań, w którym chcemy udowodnić a z następujących aksjomatów (oczywiście l ← m ≡ m → l dla wszystkich zdań l i m):

a ← b ∧ c

b

c ← d ∧ e

e ∨ f

d ∧ ¬ f

Sprowadzamy pierwszy aksjomat do postaci klauzuli:

a ← b ∧ c

a ∨ ¬ (b ∧ c) przez l → m ≡ ¬ l ∨ m

a ∨ ¬ b ∨ ¬ c zgodnie z prawem de Morgana

Pozostałe aksjomaty są zredukowane i mamy następujące klauzule:

a ∨ ¬ b ∨ ¬ c

b

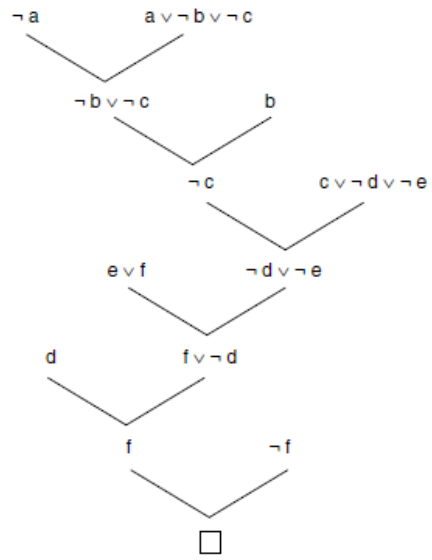
c ∨ ¬ d ∨ ¬ e

e ∨ f

re

¬ f

Dowód rozdzielczości można znaleźć na rysunku 4.



Po pierwsze, cel, który ma zostać udowodniony, a, jest negowany i dodawany do zestawu klauzul. Wyprowadzenie wskazuje, że baza danych klauzul jest niespójna. Aby użyć rozdzielczości binarnej w rachunku predykatów, gdzie każdy literał może zawierać zmienne, musi istnieć proces, w którym dwa literały o różnych nazwach zmiennych lub jeden o stałej wartości mogą być traktowane jako równoważne. Ujednolicenie zostało zdefiniowane w Części 2 jako proces określania spójnych i najbardziej ogólnych podstawień do wykonania dwa równoważne predykaty. Algorytm rozstrzygania rachunku predykatów jest bardzo podobny do algorytmu rachunku zdań, z tą różnicą, że:

1. Literał i jego negacja w klauzulach nadrzędnych dają rozwiązanie rozstrzygające tylko wtedy, gdy jednoczą się w wyniku podstawienia σ. σ jest następnie nakładane na resolvent przed dodaniem go do zestawu klauzul. Wymagamy, aby σ było najbardziej ogólnym unifer klauzul nadrzędnych.

2. Podstawienia unifikacyjne użyte do znalezienia sprzeczności oferują zmienne powiązania, w przypadku których oryginalne zapytanie jest prawdziwe. Proces ten, zwany ekstrakcją odpowiedzi, wyjaśnimy w sekcji 14.2.5.

Czasami dwa lub więcej literałów w jednej klauzuli ma ujednolicające podstawienie. W takim przypadku może nie istnieć odrzucenie zestawu klauzul zawierających tę klauzulę, nawet jeśli zestaw może być sprzeczny. Weźmy na przykład pod uwagę klauzule:

p (X) ∨ p (f (Y))

¬ p (W) ∨ ¬ p (f (Z))

Czytelnik powinien zauważyć, że przy prostym rozwiązaniu klauzule te można zredukować tylko do form równoważnych lub tautologicznych, ale nie do sprzeczności, to znaczy, że żadne zastąpienie nie może uczynić ich niespójnymi. Sytuacja ta może zostać rozwiązana poprzez faktoryzowanie takich klauzul. Jeśli podzbiór literałów w klauzuli ma najbardziej ogólny unifikator, klauzula jest zastępowana nową klauzulą, nazywaną współczynnikiem tej klauzuli. Czynnikiem jest oryginalna klauzula z najbardziej ogólną zastosowano ujednolicone podstawianie, a następnie usunięto zbędne literały. Na przykład dwa literały klauzuli p(X) ∨ p(f(Y)) ujednolicą się w wyniku podstawienia {f(Y) / X}. Dokonujemy podstawienia w obu literałach, aby otrzymać klauzulę p(f(Y)) ∨ p(f(Y)), a następnie zamieniamy tę klauzulę na jej współczynnik: p (f (Y)). Każdy system obalenia rezolucji, który obejmuje faktoring, jest

obaleniem zakończonym. Pomijając zmienne standaryzujące, sekcja 14.2.2 krok 3, można interpretować jako trywialne zastosowanie faktoringu. Faktoring może być również traktowany jako część procesu wnioskowania w hiperrozdzielczości opisanej w sekcji 14.4.2. Przedstawiamy teraz przykład obalenia rozdzielczości rachunku predykatów. Rozważ następującą historię „szczęśliwego ucznia":

Każdy, kto zda egzaminy z historii i wygra na loterii, jest szczęśliwy. Ale każdy, kto się uczy lub ma szczęście, może zdać wszystkie egzaminy. John nie uczył się, ale ma szczęście. Każdy, kto ma szczęście, wygrywa na loterii. Czy John jest szczęśliwy? Najpierw zmień zdania na formę orzeczenia:

Każdy, kto zda egzaminy z historii i wygra na loterii, jest szczęśliwy.

∀ X (pass (X, historia) ∧ wygrana (X, loteria) → happy (X))

Każdy, kto się uczy lub ma szczęście, może zdać wszystkie egzaminy.

∀ X ∀ Y (nauka (X) ∨ szczęście (X) → pass (X, Y))

John nie uczył się, ale ma szczęście.

¬ nauka (jan) ∧ szczęście (jan)

Każdy, kto ma szczęście, wygrywa na loterii.

∀ X (szczęście (X) → wygrana (X, loteria))

Te cztery instrukcje predykatów zostały teraz zamienione na formę klauzuli (sekcja 14.2.2):

1. ¬ pass (X, historia) ∨ ¬ wygrana (X, loteria) ∨ szczęśliwy (X)

2. ¬ nauka (Y) ∨ zaliczenie (Y, Z)

3. ¬ szczęśliwe (W) ∨ podanie (W, V)

4. ¬ badanie (john)

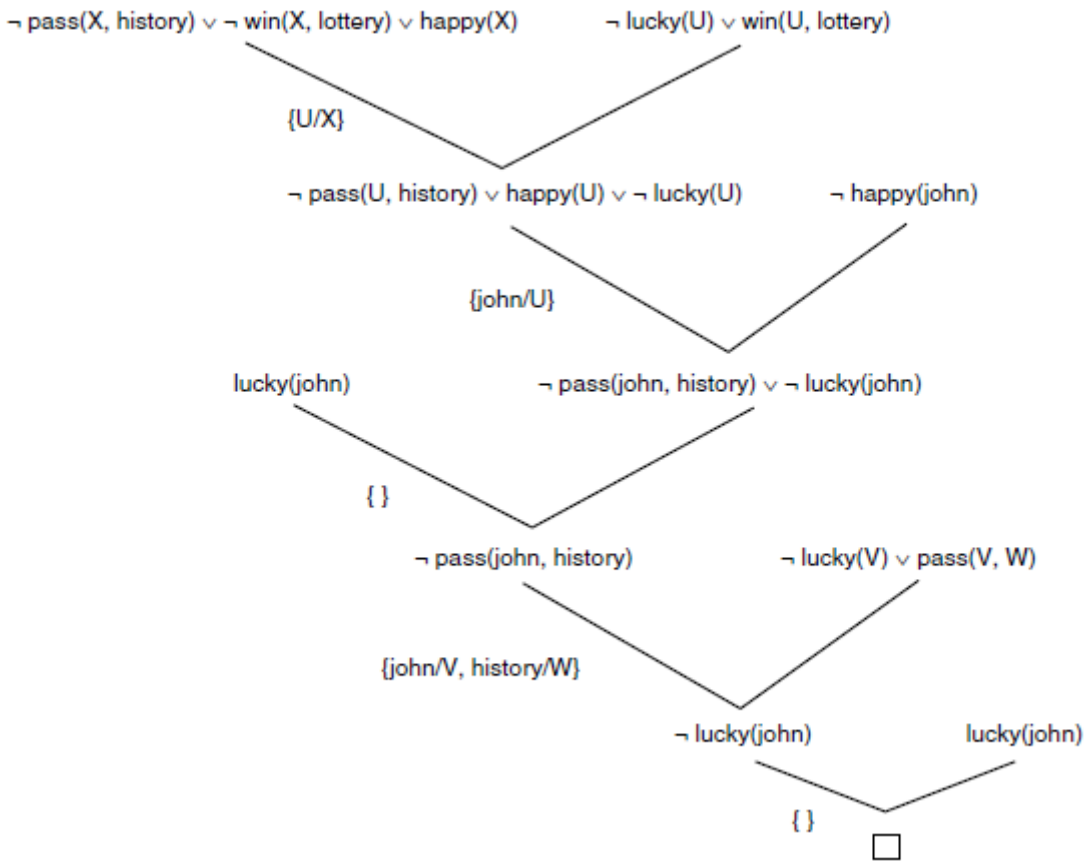
5. szczęście (jan)

6. ¬ szczęśliwy (U) ∨ wygrany (U, loteria)

Do tych klauzul wpisuje się w formie klauzuli negację konkluzji:

7. ¬ szczęśliwy (john)

Wykres obalenia rozdzielczości na rysunku 5 pokazuje wyprowadzenie sprzeczności, aw konsekwencji dowodzi, że Jan jest szczęśliwy.



Jako ostatni przykład w tym podrozdziale przedstawiamy problem „ekscytującego życia”; przypuśćmy: wszyscy ludzie, którzy nie są biedni i są mądrzy, są szczęśliwi. Ci, którzy czytają, są sprytni. John umie czytać i nie jest biedny. Szczęśliwi ludzie mają ekscytujące życie. Czy można znaleźć kogoś z ekscytującym życie?

Zakładamy ∀X (mądry (X) ≡ ¬ głupi (X)) i ∀Y (bogaty (Y) ≡ ¬ biedny (Y)) i otrzymujemy:

∀X (¬ słaby (X) ∧ inteligentny (X) → szczęśliwy (X))

∀Y (odczyt (Y) → smart (Y))

czytaj (jan) ∧ ¬ biedny (jan)

∀Z (szczęśliwy (Z) → ekscytujący (Z))

Zaprzeczeniem wniosku jest:

¬ ∃ W (ekscytujące (W))

Te wyrażenia rachunku predykatów dla problemu „ekscytującego życia” są przekształcane w następujące klauzule:

słaby (X) ∨ ¬ inteligentny (X) ∨ szczęśliwy (X)

¬ przeczytaj (Y) ∨ inteligentny (Y)

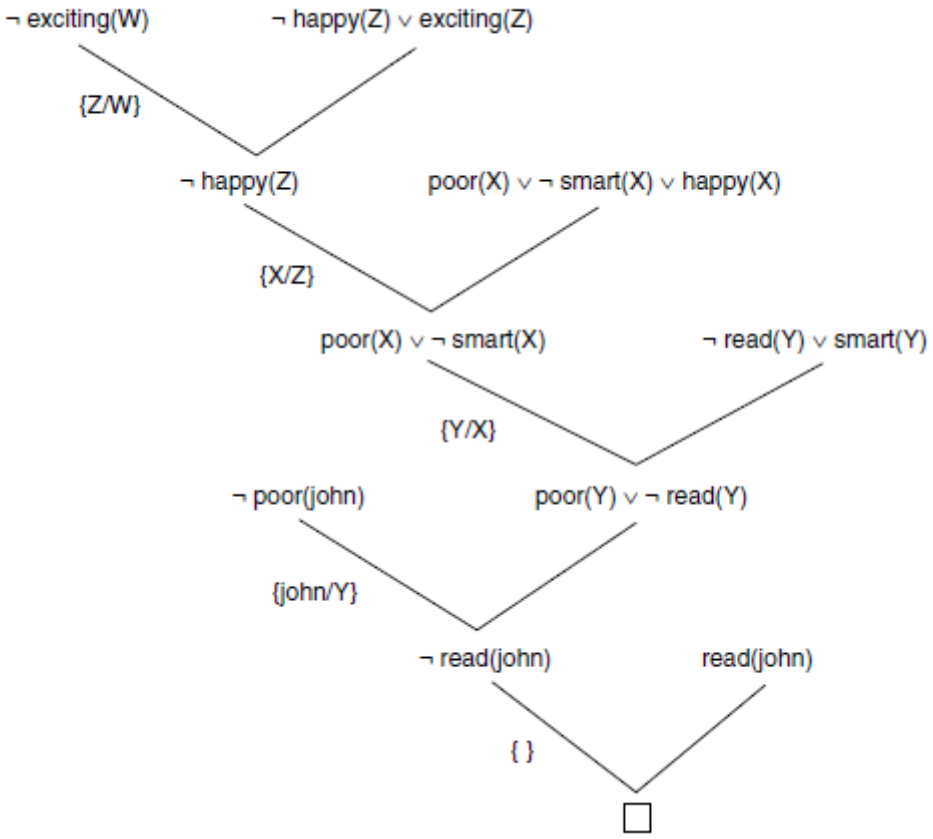
czytaj (jan)

¬ biedny (jan)

¬ szczęśliwy (Z) ∨ ekscytujący (Z)

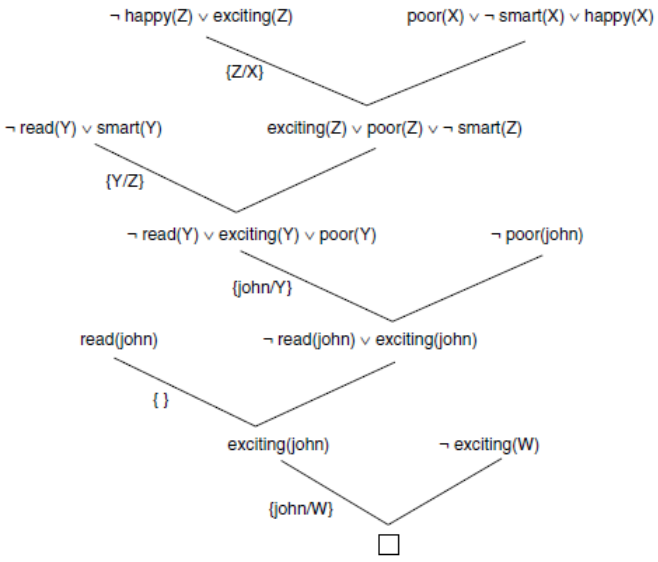
¬ ekscytujące (W)

Obalenie rozdzielczości dla tego przykładu znajduje się na rysunku 6.



14.2.4 Strategie i uproszczone techniki rozwiązywania problemów

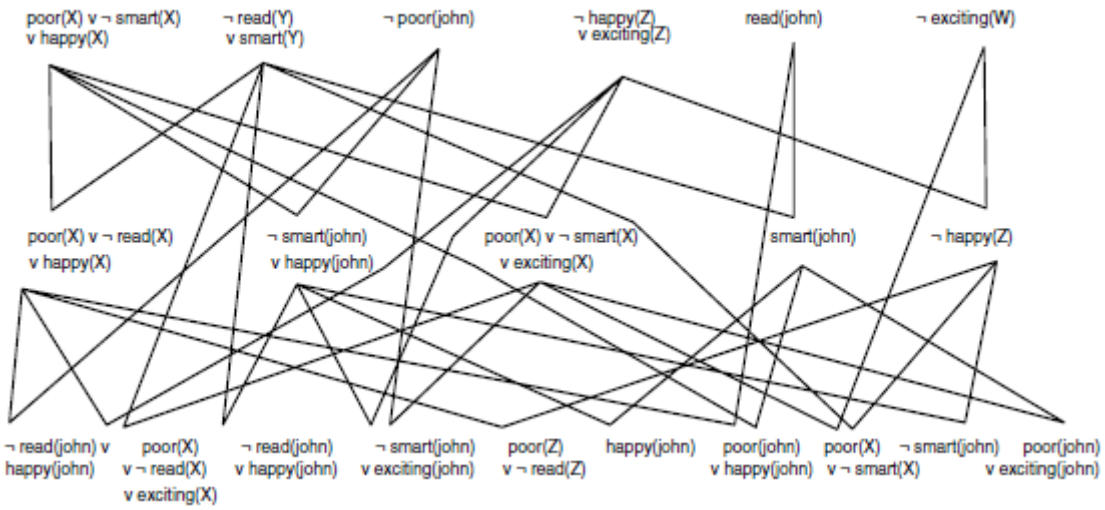
Inne drzewo dowodów w przestrzeni poszukiwań dla problemu z rysunku 6 pojawia się na rysunku 7.



Istnieją pewne podobieństwa w tych dowodach; na przykład obaj podjęli pięć kroków rozwiązywania problemów. Również asocjacyjne zastosowanie znalezionych podstawień unifikacyjnych ten John był przykładem osoby z „ekscytującym życiem” w obu dowodach. Jednak nawet te dwa podobieństwa nie muszą wystąpić. Podczas definiowania systemu sprawdzającego rozdzielczość nie sugerowano kolejności kombinacji klauzul. Jest to krytyczna kwestia: kiedy w przestrzeni klauzul znajduje się N klauzul, istnieje N^2 sposobów ich łączenia lub sprawdzania, czy można je łączyć tylko na pierwszym poziomie! Wynikowy zestaw klauzul z tego porównania jest również duży; jeśli nawet 20% z nich stworzy nowe klauzule, następna runda możliwych rozwiązań będzie zawierała jeszcze więcej kombinacji niż pierwsza. W przypadku dużego problemu ten gwałtowny wzrost szybko wymknie się spod kontroli. Z tego powodu heurystyki wyszukiwania są bardzo ważne w procedurach sprawdzania rozwiązań, ponieważ są one we wszystkich słabych metodach rozwiązywania problemów. Podobnie jak w przypadku heurystyk, które rozważaliśmy w rozdziale 4, nie ma nauki, która mogłaby określić najlepszą strategię dla konkretnego problemu. Niemniej jednak niektóre strategie ogólne mogą odnosić się do kombinatoryki wykładniczej. Zanim opiszemy nasze strategie, musimy poczynić kilka wyjaśnień. Po pierwsze, w oparciu o definicję niespełnienia wyrażenia w rozdziale 2, zestaw klauzul jest niezadowalający, jeśli nie istnieje żadna interpretacja, która określa zbiór jako spełnialny. Po drugie, reguła wnioskowania jest zaprzeczeniem kompletnym, jeśli mając niezadowalający zestaw klauzul, niezadowalalność można ustalić za pomocą samej tej reguły wnioskowania. Rozwiązanie z faktoringiem ma te własności . Wreszcie strategia jest kompletna, jeśli poprzez jej zastosowanie z zasadą refutacji-całkowitego wnioskowania możemy zagwarantować znalezienie obalenia za każdym razem, gdy zestaw klauzul jest niezadowalający. Strategia wszerz to przykład pełnej strategii.

Strategia wszerz

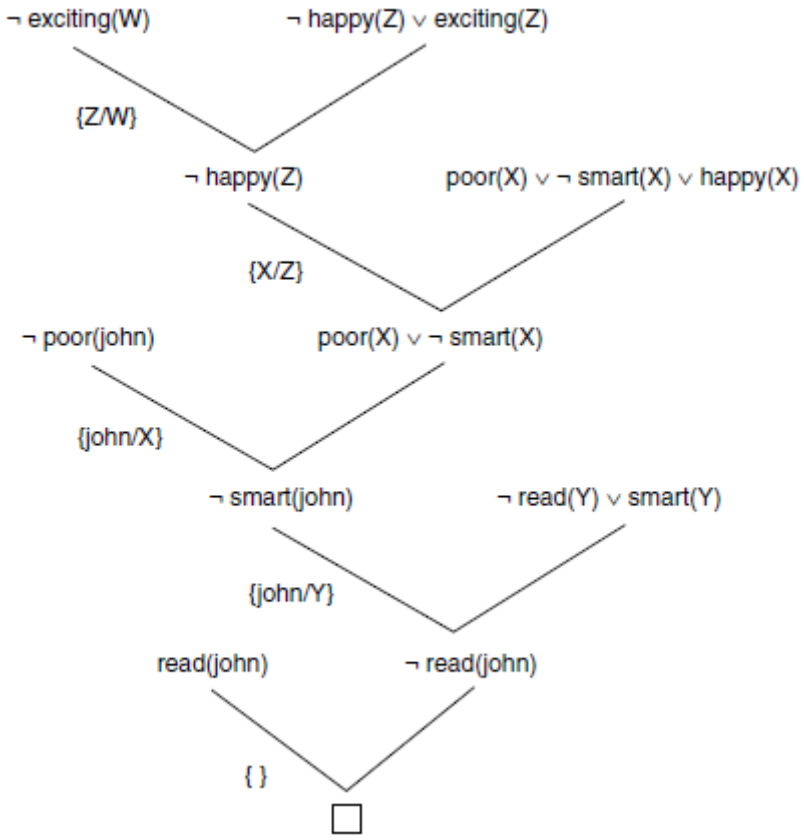
Analiza złożoności opisanego właśnie wyczerpującego porównania klauzul została oparta na przeszukiwaniu wszerz. Każda klauzula w przestrzeni klauzul jest porównywana pod kątem rozwiązania z każdą klauzulą w przestrzeni klauzul w pierwszej rundzie. Klauzule na drugim poziomie przestrzeni wyszukiwania są generowane przez rozstrzygnięcie klauzul utworzonych na pierwszym poziomie ze wszystkimi oryginalnymi klauzulami. Generujemy klauzule na n-tym poziomie, porównując wszystkie klauzule na poziomie n - 1 z elementami oryginalnego zestawu klauzul i wszystkimi poprzednio utworzonymi klauzulami. Ta strategia może szybko wymknąć się spod kontroli w przypadku dużych problemów. Ma jednak interesującą właściwość. Jak każde przeszukiwanie wszerz, gwarantuje znalezienie najkrótszej ścieżki rozwiązania, ponieważ generuje każdy stan wyszukiwania dla każdego poziomu przed przejściem głębiej. Jest to również kompletna strategia, ponieważ jeśli jest kontynuowana wystarczająco długo, gwarantuje się znalezienie odparcia, jeśli takie istnieje. Tak więc, gdy problem jest niewielki, tak jak te, które przedstawiliśmy jako przykłady, strategia „najpierw szerokość” może być dobra. Rysunek 8 stosuje strategię „najpierw wszerz” do problemu „ekscytującego życia”.



Zestaw strategii wsparcia Doskonała strategia dla dużych przestrzeni klauzul nazywana jest zestawem wsparcia. Dla zestawu klauzul wejściowych, S, możemy określić podzbiór T z S, nazywany zbiorem podpór. Strategia wymaga, aby jeden z resolventów w każdej rezolucji miał przodka w zestawie wsparcia. Można udowodnić, że jeśli S jest niezadowalające, a S - T - zadowalające, to zestaw strategii wsparcia jest kompletny. Jeśli oryginalny zestaw klauzul jest spójny, każdy zestaw obsługi, który obejmuje negację oryginalnego zapytania, spełnia te wymagania. Ta strategia opiera się na przekonaniu, że zaprzeczenie tego, co chcemy udowodnić, będzie odpowiedzialne za wywołanie sprzeczności w przestrzeni klauzul. Zbiór rozwiązań sił wspierających między klauzulami, z których przynajmniej jedna jest klauzulą zanegowanego celu lub klauzulą utworzoną przez rezolucje w sprawie zanegowanego celu. Rysunek 6 jest przykładem zestawu strategii wsparcia zastosowanych do ekscytującego problemu życiowego. Ponieważ zbiór obalenia poparcia istnieje zawsze, gdy istnieje jakiekolwiek obalenie, zbiór wsparcia może być podstawą pełnej strategii. Jednym ze sposobów jest przeprowadzenie przeszukiwania wszerz dla wszystkich możliwych zestawów obaleń wsparcia. Będzie to oczywiście znacznie bardziej wydajne niż przeszukiwanie wszerz wszystkich klauzul. Trzeba tylko mieć pewność, że wszystkie rozpatrywane są rezolwenty klauzuli zanegowanego celu wraz ze wszystkimi ich potomkami.

Strategia preferencji jednostek

Zwróć uwagę, że w dotychczasowych przykładach rozdzielczości wyprowadzenie sprzeczności jest wskazywane przez klauzulę bez literałów. Tak więc za każdym razem, gdy tworzymy klauzulę wynikową, która ma mniej literałów niż klauzule, które mają ją utworzyć, jesteśmy bliżej stworzenia klauzuli bez literałów. W szczególności rozwiązywanie za pomocą klauzuli jednego literału, zwanej klauzulą jednostkową, zagwarantuje, że rozstrzygający jest mniejszy niż największa klauzula nadrzędna. Strategia preferencji jednostek wykorzystuje jednostki do rozpatrzenia, gdy tylko są dostępne. Rysunek 9 przedstawia strategię preferencji jednostkowych dotyczącą ekscytującego problemu życiowego.



Strategia preferencji jednostkowych wraz z zestawem wsparcia może stworzyć bardziej wydajną, kompletną strategię. Rozdzielczość jednostek jest powiązaną strategią, która wymaga, aby jeden z rozwiązań zawsze był klauzulą jednostkową. Jest to silniejszy wymóg niż strategia preferencji jednostki. Możemy wykazać, że rozdzielczość jednostkowa nie jest kompletna, korzystając z tego samego przykładu, który pokazuje niekompletność liniowej formy wejściowej. Strategia liniowego formularza wejściowego Strategia liniowego formularza wejściowego jest bezpośrednim wykorzystaniem zanegowanego celu i oryginalnych aksjomatów: weź zanegowany cel i rozwiąż go jednym z aksjomatów, aby uzyskać nową klauzulę. Wynik ten jest następnie rozwiązywany za pomocą jednego z aksjomatów, aby uzyskać kolejną nową klauzulę, która jest ponownie rozwiązywana za pomocą jednego z aksjomatów. Ten proces trwa do momentu utworzenia pustej klauzuli. Na każdym etapie rozwiązujemy ostatnio uzyskaną klauzulę za pomocą aksjomatu wyprowadzonego z pierwotnego stwierdzenia problemu. Nigdy nie używamy poprzednio wyprowadzonej klauzuli ani nie rozwiązujemy razem dwóch aksjomatów. Liniowy formularz wejściowy nie jest pełną strategią, co można zobaczyć, stosując go do następującego zestawu czterech klauzul (które są oczywiście niezadowalające). Niezależnie od tego, która klauzula jest traktowana jako zaprzeczenie celu, liniowa strategia wejściowa nie może powodować sprzeczności:

¬ a V ¬ b

a V ¬ b

¬ a V b

a V b

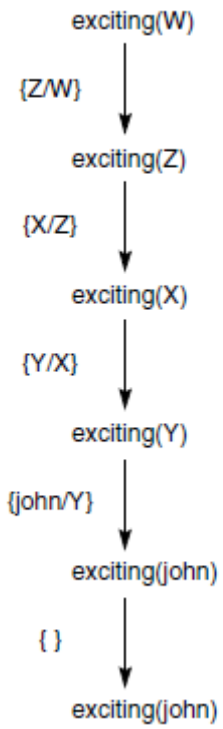
Inne strategie i techniki upraszczania Nie próbowaliśmy przedstawić wyczerpującego zestawu strategii ani nawet najbardziej wyrafinowanych technik dowodzenia twierdzeń za pomocą wnioskowania o rozdzielczości. Naszym celem jest raczej przedstawienie podstawowych narzędzi dla tego obszaru badawczego i opisanie, w jaki sposób można je wykorzystać w rozwiązywaniu problemów. Jak wspomniano wcześniej, procedura sprawdzania rozdzielczości jest kolejną słabą metodą rozwiązywania problemów. W tym sensie rozdzielczość może służyć jako mechanizm wnioskowania dla rachunku predykatów, ale silnik, który wymaga wielu analiz i starannego zastosowania strategii, aby osiągnąć sukces. W przypadku wystarczająco dużego, aby być interesującym, przypadkowe zderzanie się wyrażeń z rozwiązaniem jest równie beznadziejne, jak uderzanie losowych klawiszy terminala i nadzieja na uzyskanie dobrej jakości papieru. Przykłady użyte w tym rozdziale są banalnie małe i zawierają wszystkie klauzule niezbędne (i tylko te niezbędne) do ich rozwiązania. Rzadko dotyczy to interesujących problemów. Podaliśmy kilka prostych strategii zwalczania tych złożoności kombinatorycznych, a zakończymy ten podrozdział, opisując kilka ważniejszych rozważań przy projektowaniu rozwiązania problemu opartego na rozwiązaniach. Później pokażemy (w sekcji 14.3), w jaki sposób system obalania rozdzielczości, z interesującą kombinacją strategii wyszukiwania, zapewnia „semantykę" dla programowania logicznego, zwłaszcza do projektowania interpreterów Prolog. Połączenie strategii może być dość skuteczne w kontrolowaniu wyszukiwania - na przykład użycie zestawu wsparcia i preferencji jednostek. Heurystyki wyszukiwania mogą być również wbudowane w projekt reguł (tworząc porządek literałów od lewej do prawej do rozwiązywania). Ta kolejność może być najbardziej skuteczna przy przycinaniu przestrzeni wyszukiwania. To niejawne użycie strategii jest ważne w programowaniu Prologu (sekcja 14.3). Ogólność wniosków może być kryterium przy projektowaniu strategii rozwiązania. Z jednej strony może być ważne, aby rozwiązania pośrednie były jak najbardziej ogólne, ponieważ pozwala to na ich swobodne wykorzystanie w rozdzielczości. Dlatego wprowadzenie jakiejkolwiek rezolucji z klauzulami, które wymagają specjalizacji przez wiążące zmienne, takie jak {jan / X}, powinno być odkładane tak długo, jak to możliwe. Jeśli z drugiej strony rozwiązanie wymaga określonych zmiennych wiązań, na przykład w analizie, czy Jan ma infekcję gronkowcem, podstawienia {jan / osoba} i {gronkowiec / zakażenie} mogą ograniczyć przestrzeń poszukiwań i zwiększyć prawdopodobieństwo i szybkość znalezienia rozwiązania. Ważną kwestią przy wyborze strategii jest pojęcie kompletności. W niektórych aplikacjach może być bardzo ważne, aby wiedzieć, że zostanie znalezione rozwiązanie (jeśli takie istnieje).

Można to zagwarantować, stosując tylko kompletne strategie. Możemy również zwiększyć wydajność, przyspieszając proces dopasowywania. Możemy wyeliminować niepotrzebne (i kosztowne) ujednolicenia między klauzulami, które prawdopodobnie nie mogą generować nowych rozwiązań, poprzez indeksowanie każdej klauzuli zawartymi w niej literałami i określeniem, czy są one dodatnie czy ujemne. To pozwala nam bezpośrednio znaleźć potencjalne rozwiązania dla dowolnej klauzuli. Powinniśmy również wyeliminować niektóre klauzule, gdy tylko zostaną utworzone. Po pierwsze, żadna klauzula tautologiczna nigdy nie musi być rozważana; nigdy nie można ich sfałszować, więc nie przydają się one w próbach rozwiązania. Innym typem klauzuli, która nie dostarcza żadnych nowych informacji, jest klauzula, która może zostać podciągnięta, to znaczy, gdy nowa klauzula ma bardziej ogólną instancję już w przestrzeni klauzuli. Na przykład, jeśli p (john) jest wydedukowane dla przestrzeni, która już zawiera ∀ X (p (X)), to p (john) może zostać usunięta bez straty; w rzeczywistości jest to oszczędność, ponieważ w przestrzeni klauzul jest mniej klauzul. Podobnie p (X) zawiera klauzulę p (X) ∨ q (X). Mniej ogólne informacje nie dodają niczego do bardziej ogólnych informacji, gdy oba znajdują się w przestrzeni klauzuli. Wreszcie przywiązanie proceduralne ocenia lub w inny sposób przetwarza bez dalszego przeszukiwania żadnej klauzuli, która może dostarczyć nowych informacji. Wykonuje arytmetykę, dokonuje porównań między atomami lub klauzulami lub „uruchamia" inną deterministyczną procedurę, która może dodać konkretne informacje do rozwiązania problemu lub w

jakikolwiek sposób ograniczyć proces rozwiązywania. Na przykład, możemy użyć procedury do obliczenia powiązania dla zmiennej, gdy istnieje wystarczająca ilość informacji, aby to zrobić. To zmienne powiązanie ogranicza następnie możliwe rozdzielczości i przycina przestrzeń wyszukiwania. Następnie pokazujemy, jak można wyciągnąć odpowiedzi z procesu obalania rezolucji.

14.2.5 Wyciąganie odpowiedzi z obaleń rezolucji

Przypadki, w których hipoteza jest prawdziwa, obejmują zbiór podstawień, za pomocą których znajduje się obalenie. Dlatego zachowanie informacji o podstawieniach zjednoczeniowych dokonanych w obaleniu rezolucji daje informacje o poprawnej odpowiedzi. W tym podrozdziale podajemy trzy przykłady tego i wprowadzamy księgową metodę wydobywania odpowiedzi z obalenia rezolucji. Metoda rejestracji odpowiedzi jest prosta: zachowaj pierwotny wniosek, który miał zostać udowodniony, i wprowadź do niego każdą unifikację, która jest poczyniona w procesie rozstrzygania sporów. Zatem pierwotnym wnioskiem jest „księgowy" wszystkich unifikacji, które są dokonywane jako część obalenia. W obliczeniowych poszukiwaniach obalenia rozdzielczości może to wymagać dodatkowych wskazówek, na przykład gdy istnieje więcej niż jeden możliwy wybór w poszukiwaniu obalenia. Aby utworzyć alternatywne ścieżki rozwiązania, konieczne może być zastosowanie mechanizmu kontrolnego, takiego jak cofanie. Mimo to, przy odrobinie ostrożności, te dodatkowe informacje mogą zostać zachowane. Zobaczmy kilka przykładów tego procesu. Na rysunku 14.6, gdzie znaleziono dowód na istnienie osoby prowadzącej ekscytujące życie, dokonano unifikacji z rysunku 14.10. Jeśli zachowamy pierwotny cel i zastosujemy wszystkie podstawienia obalenia do tej klauzuli, znajdziemy odpowiedź, która to osoba ma ekscytujące życie. Rysunek 10 pokazuje, jak obalenie rezolucji nie tylko może pokazać, że „nikt nie prowadzi ekscytującego życia" jest fałszywe, ale także, w trakcie tej demonstracji, może stworzyć szczęśliwą osobę, Johna.



Jest to ogólny wynik, w którym ujednolicenia, które prowadzą do obalenia, są tymi samymi, które tworzą przypadki, w których pierwotne zapytanie jest prawdziwe. Drugi przykład to prosta historia: pies Fido idzie tam, gdzie idzie John, jego pan. John jest w bibliotece. Gdzie jest Fido?

Najpierw przedstawimy tę historię w wyrażeniach rachunku predykatów, a następnie zredukujemy te wyrażenia do postaci klauzul. Predykaty:

o (john, X) → o (fido, X)

w (jan, biblioteka)

Klauzule:

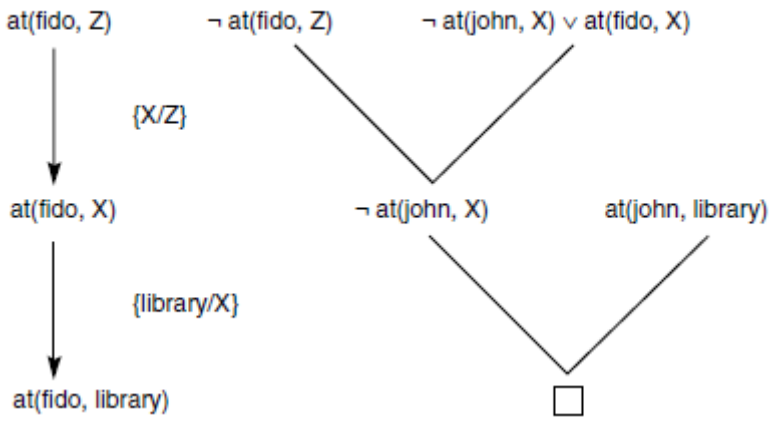
¬ w (john, Y) ∨ at (fido, Y)

w (jan, biblioteka)

Wniosek zanegowany:

¬ at (fido, Z), Fido nigdzie nie ma!

Rysunek 11 przedstawia proces wyodrębniania odpowiedzi.



Dosłowne śledzenie zjednoczeń jest pierwotnym pytaniem (gdzie jest Fido?):

w (fido, Z)

Po raz kolejny unifikacje, w ramach których znajduje się sprzeczność, mówią, jak prawdziwe jest pierwotne zapytanie: Fido jest w bibliotece. Ostatni przykład pokazuje, jak proces skolemizacji może dać instancję, w której można uzyskać odpowiedź. Rozważ następującą sytuację: Każdy ma rodzica. Rodzic rodzica jest dziadkiem. Mając osobę John, udowodnij, że John ma dziadka. Poniższe zdania przedstawiają fakty i relacje w powyższej sytuacji.

Po pierwsze, każdy ma rodzica:

(∀ X) (∃ Y) p (X, Y)

Rodzic rodzica jest dziadkiem.

(∀ X) (∀ Y) (∀ Z) p (X, Y) ∧ p (Y, Z) → gp (X, Z)

Celem jest znalezienie W takiego, że gp (jan, W) lub ∃ (W) (gp (jan, W)). Negacja

celem jest ¬ ∃ (W) (gp (john, W)) lub:

¬ gp (jan, W)

W procesie wprowadzania tego zbioru predykatów do postaci klauzuli dla obalenia rozdzielczości, kwantyfikator egzystencjalny w pierwszym predykacie (każdy ma rodzica) wymaga funkcji skolema. Ta funkcja skolema byłaby oczywistą funkcją: weź dane X i znajdź rodzica X. Nazwijmy to pa (X) od „znajdź rodzicielskiego przodka dla X". Dla Jana byłby to albo jego ojciec, albo jego matka. Forma klauzuli dla predykatów tego problemu to:

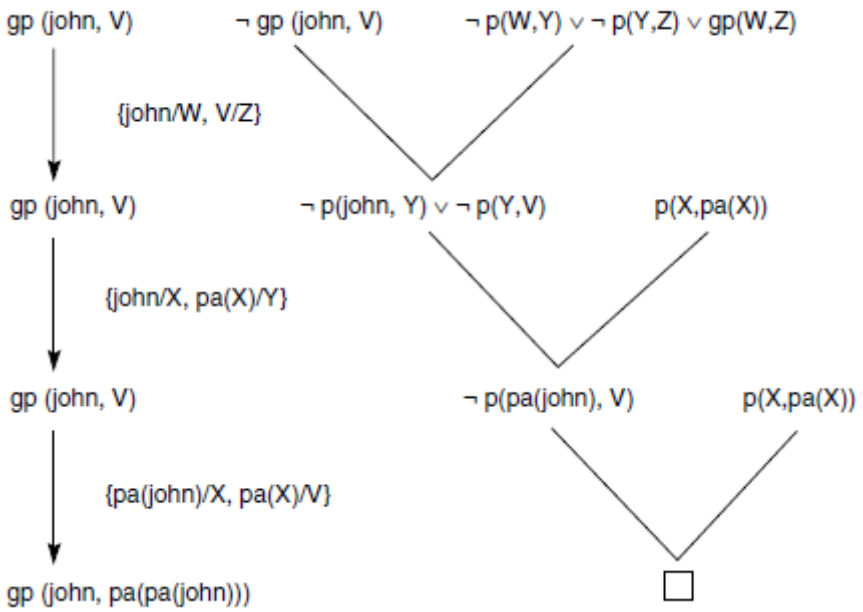
p (X, pa (X))

¬ p (W, Y) ∨ ¬ p (Y, Z) ∨ gp (W, Z)

¬ gp (jan, V)

Obalenie rozdzielczości i proces ekstrakcji odpowiedzi dla tego problemu przedstawiono na rysunku 12. Zauważ, że podstawienia unifikacji w odpowiedzi to gp (john, pa (pa (john)))

Odpowiedź na pytanie, czy Jan ma dziadka, brzmi: „znaleźć przodka rodzicielskiego przodka Jana". Skolemizowana funkcja pozwala nam obliczyć ten wynik. Opisany właśnie ogólny proces wyodrębniania odpowiedzi można zastosować we wszystkich obaleniach rozdzielczości, niezależnie od tego, czy są to ogólne ujednolicenia, jak na rysunkach 10 i 11, czy też z oceny funkcji skolema, jak na rysunku 12.



Proces daje odpowiedź. Metoda jest naprawdę dość prosta: sytuacja (zjednoczenia), w której występuje sprzeczność ,znaleziony to przypadek, w którym przeciwieństwo zaprzeczonego wniosku (pierwotne zapytanie problemu) jest prawdziwe. Chociaż w tym podrozdziale nie przedstawiono matematycznego wyjaśnienia, dlaczego jest to prawdą w każdym przypadku, pokazaliśmy kilka przykładów tego, jak działa ten proces; dalsze omówienie można znaleźć w literaturze.

14.3 Prolog i automatyczne rozumowanie

14.3.1 Wprowadzenie

Tylko poprzez zrozumienie implementacji języka komputerowego możemy właściwie pokierować jego użyciem, kontrolować jego skutki uboczne i mieć zaufanie do jego wyników. W tej sekcji opiszemy

semantykę Prologu i odniesiemy ją do zagadnień automatycznego rozumowania przedstawionych w poprzedniej sekcji. Poważna krytyka procedury sprawdzającej rozwiązanie, sekcja 14.2, dotyczy tego, że wymaga ona całkowicie jednorodnej bazy danych, aby przedstawić problem. Kiedy deskryptory rachunku predykatów są redukowane lub przekształcane do postaci klauzul, ważne informacje heurystyczne dotyczące rozwiązywania problemów są pomijane. Pominięte informacje nie są prawdą ani błędem jakiejkolwiek części problemu, ale raczej wskazówkami kontrolnymi lub opisami procedur, jak używać Informacji. Na przykład zanegowana klauzula celu w formacie rozdzielczości może mieć postać:

$$a \vee \neg b \vee c \vee \neg d$$

gdzie a, b, c i d są literałami. Mechanizm wnioskowania o rozwiązaniu stosuje strategię wyszukiwania w celu ustalenia pustej klauzuli. Wszystkie literały są otwarte na strategię, a zastosowana strategia zależy od wybranej strategii. Strategie używane do kierowania dowodzeniem twierdzeń o rozdzielczości to słabe heurystyki; nie zawierają głębokiej wiedzy na temat określonej dziedziny problemowej. Na przykład zanegowana klauzula celu w powyższym przykładzie rozwiązania może być transformacją instrukcji rachunku predykatów:

$$a \leftarrow b \wedge \neg c \wedge d$$

Można to rozumieć jako „aby zobaczyć, czy a jest prawdziwe, wyjdź i zobacz, czy b jest prawdziwe, a c jest fałszywe, a d jest prawdziwe”. Reguła została pomyślana jako procedura rozwiązywania problemów i implementuje heurystyczne informacje specyficzne dla tego zastosowania. Rzeczywiście, cel cząstkowy b może oferować najłatwiejszy sposób fałszowania całego predykatu, więc kolejność „spróbuj b, a następnie sprawdź, czy c jest fałszywe, a następnie test d” może zaoszczędzić dużo czasu i kosztów rozwiązywania problemów. Ukryta heurystyka mówi: „Najpierw przetestuj najłatwiejszy sposób zafałszowania problemu, a następnie, jeśli to się powiedzie, wygeneruj pozostałą (być może znacznie trudniejszą) część rozwiązania”. Eksperci opracowują procedury i relacje, które nie tylko są prawdziwe, ale także zawierają informacje niezbędne do wykorzystania tej prawdy. W najbardziej interesujących sytuacjach związanych z rozwiązywaniem problemów nie możemy pozwolić sobie na ignorowanie tych heurystyk. W następnej sekcji wprowadzamy klauzule Horn i używamy ich interpretacji proceduralnej jako jawnej strategii, która zachowuje te heurystyczne informacje.

14.3.2 Programowanie logiczne i Prolog

Aby zrozumieć matematyczne podstawy Prologu, najpierw zdefiniujemy programowanie logiczne. Gdy to zrobimy, dodamy jawną strategię wyszukiwania do programowania logicznego, aby przybliżyć strategię wyszukiwania, czasami nazywaną semantyką proceduralną, w Prologu. Aby uzyskać pełny Prolog, omówimy również użycie not i założenie zamkniętego świata. Rozważ bazę danych klauzul przygotowanych do odrzucenia rezolucji, jak w sekcji 14.2. Jeśli ograniczymy ten zestaw do klauzul, które mają co najwyżej jeden dodatni literał (zero lub więcej ujemnych literałów), mamy przestrzeń klauzul z kilkoma interesującymi właściwościami. Po pierwsze, problemy, które można opisać za pomocą tego zestawu klauzul, powodują, że nie można ich spełnić w przypadku odrzucenia rezolucji lub są one kompletne, sekcja 14.2. Po drugie, istotną korzyścią wynikającą z ograniczenia naszej reprezentacji do tej podklasy wszystkich klauzul jest bardzo skuteczna strategia wyszukiwania obaleń: liniowa forma wejściowa, oparta na preferencjach jednostkowych, redukcja celu od lewej do prawej i od pierwszej w głąb. Dzięki dobrze ugruntowanej rekursji (wywołania rekurencyjne, które ostatecznie się kończą) i sprawdzaniu, strategia ta gwarantuje znalezienie odrzucenia, jeśli przestrzeń klauzul jest niezaspokojona. Zdanie Horn zawiera co najwyżej jeden literał pozytywny, co oznacza, że ma postać

$$a \vee \neg b1 \vee \neg b2 \vee \cdots \vee \neg mld$$

gdzie a i wszystkie bis są literałami dodatnimi. Aby podkreślić kluczową rolę jednego pozytywnego dosłownego w rezolucjach, generalnie piszemy klauzule Horn jako implikacje z pozytywnym dosłownym jako konkluzją:

$a \leftarrow b1 \wedge b2 \wedge \cdots \wedge bn$

Zanim omówimy dalej strategię wyszukiwania, formalnie definiujemy ten podzbiór klauzul, zwany klauzulami Horna. Mówi się, że razem z niedeterministyczną strategią redukcji celu tworzą program logiczny.

DEFINICJA

PROGRAM LOGICZNY

Program logiczny jest zbiorem uniwersalnych wyrażeń ilościowych w rachunku predykatów pierwszego rzędu postaci:

$a \leftarrow b_1 \wedge b_2 \wedge b_3 \wedge \cdots \wedge b_n$

a i b_i to wszystkie dosłowne wartości dodatnie, czasami nazywane celami atomowymi. A jest nagłówkiem zdania, koniunkcją b_i, treścią. Wyrażenia te są klauzulami Horn rachunku predykatów pierwszego rzędu. Występują w trzech formach: po pierwsze, gdy oryginalna klauzula nie zawiera literałów pozytywnych; po drugie, kiedy nie ma ujemnych literałów; i po trzecie, gdy ma jeden dodatni i jeden lub więcej negatywnych literałów. Te przypadki to odpowiednio 1, 2 i 3:

1. $\leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$

nazywana klauzulą bezgłową lub celami do wypróbowania: b_1 i b_2 oraz. . . i b_n.

2. $a_1 \leftarrow$

$a_2 \leftarrow$

..

$a_n \leftarrow$

zwane faktami.

3. $a \leftarrow b_1 \wedge \cdots \wedge b_n$

nazywana relacją reguł.

Rachunek z klauzulą Horn dopuszcza tylko przedstawione formularze; może być tylko jeden literał po lewej stronie ← i ten dosłowny musi być dodatni. Wszystkie literały na prawo od ← są również pozytywne. Redukcja klauzul, które mają co najwyżej jeden literał pozytywny do postaci Horn, wymaga trzech kroków. Najpierw wybierz literał dodatni w klauzuli, jeśli istnieje literał dodatni, i przesuń ten literał w skrajną lewą stronę (używając właściwości przemiennej ∨). Ten pojedynczy literał pozytywny staje się początkiem klauzuli Horn, jak właśnie zdefiniowano. Po drugie, zmień całą klauzulę na formę Horn według reguły:

$a \vee \neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n \equiv a \leftarrow \neg (\neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n)$

Na koniec skorzystaj z prawa de Morgana, aby zmienić tę specyfikację na:

$a \leftarrow b_1 \wedge b_2 \cdots \wedge b_n$

gdzie przemienna własność ∧ może służyć do uporządkowania celów podrzędnych. Należy zauważyć, że może nie być możliwe przekształcenie klauzul z dowolnej przestrzeni klauzul do postaci Horn. Niektóre klauzule, takie jak p ∨ q, nie mają formy rogu. Aby utworzyć klauzulę Horn, w oryginalnej klauzuli może znajdować się co najwyżej jeden literał pozytywny. Jeśli to kryterium nie jest spełnione, może być konieczne ponowne przemyślenie pierwotnego rachunku predykatów specyfikacji problemu. Wypłata za reprezentację formy Horn jest skuteczną strategią obalenia, jak wkrótce zobaczymy. Algorytm obliczeniowy programów logicznych opiera się na niedeterministycznym celu zmniejszenia. Na każdym kroku obliczenia, w którym istnieje cel formularza:

$\leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_n$

tłumacz samowolnie wybiera a_i dla 1 i n. Następnie niedeterministycznie wybiera klauzulę:

$a^1 \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$

tak, że a^1 łączy się z ai z podstawieniem ς i wykorzystuje tę klauzulę do zmniejszenia celu.

Nowy cel staje się wtedy:

$\leftarrow (a_1 \wedge \cdots \wedge a_{i-1} \wedge b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge a_{i+1} \wedge \cdots \wedge an)\ \varsigma$

Ten proces niedeterministycznej redukcji celu trwa do momentu zakończenia obliczeń z pustym celem. Jeśli wyeliminujemy niedeterminizm, narzucając porządek redukcji celów cząstkowych, nie zmieniamy wyniku obliczeń. Wszystkie wyniki, które można znaleźć niedeterministycznie, można znaleźć poprzez wyczerpujące uporządkowane wyszukiwanie. Jednak zmniejszając ilość niedeterminizmu, możemy zdefiniować strategie, które wycinają niepotrzebne gałęzie z przestrzeni. Dlatego głównym problemem praktycznych języków programowania logiki jest zapewnienie programiście możliwości kontrolowania i, jeśli to możliwe, zmniejszania ilości niedeterminizmu. Te ułatwienia pozwalają programiście wpływać zarówno na kolejność, w jakiej cele są redukowane, jak i na zestaw klauzul używanych do redukcji każdego celu. (Jak przy każdym przeszukiwaniu grafów, należy podjąć środki ostrożności, aby zapobiec nieskończonym cyklom w dowodzie.) Abstrakcyjna specyfikacja programu logicznego ma czystą semantykę, taką jak system obalania rozdzielczości. van Emden i Kowalski (1976) pokazują, że najmniejszą interpretacją, na podstawie której program logiczny jest prawdziwy, jest interpretacja programu. Cena płacona przez praktyczne języki programowania, takie jak Prolog, polega na tym, że wykonywanie przez nie programów może obliczać tylko podzbiór powiązanych z nimi interpretacji (Shapiro 1987). Sequential Prolog to przybliżenie interpretera logicznego modelu programowania, zaprojektowanego do wydajnego wykonywania na komputerach von Neumanna. To jest interpreter, z którego do tej pory korzystaliśmy w tym tekście. Sequential Prolog używa zarówno kolejności celów w klauzuli, jak i kolejności klauzul w programie, aby kontrolować wyszukiwanie dowodu. Gdy dostępnych jest kilka celów, Prolog zawsze dąży do nich od lewej do prawej. W poszukiwaniu unifikowalnej klauzuli celu, możliwe klauzule są sprawdzane w kolejności, w jakiej są prezentowane przez programistę. Po dokonaniu każdego wyboru umieszczany jest wskaźnik cofania z zarejestrowanym ujednoliceniem, który umożliwia użycie innych klauzul (ponownie w kolejności programisty) w przypadku niepowodzenia pierwotnego wyboru klauzuli unifikowalnej. Jeśli ta próba zakończy się niepowodzeniem we wszystkich możliwych klauzulach w przestrzeni klauzul, obliczenia nie powiedzie się. W przypadku cut, próby efektywnego wykorzystania przeszukiwania wstecznego w pierwszej kolejności, sekcja 14.1.5, interpreter nie może w rzeczywistości odwiedzać wszystkich kombinacji klauzul (interpretacji) w przestrzeni wyszukiwania. Bardziej formalnie, biorąc pod uwagę cel:

$\leftarrow a_1 \wedge a_2 \wedge a_3 \cdots \wedge a_n$

i program P, interpreter Prologu sekwencyjnie szuka pierwszej klauzuli w P, której głowa łączy się z a_1. Ta klauzula jest następnie wykorzystywana w celu ograniczenia celów. Gdyby:

$$a^1 \leftarrow b_1 \wedge b_2 \wedge \ldots \wedge b_n$$

jest klauzulą redukującą z ξ unifikacją, klauzula celu staje się wtedy:

$$\leftarrow (b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge a_2 \wedge a_3 \wedge \cdots \wedge a_n)\xi$$

Następnie interpreter Prologu kontynuuje próbę zredukowania skrajnego lewego celu, b1 w tym przykładzie, używając pierwszej klauzuli w programie P, która łączy się z b1. Załóżmy, że jest to:

$$b^1 \leftarrow c_1 \wedge c_2 \wedge \cdots \wedge c_p$$

w trakcie zjednoczenia ϕ. Celem staje się wtedy:

$$\leftarrow (c_1 \wedge c_2 \wedge \cdots \wedge c_p \wedge b_2 \wedge \cdots \wedge b_n \wedge a_2 \wedge a_3 \wedge \cdots \wedge a_n)\xi\ \phi$$

Zauważ, że lista celów jest traktowana jako stos wymuszający przeszukiwanie w głąb. Jeśli interpreter Prologu kiedykolwiek nie znajdzie unifikacji, która rozwiązuje cel, wraca do swojego ostatniego punktu wyboru unifikacji, przywraca wszystkie powiązania utworzone od tego punktu wyboru i wybiera następną klauzulę, która ujednolici (według kolejności w P). W ten sposób Prolog implementuje przeszukiwanie od lewej do prawej, najpierw w głąb przestrzeni klauzul. Jeśli cel jest zredukowany do klauzuli null (), to skład unifikacji, które dokonały redukcji:

$\leftarrow (\square)\ \xi\ \phi \cdots \omega$

(tutaj $\xi\ \phi \cdots \omega$), zapewnia interpretację, zgodnie z którą pierwotna klauzula celu była prawdziwa. Oprócz cofania się w kolejności klauzul w programie, sekwencyjny Prolog umożliwia cięcie lub „!”. Jak opisano w sekcji 14.1.5, obniżka może być umieszczona w klauzuli jako cel sam w sobie. Tłumacz, napotykając cięcie, zobowiązuje się do aktualnej ścieżki wykonania, a w szczególności do tego podzbioru unifikacji dokonanych od momentu wyboru klauzuli zawierającej cięcie. Zobowiązuje również tłumacza do wyboru samej klauzuli jako jedynej metody redukcji celu. Jeśli wystąpi błąd w klauzuli po przecięciu, cała klauzula zawiedzie. Proceduralnie cięcie sprawia, że nie ma potrzeby zachowywania wskaźników powrotu dla klauzuli redukcji i wszystkich jej składników przed cięciem. Zatem cięcie może oznaczać, że tylko niektóre z możliwych interpretacji modelu są kiedykolwiek obliczane. Podsumowujemy naszą dyskusję na temat sekwencyjnego Prologu, porównując go z modelem obalania rezolucji w sekcji 14.2.

1. Przestrzeń klauzul rozdzielczości jest nadzbiorem wyrażeń klauzuli Horn w programowaniu logicznym. Każda klauzula musi mieć co najwyżej jeden literał pozytywny, aby mieć formę Horn.

2. Następujące struktury przedstawiają problem w postaci Horn:

a. Cele,

$\leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$

to lista stwierdzeń klauzul, które składają się na cele, które mają być sprawdzone przez odrzucenie rezolucji. Każdy ai jest z kolei zanegowany, ujednolicony z i zredukowany do momentu znalezienia pustej klauzuli (jeśli jest to możliwe).

b. Fakty,

$a_1 \leftarrow$

$a_2 \leftarrow$

...

$a_n \leftarrow$

to osobne klauzule dotyczące rozwiązania. Wreszcie,

c. Reguły lub aksjomaty klauzuli Horna,

$a \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$

pozwalają nam zmniejszyć pasujące cele pośrednie.

3. Z preferencją jednostkową, liniową strategią formularza wejściowego (zawsze preferując klauzule faktów i używając zanegowanego celu i jego pochodnych rozstrzygnięć;) i stosując porządek od lewej do prawej, w głąb (z cofaniem) wybierając klauzule do uchwał, dowódca twierdzenia o rozdzielczości pełni rolę tłumacza Prologu. Ponieważ ta strategia jest zaprzeczeniem kompletnym, jej użycie gwarantuje, że rozwiązanie zostanie znalezione (pod warunkiem, że część zbioru interpretacji nie zostanie usunięta za pomocą cięcia).

4. Wreszcie kompozycja zjednoczeń w dowodzie dostarcza odpowiedzi (interpretacji), dla której cel jest prawdziwy. Jest to dokładnie równoważne procesowi wyodrębniania odpowiedzi z sekcji 14.2.5. Zapisanie składu zjednoczeń w celu dosłownie daje interpretację każdej odpowiedzi.

Ważną kwestią związaną z Prologiem jest to, że chcemy używać negatywnych literałów po prawej stronie reguł klauzuli Horn. Wymaga to od tłumaczy egzekwowania założenia o zamkniętym świecie. W rachunku predykatów dowód p (X) jest dokładnie dowodem, że p (X) jest logicznie fałszywe. Oznacza to, że p (X) jest fałszywe w każdej interpretacji, która sprawia, że aksjomat jest prawdziwy.

Interpreter Prologu, oparty na algorytmie unifikacji z rozdziału 2, oferuje bardziej ograniczone wyniki niż odrzucenie ogólnej rozdzielczości w sekcji 14.2. Zamiast wypróbowywać wszystkie interpretacje, bada tylko te interpretacje, które są wyraźnie określone w bazie danych. Teraz aksjomatyzujemy te ograniczenia, aby dokładnie zobaczyć ograniczenia implikowane w obecnych środowiskach Prologu. Dla każdego predykatu p i każdej zmiennej X należącej do p załóżmy, że a1, a2,. . . , tworzą domenę X. Interpreter Prologu, stosując ujednolicenie, wymusza:

1. Aksjomat unikalnej nazwy. Dla wszystkich atomów domeny ai aj, chyba że są identyczne. Oznacza to, że atomy o różnych nazwach są różne.

2. Aksjomat świata zamkniętego.

$p(X) \rightarrow p(a_1) \vee p(a_2) \vee \cdots \vee p(a_n)$.

Oznacza to, że jedynymi możliwymi instancjami relacji są te implikowane przez klauzule obecne w specyfikacji problemu.

3. Aksjomat domknięcia domeny.

$(X = a_1) \vee (X = a_2) \vee \cdots \vee (X = a_n)$.

Gwarantuje to, że atomy występujące w specyfikacji problemu stanowią wszystkie i jedyne atomy.

Te trzy aksjomaty są implicite w działaniu interpretera Prologu. Można je postrzegać jako dodane do zestawu klauzul Horn tworzących opis problemu, a tym samym jako ograniczające zbiór możliwych interpretacji do zapytania Prologu. Intuicyjnie oznacza to, że Prolog przyjmuje za fałszywe wszystkie cele, których nie może udowodnić. Może to wprowadzać anomalie: jeśli aktualna baza danych faktycznie nie zna prawdziwości celu, Prolog przyjmie, że jest fałszywa. Inne ograniczenia są domniemane w Prologu, ponieważ wydają się występować we wszystkich językach komputerowych.

Najważniejsze z nich, poza problemem negacji jako niepowodzenia, stanowią naruszenie semantycznego modelu programowania logicznego. W szczególności brakuje kontroli występowania (patrz sekcja 2.3; umożliwia to ujednolicenie klauzuli z jej podzbiorem) i użycie cięcia. Obecną generację tłumaczy Prolog należy spojrzeć pragmatycznie. Niektóre problemy pojawiają się, ponieważ „obecnie nie jest znany żaden skuteczny sposób" obejścia problemu (jest to sytuacja z kontrolą występowania); inne powstają w wyniku prób zoptymalizowania użycia pierwszej operacji w głąb z przeszukiwaniem wstecznym (jawne kontrolowanie wyszukiwania za pomocą cięcia). Wiele anomalii Prologu jest wynikiem próby zaimplementowania niedeterministycznej semantyki programowania w czystej logice na komputerze sekwencyjnym. Obejmuje to problemy związane z cięciem. W ostatniej części wprowadzamy alternatywne schematy wnioskowania do automatycznego wnioskowania.

14.4 Dalsze problemy w automatycznym rozumowaniu

Opisaliśmy słabe metody rozwiązywania problemów jako wykorzystujące (a) jednolity środek reprezentacji dla (b) zasad wnioskowania dźwiękowego, które koncentrują się na składniowych cechach reprezentacji i są kierowane przez (c) metody lub strategie zwalczania kombinatoryki wyszukiwania wyczerpującego. Kończymy ten rozdział dalszymi komentarzami na temat każdego z tych aspektów procesu rozwiązywania słabych metod.

14.4.1 Jednolite reprezentacje rozwiązań słabych metod

Procedura sprawdzania rozdzielczości wymaga od nas umieszczenia wszystkich naszych aksjomatów w formie klauzuli. Ta jednolita reprezentacja pozwala nam następnie rozwiązywać klauzule i upraszcza projektowanie heurystyk rozwiązywania problemów. Jedną z głównych wad tego podejścia jest to, że wiele cennych informacji heurystycznych może zostać utraconych w tym jednolitym kodowaniu. Jeśli. . . format reguły często dostarcza więcej informacji do wykorzystania modus ponens lub wyszukiwania systemu produkcyjnego niż jeden z jej wariantów składniowych. Oferuje nam również skuteczny sposób korzystania z reguły. Na przykład, przypuśćmy, że chcemy przedstawić wnioskowanie abdukcyjne, sekcja 8.0. Jeżeli silnik się nie obraca, a światła nie zapalają się, może to oznaczać, że akumulator jest rozładowany. Ta zasada podpowiada, jak sprawdzić baterię. Rozłączna forma tej samej reguły przesłania te heurystyczne informacje o tym, jak reguła powinna być stosowana. Jeśli wyrazimy tę regułę w rachunku predykatów ¬ turn_over ∧

¬ światła → bateria, forma klauzuli tej reguły jest następująca: obroty over światła ∨ bateria.

Ta klauzula może mieć kilka równoważnych form, a każda z nich reprezentuje inną implikację.

(¬ obraca się ∧ ¬ świeci) → akumulator

(¬ obraca się → (świeci się bateria ∨))

(¬ bateria ∧ ¬ świeci) → turn_over

(¬ bateria → (włącza się świeci))

i tak dalej.

Aby zachować informacje heurystyczne w procesie automatycznego rozumowania, kilku badaczy, w tym Nilsson i Bundy, opowiada się za metodami rozumowania, które kodują heurystykę, tworząc reguły zgodnie ze sposobem, w jaki ludzki ekspert mógłby zaprojektować relacje reguł. Zaproponowaliśmy to podejście już w naszym rozumowaniu i / lub na wykresie w sekcji 3.3 oraz w formie Prologu automatycznego rozumowania w sekcji 14.3. Oparte na regułach systemy eksperckie pozwalają także programiście kontrolować przeszukiwanie struktury reguł. Rozwijamy ten pomysł w kolejnych dwóch przykładach, jednym opartym na danych, a drugim zorientowanym na cel. Oba zachowują formę implikacji i wykorzystują te informacje do kierowania wyszukiwaniem przez wykres i / lub wykres. Rozważ, do wykorzystania w rozumowaniu opartym na danych, następujące fakty, reguły (aksjomaty) i cel:

Fakt:

(a ∨ (b ∧ c))

Reguły (lub aksjomaty):

(a → (d ∧ e))

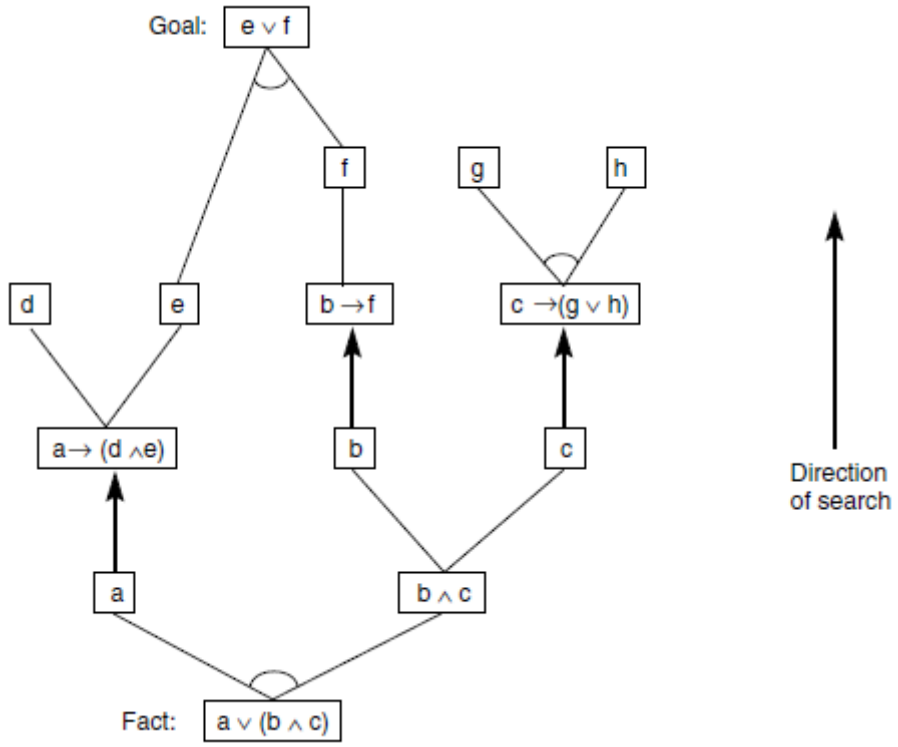
(b → f)

(c → (g ∨ h))

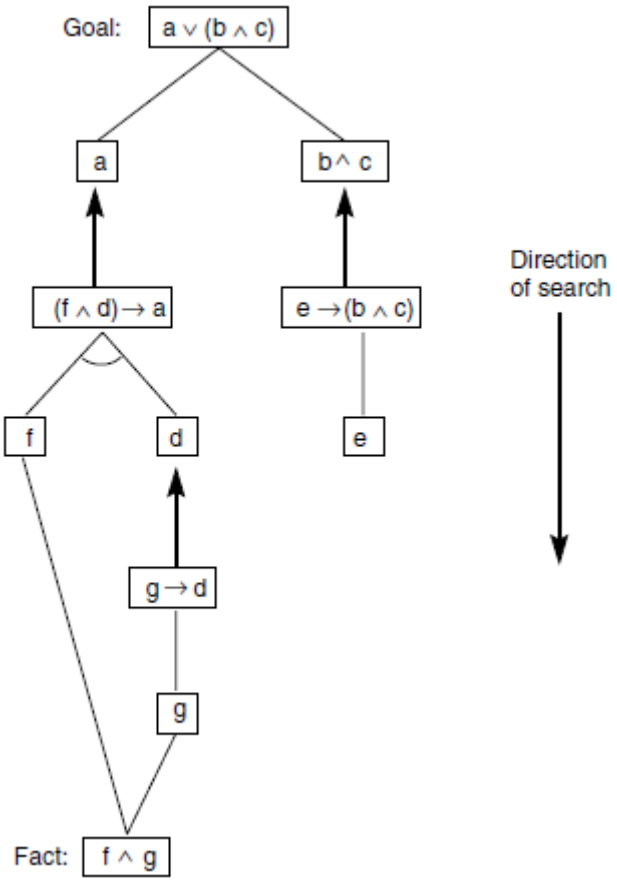
Cel:

← e ∨ f

Dowód e ∨ f można znaleźć na wykresie i / lub na rysunku 13.



Zwróć uwagę na użycie łączników i w relacjach ∨ oraz łączników lub w relacjach ∧ w przestrzeni wyszukiwania opartej na danych. Jeśli otrzymamy, że albo a albo b ∧ c jest prawdą, to musimy rozumować z oboma rozłącznikami, aby zagwarantować, że nasz argument zachowuje prawdę; stąd te dwie ścieżki są połączone. Z drugiej strony, gdy b i c są prawdziwe, możemy kontynuować badanie któregokolwiek z tych koniunkcji. Dopasowywanie reguł przyjmuje dowolny stan pośredni, taki jak c, i zastępuje go konkluzją reguły, takiej jak (g ∨ h), której przesłanka pasuje do tego stanu. Odkrycie obu stanów eif na rysunku 13 wskazuje, że cel (e ∨ f) został ustalony. W podobny sposób możemy użyć dopasowania reguł i / lub wykresów do rozumowania zorientowanego na cel. Gdy opis celu zawiera ∨, jak na przykładzie na rysunku 14,



wówczas każdą alternatywę można zbadać niezależnie, aby ustalić cel. Jeśli celem jest koniunkcja, to oczywiście oba spójniki muszą zostać ustanowione. Cel:

(a ∨ (b ∧ c))

Reguły (lub aksjomaty):

(f ∧ d) → a

(e → (b ∧ c))

(g → d)

Fakt:

f ∧ g

Chociaż przykłady te pochodzą z rachunku zdań, podobne wyszukiwanie jest generowane przy użyciu faktów i reguł rachunku predykatów. Ujednolicenie zapewnia zgodność literałów do stosowania reguł wnioskowania w różnych gałęziach przestrzeni wyszukiwania. Oczywiście unifikacje muszą być spójne (to znaczy unifikowalne) w różnych i gałęziach przestrzeni poszukiwań. W tej podsekcji zasugerowano metody rozwiązywania problemów pomagające zachować informacje heurystyczne w medium reprezentacyjnym w celu rozwiązywania problemów metodą słabych. Jest to zasadniczo sposób, w jaki silniki wnioskowania systemów ekspertowych pozwalają programiście określać informacje sterujące i heurystyczne w regule. Systemy eksperckie opierają się na formie reguły, takiej jak porządek reguł lub uporządkowanie przesłanek w regule, do kontroli wyszukiwania, a nie całkowicie polegają na ogólnych słabych metodach rozwiązywania problemów. To, co jest tracone w tym podejściu, to możliwość stosowania jednolitych procedur dowodowych, takich jak rozdzielczość, w całym zestawie reguł. Jak można zauważyć w przykładach przedstawionych na rysunku 13 i 14, modus ponens może być jednak nadal używany. Sterowanie systemem produkcyjnym przy użyciu przeszukiwania najpierw w głąb, wszerz lub przeszukiwania bestfirst oferuje jedną słabą architekturę rozumowania metod do implementacji systemów reguł .

14.4.2 Alternatywne reguły wnioskowania

Rozdzielczość jest najbardziej ogólną zasadą wnioskowania o dźwięku, jaką do tej pory przedstawiliśmy. W celu zwiększenia wydajności rozdzielczości stworzono kilka bardziej wyrafinowanych reguł wnioskowania. Omówimy krótko dwa z nich: hiper-rozdzielczość i paramodulację. Rozdzielczość, tak jak ją przedstawiliśmy, jest w rzeczywistości specjalnym wariantem zwanym rozdzielczością binarną: zderzają się dokładnie dwie klauzule nadrzędne. Pomyślne zastosowanie hiper-rozdzielczości zastępuje sekwencję rozwiązań binarnych w celu utworzenia jednej klauzuli. Hiperrozdzielczość zderza w jednym kroku klauzulę z niektórymi literałami ujemnymi, zwanymi jądrem, i pewną liczbą klauzul ze wszystkimi literałami dodatnimi, zwanymi satelitami. Te satelity muszą mieć jeden literał dodatni, który będzie zgodny z literałem ujemnym jądra. Musi być również jeden satelita dla każdego literału ujemnego jądra. Tak więc wynikiem zastosowania hiper-rozdzielczości jest klauzula zawierająca wszystkie pozytywne literały. Zaletą hiperrozdzielczości jest to, że klauzula wszystkich dodatnich literałów jest tworzona z każdego wnioskowania o hiperrozdzielczości, a sama przestrzeń klauzul jest mniejsza, ponieważ nie są tworzone żadne wyniki pośrednie. Ujednolicenia we wszystkich klauzulach na etapie wnioskowania muszą być spójne. Jako przykład hiperrozdzielczości rozważ następujący zestaw klauzul:

¬ żonaty (X, Y) ∨ ¬ matka (X, Z) ∨ ojciec (Y, Z)

żonaty (kate, george) ∨ lubi (george, kate)

matka (kate, sarah)

Wyciągamy wniosek w jednym kroku, używając hiper-rozdzielczości:

ojciec (george, sarah) ∨ lubi (george, kate)

Pierwsza klauzula w tym przykładzie to jądro; drugie dwa to satelity. Wszystkie satelity są dodatnie, aw jądrze znajduje się jeden na każdy literał ujemny. Zwróć uwagę, że jądro jest po prostu formą klauzuli dla implikacji:

żonaty (X, Y) ∧ matka (X, Z) → ojciec (Y, Z)

Zakończenie tej zasady jest częścią końcowego wyniku. Zwróć uwagę, że nie ma wyników pośrednich, takich jak:

¬ matka (kate, Z) ∨ ojciec (george, Z) ∨ lubi (george, kate)

które znaleźlibyśmy w każdym dowodzie rozdzielczości binarnej zastosowanym do tej samej przestrzeni klauzul. Hiperrozdzielczość jest solidna i kompletna, gdy jest używana samodzielnie. W połączeniu z innymi strategiami, takimi jak zestaw wsparcia, kompletność może być zagrożona (Wos i in. 1984). Wymaga to specjalnych strategii wyszukiwania w celu uporządkowania klauzul satelity i jądra, chociaż w większości środowisk, w których używana jest hiper-rozdzielczość, klauzule są często indeksowane według nazwy i dodatniej lub ujemnej właściwości każdego literału. To sprawia, że przygotowanie klauzul jądra i satelity do wnioskowania o hiper-rozdzielczości jest wydajne. Ważną i trudną kwestią w projektowaniu mechanizmów dowodzenia twierdzeń jest kontrola równości. Szczególnie złożone są obszary zastosowań, takie jak matematyka, w których większość faktów i relacji ma wiele reprezentacji, na przykład można je uzyskać, stosując do wyrażeń właściwości asocjacyjne i przemienne. Aby przekonać się o tym za pomocą bardzo prostego przykładu, rozważ wiele sposobów przedstawienia wyrażenia arytmetycznego 3 + (4 + 5), w tym 3 + ((4 + 0) + 5). Jest to złożona kwestia, ponieważ wyrażenia należy zastępować, ujednolicać i sprawdzać pod kątem równości z innymi wyrażeniami w ramach automatycznego rozwiązywania problemów matematycznych. Demodulacja to proces przeformułowywania lub przepisywania wyrażeń, aby automatycznie przyjmowały wybraną formę kanoniczną. Klauzule jednostek używane do tworzenia tej formy kanonicznej nazywane są demodulatorami. Demodulatory określają równość różnych wyrażeń, dzięki czemu możemy zastąpić wyrażenie jego formą kanoniczną. Przy odpowiednim użyciu demodulatorów wszystkie nowo utworzone informacje są redukowane do określonej postaci, zanim zostaną umieszczone w przestrzeni klauzul. Na przykład możemy mieć demodulator:

równy (ojciec (ojciec (X)), dziadek (X))

i nowa klauzula:

wiek (ojciec (ojciec (sarah)), 86).

Przed dodaniem tej nowej klauzuli do przestrzeni klauzul stosujemy demodulator i zamiast tego dodajemy:

wiek (dziadek (sarah), 86).

Problem równości dotyczy tutaj nazewnictwa. Czy chcemy zaklasyfikować osobę jako ojca (ojciec (X)) czy dziadka (X)? Podobnie, możemy wybrać nazwy kanoniczne dla wszystkich relacji rodzinnych: brat (ojciec (Y)) to wujek (Y) itd. Po wybraniu nazw kanonicznych do przechowywania informacji, projektujemy demodulatory, takie jak klauzula równości zredukuj wszystkie nowe informacje do tej określonej formy. Zauważ, że demodulatory są zawsze klauzulami jednostkowymi. Paramodulacja jest uogólnieniem zastępowania równości na poziomie terminu. Na przykład, biorąc pod uwagę wyrażenie:

starszy (matka (Y), Y)

i relacja równości:

równy (matka (Sarah), kate)

możemy zakończyć paramodulacją:

starszy (kate, sarah)

Zwróć uwagę na dopasowanie na poziomie terminów i zamianę {Sarah / Y} i matka (Sarah) na kate. Istotną różnicą między demodulacją a paramodulacją jest to, że ta ostatnia pozwala na nietrywialne zastępowanie zmiennych zarówno w argumentach predykatu równości, jak i predykatu, w którym

następuje podstawienie. Demodulacja polega na wymianie na podstawie demodulatora. Można użyć wielu demodulatorów, aby uzyskać wyrażenie w jego ostatecznej postaci; paramodulacja jest zwykle używana tylko raz w każdej sytuacji. Podaliśmy kilka prostych przykładów tych potężnych mechanizmów wnioskowania. Powinny być postrzegane jako bardziej ogólne techniki stosowane w klauzuli rozdzielczości. Podobnie jak wszystkie inne reguły wnioskowania, które widzieliśmy, są one ściśle powiązane z wybraną reprezentacją i muszą być kontrolowane przez odpowiednie strategie.

14.4.3 Udzielanie odpowiedzi na pytania poparte odrzuceniem rezolucji

W sekcji 14.2.5 wykazaliśmy naturalne dopasowanie między procesem obalania rozdzielczości a generowaniem odpowiedzi na pierwotne zapytanie (twierdzenie, które ma zostać udowodnione). Stuart Shapiro i jego koledzy pokazują, że obalanie rezolucji jest również użytecznym paradygmatem odpowiedzi na pytania. Burhans i Shipiro zauważają, że proces obalenia dzieli klauzule wygenerowane podczas obalania rezolucji na trzy klasy, z których każda jest odpowiednia do udzielenia odpowiedzi na różne rodzaje pytań dotyczących rozważanego twierdzenia. W swoich klauzulach badawczych, które są istotne, są identyfikowane jako możliwe odpowiedzi, gdzie trafność jest określana na podstawie pytania (twierdzenia, które ma zostać udowodnione) i bazy wiedzy (zestaw klauzul użytych w dowodzie), gdzie każda klauzula pochodzi z klauzuli uznaje się formę zanegowanej bramki. Dowód obalenia jest podzielony na trzy klasy odpowiedzi: specyficzne, ogólne i hipotetyczne. Klasy te rozróżnia się sposobem, w jaki literały tworzące klauzulę współużytkują zmienne. Rezultatem jest niezależność od kontekstu, tj. może być używana z dowolnym obaleniem opartym na zestawie wsparcia, pragmatyką do odpowiadania na pytania. Konkretne odpowiedzi, czasami nazywane odpowiedziami ekstensjonalnymi ,są powiązane ze zbiorem terminów podstawowych odpowiadających stałym podstawionym wartościom zmiennym w procesie tworzenia refutacji. Na przykład zapytanie „czy jest ktoś w domu?" zostałby zastąpiony przez zanegowany cel „nie ma nikogo w domu" i prowadziłby do obalenia, gdy zostanie pokazane, że „Amy jest w domu". Ten zbiór osób, które można wytworzyć w procesie obalenia, dostarcza konkretnych odpowiedzi. Odpowiedzi ogólne, zwane także odpowiedziami intensywnymi (Green 1969a, b), odpowiadają wszystkim nieuziemionym przypadkom klauzul odpowiedzi, które mogą być powiązane z odrzuceniem. Na przykład w przypadku zapytania „czy jest ktoś w domu?" Zbiór składa się z klauzul opartych na zmiennych, takich jak „wszystkie dzieci są w domu", pod warunkiem ogólnych odpowiedzi. Wreszcie, hipotetyczne odpowiedzi, czasami nazywane warunkowymi lub kwalifikowanymi, przyjmują postać reguły, której poprzednik ma pewne zmienne bazowe i której następnik pasuje do zanegowanego zapytania o odpowiedź. Na przykład „czy jest ktoś w domu?" może pasować do klauzuli „jeśli Allen jest w domu, to Amy jest w domu" podczas późniejszego wyszukiwania w celu ustalenia, czy Allen jest w domu. Więcej szczegółów w Burhans i Shiparo.

Czasami dziedzina aplikacji stawia szczególne wymagania regułom wnioskowania i heurystyce w celu kierowania ich użyciem. Widzieliśmy już użycie demodulatorów do pomocy w zastępowaniu równości. Bledsoe w swoim naturalnym systemie dedukcji identyfikuje dwie ważne strategie przygotowywania twierdzeń do dowodu rozdzielczości. Nazywa te strategie dzieleniem i ograniczaniem. Blades zaprojektował swoje strategie do zastosowania w matematyce, a zwłaszcza do zastosowania w teorii mnogości. Efektem tych strategii jest rozbicie twierdzenia na części, aby ułatwić jego udowodnienie konwencjonalnymi metodami, takimi jak rozdzielczość. Split przyjmuje różne formy matematyczne i dzieli je na odpowiednie części. Dowód $A \wedge B$ jest równoważny z dowodem A i dowodem B. Podobnie, dowód $A \leftrightarrow B$ jest dowodem $A \rightarrow B$ i dowodem $A \leftarrow B$. Redukcja heurystyczna również próbuje przełamać duże dowody na ich elementy. Na przykład dowód $s \in A \cap B$ można rozłożyć na dowody $s \in A$ i $s \in B$.Znowu, aby udowodnić właściwość $\neg (A \cup B)$ udowodnić właściwość dla $\neg A$ i dla $\neg B$. Bledsoe ma nadzieję, że dzieląc większe dowody na mniejsze części, ograniczy przestrzeń poszukiwań. Jego

heurystyki obejmują również ograniczone użycie zastępowania równości. Jak wspomniano w tej książce, właściwe użycie heurystyki jest w dużej mierze sztuką, która bierze pod uwagę obszar zastosowania, a także zastosowane reguły reprezentacji i wnioskowania. Zamykamy ten rozdział, przytaczając kilka ogólnych przysłów, z których wszystkie są czasami fałszywe, ale które przy ostrożnym użyciu mogą być bardzo skuteczne. Przysłowia te podsumowują przemyślenia badaczy z tego obszaru, a także nasze własne refleksje na temat słabych metod rozwiązywania problemów. Podajemy je bez dalszych komentarzy. Jeśli to możliwe, używaj klauzul z mniejszą liczbą literałów. Przed zastosowaniem wnioskowania ogólnego podziel zadanie na podzadania.

Używaj predykatów równości, gdy jest to właściwe.

Użyj demodulatorów do tworzenia form kanonicznych.

Użyj paramodulacji podczas wnioskowania z predykatami równości.

Używaj strategii, które zachowują „kompletność".

Użyj zestawu strategii wsparcia, ponieważ zawierają one potencjalną sprzeczność.

Używaj jednostek w ramach rozdzielczości, ponieważ skracają one wynikową klauzulę.

Przeprowadź kontrole podporządkowania z nowymi klauzulami.

Użyj mechanizmu porządkowania klauzul i literałów w klauzulach, które odzwierciedlają Twoją intuicję i doświadczenie w rozwiązywaniu problemów.

14.6 Ćwiczenia

Sztuczna inteligencja : Rozumienie języka naturalnego

(XV / XVI)

ĆWICZENIA

1. Sklasyfikuj każde z poniższych zdań jako niepoprawne składniowo, poprawne składniowo, ale bez znaczenia, znaczące, ale nieprawdziwe lub prawdziwe. Na którym etapie procesu zrozumienia każdy z tych problemów jest wykrywany?

Bezbarwne zielone pomysły śpią wściekle.

Owoce lecą jak banan.

Dogs The Bite Man a.

George Washington był piątym prezydentem USA.

To ćwiczenie jest łatwe.

Chcę być pod wodą w zacienionym ogrodzie ośmiornicy.

2. Omów struktury reprezentacyjne i wiedzę niezbędną do zrozumienia poniższych zdań.

Brązowy pies zjadł kość.

Przymocuj duże koło do osi za pomocą nakrętki sześciokątnej.

Mary podlała rośliny.

Duch jest chętny, ale ciało jest słabe.

Moje królestwo za konia!

3. Przeanalizuj każde z tych zdań, używając gramatyki „świata psów" z sekcji 15.2.1. Które z poniższych wyroków są nielegalne? Czemu?

Pies gryzie psa.

Duży pies gryzie mężczyznę.

Emma lubi chłopca.

Mężczyzna lubi.

Ugryź mężczyznę.

4. Rozszerz gramatykę świata psów, tak aby zawierała niedozwolone zdania w ćwiczeniu 3.

5. Przeanalizuj każde z tych zdań, korzystając z gramatyki kontekstowej z sekcji 15.2.3.

Mężczyźni lubią psa.

Pies gryzie mężczyznę.

6. Utwórz drzewo parsowania dla każdego z poniższych zdań. Będziesz musiał rozszerzyć nasze proste gramatyki o bardziej złożone konstrukcje językowe, takie jak przysłówki, przymiotniki i frazy przyimkowe. Jeśli zdanie ma więcej niż jedno parsowanie, zrób diagram wszystkich z nich i wyjaśnij semantyczne informacje, które zostaną użyte do wybrania parsowania.

Czas leci jak strzała, ale owoce lecą jak banan.

Tom dał Mary we wtorek dużą, czerwoną książkę.

Rozumowanie to sztuka, a nie nauka.

Błądzić jest rzeczą ludzką, przebaczać boskości.

7. Rozszerz gramatykę świata psów o przymiotniki we frazach rzeczownikowych. Upewnij się, że dopuszczasz nieokreśloną liczbę przymiotników. Wskazówka: użyj reguły rekurencyjnej adjective_list, która jest pusta lub zawiera przymiotnik, po którym następuje lista przymiotników. Zmapuj tę gramatykę na sieci przejściowe.

8. Dodaj następujące bezkontekstowe reguły gramatyczne do gramatyki świata psów w sekcji 15.2.1. Zmapuj wynikową gramatykę na sieci przejściowe.

zdanie ↔ noun_phrase verb_phrase prepositional_phrase

przyimek_fraza ↔ przyimek noun_phrase

przyimek ↔ z

przyimek ↔ do

przyimek ↔ on

9. Zdefiniuj parser ATN dla gramatyki świata psów z przymiotnikami (ćwiczenie 7) i wyrażeniami przyimkowymi (ćwiczenie 8).

10. Zdefiniuj pojęcia i relacje w grafach pojęciowych potrzebnych do przedstawienia znaczenia gramatyki Ćwiczenia 9. Zdefiniuj procedury budowania reprezentacji semantycznej z drzewa parsowania.

11. Rozszerz gramatykę kontekstualną w sekcji 15.2.3, aby sprawdzić zgodność semantyczną między podmiotem a czasownikiem. W szczególności mężczyźni nie powinni gryźć psów, chociaż psy mogą lubić lub gryźć mężczyzn. Wykonaj podobną modyfikację do gramatyki ATN.

12. Rozwiń gramatykę ATN w sekcji 15.2.4, aby uwzględnić pytania dotyczące kogo i jakie.

13. Opisz, jak można połączyć modele Markowa z sekcji 15.4 z bardziej symbolicznym podejściem do rozumienia języka z sekcji 15.1-15.3. Opisać, w jaki sposób modele stochastyczne można połączyć z tradycyjnymi systemami opartymi na wiedzy, rozdział 8.

14. Rozszerz przykład interfejsu bazy danych z sekcji 15.5.2, tak aby odpowiadał na pytania w formularzu „Ile zarabia Don Morrison?” Będziesz musiał rozszerzyć gramatykę, język reprezentacji i bazę wiedzy.

15. Weź poprzednie zadanie i ułóż jego słowa, w tym znaki interpunkcyjne, w przypadkowej kolejności.

16. Załóżmy, że menedżerowie są wymienieni w relacji pracownik_ wynagrodzenie z innymi pracownikami na przykładzie w sekcji 15.5.2. Rozszerz przykład, aby obsługiwał zapytania, takie jak „Znajdź pracownika, który zarabia więcej niż jego przełożony”.

17. W jaki sposób można połączyć podejścia stochastyczne z sekcji 15.4 z technikami analizy bazy danych, które opisano w sekcji 15.5.2.

18. Wykorzystanie podejścia stochastycznego do odkrywania wzorców w relacyjnej bazie danych jest ważnym obszarem obecnych badań, czasami nazywanym eksploracją danych (patrz Rozdział 10.3). Jak można wykorzystać tę pracę do udzielenia odpowiedzi na pytania, takie jak te postawione w sekcji 15.5.2, dotyczące relacyjnych baz danych?

19. Jako projekt zbuduj system wyodrębniania informacji dla jakiejś domeny wiedzy do wykorzystania w sieci WWW. Sugestie znajdują się w sekcji 15.5.3.

20. Z materiałów pomocniczych weźmy strukturę Prologu bezkontekstowych i kontekstowych parserów i przymiotników, przysłówków i zdań zdaniowych do gramatyki.

21. Z materiałów pomocniczych tej książki, przyjmij do gramatyki strukturę probabilistycznego parsera bezkontekstowego Prologu oraz przymiotników, przysłówków i zdań zdaniowych.

ROZUMIENIE JĘZYKA NATURALNEGO

15.0 Problem rozumienia języka naturalnego

Komunikowanie się za pomocą języka naturalnego, czy to w formie tekstu, czy mowy, w dużej mierze zależy od naszych umiejętności językowych, znajomości interesującej nas dziedziny i oczekiwań w tej dziedzinie. Zrozumienie języka to nie tylko przekazywanie słów: wymaga również wnioskowania o celach, wiedzy i założeniach mówcy, a także o kontekście interakcji. Wdrożenie programu rozumienia języka naturalnego wymaga, abyśmy przedstawiali wiedzę i oczekiwania dotyczące domeny oraz skutecznie je uzasadniali. Musimy wziąć pod uwagę takie kwestie, jak brak monotoniczności, weryfikacja przekonań, metafora, planowanie, uczenie się i praktyczne zawiłości interakcji międzyludzkich. Ale to są główne problemy samej sztucznej inteligencji! Rozważmy jako przykład następujące wersety z Sonetu XVIII Szekspira:

Czy mam cię porównać do letniego dnia?

Jesteś piękniejszy i bardziej umiarkowany:

Ostry wiatr wstrząsa ukochanymi pąkami maja,

A letnia dzierżawa ma zbyt krótki termin:

Nie możemy zrozumieć tych linii poprzez uproszczone, dosłowne potraktowanie znaczenia. Zamiast tego musimy zająć się takimi kwestiami, jak:

1. Jakie były intencje Szekspira na piśmie? Musimy poznać ludzką miłość i jej społeczne konwencje, aby zacząć rozumieć te kwestie. A może Szekspir próbował po prostu dostać coś do swojego wydawcy, żeby mu zapłacić?

2. Dlaczego Szekspir porównał swoją ukochaną do letniego dnia? Czy chodzi mu o to, że ma 24 godziny i może powodować oparzenia słoneczne, czy też sprawia, że czuje ciepło i piękno lata?

3. Jakich wniosków wymaga ten fragment? Zamierzone znaczenie Szekspira nie znajduje się bezpośrednio w tekście; należy ją wywnioskować za pomocą metafor, analogii i wiedzy podstawowej. Na przykład, jak możemy zinterpretować odniesienia do gwałtownych wiatrów i krótkiego lata jako lamentujące nad krótkością ludzkiego życia i miłości?

4. Jak metafora kształtuje nasze rozumienie? Słowa nie są jedynie odniesieniami do wyraźnych przedmiotów, takich jak klocki na stole: znaczenie wiersza polega na selektywnym przypisywaniu ukochanemu właściwości letniego dnia. Które właściwości są przypisywane, a które nie, a przede wszystkim dlaczego niektóre właściwości są ważne podczas gdy inni są ignorowani?

5. Czy komputerowy system zamiany tekstu na mowę musi wiedzieć coś na temat pentametru jambicznego? Jak komputer mógłby podsumować, o czym „jest" ten wiersz, lub w inteligentny sposób wydobyć to z korpusu poezji?

Nie możemy po prostu połączyć ze sobą słownikowych znaczeń słów Szekspira i nazwać wyniku zrozumieniem. Zamiast tego musimy zastosować złożony proces wychwytywania wzorców słów, analizowania zdań, konstruowania reprezentacji znaczeń semantycznych i interpretowania tych znaczeń w świetle naszej wiedzy o dziedzinie problemowej. Nasz drugi przykład to część reklamy internetowej na stanowisko wydziału informatyki. Wydział Informatyki Uniwersytetu Nowego Meksyku. . . prowadzi poszukiwania w celu obsadzenia dwóch etatów. Jesteśmy zainteresowani zatrudnieniem osób z zainteresowaniami: Oprogramowanie, w tym narzędzia analityczne, projektowe i programistyczne. . . Systemy, w tym architektura, kompilatory, sieci. . .

... Kandydaci musieli mieć doktorat w.. . Wydział posiada uznane na całym świecie programy badawcze z zakresu obliczeń adaptacyjnych, sztucznej inteligencji,. . . i cieszy się ścisłą współpracą badawczą z Santa Fe Institute i kilkoma krajowymi laboratoriami. . .

Przy zrozumieniu tego ogłoszenia o pracę pojawia się kilka pytań:

1. Skąd czytelnik może wiedzieć, że ta reklama dotyczy stanowiska naukowego, skoro wyraźnie podano tylko „tytuł stażu"? Na jak długo ludzie są zatrudniani na podstawie umowy o pracę?

2. Jakie oprogramowanie i narzędzia programowe są potrzebne do pracy w środowisku uniwersyteckim, gdy żadne z nich nie zostało wyraźnie wymienione? Cobol, Prolog, UML? Aby zrozumieć te oczekiwania, potrzebowałaby dużej wiedzy na temat nauczania i badań uniwersyteckich.

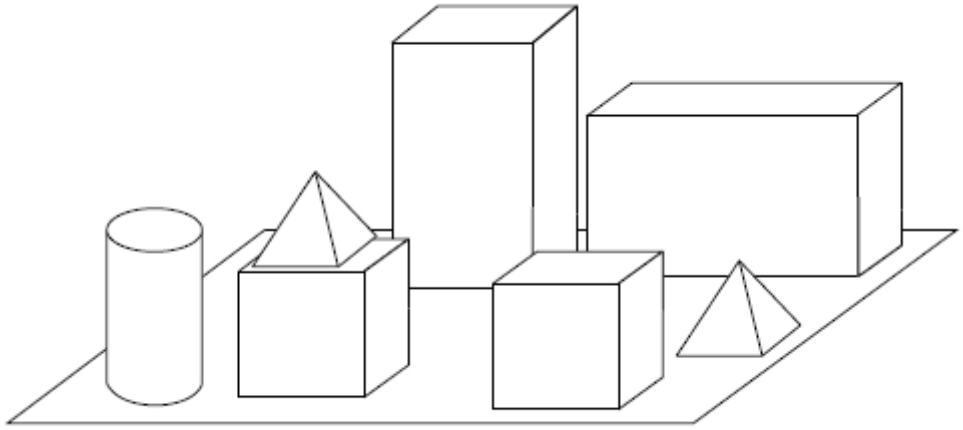
3. Co wspólnego z ogłoszeniem o pracy na uczelni mają uznane na arenie międzynarodowej programy i współpraca z interesującymi instytucjami?

4. Jak komputer mógłby podsumować, o czym jest ta reklama? Co musi wiedzieć, aby w inteligentny sposób pobrać z sieci to ogłoszenie dla doktoranta poszukującego pracy?

Istnieją (przynajmniej!) Trzy główne kwestie związane ze zrozumieniem języka. Po pierwsze, zakłada się dużą ilość wiedzy ludzkiej. Akty językowe opisują relacje w często złożonym świecie. Znajomość tych relacji musi być częścią każdego zrozumienia system. Po drugie, język jest oparty na wzorcach: fonemy są składnikami słów, a słowa tworzą frazy i zdania. Porządki fonemów, słów i zdań nie są przypadkowe. Komunikacja jest niemożliwa bez raczej ograniczonego użycia tych komponentów. Wreszcie akty językowe są wytworem agentów, czy to człowieka, czy komputera. Agenci są osadzeni w złożonych środowiskach o wymiarze indywidualnym i socjologicznym. Akty językowe są celowe.

Ta Część zawiera wprowadzenie do problemów rozumienia języka naturalnego i technik obliczeniowych opracowanych w celu ich rozwiązania. Chociaż w tym rozdziale skupiamy się przede wszystkim na rozumieniu tekstu, systemy rozumienia mowy i generowania również muszą rozwiązać

te problemy, a także dodatkowe trudności związane z rozpoznawaniem i ujednoznacznianiem słów osadzonych w określonym kontekście. Wczesne programy AI, ze względu na wiedzę wymaganą do zrozumienia nieograniczonego języka, poczyniły postęp, ograniczając się do mikroświatów, ograniczonych zastosowań która wymagała minimalnej wiedzy o domenie. Jednym z najwcześniejszych programów stosujących to podejście był SHRDLU Terry'ego Winograda, który mógł rozmawiać o świecie bloków składającym się z bloków o różnych kształtach i kolorach oraz dłoni do poruszania nimi, jak na rysunku 1.



SHRDLU może odpowiadać na zapytania w języku angielskim, takie jak „Co jest na czerwonym bloku?" „Jaki kształt ma niebieski klocek na stole?" lub „Umieść zieloną piramidę na czerwonej cegle". Może obsługiwać odniesienia do zaimków, takie jak „Czy istnieje czerwony blok? Podnieś to." Mógłby nawet zrozumieć elipsy, takie jak „Jakiego koloru jest blok na niebieskiej cegle? Kształt?" Ze względu na prostotę świata bloków udało się zapewnić systemowi odpowiednią wiedzę. Ponieważ świat bloków nie obejmował trudniejszych problemów zdroworozsądkowego rozumowania, takich jak zrozumienie czasu, przyczynowości, możliwości czy przekonań, techniki przedstawiania tej wiedzy były stosunkowo proste. Pomimo swojej ograniczonej dziedziny, SHRDLU dostarczył model integracji składni i semantyki oraz wykazał, że program z wystarczającą znajomością dziedziny dyskursu może komunikować się w sposób znaczący w języku naturalnym. Uzupełnieniem opisanego właśnie składnika rozumienia języka wymagającego dużej wiedzy jest modelowanie wzorców i oczekiwań samych wyrażeń językowych. Łańcuchy Markowa oferują potężne narzędzie do wychwytywania tych prawidłowości. Na przykład w używaniu języka przedimki i przymiotniki zazwyczaj poprzedzają rzeczowniki, a nie następują po nich, a niektóre rzeczowniki i czasowniki występują razem. Modele Markowa mogą również przechwytywać relacje między wzorcami językowymi a światami, które opisują. W sekcji 15.1 przedstawiamy analizę wysokiego poziomu rozumienia języka. Sekcja 15.2 przedstawia analizę składniową; sekcja 15.3 łączy składnię i semantykę przy użyciu rozszerzonego analizowania sieci przejść. Sekcja 15.4 przedstawia stochastyczne podejście do wychwytywania prawidłowości w wyrażeniach językowych. Wreszcie, w sekcji 15.5 rozważymy kilka zastosowań, w których przydatne są programy do rozumienia języka naturalnego: odpowiadanie na pytania, dostęp do informacji w bazach danych, zapytania internetowe i podsumowanie tekstu.

15.1 Dekonstruowanie języka: analiza

Język jest skomplikowanym zjawiskiem, obejmującym procesy tak zróżnicowane, jak rozpoznawanie dźwięków lub drukowanych liter, analiza składniowa, wysokiego poziomu wnioskowania semantyczne,

a nawet przekazywanie treści emocjonalnych poprzez rytm i fleksję. Aby sobie z tym poradzić złożoność, lingwiści opisali różne poziomy analizy języka naturalnego:

1. Prozodia zajmuje się rytmem i intonacją języka. Ten poziom analizy jest trudny do sformalizowania i często zaniedbywany; jednak jego znaczenie jest ewidentne w potężnym wpływie poezji lub pieśni religijnych, a także w roli odgrywanej przez rytm w dziecięcej zabawce słownej i gaworzeniu niemowląt.

2. Fonologia bada dźwięki, które są łączone w język. Ta gałąź językoznawstwa jest ważna dla komputerowego rozpoznawania i generowania mowy.

3. Morfologia dotyczy składników (morfemów), z których składają się słowa. Obejmują one zasady rządzące tworzeniem słów, takie jak wpływ przedrostków (un-, non-, anti- itp.) I sufiksów (-ing, -ly itp.), Które modyfikują znaczenie słów rdzeniowych. Analiza morfologiczna jest ważna przy określaniu roli słowa w zdaniu, w tym jego czasu, liczby i części mowy.

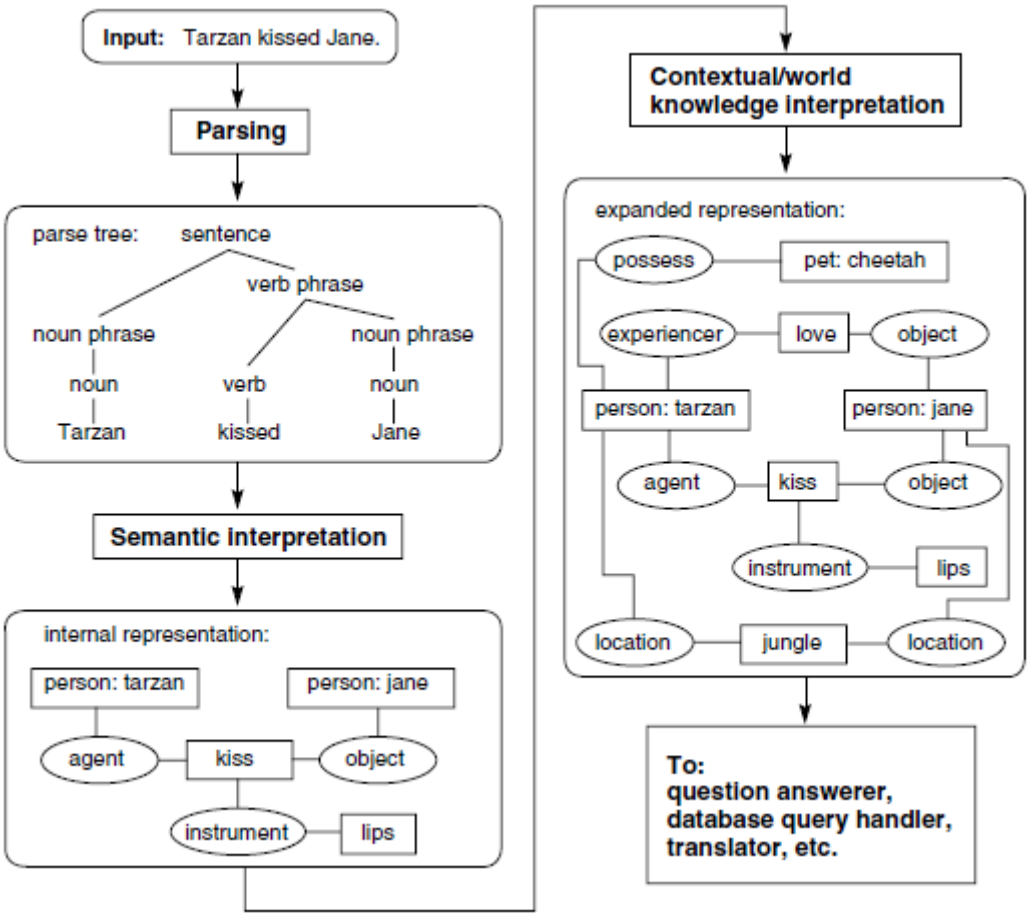
4. Składnia bada zasady łączenia słów w frazy prawne i zdania oraz wykorzystanie tych reguł do analizowania i generowania zdań. Jest to najlepiej sformalizowany, a tym samym najskuteczniejszy zautomatyzowany element analizy językowej.

5. Semantyka rozważa znaczenie słów, fraz i zdań oraz sposoby przekazywania znaczenia w wyrażeniach języka naturalnego.

6. Pragmatyka to badanie sposobów używania języka i jego wpływu na słuchacza. Na przykład pragmatyka odnosi się do przyczyny, dla której „Tak" jest zwykle niewłaściwą odpowiedzią na pytanie „Czy masz zegarek?"

7. Wiedza o świecie obejmuje wiedzę o świecie fizycznym, świecie ludzkich interakcji społecznych oraz roli celów i intencji w komunikacji. Ta ogólna wiedza jest niezbędna do zrozumienia pełnego znaczenia tekstu lub rozmowy.

Chociaż te poziomy analizy wydają się naturalne i są poparte dowodami psychologicznymi, są one do pewnego stopnia sztucznymi podziałami, które zostały narzucone językowi. Wszystkie te elementy intensywnie oddziałują na siebie, a nawet niskie intonacje i wariacje rytmiczne mają wpływ na znaczenie wypowiedzi, na przykład użycie sarkazmu. Ta interakcja jest ewidentna w związku między składnią a semantyką i chociaż pewien podział wzdłuż tych linii wydaje się niezbędny, dokładna granica jest trudna do scharakteryzowania. Na przykład zdania takie jak „Oni jedzą jabłka" mają wiele analiz, rozwiązane tylko przez zwrócenie uwagi na znaczenie w kontekście. Składnia wpływa również na semantykę, o czym świadczy rola struktury frazy w interpretacji znaczenia zdania. Chociaż dokładna natura rozróżnienia między składnią a semantyką jest często dyskutowana, zarówno dowody psychologiczne, jak i ich użyteczność w zarządzaniu złożonością problemu przemawiają za jego zachowaniem. Te głębsze kwestie dotyczące rozumienia i interpretacji języka omówimy ponownie w rozdziale 16. Chociaż specyficzna organizacja programów rozumienia języka naturalnego różni się w zależności od różnych filozofii i zastosowań - np. Interfejs dla bazy danych, system automatycznego tłumaczenia, program do rozumienia historii - wszyscy muszą przetłumaczyć oryginał zdania w wewnętrzną reprezentację jego znaczenia. Zasadniczo rozumienie języka naturalnego w oparciu o symbole przebiega według etapów przedstawionych na rysunku 2.



Pierwszym etapem jest parsowanie, które polega na analizie składniowej struktury zdań. Przetwarzanie obu weryfikuje, czy zdania są poprawnie sformułowane pod względem składniowym, a także określa strukturę językową. Identyfikując główne relacje językowe, takie jak podmiot – czasownik, czasownik – dopełnienie i rzeczownik – modyfikator, analizator składni zapewnia ramy dla interpretacji semantycznej. Jest to często przedstawiane jako drzewo parsowania. Parser języka wykorzystuje znajomość składni języka, morfologii i pewnej semantyki. Drugi etap to interpretacja semantyczna, w wyniku której powstaje przedstawienie znaczenia tekstu. Na rysunku 15.2 przedstawiono to jako wykres koncepcyjny. Inne powszechnie używane reprezentacje obejmują zależności koncepcyjne, ramki i reprezentacje oparte na logice.

Interpretacja semantyczna wykorzystuje wiedzę o znaczeniu słów i strukturze językowej, np. Rolach przypadków rzeczowników czy przechodniości czasowników. Na rysunku 2 program wykorzystał znajomość znaczenia pocałunku, aby dodać domyślną wartość ust dla instrumentu całowania. Na tym etapie wykonywane są również kontrole spójności semantycznej. Na przykład definicja czasownika pocałunek może zawierać ograniczenia, że przedmiotem jest osoba, jeśli agent jest osobą, to znaczy Tarzan całuje Jane i (normalnie) nie całuje Cheetah. Na trzecim etapie struktury z bazy wiedzy są dodawane do wewnętrznej reprezentacji zdania, aby uzyskać rozszerzoną reprezentację znaczenia zdania. To dodaje niezbędnej wiedzy o świecie wymaganej do pełnego zrozumienia, na przykład faktów, że Tarzan kocha Jane, że Jane i Tarzan żyją w dżungli oraz że Cheetah jest zwierzęciem Tarzana. Ta wynikowa struktura przedstawia znaczenie tekstu w języku naturalnym i jest wykorzystywana przez system do dalszego przetwarzania. Na przykład w interfejsie bazy danych rozszerzona struktura łączyłaby reprezentację znaczenia zapytania z wiedzą o organizacji bazy danych. Można to następnie przetłumaczyć na odpowiednie zapytanie w języku bazy danych. W programie rozumienia historii ta

rozszerzona struktura reprezentowałaby znaczenie historii i służyłaby do udzielania odpowiedzi na jej pytania (zobacz omówienie skryptów w rozdziale 7 i podsumowanie tekstu w sekcji 15.5.3). Etapy te istnieją w większości (nie probabilistycznych) systemów rozumienia języka naturalnego, chociaż mogą, ale nie muszą, odpowiadać odrębnym modułom oprogramowania. Na przykład wiele programów nie tworzy jawnego drzewa analizy, ale bezpośrednio generuje wewnętrzną reprezentację semantyczną. Niemniej jednak drzewo jest implicite w analizie zdania. Parsowanie przyrostowe (Allen 1987) jest powszechnie stosowaną techniką, w której fragment reprezentacji wewnętrznej jest tworzony, gdy tylko zostanie przeanalizowana znaczna część zdania. Te fragmenty są łączone w kompletną strukturę w miarę postępu analizy. Te fragmenty są również używane do rozwiązywania niejednoznaczności i kierowania parserem.

15.2 Składnia

15.2.1 Specyfikacja i analiza z użyciem gramatyki bezkontekstowej

Część 3 wprowadził użycie reguł przepisywania w celu określenia gramatyki. Poniższe zasady definiują gramatykę prostych zdań przechodnich, takich jak „Mężczyzna lubi psa”. Zasady są ponumerowane w celach informacyjnych.

1. sentence

↔ noun_phrase verb_phrase

2. noun_phrase

↔ rzeczownik

3. noun_phrase

↔ artykuł rzeczownik

4. verb_phrase

↔ czasownik

5. verb_phrase

↔ czasownik noun_phrase

6. artykuł

↔ a

7. artykuł

↔

8. rzeczownik

↔ człowiek

9. rzeczownik

↔ pies

10. czasownik

↔ polubień

11. czasownik

↔ ukąszenia

Reguły od 6 do 11 mają angielskie słowa po prawej stronie; reguły te tworzą słownik słów, które mogą występować w zdaniach. Te słowa są końcami gramatyki i określają leksykon języka. Terminy opisujące pojęcia lingwistyczne wyższego poziomu (zdanie, fraza_ rzeczownika itp.) Nazywane są nieterminalami. Nieterminale pojawiają się w tym kroju. Zwróć uwagę, że terminale nie pojawiają się po lewej stronie żadnej reguły. Zdaniem prawnym jest dowolny ciąg terminali, które można wyprowadzić za pomocą tych reguł. Derywacja zaczyna się od nieterminalnego zdania symbolu i tworzy ciąg terminali poprzez serię podstawień określonych regułami gramatyki. Legalna zmiana zastępuje symbol, który pasuje do lewej strony reguły, na symbole po prawej stronie tej reguły. Na pośrednich etapach wyprowadzania łańcuch może zawierać zarówno terminale, jak i nieterminale i jest nazywany formą zdaniową. Wyprowadzenie zdania „Mężczyzna gryzie psa" daje:

STRING: ZASTOSUJ REGUŁĘ NR

sentence 1

noun_phrase verb_phrase 3

article noun verb_phrase 7

The noun verb_phrase 8

The man verb_phrase 5

The man verb noun_phrase 11

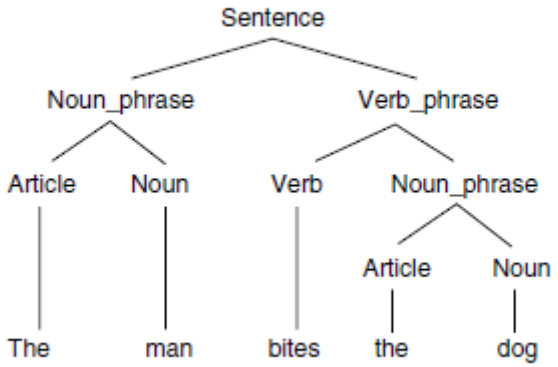
The man bites noun_phrase 3

The man bites article noun 7

The man bites the noun 9

The man bites the dog

To jest przykład wyprowadzenia odgórnego: zaczyna się od symbolu zdania i przechodzi do ciągu terminali. Derywacja oddolna zaczyna się od ciągu terminali i zastępuje wzorce po prawej stronie wzorami z lewej strony, kończąc się, gdy pozostaje tylko symbol zdania. Wyprowadzenie można przedstawić jako strukturę drzewa, znaną jako drzewo analizy, w której każdy węzeł jest symbolem z zestawu reguł gramatyki. Węzły wewnętrzne drzewa są nieterminalami; każdy węzeł i jego dzieci odpowiadają, odpowiednio, lewej i prawej stronie reguły w gramatyce. Węzły liści to terminale, a symbol zdania to korzeń drzewa. Drzewo parsowania „Mężczyzna gryzie psa" pokazano na rysunku 3.



Istnienie drzewa derywacji lub parsowania nie tylko świadczy o tym, że zdanie jest legalne w gramatyce, ale także określa jego strukturę. Struktura fraz w gramatyce definiuje głębszą organizację językową języka. Na przykład rozbicie zdania na noun_phrase i verb_phrase określa relację między czynnością a jej agentem. Ta struktura frazy odgrywa zasadniczą rolę w interpretacji semantycznej, definiując etapy pośrednie w derywacji, na których może mieć miejsce przetwarzanie semantyczne. Parsowanie to problem konstruowania wyprowadzenia lub drzewa parsowania dla ciągu wejściowego z formalnej definicji gramatyki. Algorytmy analizujące dzielą się na dwie klasy: parsery odgórne, które rozpoczynają się od symbolu zdania najwyższego poziomu i próbują zbudować drzewo, którego liście pasują do zdania docelowego, oraz parsery oddolne, które rozpoczynają się od słów w zdaniu (terminale) i spróbuj znaleźć serię redukcji, które dają symbol zdania. Jedną z trudności, która może zwiększyć złożoność problemu analizy, jest określenie, która z kilku potencjalnie mających zastosowanie reguł powinna być używana na dowolnym etapie wyprowadzania. W przypadku złego wyboru parser może nie rozpoznać wyroku prawnego. Na przykład, próbując przeanalizować zdanie „Pies gryzie” w sposób oddolny, zasady 7, 9, a 11 tworzy czasownik rzeczownik rzeczownik. W tym momencie błędne zastosowanie reguły 2 spowodowałoby wytworzenie czasownika przedimek rzeczownik_phrase; nie można tego sprowadzić do symbolu zdania. Zamiast tego parser powinien zastosować regułę 3. Podobne problemy mogą wystąpić w analiza odgórna. Problem z wyborem prawidłowej reguły na dowolnym etapie parsowania jest rozwiązywany albo przez zezwolenie parserowi na ustawienie wskaźników powrotu i powrót do sytuacji problemowej, jeśli dokonano niewłaściwego wyboru, albo przez użycie funkcji look-ahead do sprawdzenia strumień wejściowy dla funkcji pomagających określić właściwą regułę do zastosowania. W obu przypadkach musimy zadbać o kontrolę złożoności wykonania, gwarantując jednocześnie poprawną analizę. Odwrotnym problemem jest generowanie lub tworzenie wyroków prawnych z wewnętrznej reprezentacji. Generowanie rozpoczyna się od przedstawienia pewnych znaczących treści (takich jak sieć semantyczna lub graf zależności pojęciowych) i konstruuje poprawne gramatycznie zdanie, które przekazuje to znaczenie. Generowanie to nie tylko odwrotność analizy; napotyka wyjątkowe trudności i wymaga oddzielnych metodologii. W materiałach pomocniczych prezentujemy bezkontekstowe i kontekstowe parsery zejścia rekursywnego. Ponieważ parsowanie jest szczególnie ważne w przetwarzaniu języków programowania, a także języka naturalnego, naukowcy opracowali szereg różnych algorytmów analizowania, w tym zarówno strategie odgórne, jak i oddolne. Chociaż pełne omówienie algorytmów analizowania wykracza poza zakres tego rozdziału, rozważymy kilka podejść do analizowania bardziej szczegółowo. W następnej sekcji pokażemy parser Earley, ważny parser wielomianowy oparty na programowaniu dynamicznym. W sekcji 15.3 wprowadzamy ograniczenia semantyczne do parserów z parserami sieci przejściowej. Te parsery nie są wystarczająco wydajne do

semantycznej analizy języka naturalnego, ale stanowią podstawę dla rozszerzonych sieci przejściowych, które okazały się użytecznym i potężnym narzędziem w pracy z językiem naturalnym.

15.2.2 The Earley Parser: programowanie dynamiczne ponownie

Programowanie dynamiczne (DP) zostało pierwotnie zaproponowane przez Richarda Bellmana i przedstawione z kilkoma przykładami. Pomysł jest prosty: rozwiązując złożony problem, który można podzielić na wiele powiązanych ze sobą podproblemów, należy zapisać generowane rozwiązania częściowe, aby można je było ponownie wykorzystać w ciągłym procesie rozwiązywania. To podejście jest czasami nazywane zapamiętywaniem wyników podproblemu w celu ponownego wykorzystania. Istnieje wiele przykładów programowania dynamicznego w dopasowywaniu wzorców, na przykład przy określaniu miary różnicy między dwoma ciągami bitów lub znaków. Ogólna różnica między strunami będzie funkcją różnic między ich określonymi składnikami. Przykładem tego jest moduł sprawdzania pisowni sugerujący słowa, które są „bliskie" błędnie napisanego słowa. Innym przykładem DP jest rozpoznawanie słów w rozumieniu mowy jako funkcji możliwych fonemów ze strumienia wejściowego. Ponieważ fonemy są rozpoznawane (z powiązanymi z nimi prawdopodobieństwami), najbardziej odpowiednie słowo jest często funkcją połączonych połączonych miar probabilistycznych poszczególnych telefonów. W tej sekcji używamy DP do pokazania parser Earleya określającego, czy ciągi słów tworzą zdania poprawne składniowo. Nasz pseudokod został zaadaptowany z tego autorstwa Jurafsky'ego i Martina. Algorytmy analizowania z sekcji 15.2.1 są często implementowane z przeszukiwaniem rekurencyjnym, przeszukiwania najpierw w głąb i od lewej do prawej możliwych akceptowalnych struktur składniowych. To wyszukiwanie podejście może oznaczać, że wiele z możliwych do zaakceptowania częściowych analiz pierwszych (najbardziej po lewej) składników ciągu jest wielokrotnie odtwarzanych. Ta ponowna weryfikacja wczesnych rozwiązań częściowych w ramach pełnej struktury analizy jest wynikiem późniejszych wymagań dotyczących wycofywania wyszukiwanie i może stać się wykładniczo kosztowne i wymagać większych analiz. Programowanie dynamiczne zapewnia wydajną alternatywę, w której częściowe analizy, po wygenerowaniu, są zapisywane do ponownego wykorzystania w ogólnej końcowej analizie ciągu słów. Utworzono pierwszy parser oparty na DP przez Earley.

Zapamiętywanie i pary kropek

Podczas analizowania za pomocą algorytmu Earleya zapamiętywanie rozwiązań częściowych (częściowe analizy) odbywa się za pomocą struktury danych zwanej wykresem. Dlatego podejście Earley do analizy składniowej jest często nazywane analizowaniem wykresów. Wykres jest generowany za pomocą kropkowanych reguł gramatycznych. Kropkowana reguła gramatyki zapewnia reprezentację, która wskazuje na wykresie stan procesu analizowania w dowolnym momencie. Każda reguła z kropkami należy do jednej z trzech kategorii, w zależności od tego, czy pozycja kropki znajduje się na początku, gdzieś w środku, czy na końcu prawej strony reguły gramatycznej, RHS. Te trzy kategorie nazywamy odpowiednio początkowymi, częściowymi lub zakończonymi etapami analizy:

Wstępna prognoza: Symbol →. ●RHS_unseen

Częściowa analiza: Symbol → RHS_seen ●RHS_unseen

Ukończono analizę: Symbol → RHS_seen ●

Ponadto istnieje naturalna zgodność między stanami zawierającymi różne reguły z kropkami a krawędziami drzewa (-ów) analizy utworzonej przez analizę. Rozważ następującą bardzo prostą gramatykę, w której symbole terminali są otoczone cudzysłowami, jak w „mary":

Zdanie → Rzeczownik Czasownik

Rzeczownik → „mary”

Czasownik → „działa”

Gdy wykonujemy analizę z góry na dół tego zdania od lewej do prawej, powstaje następująca sekwencja stanów:

Zdanie →. ●Rzeczownik Czasownik przewidzieć: rzeczownik, po którym następuje czasownik

Rzeczownik → ● mary przewiduje: mary

Rzeczownik → mary ●zeskanowano: mary

Zdanie → Rzeczownik ●Czasownik ukończony: rzeczownik; przewidzieć: czasownik

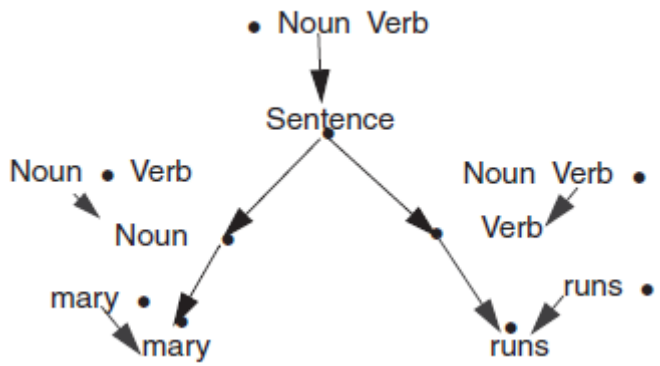
Czasownik → ●przebiegi przewidywać: przebiegi

Czasownik → działa ● skanowano: przebiegi

Zdanie → Rzeczownik Czasownik ● zakończone: Czasownik zakończone: zdanie

Zwróć uwagę, że skanowanie i wykonywanie procedur deterministycznie daje wynik. Procedura przewidywania opisuje możliwe reguły analizy, które można zastosować do bieżącej sytuacji. Skanowanie i przewidywanie tworzą stany w drzewie analizy przedstawionym na rysunku 4. Algorytm Earleya działa na zasadzie odgórnych i od lewej do prawej przewidywań dotyczących analizowania danych wejściowych. Każda prognoza jest zapisywana jako stan zawierający wszystkie istotne informacje o prognozie, gdzie kluczowym składnikiem każdego stanu jest reguła kropkowa. (Drugi komponent implementacji zostanie przedstawiony w następnej sekcji). Wszystkie predykcje generowane po zbadaniu określonego słowa z wejścia są zbiorczo nazywane zestawem stanów. Dla danego zdania wejściowego zawierającego n słów, od w1 do wn, generowanych jest łącznie n + 1 zestawów stanów: [S_0, S_1,…, S_n]. Zestaw stanu początkowego S_0 zawiera przewidywania, które są dokonywane przed sprawdzeniem jakichkolwiek słów wejściowych, S_1 zawiera prognozy wykonane po zbadaniu w_1 i tak dalej. Cały zbiór zestawów stanów nazywamy wykresem utworzonym przez parser.

Rysunek 4 ilustruje zależność między generowaniem zestawu stanów a sprawdzaniem słów wejściowych.



Chociaż tradycyjnie zestawy stanów, które składają się na każdy składnik parsowania, nazywane są zbiorami stanów, kolejność generowania tych stanów jest ważna. Dlatego każdy składnik wykresu nazwiemy listą stanów i opiszemy go jako [State1, State2,…, Staten]. Działa to również dobrze z implementacją Prologu w materiałach pomocniczych, gdzie listy stanów będą utrzymywane jako listy

Prologu. Na koniec opisujemy każdy stan z listy stanów jako sekwencję określonych symboli ujętych w nawiasy, na przykład ($ → ● S).

Rozważmy teraz algorytm Earley analizujący proste zdania wykonywane przez Mary, używając powyższej gramatyki. Algorytm rozpoczyna się od utworzenia fikcyjnego stanu początkowego ($ → ●S), czyli pierwszego elementu listy stanów S0. Stan ten reprezentuje przewidywanie, że ciąg wejściowy może być analizowany jako zdanie i jest wstawiany do S0 przed sprawdzeniem jakichkolwiek słów wejściowych. Pomyślna analiza tworzy ostateczną listę stanów Sn, która zawiera stan ($ → S ●). Zaczynając od S0, parser wykonuje pętlę, w której każdy stan Si na bieżącej liście stanów jest sprawdzany w kolejności i używany do generowania nowych stanów. Każdy nowy stan jest generowany przez jedną z trzech procedur nazywanych predyktorem, skanerem i zakończeniem. Odpowiednią procedurę określa reguła z kropkami w stanie S, a konkretnie symbol gramatyki (jeśli występuje) następujący po kropce w regule. W naszym przykładzie pierwszy badany stan zawiera regułę ($ → ● S). Ponieważ po kropce występuje symbol S, stan ten „oczekuje", że wystąpi instancja S jako następna na wejściu. Ponieważ S jest nieterminalnym symbolem gramatyki, procedura predykcyjna generuje wszystkie stany odpowiadające możliwemu parsowi S. W tym przypadku, ponieważ istnieje tylko jedna alternatywa dla S, a mianowicie, że S → Rzeczownik Czasownik, tylko jeden stan (S → ● Rzeczownik Czasownik), jest dodawany do S0. Ponieważ stan ten oczekuje części mowy, oznaczonej przez nieterminalny symbol Rzeczownik następujący po kropce, algorytm sprawdza następne słowo wejściowe, aby zweryfikować to przewidywanie. Odbywa się to przez procedurę skanera, a ponieważ następne słowo pasuje do predykcji, mary jest rzeczywiście rzeczownikiem, skaner generuje nowy stan zapisujący dopasowanie: (Rzeczownik → mary ●). Ponieważ ten stan zależy od słowa wejściowego w1, staje się pierwszym stanem w liście stanów S1, a nie dodawanie do S0. W tym miejscu wykres zawierający dwie listy stanów wygląda następująco, gdzie po każdym stanie nazywamy procedurę, która go wygenerowała:

S0: [($ → ● S), fikcyjny stan początkowy

(S → ● Rzeczownik Czasownik)] predyktor

S1: [(rzeczownik → mary ●)] Skaner

Każdy stan na liście stanów S0 został już przetworzony, więc algorytm przechodzi do S1 i rozważa stan (Rzeczownik → Mary ●). Ponieważ jest to stan ukończony, stosowana jest procedura kompletna. Dla każdego stanu oczekującego rzeczownika, to znaczy, że ma ● Wzorzec rzeczownika, element uzupełniający generuje nowy stan, który rejestruje odkrycie rzeczownika, przesuwając kropkę nad symbol rzeczownika. W tym przypadku wypełniacz tworzy stan (S → ● Rzeczownik Czasownik) w S0 i generuje nowy stan (S → Rzeczownik ●Czasownik) na liście S1. Stan ten oczekuje części mowy, co powoduje, że skaner bada następne słowo wejściowe w2. Ponieważ w2 jest czasownikiem, skaner generuje stan (czasownik → działa ●) i dodaje go do S2, co daje następujący wykres:

S0: [($ → ●S), start

(S → ● Rzeczownik Czasownik)] predyktor

S1: [(rzeczownik → mary ●), Scanner

(S → Rzeczownik ●Verb)] completeer

S2: [(Verb → run ●)] Skaner

Przetwarzając nowy stan S2, wypełniacz przesuwa kropkę w (S → Rzeczownik ● Verb) w celu wytworzenia (S → Rzeczownik Czasownik ●), Z którego kompletujący generuje stan ($ → S ●)

Oznaczający pomyślną analizę zdania. Ostateczny wykres dla biegów Mary, z trzema listami stanów, to:

S0: [($ → ● S), start

(S → ● Rzeczownik Czasownik)] predyktor

S1: [(rzeczownik → mary ●), Scanner

(S → Rzeczownik ●Verb)] completeer

S2: [(Verb → run ●)] Skaner

(S → Rzeczownik Czasownik ●), completeer

($ → S ●.)] Dopełniacz

15.2.2.2 Algorytm Earleya: pseudokod

Aby przedstawić obliczeniowo listy stanów utworzone przez powyższe reguły par z kropkami, tworzymy indeksy pokazujące, jaka część prawej strony reguły gramatycznej została przeanalizowana. Najpierw opisujemy tę reprezentację, a następnie oferujemy pseudokod do jej implementacji w algorytmie Earley. Każdy stan na liście stanów jest uzupełniony o indeks wskazujący, jak daleko przetworzono strumień wejściowy. W ten sposób rozszerzamy każdy opis stanu do reprezentacji (reguła kropkowana [i, j]), w której para [i, j] oznacza, ile prawej strony, RHS, reguły gramatyki zostało zobaczone lub przeanalizowane do chwili obecnej . Po prawej stronie przeanalizowanej reguły, która zawiera zero lub więcej widocznych i niewidocznych elementów wskazanych przez ●, mamy (A → Seen ● Unseen, [i, j]), gdzie i jest początkiem Seen, a j jest pozycją ● w sekwencji słów. Teraz dodajemy indeksy do omówionych wcześniej stanów analizy dla zdania, które wykonuje mary: ($ →● S, [0, 0]) utworzone przez predyktor, i = j = 0, nic nie zostało przeanalizowane (Rzeczownik → mary●, [0,1]) skaner widzi w1 między indeksami słów 0 i 1 (S → Rzeczownik ● Verb, [0,1]) uzupełniający widział Rzeczownik (mary) między 0 a 1 (S → Rzeczownik Czasownik ●., [0,2]) Completeer widział zdanie S między 0 a 2 .W ten sposób wzorzec indeksowania stanu przedstawia wyniki uzyskane przez każdy z trzech generatorów stanu przy użyciu reguł kropkowych wraz z indeksem słowa wi. Podsumowując, trzy procedury generowania stanów listy stanów to: predyktor generujący stany z indeksem [j, j] przechodzący do wykresu [j], skaner uwzględniający słowo wj + 1 w celu wygenerowania stanów indeksowanych przez [j, j + 1] do wykresu [j + 1] i kompletator działający na regułach z indeksem [i, j], i <j, dodając wpis stanu do wykresu [j]. Zauważ, że stan z reguły kropkowanej [i, j] zawsze trafia do wykresu listy stanów [j]. Zatem listy stanów zawierają chart [0], ..., chart [n] dla zdania zawierającego n słów. Teraz, gdy przedstawiliśmy schemat indeksowania do reprezentacji wykresu, podajemy pseudokod dla parsera Earley.

function EARLEY-PARSE(words, grammar) returns chart

begin

chart := empty

ADDTOCHART(($ →● S, [0, 0]), chart[0]) % dummy start state

for i from 0 to LENGTH(words) do

for each state in chart[i] do

if rule_rhs(state) = … ●A … and A is not a part of speech

```
then PREDICTOR(state)
else if rule_rhs(state) = … ● L … % L is part of speech
then SCANNER(state)
else COMLETER(state) % rule_rhs = RHS .
end
procedure PREDICTOR((A → … ● B …, [i, j]))
begin
for each (B → RHS) in grammar do
ADDTOCHART((B→ ● RHS, [j, j]), chart[j])
end
procedure SCANNER((A → … ● L …, [i, j]))
begin
if (L → word[j]) is_in grammar
then ADDTOCHART((L→ word[j] ●, [j, j + 1]), chart[j + 1])
end
procedure COMPLETER((B → … ●, [j, k]))
begin
for each (A → … ● B …, [i, j]) in chart[j] do
ADDTOCHART((A→ … B ● …, [i, k]), chart[k])
end
procedure ADDTOCHART(state, state-list)
begin
if state is not in state-list
then ADDTOEND(state, state-list)
end
```

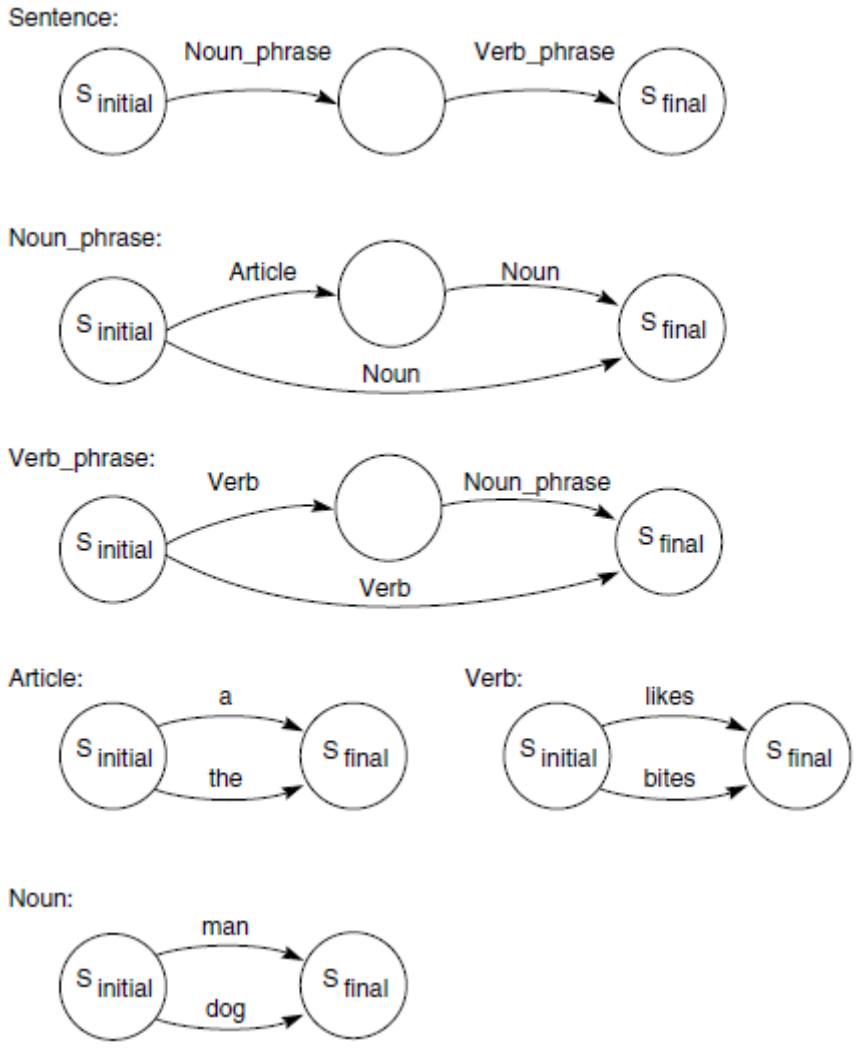
Nasz pierwszy przykład, analiza Earley zdania, które prowadzi Mary, miała być prosta, ale ilustracyjna, z bardzo szczegółową prezentacją list stanów i ich indeksów. bardziej złożone - i niejednoznaczne - zdanie, John zwany Mary z Denver jest przedstawiony w materiałach pomocniczych wraz z kodem do implementacji parsowania przy użyciu algorytmu Earley zarówno w języku Prolog, jak i Java. Dwuznaczność tego zdania znajduje odzwierciedlenie w dwóch różnych analizach, które odzwierciedlają tę niejednoznaczność. Wyboru jednej z tych analiz dokonuje się przez uruchomienie wstecznego składnika parsera Earleya opartego na programowaniu dynamicznym.

15.3 Parsery i semantyka sieci przejścia

Do tej pory zaoferowaliśmy reprezentacje i algorytmy, które wspierają syntaktyczną analizę składniową opartą na symbolach. Analiza relacji składniowych, nawet jeśli ogranicza się do analizy kontekstowej (np. Zgodność rzeczownik-czasownik), nie uwzględnia relacji semantycznych. W tej sekcji przedstawiamy sieci przejściowe, aby rozwiązać ten problem.

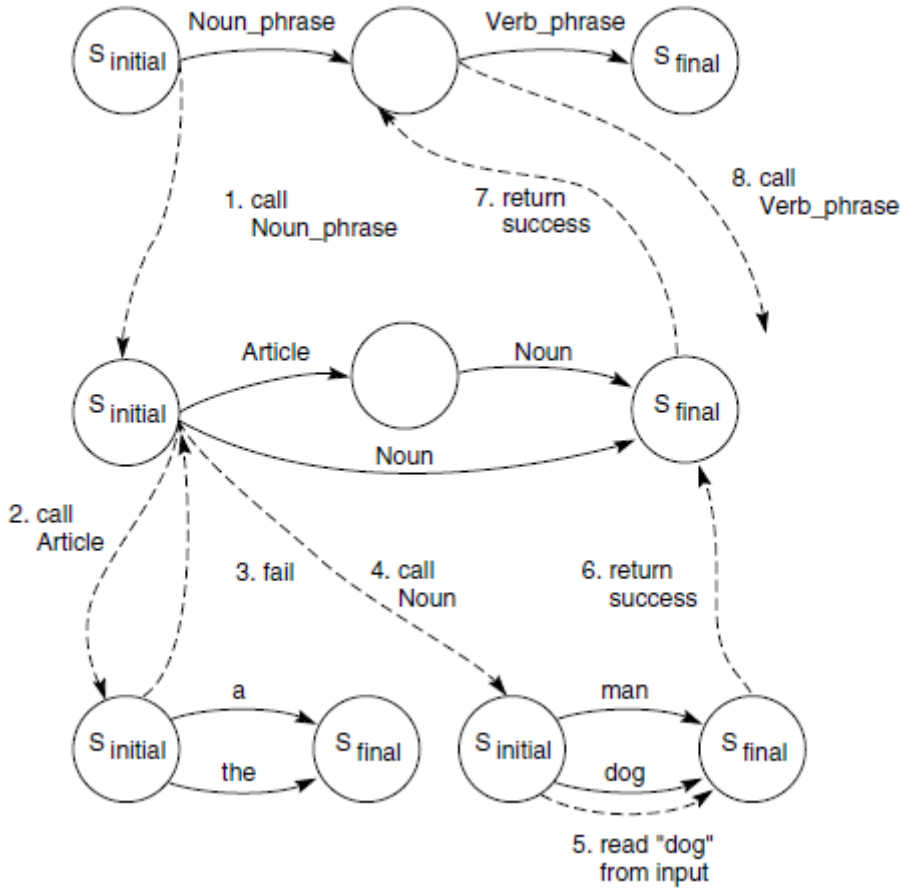
15.3.1 Parsery sieci przejściowej

Parser sieci przejściowej reprezentuje gramatykę jako zbiór maszyn skończonych lub sieci przejściowych. Każda sieć odpowiada pojedynczemu nieterminalowi w gramatyce. Łuki są oznaczone symbolami końcowymi lub nieterminalnymi. Każda ścieżka w sieci, stan początkowy do stanu końcowego, odpowiada pewnej regule dla tego nieterminala; sekwencja etykiet łuków na ścieżce to sekwencja symboli po prawej stronie reguły. Gramatyka sekcji 15.2.1 jest reprezentowana przez sieci z rysunku 5.



Gdy istnieje więcej niż jedna reguła dla nieterminala, odpowiednia sieć ma wiele ścieżek od początku do celu, np. Reguły noun_phrase ↔ noun i noun_phrase ↔ rzeczownik są przechwytywane przez alternatywne ścieżki w sieci noun_phrase. Znalezienie udanego przejścia przez sieć dla nieterminala odpowiada zastąpieniu tego nieterminala prawą stroną reguły gramatycznej. Na przykład, aby

przeanalizować zdanie, parser sieci przejściowej musi znaleźć przejście przez sieć zdań. Rozpoczyna się w stanie początkowym (Sinitial) i przyjmuje przejście noun_phrase, a następnie przejście verb_phrase, aby osiągnąć stan końcowy (Sfinal). Jest to równoważne zamianie oryginalnego symbolu zdania na noun_phrase verb_phrase. Aby przejść przez łuk, parser sprawdza jego etykietę. Jeśli etykieta jest symbolem terminala, parser sprawdza strumień wejściowy, aby zobaczyć, czy następne słowo pasuje do etykiety łuku. Jeśli nie pasuje, przejście nie może zostać wykonane. Jeśli łuk jest oznaczony nie terminalem symbol, parser wyszukuje sieć dla tego nieterminala i rekurencyjnie próbuje znaleźć ścieżkę przez nią. Jeśli parser nie znajdzie ścieżki w tej sieci, nie można przejść przez łuk najwyższego poziomu. Powoduje to, że parser cofa się i próbuje użyć innej ścieżki przez sieć. W ten sposób parser próbuje znaleźć ścieżkę w sieci zdań; jeśli się powiedzie, ciąg wejściowy jest zdaniem prawnym w gramatyce. Rozważmy proste zdanie Ukąszenia psa, przeanalizowane i zilustrowane na rysunku 6:



1. Parser zaczyna od sieci zdań i próbuje poruszać się po łuku oznaczonym noun_phrase. Aby to zrobić, pobiera sieć dla noun_phrase.

2. W sieci noun_phrase parser najpierw próbuje przejść do zaznaczonego artykułu. To powoduje, że rozgałęzia się do sieci w celu uzyskania artykułu.

3. Nie można znaleźć ścieżki do węzła końcowego sieci artykułów, ponieważ pierwsze słowo „Dog” nie pasuje do żadnej z etykiet łuków. Parser kończy się niepowodzeniem i wraca do sieci noun_phrase.

4. Parser próbuje podążać za łukiem oznaczonym rzeczownikiem w sieci noun_phrase i rozgałęzia się do sieci dla rzeczownika.

5. Parser pomyślnie przechodzi przez łuk oznaczony „pies”, ponieważ odpowiada to pierwszemu słowu strumienia wejściowego.

6. Sieć rzeczowników zwraca sukces. Pozwala to na przejście łuku oznaczonego rzeczownikiem w sieci noun_phrase do stanu końcowego.

7. Sieć noun_phrase zwraca sukces do sieci najwyższego poziomu, umożliwiając przejście łuku oznaczonego noun_phrase.

8. Sekwencja podobnych kroków jest wykonywana w celu przeanalizowania części zdania verb_phrase

Pseudokod parsera przejściowego znajduje się poniżej. Jest definiowany za pomocą dwóch wzajemnie rekurencyjnych funkcji, parsowania i przejścia. Parse przyjmuje symbol gramatyki jako argument: jeśli symbol jest terminalem, parse sprawdza go względem następnego słowa w strumieniu wejściowym. Jeśli jest to nieterminal, analiza składniowa pobiera sieć przejściową skojarzoną z symbolem i wywołuje przejście w celu znalezienia ścieżki w sieci. Aby przeanalizować zdanie, wywołaj parse (zdanie).

```
function parse(grammar_symbol);

begin

save pointer to current location in input stream;

case

grammar_symbol is a terminal:

if grammar_symbol matches the next word in the input stream

then return (success)

else begin

reset input stream;

return (failure)

end;

grammar_symbol is a nonterminal:

begin

retrieve the transition network labeled by grammar symbol;

state := start state of network;

if transition(state) returns success

then return (success)

else begin

reset input stream;

return (failure)

end

end
```

```
end

end.

function transition (current_state);

begin

case

current_state is a final state:

return (success)

current_state is not a final state:

while there are unexamined transitions out of current_state

do begin

grammar_symbol := the label on the next unexamined transition;

if parse(grammar_symbol) returns (success)

then begin

next_state := state at end of the transition;

if transition(next_state) returns success;

then return (success)

end

end

return (failure)

end

end.
```

Przejście przyjmuje stan w sieci przejściowej jako argument i próbuje znaleźć ścieżkę przez tę sieć w sposób głęboki. Ponieważ parser może popełnić błąd i wymagać śledzenia wstecznego, parse zachowuje wskaźnik do bieżącej lokalizacji w strumieniu wejściowym. Pozwala to na zresetowanie strumienia wejściowego do tej lokalizacji w przypadku cofnięcia się parsera. Ten parser sieci przejściowej określa, czy zdanie jest poprawne gramatycznie, ale nie tworzy drzewa parsowania. Można to osiągnąć poprzez zwrócenie przez funkcje poddrzewa drzewa parsowania zamiast sukcesu symbolu. Modyfikacje, które mogłyby to osiągnąć, to:

1. Za każdym razem, gdy parse funkcji jest wywoływana z symbolem terminala jako argumentem i ten terminal pasuje do następnego symbolu wejścia, zwraca drzewo składające się z pojedynczego węzła-liścia oznaczonego tym symbolem.

2. Kiedy parse jest wywoływana z nieterminalnym, grammar_symbol, wywołuje przejście. Jeśli przejście się powiedzie, zwraca uporządkowany zestaw poddrzew (opisanych poniżej). Parse łączy je w drzewo, którego korzeniem jest grammar_symbol i którego dziećmi są poddrzewa zwrócone przez przejście.

3. Podczas wyszukiwania ścieżki w sieci, wywołania przejścia analizują etykiety każdego łuku. Po pomyślnym przeprowadzeniu analizy składni zwraca drzewo reprezentujące analizę tego symbolu. Przejście zapisuje te poddrzewa w uporządkowanym zestawie i po znalezieniu ścieżki w sieci zwraca uporządkowany zestaw drzew parsowania odpowiadający sekwencji etykiet łuków na ścieżce.

15.3.2 Hierarchia Chomsky'ego i gramatyki kontekstualne

W sekcji 15.3.1 zdefiniowaliśmy niewielki podzbiór języka angielskiego przy użyciu gramatyki bezkontekstowej. Gramatyka bezkontekstowa pozwala regułom mieć tylko jeden nieterminal po lewej stronie. W konsekwencji regułę można zastosować do każdego wystąpienia tego symbolu, niezależnie od kontekstu. Chociaż gramatyki bezkontekstowe okazały się potężnym narzędziem do definiowania języków programowania i innych formalizmów w informatyce, istnieją powody, by sądzić, że same w sobie nie są wystarczająco potężne, aby reprezentować reguły składni języka naturalnego. Na przykład zastanówmy się, co się stanie, jeśli dodamy liczbę pojedynczą oraz rzeczowniki i czasowniki w liczbie mnogiej do gramatyki sekcji 15.2.1:

rzeczownik ↔ mężczyzn

rzeczownik ↔ psy

czasownik ↔ ukąszenia

czasownik ↔ jak

Wynikowa gramatyka przeanalizuje zdania, takie jak „Psy lubią mężczyzn”, ale akceptuje również „A mężczyzna gryzie psy”. Parser akceptuje te zdania, ponieważ reguły nie używają kontekstu do określenia, kiedy należy skoordynować liczbę pojedynczą i mnogą. Reguła definiująca zdanie jako noun_phrase, po którym następuje verb_phrase, nie wymaga, aby rzeczownik i czasownik zgadzały się co do liczby lub żeby przedimki zgadzały się z rzeczownikami. Języki wolne od Conext można rozszerzyć, aby poradzić sobie z takimi sytuacjami, ale bardziej naturalne podejście polega na tym, że grammer jest wrażliwy na kontekst, gdzie komponenty drzewa parsowania są zaprojektowane tak, aby wzajemnie się ograniczać. Chomsky (1965) jako pierwszy zaproponował świat hierarchicznych i coraz potężniejszych gramofonów (Hopcroft i Ullman 1979). Na dole tej hierarchii znajduje się klasa języków regularnych, których gramatykę można zdefiniować za pomocą maszyny skończonej, sekcja 3.1. Zwykłe języki mają wiele zastosowań w informatyce, ale nie są wystarczająco potężne, aby reprezentować składnię większości języków programowania. Języki bezkontekstowe znajdują się ponad zwykłymi językami w hierarchii Chomsky'ego. Języki bezkontekstowe są definiowane za pomocą reguł przepisywania, takich jak w sekcji 15.2.1; reguły bezkontekstowe mogą mieć tylko jeden symbol nieterminalny po lewej stronie. Parsery sieci przejściowej są w stanie przeanalizować klasę języków bezkontekstowych. Warto zauważyć, że jeśli nie pozwolimy na rekurencję w parserze sieci przejściowej, tj. Łuki mogą być etykietowane tylko symbolami terminala, a przejścia nie mogą „wywoływać” innej sieci, to klasa języków, które mogą być tak zdefiniowane, odpowiada zwykłej wyrażenia. Tak więc języki regularne stanowią właściwy podzbiór języków bezkontekstowych. Języki kontekstowe stanowią właściwy nadzbiór języków bezkontekstowych. Są one definiowane za pomocą gramatyki kontekstowej, która zezwala na więcej niż jeden symbol po lewej stronie reguły i umożliwia zdefiniowanie kontekstu, w którym można zastosować tę regułę. Zapewnia to spełnienie globalnych ograniczeń, takich jak zgodność liczby i inne kontrole semantyczne. Jedynym ograniczeniem reguł gramatycznych zależnych od kontekstu jest to, że prawa strona jest co najmniej tak długa, jak lewa (Hopcroft i Ullman 1979). Czwarta klasa, tworząca nadzbiór języków zależnych od kontekstu, to klasa języków rekurencyjnie wyliczalnych. Rekurencyjnie wyliczalne języki mogą być definiowane przy użyciu nieograniczonych reguł produkcji; ponieważ te reguły są mniej ograniczone niż reguły kontekstowe,

rekurencyjnie wyliczalne języki są właściwym nadzbiorem języków kontekstowych. Ta klasa nie jest interesująca w definiowaniu składni języka naturalnego, chociaż jest ważna w teorii informatyki. Pozostała część tej sekcji skupia się na języku angielskim jako języku kontekstowym. Prosta gramatyka bezkontekstowa dla zdań z formy rzeczownik czasownik, która wymusza zgodność liczbową między przedimkiem i rzeczownikiem oraz podmiotem i czasownikiem, jest dana przez:

zdanie ↔ noun_phrase verb_phrase

noun_phrase ↔ numer artykułu rzeczownik

noun_phrase ↔ liczba rzeczownik

liczba ↔ liczba pojedyncza

liczba ↔ liczba mnoga

artykuł liczba pojedyncza ↔ a liczba pojedyncza

artykuł liczba pojedyncza ↔ liczba pojedyncza

artykuł liczba mnoga ↔ pewna liczba mnoga

artykuł w liczbie mnogiej ↔ liczba mnoga

rzeczownik w liczbie pojedynczej ↔ pies w liczbie pojedynczej

rzeczownik w liczbie pojedynczej ↔ człowiek w liczbie pojedynczej

rzeczownik ↔ mężczyźni liczba mnoga

rzeczownik w liczbie mnogiej ↔ psy w liczbie mnogiej

czasownik w liczbie pojedynczej_fraza ↔ czasownik w liczbie pojedynczej

czasownik w liczbie mnogiej_fraza ↔ czasownik w liczbie mnogiej

czasownik w liczbie pojedynczej ↔ ugryzienia

czasownik w liczbie pojedynczej ↔ lubi

czasownik w liczbie mnogiej ↔ zgryz

czasownik w liczbie mnogiej ↔ podobny

W tej gramatyce nieterminale liczby pojedynczej i mnogiej oferują ograniczenia do określenia, kiedy można zastosować różne reguły przedimka, rzeczownika i czasownika_fraza, zapewniając zgodność liczb. Wyprowadzenie zdania „Psy gryzą” przy użyciu tej gramatyki jest podane przez:

zdanie.

noun_phrase verb_phrase.

artykuł liczba mnoga rzeczownik czasownik_fraza.

Rzeczownik w liczbie mnogiej verb_phrase.

Psy w liczbie mnogiej verb_phrase.

Psy w liczbie mnogiej.

Psy gryzą.

Podobnie, możemy użyć gramatyki kontekstowej do sprawdzania zgodności semantycznej. Na przykład moglibyśmy zabronić wykonywania zdań, takich jak „Mężczyzna gryzie psa", dodając do gramatyki nieterminal „act_of_biting". Ten nieterminal można by zaznaczyć w regułach, aby zapobiec sytuacji, w której w zdaniu zawierającym „ugryzienia" słowo „człowiek" jest przedmiotem.

Chociaż gramatyki kontekstowe mogą definiować struktury językowe, których nie można uchwycić za pomocą gramatyk bezkontekstowych, mają szereg wad przy projektowaniu praktycznych parserów:

1. Gramatyki kontekstowe drastycznie zwiększają liczbę reguł i nieterminali w gramatyce. Wyobraź sobie złożoność gramatyki kontekstowej, która zawierałaby liczbę, osobę i wszystkie inne formy porozumienia wymagane w języku angielskim.

2. Zasłaniają strukturę wyrażeń języka, który jest tak jasno przedstawiony w zasadach bezkontekstowych.

3. Próbując poradzić sobie z bardziej skomplikowanymi sprawdzeniami zgodności i spójności semantycznej w samej gramatyce, tracą wiele korzyści z oddzielenia składniowych i semantycznych składników języka.

4. Gramatyki kontekstualne nie rozwiązują problemu budowania semantycznej reprezentacji znaczenia tekstu. Parser, który po prostu akceptuje lub odrzuca zdania, nie jest wystarczający; musi zwracać użyteczną reprezentację znaczenia semantycznego zdania.

W materiałach pomocniczych do tej książki przedstawiamy zarówno parsery bezkontekstowe, jak i kontekstowe, a także generatory zdań. Reżim kontroli dla tych programów Prologu to rekurencyjne zejście najpierw w głąb. Następnie przeanalizujemy rozszerzone sieci przejściowe (ATN), rozszerzenie sieci przejściowych, które mogą definiować języki kontekstowe, ale ma kilka zalet w porównaniu z gramatyką kontekstową w projektowaniu parserów.

15.3.3 Semantyka: parsery sieci rozszerzonego przejścia

Alternatywą dla gramatyki kontekstualnej jest zachowanie prostszej struktury bezkontekstowych reguł gramatycznych, ale rozszerzenie tych reguł o dołączone procedury, które wykonują niezbędne testy kontekstowe. Te procedury są wykonywane, gdy reguła jest wywoływana podczas analizowania. Zamiast używać gramatyki do opisu takich pojęć, jak liczba, czas i osoba, przedstawiamy je jako cechy dołączone do terminali i nieterminali gramatyki. Procedury dołączone do reguł gramatyki mają dostęp do tych funkcji w celu przypisania wartości i wykonania niezbędnych testów. Gramatyki, które wykorzystują rozszerzenia gramatyk bezkontekstowych do implementacji wrażliwości kontekstowej, obejmują gramatykę rozszerzonej struktury fraz, rozszerzenia gramatyki logicznej i rozszerzoną sieć przejść (ATN). W tej sekcji przedstawiamy parsowanie ATN i zarys projektu prostego parsera ATN dla zdań dotyczących „świata psów" przedstawionych w sekcji 15.2.1. Zajmujemy się pierwszymi dwoma krokami przedstawionymi na rysunku 2: utworzeniem drzewa parsowania i jego wykorzystaniem do skonstruowania reprezentacji znaczenia zdania. W tym przykładzie używamy grafów koncepcyjnych, chociaż parsery ATN mogą być również używane z reprezentacjami opartymi na sieci semantycznej, skrypcie, ramce lub logice. Rozszerzone sieci przejściowe rozszerzają sieci przejściowe, umożliwiając dołączanie procedur do łuków sieci. Parser ATN wykonuje te dołączone procedury podczas przechodzenia przez łuki. Procedury mogą przypisywać wartości cechom gramatycznym i wykonywać testy, powodując niepowodzenie przejścia, jeśli określone warunki (takie jak zgodność liczb) nie są spełnione. Te procedury również konstruują drzewo parsowania, które jest używane do generowania wewnętrznej semantycznej reprezentacji znaczenia zdania. Reprezentujemy zarówno terminale, jak i

nieterminale jako identyfikatory (np. Czasownik, fraza_ rzeczownika) z dołączonymi funkcjami. Na przykład słowo jest opisywane za pomocą jego korzenia morfologicznego, wraz z cechami jego części mowy, liczby, osoby itp. Nieterminale w gramatyce są podobnie opisane. Fraza rzeczownikowa jest opisywana przez rodzajnik, rzeczownik, liczbę i osobę. Zarówno terminale, jak i nieterminale mogą być reprezentowane za pomocą struktur przypominających ramkę z nazwanymi gniazdami i wartościami. Wartości tych szczelin określają cechy gramatyczne lub wskaźniki do innych struktur. Na przykład pierwsza sekcja ramki zdania zawiera wskaźnik do definicji wyrażenia rzeczownikowego. Rysunek 7 pokazuje ramki dla nieterminali zdania, noun_phrase i verb_phrase w naszej prostej gramatyce.

Sentence
Noun phrase:
Verb phrase:

Noun phrase
Determiner:
Noun:
Number:

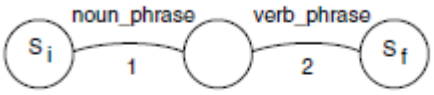
Verb phrase
Verb:
Number:
Object:

Poszczególne słowa są przedstawiane za pomocą podobnych struktur. Każde słowo w słowniku jest zdefiniowane ramką, która określa jego część mowy (rodzajnik, rzeczownik itp.), Jego rdzeń morfologiczny i istotne cechy gramatyczne. W naszym przykładzie sprawdzamy tylko zgodność numeru i rejestrujemy tylko tę funkcję. Bardziej wyrafinowane gramatyki wskazują osobę i inne cechy. Te hasła słownikowe mogą również wskazywać na koncepcyjną definicję grafową znaczenia słowa do użycia w interpretacji semantycznej. Kompletny słownik naszej gramatyki przedstawiono na rysunku 8.

Word	Definition
a	PART_OF_SPEECH: article ROOT: a NUMBER: singular
bite	PART_OF_SPEECH: verb ROOT: bite NUMBER: plural
bites	PART_OF_SPEECH: verb ROOT: bite NUMBER: singular
dog	PART_OF_SPEECH: noun ROOT: dog NUMBER: singular
dogs	PART_OF_SPEECH: noun ROOT: dog NUMBER: plural
like	PART_OF_SPEECH: verb ROOT: like NUMBER: plural
likes	PART_OF_SPEECH: verb ROOT: like NUMBER: singular
man	PART_OF_SPEECH: noun ROOT: man NUMBER: singular
men	PART_OF_SPEECH: noun ROOT: man NUMBER: plural
the	PART_OF_SPEECH: article ROOT: the NUMBER: plural or singular

Rysunek 9 przedstawia ATN dla naszej gramatyki, z pseudokodowymi opisami testów wykonanych na każdym łuku.

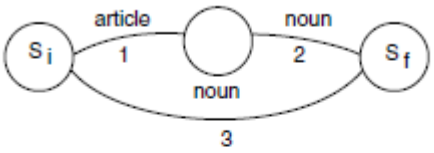
sentence:




```
function sentence-1;
  begin
    NOUN_PHRASE := structure returned by
          noun_phrase network;

    SENTENCE.SUBJECT := NOUN_PHRASE;
  end.
```

```
function sentence-2;
  begin
    VERB_PHRASE := structure returned by
          verb_phrase network;

    if NOUN_PHRASE.NUMBER =
                              VERB_PHRASE.NUMBER
      then begin
        SENTENCE.VERB_PHRASE := VERB_PHRASE;
        return SENTENCE
      end
      else fail
  end.
```

noun_phrase:

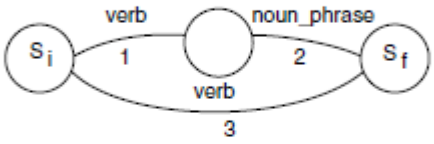



```
function noun_phrase-1;
  begin
    ARTICLE := definition frame for next word of input;

    if ARTICLE.PART_OF_SPEECH=article
      then NOUN_PHRASE.DETERMINER := ARTICLE
      else fail
  end.
```

```
function noun_phrase-2;
  begin
    NOUN := definition frame for next word of input;

    if NOUN.PART_OF_SPEECH=noun and
          NOUN.NUMBER agrees with
              NOUN_PHRASE.DETERMINER.NUMBER
      then begin
        NOUN_PHRASE.NOUN := NOUN;
        NOUN_PHRASE.NUMBER := NOUN.NUMBER
        return NOUN_PHRASE
      end
      else fail
  end.
```

```
function noun_phrase-3
  begin
    NOUN := definition frame for next word of input;

    if NOUN.PART_OF_SPEECH=noun
      then begin
        NOUN_PHRASE.DETERMINER := unspecified;
        NOUN_PHRASE.NOUN := NOUN
        NOUN_PHRASE.NUMBER := NOUN.NUMBER
      end
      else fail
  end.
```

verb_phrase:



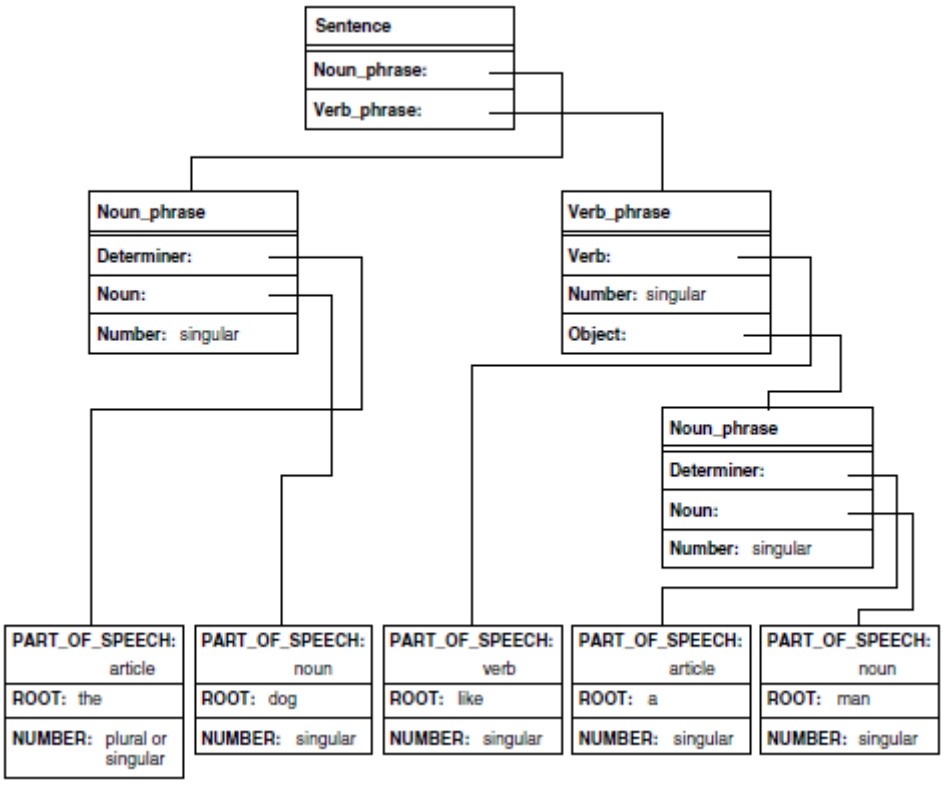

```
function verb_phrase-1
  begin
    VERB := definition frame for next word of input;

    if VERB.PART_OF_SPEECH=verb
      then begin
        VERB_PHRASE.VERB := VERB;
        VERB_PHRASE.NUMBER := VERB.NUMBER;
      end;
  end.
```

```
function verb_phrase-2
  begin
    NOUN_PHRASE := structure returned by
                                  noun_phrase network;

    VERB_PHRASE.OBJECT := NOUN_PHRASE;
    return VERB_PHRASE
  end.
```

```
function verb_phrase-3
  begin
    VERB := definition frame for next word of input;

    if VERB.PART_OF_SPEECH=verb
      then begin
        VERB_PHRASE.VERB := VERB;
        VERB_PHRASE.NUMBER := VERB.NUMBER;
        VERB_PHRASE.OBJECT := unspecified;
        return VERB_PHRASE;
      end;
  end.
```

Łuki są oznaczone zarówno nieterminalami gramatyki, jak i liczbami; liczby te służą do wskazania funkcji przypisanej do każdego łuku. Funkcje te muszą działać pomyślnie, aby przejść po łuku. Kiedy parser wywołuje sieć w celu uzyskania nieterminala, tworzy nową ramkę dla tego nieterminala. Na przykład po wejściu do sieci noun_phrase tworzy nową ramkę noun_phrase. Szczeliny w ramce są wypełnione funkcjami dla tej sieci. Szczelinom tym można przypisać wartości funkcji gramatycznych lub wskaźniki do składników struktury składniowej (np. Verb_phrase może składać się z czasownika i rzeczownik_phrase). Po osiągnięciu stanu końcowego sieć zwraca tę strukturę. Kiedy sieć przechodzi przez łuki oznaczone rzeczownikiem, przedimkiem i czasownikiem, odczytuje następne słowo ze strumienia wejściowego i pobiera definicję tego słowa ze słownika. Jeśli słowo nie jest oczekiwaną częścią mowy, reguła zawodzi; w przeciwnym razie ramką definicji jest zwrócony. Na rysunku 9 ramki i szczeliny są oznaczone przy użyciu notacji Frame.Slot; np. miejsce na numer ramki czasownika jest wskazywane przez VERB.NUMBER. W miarę postępu analizy każda funkcja buduje i zwraca ramkę opisującą powiązaną strukturę składniową. Ta struktura zawiera wskaźniki do struktur zwracanych przez sieci niższego poziomu. Funkcja zdania najwyższego poziomu zwraca strukturę zdania reprezentującą drzewo analizy dla danych wejściowych. Ta struktura jest przekazywana do

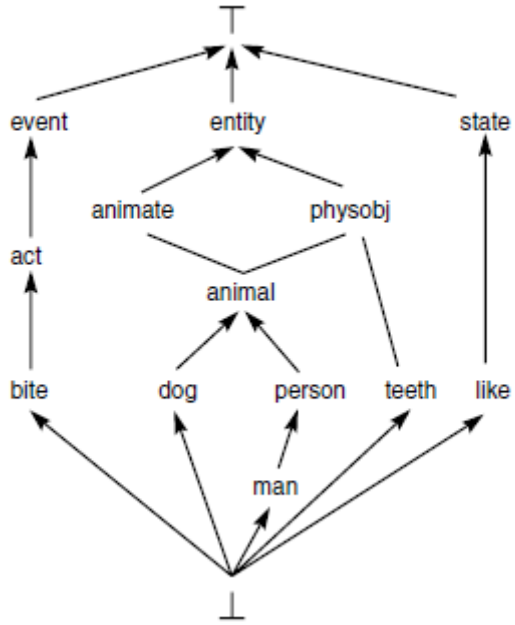
interpretera semantycznego. Rysunek 10 przedstawia drzewo analizy, które jest zwracane dla zdania „Pies lubi człowieka”.



Następna faza przetwarzania języka naturalnego obejmuje drzewo parsowania, takie jak na rysunku 10, i konstruuje semantyczną reprezentację wiedzy dziedzinowej i treści znaczeniowej zdania.

15.3.4 Łączenie wiedzy składniowej i semantycznej z sieciami ATN

Interpreter semantyczny konstruuje reprezentację znaczenia ciągu wejściowego, rozpoczynając od korzenia lub węzła zdania i przechodząc przez drzewo analizy. W każdym węźle rekurencyjnie interpretuje elementy potomne tego węzła i łączy wyniki w jeden wykres pojęciowy; ten wykres jest przekazywany w górę drzewa. Na przykład interpreter semantyczny buduje reprezentację verb_phrase, rekurencyjnie budując reprezentacje elementów potomnych węzła, czasownika i noun_phrase, i łącząc je w celu utworzenia interpretacji fraza czasownikowa. Jest to przekazywane do węzła zdań i łączone z reprezentacją podmiotu. Rekursja zatrzymuje się na terminalach drzewa parsowania. Niektóre z nich, takie jak rzeczowniki, czasowniki i przymiotniki, powodują pobieranie pojęć z bazy wiedzy. Inne, takie jak artykuły, nie odpowiadają bezpośrednio pojęciom w bazie wiedzy, ale kwalifikują inne pojęcia na wykresie. Tłumacz semantyczny w naszym przykładzie korzysta z bazy wiedzy dotyczącej „świata psów”. Pojęcia w bazie wiedzy obejmują obiekty pies i człowiek oraz czynności takie jak i gryzienie. Pojęcia te są opisane w hierarchii typów na rysunku 11.



Oprócz pojęć musimy zdefiniować relacje, które będą używane na naszych grafach koncepcyjnych. W tym przykładzie używamy następujących pojęć: agent łączy akt z pojęciem typu animate. agent definiuje relację pomiędzy akcją a animowanym obiektem, który ją wywołuje. Doświadczający łączy stan z koncepcją typu animowanego. Określa relację między stanem psychicznym a jego doświadczającym.

instrument łączy czynność z bytem i definiuje instrument używany w czynności.

obiekt łączy zdarzenie lub stan z bytem i reprezentuje relację czasownik-obiekt.

part łączy pojęcia typu physobj i definiuje relację między całością a częścią.

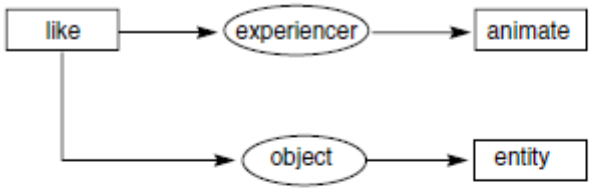
Czasownik odgrywa szczególnie ważną rolę w budowaniu interpretacji, ponieważ określa relacje między podmiotem, przedmiotem i innymi składnikami zdania. Reprezentujemy każdy czasownik za pomocą ramki wielkości liter, która określa:

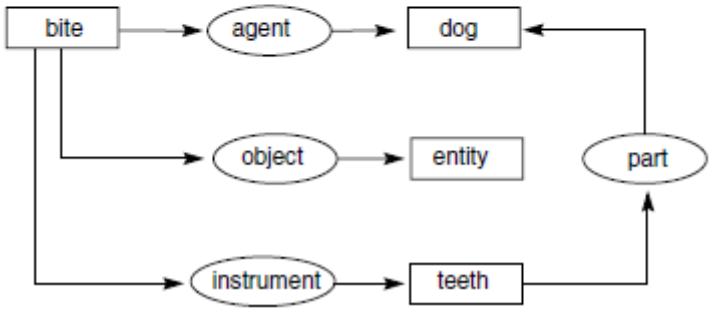
1. Relacje językowe (agent, przedmiot, instrument itd.) Właściwe dla danego czasownika. Na przykład czasowniki przechodnie mają dopełnienie; czasowniki nieprzechodnie nie.

2. Ograniczenia wartości, które mogą być przypisane do dowolnego komponentu ramy obudowy. Na przykład w ramce przypadku dla czasownika „ugryzienia” stwierdziliśmy, że agent musi być w typie pies. Powoduje to odrzucenie wyrażenia „Człowiek gryzie psa” jako niepoprawne semantycznie.

3. Wartości domyślne na elementach ramy obudowy. W ramce „zgryz” mamy domyślną wartość zębów dla pojęcia powiązanego z relacją instrumentu.

Ramki przypadków dla czasowników like i bite pojawiają się na rysunku 12.





Definiujemy działania, które budują reprezentację semantyczną z regułami lub procedurami dla każdego potencjalnego węzła w drzewie parsowania. Reguły dla naszego przykładu są opisane jako procedury pseudokodu. W każdej procedurze, jeśli określone łączenie lub inny test nie powiedzie się, interpretacja ta jest odrzucana jako semantycznie niepoprawna:

procedure sentence;

begin

call noun_phrase to get a representation of the subject;

call verb_phrase to get a representation of the verb_phrase;

using join and restrict, bind the noun concept returned for the subject to

the agent of the graph for the verb_phrase

end.

procedure noun_phrase;

begin

call noun to get a representation of the noun;

case

the article is indefinite and number singular: the noun concept is generic;

the article is definite and number singular: bind marker to noun concept;

number is plural: indicate that the noun concept is plural

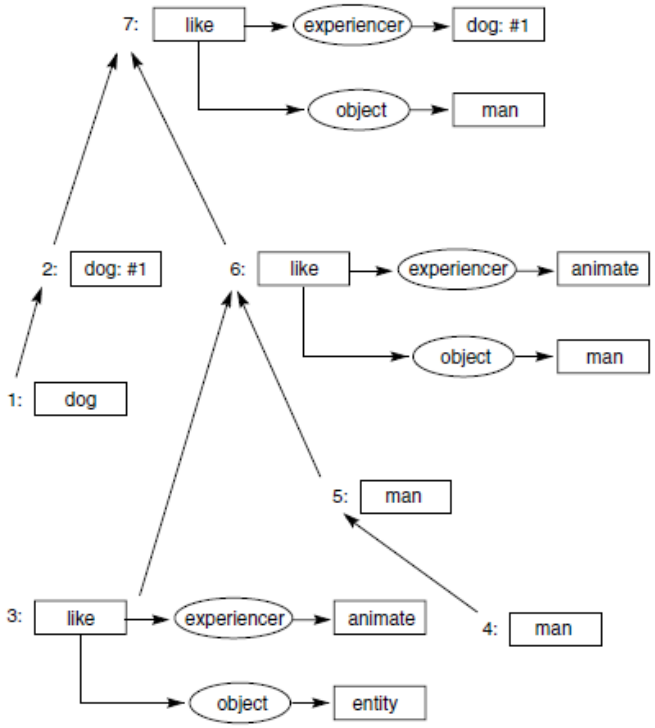
end case

end.

procedure verb_phrase;

```
begin
call verb to get a representation of the verb;
if the verb has an object
then begin
call noun_phrase to get a representation of the object;
using join and restrict, bind concept for object to object of the verb
end
end.
procedure verb;
begin
retrieve the case frame for the verb
end.
procedure noun;
begin
retrieve the concept for the noun
end.
```

Przedimki nie odpowiadają pojęciom w bazie wiedzy, ale określają, czy ich pojęcie rzeczownika jest ogólne czy specyficzne. Nie omawialiśmy reprezentacji pojęć w liczbie mnogiej; odnieś się do Sowy (1984), aby je traktować jako grafy koncepcyjne. Korzystając z tych procedur, wraz z hierarchią pojęć na rysunku 11 i ramkami przypadków z rysunku 12, śledzimy działania interpretera semantycznego w budowaniu semantycznej reprezentacji zdania „Pies lubi człowieka” z drzewa parsowania na rysunku 10 . Ten ślad pojawia się na rysunku 13. Działania podjęte w śladzie to (liczby odnoszą się do rysunku 13):



1. Zaczynając od węzła zdania, nazwij zdanie.

2. zdanie wywołuje noun_phrase.

3. noun_phrase wywołuje rzeczownik.

4. rzeczownik zwraca pojęcie psa rzeczownika (1 na rysunku 13).

5. Ponieważ przedimek jest określony, noun_phrase wiąże indywidualny znacznik z pojęciem (2) i zwraca to pojęcie do zdania.

6. zdanie wywołuje verb_phrase.

7. verb_phrase wywołuje czasownik, który pobiera ramkę wielkości liter dla like (3).

8. verb_phrase wywołuje noun_phrase, które następnie wywołuje rzeczownik, aby pobrać koncepcję man (4).

9. Ponieważ przedimek jest nieokreślony, noun_phrase pozostawia to pojęcie rodzajowe (5).

10. Procedura verb_phrase ogranicza pojęcie bytu w ramce przypadku i łączy je z pojęciem człowieka (6). Ta struktura powraca do zdania.

11. zdanie łączy koncepcję psa: # 1 z węzłem doświadczającym ramki przypadku (7).

Ten wykres pojęciowy przedstawia znaczenie zdania.

Generowanie języka jest pokrewnym problemem, którego dotyczą programy rozumienia języka naturalnego. Generowanie zdań angielskich wymaga skonstruowania semantycznie poprawnego wyniku z wewnętrznej reprezentacji znaczenia. Na przykład agent relacja wskazuje na relację podmiot-czasownik między dwoma pojęciami. Proste podejścia pozwalają na podłączenie odpowiednich słów do zapisanych szablonów zdań. Te szablony są wzorcami zdań i fragmentów, takich jak frazy rzeczownikowe i wyrażenia przyimkowe. Dane wyjściowe są konstruowane, przechodząc po grafie

pojęciowym i łącząc te fragmenty. Bardziej wyrafinowane podejścia do generowania języka wykorzystują gramatykę transformacyjną, aby odwzorować znaczenie na szereg możliwych zdań (Winograd 1972, Allen 1987). W materiałach pomocniczych do tej książki zbudowaliśmy w Prologu pełny semantyczny parser sieci z rekursywnym zstępowaniem. Ten parser używa operatorów grafów, takich jak łączenie i ograniczanie, w celu ograniczenia semantycznej reprezentacji sieci dołączonej do węzłów-liści drzewa parsowania. W części 15.5 pokażemy, jak program może budować wewnętrzne reprezentacje zjawisk językowych. Reprezentacje te są używane przez programy na wiele sposobów, w zależności od aplikacji, z których kilka przedstawiamy. Ale najpierw, w sekcji 15.4, przedstawiamy stochastyczne podejścia do uchwycenia wzorców i prawidłowości języka.

15.4 Stochastyczne narzędzia do rozumienia języka

W rozdziale 15.1 omówiliśmy naturalny rozkład konstrukcji zdań, który wspierał rozumienie języka. W rozdziałach 2 i 3 zauważyliśmy, że semantyczne i wymagające wiedzy aspekty języka mogą być wspierane przez struktury reprezentacyjne na poziomie słów i zdań. W tej sekcji przedstawiamy narzędzia stochastyczne, które na tych samych poziomach wspierają opartą na wzorcach analizę struktur umożliwiających zrozumienie języka.

15.4.1 Wprowadzenie: techniki statystyczne w analizie języka

Statystyczne techniki językowe to metody, które pojawiają się, gdy postrzegamy język naturalny jako proces losowy. W języku potocznym losowość sugeruje brak struktury, definicji lub zrozumienia. Jednak postrzeganie języka naturalnego jako procesu losowego uogólnia deterministyczny punkt widzenia. Oznacza to, że techniki statystyczne (lub stochastyczne) mogą dokładnie modelować zarówno te części języka, które są dobrze zdefiniowane, jak i te, które rzeczywiście mają pewien stopień losowości. Postrzeganie języka jako procesu losowego pozwala nam przedefiniować wiele podstawowych problemów w rozumieniu języka naturalnego w rygorystyczny, matematyczny sposób. Ciekawym ćwiczeniem jest na przykład zrobienie kilku zdań, wypowiedzenie poprzedniego akapitu z kropkami i nawiasami oraz wydrukowanie tych samych słów i symboli językowych uporządkowanych przez generator liczb losowych. Wynik nie będzie miał większego sensu. Warto zauważyć (Jurafsky i Martin 2008), że te same ograniczenia oparte na wzorcach działają na wielu poziomach analizy językowej, w tym na wzorach akustycznych, kombinacjach fonemicznych, analizie struktury gramatycznej i tak dalej. Jako przykład użycia narzędzi stochastycznych, najpierw rozważymy problem tagowania części mowy. Większość ludzi zna ten problem z gramatyki. Chcemy oznaczyć każde słowo w zdaniu jako rzeczownik, czasownik, przyimek, przymiotnik i tak dalej. Ponadto, jeśli słowo jest czasownikiem, możemy chcieć wiedzieć, czy jest czynne, bierne, przechodnie czy nieprzechodnie. Jeśli słowo jest rzeczownikiem, czy jest w liczbie pojedynczej czy mnogiej i tak dalej. Trudność pojawia się w przypadku słów takich jak „swing". Jeśli mówimy „huśtawka na ganku", huśtawka jest rzeczownikiem, ale jeśli mówimy „huśtawka na piłce", huśtawka jest czasownikiem. Przedstawiamy kolejny cytat z Picassa wraz z odpowiednią częścią etykiet mowy:

Art is a lie that lets us see the truth.

Rzeczownik Czasownik Artykuł Rzeczownik Zaimek Czasownik Zaimek Czasownik Artykuł Rzeczownik

Aby rozpocząć naszą analizę, najpierw formalnie zdefiniujemy problem. Mamy zestaw słów w naszym języku $S_w = \{w_1, ..., w_n\}$, na przykład {a, aardvark, ..., zygota} oraz zestaw części mowy lub znaczników $S_t = \{t_1,. .., t_m\}$. Zdanie zawierające n słów jest ciągiem n zmiennych losowych W_1, W_2, ... W_n. Nazywa się to zmiennymi losowymi, ponieważ z pewnym prawdopodobieństwem mogą przyjmować dowolną wartość w S_w. Znaczniki T_1, T_2, ..., T_n są również sekwencją zmiennych losowych. Wartość, jaką przyjmuje T_i, będzie oznaczona t_i, a wartość W_i to wi. Chcemy znaleźć sekwencję wartości dla tych

tagów, co jest najbardziej prawdopodobne, biorąc pod uwagę słowa w zdaniu. Formalnie chcemy wybrać $t_1, ..., t_n$, aby zmaksymalizować:

$p (T_1 = t_1, ..., T_n = t_n \mid W_1 = w_1, ..., W_n = w_n)$

Przypomnijmy, że p (X | Y) oznacza prawdopodobieństwo X przy założeniu, że Y wystąpiło. Zwyczajowo odrzuca się odniesienie do zmiennych losowych i po prostu pisze:

$p (t_1, ..., t_n \mid w_1, ..., w_n)$ równanie 1

Zauważ, że gdybyśmy dokładnie znali ten rozkład prawdopodobieństwa i mielibyśmy wystarczająco dużo czasu, aby zmaksymalizować wszystkie możliwe zestawy znaczników, zawsze otrzymalibyśmy najlepszy możliwy zestaw znaczników dla rozważanych słów. Ponadto, jeśli naprawdę istniałaby tylko jedna poprawna sekwencja znaczników dla każdego zdania, pomysł, który mógł zaakceptować twój nauczyciel gramatyki, ta probabilistyczna technika zawsze znajdowałaby tę poprawną sekwencję! Zatem prawdopodobieństwo prawidłowej sekwencji wyniesie 1, a dla wszystkich pozostałych ciągów 0. To właśnie mieliśmy na myśli, mówiąc, że statystyczny punkt widzenia może uogólniać deterministyczny. W rzeczywistości, z powodu ograniczonej przestrzeni dyskowej, danych i czasu, nie możemy zastosować tej techniki i musimy wymyślić jakiś rodzaj przybliżenia. Reszta tej sekcji dotyczy coraz lepszych sposobów przybliżania równania 1. Po pierwsze, zauważ, że możemy przepisać równanie 1 w bardziej użyteczny sposób:

$$p(t_1, ..., t_n \mid w_1, ..., w_n) = p(t_1, ..., t_n , w_1, ..., w_n) / p(w_1, ..., w_n)$$

a ponieważ maksymalizujemy to, wybierając $t_1, ..., t_n$, możemy uprościć równanie 1 do:

$$p(t_1, ..., t_n, w_1,..., w_n) =$$

$$p(t_1)p(w_1 \mid t_1)p(t_2 \mid t_1,w_1) ... p(t_n \mid w_1, ..., w_n , t_1, ..., t_{n-1}) =$$

$$\prod_{i=1}^{n} p(t_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1}) \, p(w_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1}) \qquad \text{equation 2}$$

Zauważ, że równanie 2 jest równoważne równaniu 1.

15.4.2 Podejście modelowe Markowa

W praktyce, jak widzieliśmy wcześniej w naszej dyskusji na temat rozdziałów 9.3 i 13, maksymalizacja równań z prawdopodobieństwami uwarunkowanymi wieloma innymi zmiennymi losowymi, takimi jak te, które znajdujemy w równaniu 2., jest zwykle złożonym zadaniem: po pierwsze , to jest trudne do przechowywania prawdopodobieństwa zmiennej losowej uwarunkowanej wieloma innymi zmiennymi losowymi, ponieważ liczba możliwych prawdopodobieństw rośnie wykładniczo wraz z liczbą zmiennych warunkujących. Po drugie, nawet gdybyśmy mogli przechowywać wszystkie wartości prawdopodobieństwa, często trudno jest oszacować ich wartości. Szacowanie jest zwykle wykonywane empirycznie, licząc liczbę wystąpień zdarzenia w ręcznie oznaczonym korpusie treningowym, a zatem, jeśli zdarzenie występuje tylko kilka razy w tym zbiorze uczącym, nie uzyskamy dobrego oszacowania jego prawdopodobieństwa. Oznacza to, że łatwiej jest oszacować p (kot | the) niż p (kot | Pies gonił), ponieważ będzie mniej wystąpień tego ostatniego w zestawie szkoleniowym. Wreszcie, znalezienie łańcucha znaczników, który maksymalizuje struktury takie jak równanie 2, zajęłoby zbyt dużo czasu, co zostanie pokazane poniżej.

Najpierw wykonujemy użyteczne przybliżenia równania 2. Pierwsza zgrubna próba to:

$p(t_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1})$ podejścia $p(t_i \mid t_{i-1})$ i $p(w_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1})$ zbliża się do $p(w_i \mid t_i)$.

Są to założenia Markowa pierwszego rzędu, gdzie zakłada się, że rozważana rzecz obecna jest zależna tylko od rzeczy bezpośrednio poprzedzającej, tj. Jest niezależna od rzeczy z bardziej odległej przeszłości. Podłączając te przybliżenia z powrotem do równania 2, otrzymujemy

$$\prod_{I=1}^{n} p(t_i \mid t_{i-1}) \, p(w_i \mid t_i) \qquad \text{equation 3}$$

Równanie 3 jest proste w obsłudze, ponieważ jego prawdopodobieństwa można łatwo oszacować i zapisać. Przypomnijmy, że równanie 3 jest tylko oszacowaniem $P(t_1, ..., t_n \mid w_1, ..., w_n)$ i nadal musimy je zmaksymalizować, wybierając tagi, tj. $t_1, ..., t_n$. Na szczęście istnieje algorytm programowania dynamicznego (DP) zwany algorytmem Viterbiego który pozwoli nam to zrobić. Algorytm Viterbiego oblicza prawdopodobieństwo wystąpienia sekwencji znaczników t2 dla każdego słowa w zdaniu, gdzie t jest liczbą możliwych tagów. W przypadku konkretnego kroku rozważane sekwencje znaczników mają następującą postać:

artykuł artykuł {best tail}

artykuł czasownik {best tail}

...

artykuł rzeczownik {best tail}

...

...

artykuł rzeczownikowy {best tail}

...

rzeczownik rzeczownik {best tail}

gdzie {best tail} jest najbardziej prawdopodobną sekwencją tagów znalezionych dynamicznie dla ostatnich n - 2 słów dla danego znacznika n - 1. W tabeli znajduje się wpis dla każdej możliwej wartości numeru tagu n - 1 i tagu liczba n (stąd mamy sekwencje znaczników t^2). Na każdym kroku algorytm znajduje maksymalne prawdopodobieństwo i dodaje jeden znacznik do każdej najlepszej sekwencji ogona. Algorytm ten gwarantuje znalezienie sekwencji znaczników, która maksymalizuje równanie 3 i działa w $O(t^2 s)$, gdzie t jest liczbą znaczników, a s jest liczbą słów w zdaniu. Jeśli $p(t_i)$ jest uwarunkowane na ostatnich n znacznikach, a nie na ostatnich dwóch, algorytm Viterbiego przyjmie $O(t^n s)$. W ten sposób widzimy, dlaczego warunkowanie zbyt wielu przeszłych zmiennych wydłuża czas potrzebny do znalezienia wartości maksymalizującej. Na szczęście przybliżenia użyte w równaniu 3 działają dobrze. Z około 200 możliwymi tagami i dużym zestawem uczącym do szacowania prawdopodobieństwa, tagger używający tych metod jest dokładny w około 97%, co jest zbliżone do ludzkiej dokładności. Zaskakująca dokładność przybliżenia Markowa wraz z jego prostotą sprawia, że jest ono przydatne w wielu zastosowaniach. Na przykład większość systemów rozpoznawania mowy wykorzystuje tzw. Model trygramowy, aby dostarczyć systemowi pewnej „wiedzy gramatycznej" do przewidywania słów, które wypowiedział użytkownik. Model trygramowy to prosty model Markowa, który szacuje prawdopodobieństwo aktualnego słowa uwarunkowane dwoma poprzednimi słowami. Wykorzystuje

algorytm Viterbiego i inne właśnie opisane techniki. Aby uzyskać więcej informacji na temat tej i pokrewnych technik, zobacz Jurasky i Martin.

15.4.3 Podejście drzewa decyzyjnego

Oczywistym problemem związanym z podejściem Markowa jest to, że uwzględnia ono tylko kontekst lokalny. Jeśli zamiast oznaczać słowa prostymi częściami mowy, chcemy np. Zidentyfikować agenta, zidentyfikować przedmiot lub zdecydować, czy czasowniki są aktywne, czy pasywne, to wymagany jest bogatszy kontekst. Poniższe zdanie ilustruje ten problem: Polityka ogłoszona w grudniu przez Prezydenta gwarantuje niższe podatki. W rzeczywistości prezydent jest agentem, ale program wykorzystujący model Markowa prawdopodobnie zidentyfikowałby politykę jako agenta i ogłosił jako aktywny czasownik. Możemy sobie wyobrazić, że program byłby lepszy w probabilistycznym wyborze agenta tego typu zdania, gdyby można zadawać pytania typu: „Czy obecny rzeczownik jest nieożywiony?" lub „Czy słowo obok występuje kilka słów przed rozważanym rzeczownikiem?" Przypomnij sobie, że problem tagowania jest równoważny maksymalizowaniu równania 2, tj.

$$\prod_{i=1}^{n} p(t_i \mid t_1, \ldots, t_{i-1}, w_1, \ldots, w_{i-1})\, p(w_i \mid t_1, \ldots, t_{i-1}, w_1, \ldots, w_{i-1}).$$

Teoretycznie rozważenie szerszego kontekstu wymaga po prostu znalezienia lepszych szacunków dla tych prawdopodobieństw. Sugeruje to, że moglibyśmy chcieć użyć odpowiedzi na powyższe pytania gramatyczne, aby sprecyzować prawdopodobieństwa. Istnieje kilka sposobów rozwiązania tego problemu. Po pierwsze, możemy połączyć podejście Markowa z technikami analizy przedstawionymi w pierwszych trzech sekcjach tego rozdziału. Druga metoda pozwala nam znaleźć prawdopodobieństwa uwarunkowane pytaniami tak lub nie za pomocą algorytmu ID3 lub innego równoważnego algorytmu. Drzewa ID3 mają tę dodatkową zaletę, że z bardzo dużego zestawu możliwych pytań wybiorą tylko te, które są dobre w precyzowaniu szacunków prawdopodobieństwa. W przypadku bardziej skomplikowanych zadań przetwarzania języka naturalnego, takich jak parsowanie, drzewa oparte na ID3 są często preferowane zamiast modeli Markowa. Następnie opiszemy, jak używać ID3 do konstruowania drzewa decyzyjnego używanego podczas analizowania. Przypomnij sobie, że w powyższej sekcji zadaliśmy pytanie „Czy obecny rzeczownik jest nieożywiony?" Takie pytania możemy zadawać tylko wtedy, gdy wiemy, które słowa są ożywione, a które nieożywione. W rzeczywistości istnieje zautomatyzowana technika, która może przypisywać nam słowa do tego typu zajęć. Technika ta nazywa się wzajemnym grupowaniem informacji. Informacje o wzajemnej wymianie między dwiema zmiennymi losowymi X i Y są zdefiniowane w następujący sposób:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 \frac{p(x, y)}{p(x)p(y)}$$

Aby dokonać wzajemnego grupowania informacji na podstawie słownika słów, zaczynamy od umieszczenia każdego słowa ze słownika w odrębnym zestawie. Na każdym kroku obliczamy średnią wzajemną informację między zbiorami za pomocą bigramu, czyli modelu następnego słowa, i wybierane jest połączenie dwóch zestawów słów, co minimalizuje utratę średniej wzajemnej informacji dla wszystkich klas. Na przykład, jeśli początkowo mamy słowa cat, kitten, run i green, to w pierwszym kroku algorytmu mamy zestawy:

{kot} {kotek} {bieg} {zielony}.

Jest prawdopodobne, że prawdopodobieństwo wystąpienia następnego słowa, biorąc pod uwagę, że poprzednie słowo było kotem, jest prawie równe prawdopodobieństwu następnego słowa, biorąc pod uwagę, że poprzednie słowo było kociakiem. Innymi słowy:

p (eats | cat) jest mniej więcej takie samo jak p (eats | kitten)

p (miauczy | kot) jest mniej więcej tym samym, co p (miauczy | kotek)

Tak więc, jeśli pozwolimy X1, X2, Y1 i Y2 być zmiennymi losowymi takimi, że:

X1 = {{kot}, {kotek}, {bieg}, {zielony}}

Y1 = słowo następujące po X1

X2 = {{kot, kotek}, {bieg}, {zielony}}

Y2 = słowo następujące po X2,

wtedy wzajemne informacje między X2 i Y2 są niewiele mniejsze niż wzajemne informacje między X1 i Y1, więc kot i kociak prawdopodobnie zostaną połączone. Jeśli będziemy kontynuować tę procedurę, dopóki nie połączymy wszystkich możliwych klas, otrzymamy drzewo binarne. Następnie można przypisać kody bitowe do słów w oparciu o gałęzie pobrane w drzewie, które docierają do węzła-liścia, w którym znajduje się to słowo. Odzwierciedla to semantyczne znaczenie tego słowa. Na przykład:

cat = 01100011

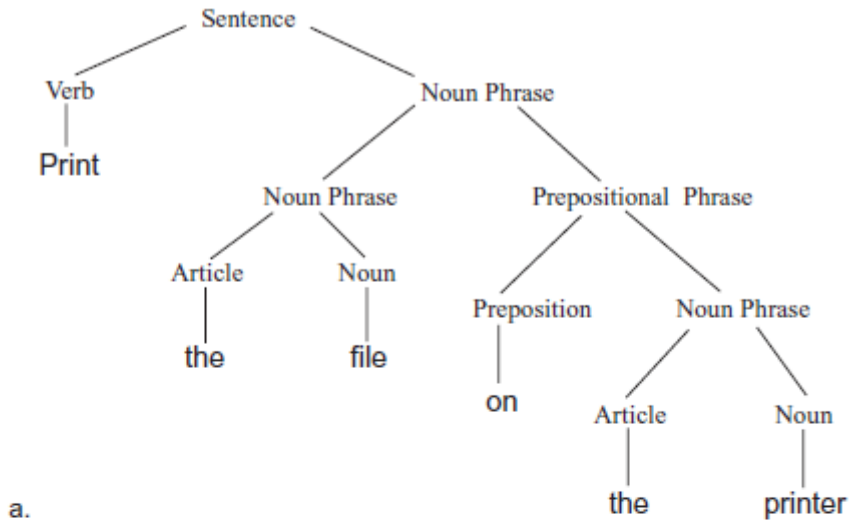
kociak = 01100010

Co więcej, możemy stwierdzić, że słowa „podobne do rzeczowników" to wszystkie te słowa, które mają 1 w lewym bicie, a słowami, które najprawdopodobniej reprezentują przedmioty nieożywione, mogą być te, których trzecim bitem jest 1.

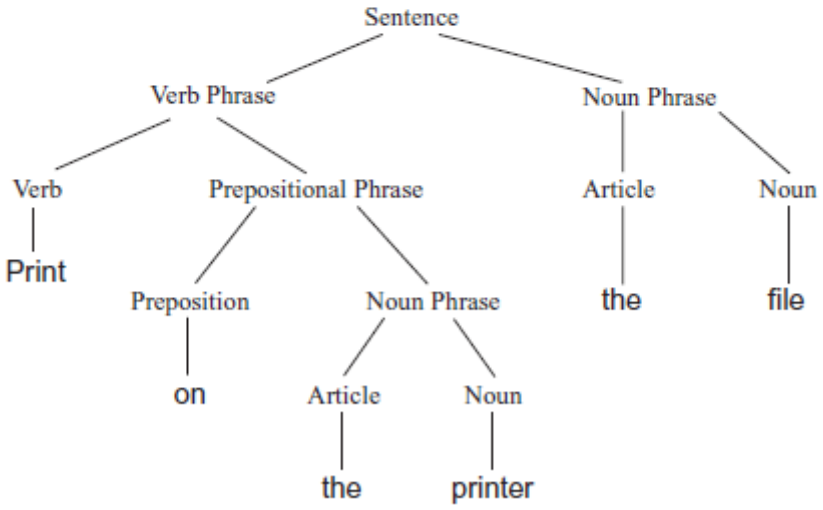
To nowe kodowanie słów ze słownika umożliwia bardziej efektywne zadawanie pytań przez parser. Zwróć uwagę, że grupowanie nie bierze pod uwagę kontekstu, więc nawet jeśli „książka" może być skupiona jako słowo „podobne do rzeczownika", będziemy chcieli, aby nasz model oznaczył go jako czasownik, gdy zostanie znaleziony w wyrażeniu „książka lot."

15.4.4 Probabilistyczne podejścia do analizy

Techniki stochastyczne były już stosowane w wielu dziedzinach lingwistyki komputerowej i nadal istnieje wiele możliwości zastosowania ich w obszarach, które oparły się tradycyjnym, symbolicznym podejściom. Zastosowanie metod statystycznych w parsowaniu było po raz pierwszy motywowane problemem niejednoznaczności. Niejednoznaczność wynika z faktu, że często istnieje kilka możliwych parsów dla danego zdania i musimy wybrać, która pars może być najlepsza. Na przykład zdanie Drukuj plik na drukarce można przeanalizować za pomocą jednego z dwóch drzew przedstawionych na rysunku 14.



a.



b.

W takich sytuacjach same reguły gramatyczne nie wystarczą, aby wybrać poprawną analizę. W przypadku Drukuj plik na drukarce musimy wziąć pod uwagę pewne informacje dotyczące kontekstu i semantyki. W rzeczywistości podstawowym zastosowaniem technik stochastycznych w dziedzinie analizowania jest pomoc w rozwiązywaniu niejednoznaczności. W obecnym przykładzie możemy użyć tego samego narzędzia, które jest używane w części znakowania mowy, algorytmu ID3. ID3 pomaga nam w przewidywaniu prawdopodobieństwa, że parsowanie jest poprawne na podstawie semantycznych pytań dotyczących zdania. W przypadku, gdy w zdaniu występuje pewna niejednoznaczność składniowa, możemy to wybrać parse, która ma największe prawdopodobieństwo, że będzie poprawna. Jak zwykle ta technika wymaga dużego korpusu szkoleniowego zdań wraz z ich poprawnymi analizami. Ostatnio ludzie ze społeczności zajmujących się modelowaniem statystycznego języka naturalnego stali się bardziej ambitni i próbowali używać technik statystycznych bez gramatyki do analizowania. Chociaż szczegóły analizy bez gramatyki wykraczają poza zakres tej książki, wystarczy powiedzieć, że jest to bardziej związane z rozpoznawaniem wzorców niż z tradycyjnymi technikami analizowania opisanymi wcześniej w tej części. Analiza gramatyczna była całkiem udana. W eksperymentach porównujących tradycyjny parser oparty na gramatyce z parserem bez gramatyki, zajmującym się analizowaniem tego samego zestawu zdań, parser gramatyczny uzyskał wynik, używając popularnej miary, w nawiasach 69%, a parser bez gramatyki 78%. Wyniki te są dobre, choć

nie wybitne. Co ważniejsze, gramatyka w tradycyjnym parserze została skrupulatnie opracowana przez wyszkolonego językoznawcę w ciągu około dziesięciu lat, podczas gdy parser bez gramatyki zasadniczo nie używał zakodowanych na stałe informacji językowych, a jedynie wyrafinowane modele matematyczne, które mogłyby wywnioskować potrzebne informacje ze szkolenia. dane. Więcej informacji na temat parsowania bez gramatyki i pokrewnych zagadnień można znaleźć w Manning i Schutze. Rozumienie mowy, konwersja mowy na tekst i rozpoznawanie pisma ręcznego to trzy kolejne obszary, które mają długą historię stosowania stochastycznych metod modelowania języka. Najczęściej stosowaną metodą statystyczną w tych obszarach jest model trygramowy do przewidywania następnego słowa. Siła tego modelu tkwi w jego prostocie: przewiduje on następne słowo na podstawie dwóch ostatnich słów. Ostatnio w społeczności języków statystycznych podjęto prace nad utrzymaniem prostoty i łatwości użycia tego modelu, przy jednoczesnym uwzględnieniu ograniczeń gramatycznych i zależności z większej odległości. To nowe podejście wykorzystuje tak zwane trygramy gramatyczne. Trygramy gramatyczne są oparte na podstawowych skojarzeniach między parami słów (tj. Podmiot-czasownik, rodzajnik-rzeczownik i czasownik-przedmiot). Łącznie te skojarzenia nazywane są gramatyką linków. Gramatyka linków jest znacznie prostsza i łatwiejsza do skonstruowania niż tradycyjne gramatyki używane przez językoznawców i dobrze działa z metodami probabilistycznymi. Berger i inni opisują program statystyczny Candide, który tłumaczy tekst francuski na tekst angielski. Kandyd używa zarówno statystyki, jak i teorii informacji, aby opracować model prawdopodobieństwa procesu tłumaczenia. Szkoli tylko na dużym korpusie francuskim i Angielskie pary zdań i dają wyniki porównywalne, aw niektórych przypadkach lepsze niż Systran, komercyjny program tłumaczeniowy. Szczególnie interesujący jest fakt, że system Candide nie wykonuje tradycyjnej analizy w procesie tłumaczenia. Zamiast używać trygramów gramatycznych i gramatyk linków, o których właśnie wspomniałem.

15.4.5 Probabilistyczne parsery bezkontekstowe

W tej sekcji przedstawiamy dwa różne probabilistyczne parsery bezkontekstowe, oparte na strukturze i leksykalizowane. Parser oparty na strukturze używa miar prawdopodobieństwa dla każdej występującej reguły analizy gramatycznej. Prawdopodobieństwo każdej reguły parsowania jest funkcją tej reguły w połączeniu z prawdopodobieństwami jej składników. Demonstrujemy probabilistyczny parser oparty na strukturze, rozszerzając zestaw bezkontekstowych reguł gramatycznych z sekcji 15.2 o miary prawdopodobieństwa:

1. zdanie ↔ rzeczownik_fraza czasownik_fraza p (s) = p (r1) p (np) p (vp)

2. noun_phrase ↔ rzeczownik p (np) = p (r2) p (rzecz.)

3. noun_phrase ↔ rzeczownik przedimka p (np) = p (r3) p (przedimek) p (rzeczownik)

4. verb_phrase ↔ czasownik p (vp) = p (r4) p (czasownik)

5. verb_phrase ↔ czasownik rzeczownik_phrase p (vp) = p (r5) p (czasownik) p (np)

6. artykuł ↔ a p (artykuł) = p (a)

7. artykuł ↔ p (artykuł) = p (the)

8. rzeczownik ↔ mężczyzna p (rzecz.) = P (mężczyzna)

9. rzeczownik ↔ dog p (rzecz.) = P (dog)

10. czasownik ↔ lubi p (rzeczownik) = p (lubi)

11. czasownik ↔ ukąszenia p (rzecz.) = P (ugryzienia)

Miary prawdopodobieństwa dla poszczególnych słów można wyliczyć z prawdopodobieństwa wystąpienia w określonym korpusie zdań angielskich. Miarę prawdopodobieństwa dla każdej reguły gramatycznej, p(ri), można określić na podstawie tego, jak często ta reguła gramatyczna występuje w zdaniach korpusu języka. Oczywiście przykład tutaj jest prosty - ma na celu scharakteryzowanie tego typu probabilistycznego parsera bezkontekstowego. Drugi przykład, probabilistyczny, leksykalizowany bezkontekstowy parser, również zostanie zademonstrowany poprzez rozszerzenie reguł gramatycznych z sekcji 15.2. W tej sytuacji chcemy wziąć pod uwagę wiele aspektów wyroku. Po pierwsze, możemy mieć prawdopodobieństwa użycia języka w bigramie lub trygramie. Ponownie będziemy zależni od odpowiedniego korpusu językowego, aby uzyskać miary bigramów lub trygramów. Podobnie jak w naszym poprzednim przykładzie, użyjemy miar prawdopodobieństwa, uzyskanych z korpusu języka, dla wystąpienia każdej z reguł analizy. Na koniec będziemy mieli probabilistyczne miary poszczególnych rzeczowników i czasowników występujących razem. Na przykład, jeśli rzeczownikiem podmiotu jest pies. prawdopodobieństwo, że czasownik będzie ukąszony. Zwróć uwagę, że ten środek, nawet jeśli pochodzi z korpusu języka, może również wymusić zgodność rzeczownika-czasownika. Nasz parser nazywamy leksykalizacją, ponieważ jesteśmy w stanie skupić się na prawdopodobieństwach występowania określonych wzorców słów. Dla uproszczenia przedstawiamy nasz parser tylko dla bigramów:

1. sentence ↔ noun_phrase(noun) verb_phrase(verb)

p(sentence) = p(r1) p(np) p(vp) p(verb | noun)

2. noun_phrase ↔ noun

p(np) = p(r2) p(noun)

3. noun_phrase ↔ article noun

p(np) = p(r3) p(article) p(noun) p(noun | article)

4. verb_phrase ↔ verb

p(vp) = p(r4) p(verb)

5. verb_phrase ↔ verb noun_phrase

p(vp) = p(r5) p(verb) p(noun_phrase(noun)) p(noun |verb)

6. article ↔ a p(article) = p(a)

7. article ↔ the p(article) = p(the)

8. noun ↔ man p(noun) = p(man)

. p(man | a)

p(man | the)

etc.

p(likes | man)

etc.

Każdy z dwóch probabilistycznych parserów tej sekcji został zbudowany w Prologu w materiałach pomocniczych do tego rozdziału. Istnieje wiele innych obszarów, w których rygorystyczne techniki

modelowania języka stochastycznego nie zostały jeszcze wypróbowane, ale mogą przynieść użyteczne wyniki. Jednym z potencjalnych zastosowań jest ekstrakcja informacji, czyli problem uzyskania określonej ilości konkretnych informacji z tekstu pisanego. Po drugie, można dopasować wzorce w mowie dźwięcznej, aby połączyć je z koncepcjami określonej bazy wiedzy. Wreszcie jest oczywiście wyszukiwanie w sieci semantycznej. Więcej szczegółów na temat stochastycznych podejść do przetwarzania języka naturalnego można znaleźć w Jurafsky i Martin oraz Manning i Schutze.

15.5 Aplikacje języka naturalnego

15.5.1 Rozumienie historii i odpowiadanie na pytania

Ciekawym testem na technologię rozumienia języka naturalnego jest napisanie programu, który potrafi czytać opowiadanie lub inny fragment tekstu w języku naturalnym i odpowiadać na pytania na jego temat. W rozdziale Części 7 omówiliśmy niektóre kwestie reprezentacyjne związane ze zrozumieniem historii, w tym znaczenie połączenia wiedzy podstawowej z wyraźną treścią tekstu. Jak pokazano na rysunku 2, program może to osiągnąć, wykonując połączenia sieciowe między semantyczną interpretacją danych wejściowych a koncepcyjnymi strukturami grafów w bazie wiedzy. Bardziej wyrafinowane reprezentacje, takie jak skrypty, mogą modelować bardziej złożone sytuacje obejmujące zdarzenia zachodzące w czasie. Gdy program zbuduje rozszerzoną reprezentację tekstu, może inteligentnie odpowiadać na pytania o to, co przeczytał. Program analizuje pytanie w wewnętrzną reprezentację i dopasowuje to zapytanie do rozszerzonej reprezentacji historii. Rozważmy przykład z rysunku 2. Program odczytał zdanie „Tarzan pocałował Jane” i zbudował rozszerzoną reprezentację. Załóżmy, że pytamy program „Kto kocha Jane?” Analizując pytanie, pytający, kto, co, dlaczego itp. Wskazuje na intencję pytania. Kto pyta o agenta akcji; jakie pytania dotyczą przedmiotu działania; jak pytania dotyczą środków, za pomocą których czynność została wykonana i tak dalej. Pytanie „Kto kocha Jane?” tworzy wykres z rysunku 15. Węzeł agenta na wykresie jest oznaczony? aby wskazać, że jest to cel pytania. Ta struktura jest następnie łączona z rozszerzoną reprezentacją tekstu. Pojęcie, które wiąże się z osobą:? pojęcie na wykresie zapytań to odpowiedź na pytanie: „Tarzan kocha Jane”.

15.5.2 Interfejs bazy danych

Głównym wąskim gardłem w projektowaniu programów rozumienia języka naturalnego jest zdobycie wystarczającej wiedzy na temat domeny dyskursu. Obecna technologia jest ograniczona do wąskich dziedzin z dobrze zdefiniowaną semantyką. Obszarem zastosowań spełniającym te kryteria jest rozwój interfejsów w języku naturalnym dla baz danych. Chociaż bazy danych przechowują ogromne ilości informacji, informacje te są bardzo regularne i mają wąski zakres; ponadto semantyka bazy danych jest dobrze zdefiniowana. Te funkcje, wraz z użytecznością bazy danych, która może akceptować zapytania w języku naturalnym, sprawia, że interfejsy baz danych są ważnym zastosowaniem technologii rozumienia języka naturalnego. Zadaniem frontendu bazy danych jest przetłumaczenie pytania w języku naturalnym na dobrze sformułowane zapytanie w języku bazy danych. Na przykład przy użyciu bazy danych SQL język jako cel (Ullman 1982), interfejs języka naturalnego tłumaczyłby pytanie „Kto zatrudnił Johna Smitha?” do zapytania:

```
SELECT MANAGER

FROM MANAGER_OF_HIRE

WHERE EMPLOYEE = John Smith
```

Wykonując to tłumaczenie, program musi zrobić coś więcej niż tylko przetłumaczyć oryginalne zapytanie; musi również zdecydować, gdzie szukać w bazie danych (relacja MANAGER_OF_HIRE), nazwa pola, do którego ma być dostęp (MANAGER), oraz ograniczenia zapytania (PRACOWNIK = „Jan

Kowalski"). Żadna z tych informacji nie znajdowała się w pierwotnym pytaniu; został znaleziony w bazie wiedzy, która wiedziała o organizacji bazy danych i znaczeniu potencjalnych pytań. Relacyjna baza danych organizuje dane w relacjach między domenami jednostek. Załóżmy na przykład, że tworzymy bazę danych pracowników i chcielibyśmy uzyskać dostęp do wynagrodzenia każdego pracownika oraz menedżera, który ją zatrudnił. Ta baza danych składałaby się z trzech domen lub zbiorów jednostek: zbioru menedżerów, zbioru pracowników i zbioru wynagrodzeń. Moglibyśmy uporządkować te dane w dwie relacje, wynagrodzenie_pracownika, które odnosi się do pracownika i jej pensji, oraz manager_of_hire, które odnosi się do pracownika i jego kierownika. W relacyjnej bazie danych relacje są zwykle wyświetlane jako tabele, które wyliczają wystąpienia relacji. Kolumny tabel są często nazywane; nazwy te nazywane są atrybutami relacji. Rysunek 16 przedstawia tabele dla relacji wynagrodzenie_pracownika i wynagrodzenia_zaawansowanego_w hrabstwie.

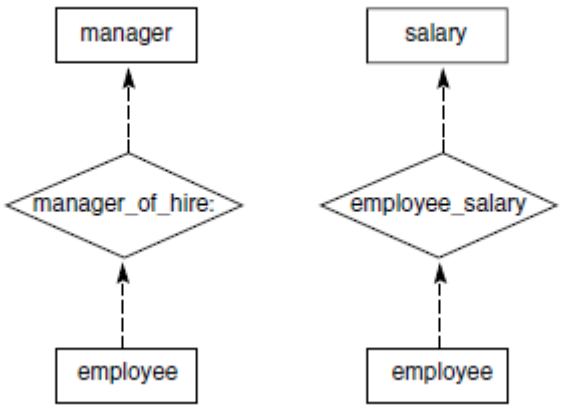
manager_of_hire:

employee	manager
John Smith	Jane Martinez
Alex Barrero	Ed Angel
Don Morrison	Jane Martinez
Jan Claus	Ed Angel
Anne Cable	Bob Veroff

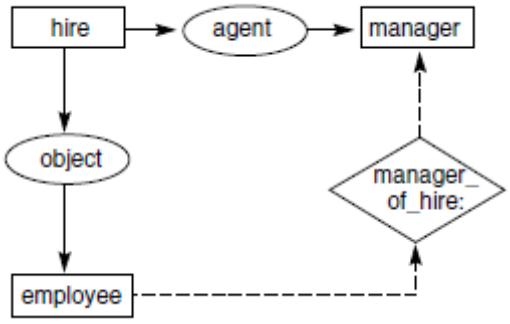
employee_salary:

employee	salary
John Smith	$35,000.00
Alex Barrero	$42,000.00
Don Morrison	$50,000.00
Jan Claus	$40,000.00
Anne Cable	$45,000.00

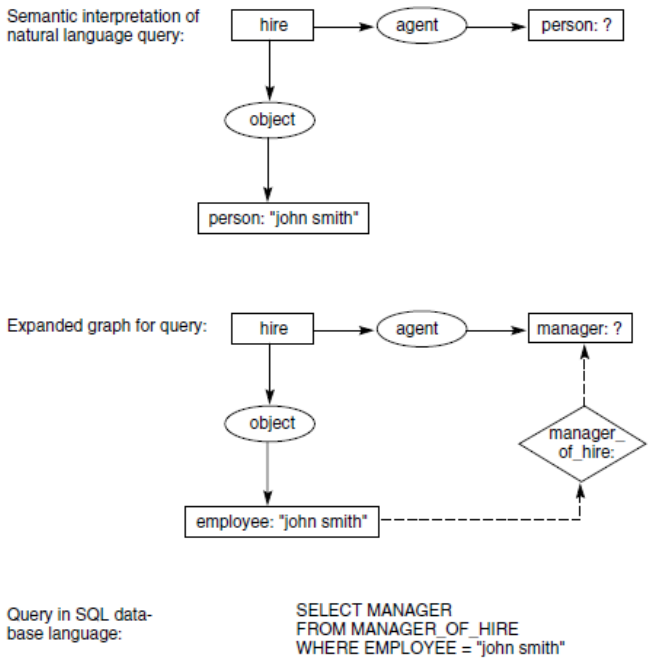
Manager_of_hire ma dwa atrybuty: pracownika i kierownika. Wartości relacji są parami pracowników i menedżerów. Jeśli założymy, że pracownicy mają unikalne imię i nazwisko, menedżera i pensję, to nazwisko pracownika może służyć jako klucz zarówno do wynagrodzenia, jak i do atrybutów menedżera. Atrybut jest kluczem do innego atrybutu, jeśli jednoznacznie określa wartość elementów innego atrybutu. Prawidłowe zapytanie wskazuje atrybut docelowy i określa wartość lub zestaw ograniczeń; baza danych zwraca określone wartości atrybutu docelowego. Zależności między kluczami i innymi atrybutami możemy wskazać graficznie na wiele sposobów, w tym na diagramach relacji encja-jednostka i diagramach przepływu danych . Oba te podejścia wyświetlają mapowanie kluczy na atrybuty przy użyciu skierowanych wykresów. Grafy pojęciowe możemy rozszerzyć o diagramy tych zależności . Relacja bazy danych definiująca odwzorowanie jest oznaczona rombem, na którym znajduje się nazwa relacji. Atrybuty relacji są wyrażone jako pojęcia na grafie pojęciowym, a kierunek strzałek wskazuje przyporządkowanie kluczy do innych atrybutów. Wykresy relacji jednostka-jednostka dla relacji wynagrodzenie_pracownik i kierownik_w_hire można zobaczyć na rysunku 17.



Podczas tłumaczenia z języka angielskiego na formalne zapytanie musimy określić rekord zawierający odpowiedź, pole tego rekordu, które ma zostać zwrócone, oraz wartości kluczy określających to pole. Zamiast tłumaczyć bezpośrednio z języka angielskiego na język bazy danych, najpierw tłumaczymy na bardziej wyrazisty język, taki jak grafy pojęciowe. Jest to konieczne, ponieważ wiele zapytań w języku angielskim jest niejednoznacznych lub wymaga dodatkowej interpretacji w celu utworzenia poprawnie sformułowanego zapytania do bazy danych. Użycie bardziej wyrazistej reprezentacji język pomaga w tym procesie. Interfejs użytkownika języka naturalnego analizuje i interpretuje zapytanie w postaci grafu pojęciowego, jak opisano wcześniej w tym rozdziale. Następnie łączy ten wykres z informacjami w bazie wiedzy za pomocą operacji łączenia i ograniczania. W tym przykładzie chcemy obsługiwać zapytania, takie jak „Kto zatrudnił Jana Kowalskiego?" lub „Ile zarabia Jan Kowalski?" Dla każdego potencjalnego zapytania przechowujemy wykres, który definiuje jego czasownik, role przypadków dla tego czasownika oraz wszelkie istotne diagramy relacji encji dla pytania. Rysunek 18 przedstawia wpis w bazie wiedzy dla czasownika „zatrudniać".



Interpreter semantyczny tworzy wykres zapytania użytkownika i łączy ten wykres z odpowiednim wpisem w bazie wiedzy. Jeśli istnieje dołączony wykres relacji encji, który odwzorowuje klucze w celu pytania, program może użyć tego wykresu relacji encji do utworzenia zapytania do bazy danych. Rysunek 19 przedstawia wykres zapytań dla pytania „Kto zatrudnił Johna Smitha?" oraz wynik połączenia tego z wpisem do bazy wiedzy z rysunku 18.



Pokazuje również zapytanie SQL utworzone na podstawie tego wykresu. Zwróć uwagę, że nazwa odpowiedniego rekordu, pole docelowe i klucz zapytania nie zostały określone w zapytaniu w języku naturalnym. Zostały one wywiedzione z bazy wiedzy. Na rysunku 18 agent i obiekt pierwotnego zapytania były znane tylko jako typ person. Aby połączyć je z wpisem do bazy wiedzy do wynajęcia, były one najpierw ograniczone odpowiednio do typów menedżerów i pracowników. Hierarchia typów może zatem służyć do sprawdzania typów w możliwych rozwiązaniach pierwotnego zapytania. Gdyby John Smith nie był typem pracownika, pytanie byłoby niepoprawne i program mógłby to wykryć. Po zbudowaniu rozszerzonego wykresu zapytania program sprawdza koncepcję docelową oznaczoną znakiem? I ustala, że relacja manager_of_hire mapowała klucz na tę koncepcję. Ponieważ klucz jest powiązany z wartością john smith, pytanie było poprawne i program utworzyłby prawidłowe zapytanie do bazy danych. Tłumaczenie tego wykresu relacji encji na język SQL lub inny dowolny inny język baz danych ogólnego przeznaczenia jest proste. Chociaż ten przykład jest uproszczony, ilustruje użycie podejścia opartego na wiedzy do budowania frontonu bazy danych języka naturalnego. Idee w naszym przykładzie są wyrażone na grafach pojęciowych, ale równie dobrze można je odwzorować na inne reprezentacje, takie jak ramy lub formalizmy oparte na logice predykatów.

15.5.3 System pozyskiwania i podsumowywania informacji w sieci WWW

Sieć WWW oferuje wiele ekscytujących wyzwań, a także interesujące możliwości badań nad sztuczną inteligencją i rozumieniem języka naturalnego. Jedną z największych jest potrzeba zaprojektowania inteligentnego oprogramowania, które potrafi podsumować „ciekawe” materiały internetowe. W rzeczywistości problem ten składa się z dwóch części: lokalizowania potencjalnie interesujących informacji w Internecie, a następnie wydobywania (lub wydobywania) krytycznych składników tych informacji. Chociaż pierwszy problem związany z lokalizacją potencjalnie interesujących informacji jest krytyczny), teraz rozważymy podejście do dopasowywania i wypełniania szablonów w celu rozwiązania drugiego problemu. Po zlokalizowaniu informacji, być może przez dopasowanie słów kluczowych lub użycie bardziej wyrafinowanej wyszukiwarki, system wyodrębniania informacji przyjmuje jako dane wejściowe ten nieograniczony tekst, a następnie podsumowuje go w odniesieniu do wcześniej określonej dziedziny lub interesującego tematu. Znajduje użyteczne informacje o domenie i koduje ten formularz informacyjny odpowiedni do raportowania użytkownikowi lub do zapełniania

ustrukturyzowanej bazy danych. W przeciwieństwie do dogłębnego systemu rozumienia języka naturalnego, systemy ekstrakcji informacji przeglądają tekst, aby znaleźć odpowiednie sekcje, a następnie koncentrują się tylko na przetwarzaniu tych sekcji. Proponujemy podejście oparte na szablonach, podobne do technik ekstrakcji proponowanych przez kilku badaczy. Przegląd naszego systemu ekstrakcji informacji przedstawiono na rysunkach 20 i 21. Załóżmy, że chcemy znaleźć i podsumować na stronie WWW informacje dotyczące stanowisk wydziałowych na poziomie uniwersytetu informatycznego. Rysunek 20 przedstawia docelowe ogłoszenie o pracę, a także przedstawia szablon informacji, które chcemy, aby nasze oprogramowanie wyodrębniło z tego i podobnych ogłoszeń o pracę

Przykładowe ogłoszenie o pracę w informatyce (fragment):

The Department of Computer Science of the University of New Mexico. . . is conducting

a search to fill two tenure-track positions. We are interested in hiring people with research

interests in:

Software, including analysis, design, and development tools. . .

Systems, including architecture, compilers, networks. . .

...

Candidates must have completed a doctorate in. . .

The department has internationally recognized research programs in adaptive computation,

artificial intelligence, . . . and enjoys strong research collaborations with the Santa Fe

Institute and several national laboratories. ...

Przykładowy częściowo wypełniony szablon:

Employer: Department of Computer Science, University of New Mexico

Location City: Albuquerque

Location State: NM 87131

Job Description: Tenure track faculty

Job Qualifications: PhD in . . .

Skills Required: software, systems, . . .

Platform Experience: . . .

About the Employer: (text attached)

1. Tekst: Katedra Informatyki Uniwersytetu

of New Mexico prowadzi poszukiwania, aby zapełnić dwa

pozycje na torze. Jesteśmy zainteresowani wypełnieniem

na stanowiskach adiunkta. . .

2.Tokenizacja i tagowanie: / det Department / rzeczownik / prep ...

3. Analiza zdań: Dział / podmiot przeprowadza / wyszukiwanie czasownika / obj

Komputer / rzeczownik ...

4. Wydobycie: Pracodawca: Katedra Informatyki

Opis stanowiska: Pozycja na torze zatrudnienia ...

5. Łączenie: pozycja toru pracy = wydział

Nowy Meksyk = NM ...

6. Generowanie szablonu: jak na rysunku 19

We wczesnych próbach ekstrakcji informacji systemy języka naturalnego różniły się pod względem podejść. Z jednej strony systemy przetwarzały tekst przy użyciu tradycyjnych narzędzi: pełne rozbicie składniowe każdego zdania, któremu towarzyszyła szczegółowa analiza semantyczna. Często następowało przetwarzanie na poziomie dyskursu. Z drugiej strony systemy wykorzystywały techniki dopasowywania słów kluczowych z niewielką lub żadną wiedzą lub analizą na poziomie językowym. Jednak wraz z budową i oceną większej liczby systemów ograniczenia tych ekstremalnych podejść do ekstrakcji informacji stały się oczywiste. Bardziej nowoczesne podejście do ekstrakcji informacji, zaadaptowane z Cardie, zostało odzwierciedlone na rysunkach 20 i 21. Chociaż szczegóły tej architektury mogą się różnić w różnych aplikacjach, rysunki wskazują główne funkcje wykonywane podczas ekstrakcji. Najpierw każde zdanie „interesującej" witryny internetowej jest tokenizowane i tagowane. W tym celu można użyć taggera stochastycznego przedstawionego w sekcji 15.4. Etap analizy zdań, który następuje i przeprowadza analizę, tworzy następnie grupy rzeczowników, czasowniki, frazy przyimkowe i inne konstrukcje gramatyczne. Następnie faza wyodrębniania znajduje i określa semantyczne encje istotne dla tematu ekstrakcji. W naszym przykładzie będzie to identyfikacja nazwy pracodawcy, lokalizacji stanowiska wydziału, wymagań dotyczących stanowiska (wykształcenie, umiejętności obsługi komputera itp.), Czasu rozpoczęcia spotkania itp. Faza ekstrakcji jest pierwszym całkowicie specyficznym dla domeny etapem proces. Podczas ekstrakcji system identyfikuje określone relacje między odpowiednimi składnikami tekstu. W naszym przykładzie Wydział Informatyki jest postrzegany jako pracodawca, a lokalizacja jest postrzegana jako Uniwersytet Nowego Meksyku. Faza łączenia musi obejmować takie kwestie, jak odniesienie do synonimów i rozwiązanie anafory. Przykładami synonimów są teuretrack stanowisko i pozycja na wydziale, a także Nowy Meksyk i NM. Rezolucja Anafora łączy Wydział Informatyki w pierwszym zdaniu z nami, to jest przedmiotem drugiego zdania. Wnioski na poziomie dyskursu dokonane podczas scalania wspomagają fazę generowania szablonu, która określa liczbę odrębnych relacji w tekście, odwzorowuje te wyodrębnione fragmenty informacji na każde pole szablonu i tworzy ostateczny szablon wyjściowy. Pomimo ostatnich postępów, obecne systemy ekstrakcji informacji nadal mają problemy. Po pierwsze, dokładność i niezawodność tych systemów można znacznie poprawić, ponieważ błędy w ekstrakcji wydają się wynikać z raczej płytkiego zrozumienia znaczenia (semantyki) tekstu wejściowego. Po drugie, zbudowanie systemu ekstrakcji informacji w nowej domenie może być trudne i czasochłonne. Oba te problemy są związane z dziedzinowym charakterem zadania ekstrakcji. Proces wyodrębniania informacji poprawia się, jeśli źródła wiedzy językowej są dostrojone do określonej domeny, ale ręczne modyfikowanie wiedzy językowej specyficznej dla domeny jest zarówno trudne, jak i podatne na błędy.

Niemniej jednak istnieje obecnie szereg interesujących aplikacji. Glasgow i inni zbudowali system wspomagający ubezpieczycieli w analizie wniosków dotyczących ubezpieczeń na życie. Soderland i inni) zbudowali system do wyodrębniania objawów, ustaleń fizycznych, wyników testów i diagnoz z

dokumentacji medycznej pacjenta w celu przetworzenia ubezpieczenia. Istnieją programy do analizy artykułów prasowych w celu znalezienia i podsumowania wspólnych przedsięwzięć biznesowych, systemy, które automatycznie klasyfikują dokumenty prawne oraz programy, które wyodrębniają szczegółowe informacje z komputerowych ofert pracy.

15.5.4 Wykorzystanie algorytmów uczenia się do uogólniania wyodrębnionych informacji

Inna aplikacja łączy kilka pomysłów przedstawionych w tym rozdziale, a także algorytmy z uczenia maszynowego (sekcja 10.3). Cardie i Mooney oraz Nahm i Mooney zasugerowali, że informacje wyodrębnione z tekstu można uogólniać algorytmy uczenia maszynowego, a wynik ponownie wykorzystany w zadaniu ekstrakcji informacji. Podejście jest proste. Gotowe lub nawet częściowo wypełnione szablony podsumowania tekstu, jak pokazano na przykład na rysunku 20, są zbierane z odpowiednich witryn internetowych. Wynikowe informacje z szablonu są następnie przechowywane w relacyjnej bazie danych, w której algorytmy uczące się, takie jak ID3 lub C4.5, są używane do wyodrębniania drzew decyzyjnych, które, mogą odzwierciedlać relacje reguł ukryte w zbiorach danych. (Technikę tę nazwaliśmy eksploracją danych). Mooney i jego koledzy proponują, aby te nowo odkryte relacje można było następnie wykorzystać do udoskonalenia oryginalnych szablonów i struktur wiedzy wykorzystywanych do ekstrakcji informacji. Przykłady tego typu informacji, które można znaleźć na podstawie analizy podań o pracę w informatyce w sekcji 15.5.3, mogą obejmować: jeśli stanowisko jest wydziałem informatyki, to doświadczenie na określonej platformie komputerowej nie jest wymagane; jeśli uniwersytety zatrudniają wykładowców, wymagane jest doświadczenie badawcze, itp. Więcej szczegółów na temat tego podejścia do generalizacji informacji można znaleźć w Nahm i Mooney. Istnieje wiele ekscytujących problemów związanych z lingwistyką komputerową, których rozwiązanie nie jest naszym celem. Kończymy trzema otwartymi kwestiami. Po pierwsze, jaka jest kluczowa jednostka dla rozumienia języka mówionego? Podejmowano różne próby odpowiedzi na to pytanie, począwszy od pełnej analizy zdania w języku angielskim, przez próby rozpoznawania pojedynczych słów, po skupienie się na poszczególnych telefonach. Czy to możliwe, że skupienie się na sylabie jest kluczem do zrozumienia ludzkiego języka? Ile słowa musimy usłyszeć, zanim zostanie rozpoznane? Jaka jest optymalna ziarnistość do analizy mowy dźwięcznej (Greenberg 1999)? Po drugie, czy dostęp do pojęć w bazie wiedzy poprzez rozpoznawanie dźwięku może prowadzić do dalszych uwarunkowań późniejszej interpretacji języka? Załóżmy, że sylaba okaże się użyteczną jednostką do analizy języka. Załóżmy, że sylaby bigramów są przydatne do sugerowania słów mówcy. Czy te słowa można odwzorować w pojęciach bazy wiedzy, a powiązaną wiedzę można wykorzystać do ulepszenia interpretacji kolejnych sylab? Osoba dzwoniąca do organizatora podróży korzystającego z komputera może rozpoznać sylaby sugerujące nazwę miasta, a komputer może wtedy przewidzieć pytania związane z odwiedzaniem tego miasta. Wreszcie długoterminowym celem przetwarzania języka naturalnego jest stworzenie sieci semantycznej; jak to się stanie? Czy to się stanie? Czy istnieje metodologia, według której można powiedzieć, że komputery „coś rozumieją”? Czy też zawsze będą istnieć oparte na wzorcach „sztuczki” oparte na wyszukiwaniu, które są wystarczająco dobre, aby „przekonać” ludzi, że komputer rozumie (ponownie test Turinga)? Czy to „wystarczająco dobre” podejście nie jest tym, co my, ludzie, i tak robimy dla siebie nawzajem? Ostatni rozdział kontynuuje tę i powiązane dyskusje

SZTUCZNA INTELIGENCJA JAKO PYTANIE EMPIRYCZNE

16.0 Wprowadzenie

Dla wielu osób jednym z najbardziej zaskakujących aspektów pracy w sztucznej inteligencji jest stopień, w jakim sztuczna inteligencja, a właściwie większość informatyki, okazuje się dyscypliną empiryczną. Jest to zaskakujące, ponieważ większość ludzi początkowo myśli o tych dziedzinach w kategoriach ich podstaw matematycznych lub, alternatywnie, inżynierskich. Z matematycznego punktu widzenia, czasami nazywanego „schludną" perspektywą, istnieje racjonalistyczne pragnienie wprowadzenia standardów dowodu i analizy do projektowania inteligentnych urządzeń obliczeniowych. Z perspektywy inżynierskiej lub „niechlujnej", zadanie jest często postrzegane jako po prostu tworzenie udanych artefaktów, które społeczeństwo chce nazwać „inteligentnymi". Niestety, lub na szczęście, w zależności od waszej filozofii, złożoność inteligentnego oprogramowania i niejasności związane z jego interakcjami ze światem ludzkiej działalności utrudniają analizę z czysto matematycznego lub czysto inżynierskiego punktu widzenia. Ponadto, jeśli sztuczna inteligencja ma osiągnąć poziom nauki i stać się krytycznym składnikiem nauki o inteligentnych systemach, przy projektowaniu, wykonywaniu i analizie artefaktów należy uwzględnić mieszankę metod analitycznych i empirycznych. Z tego punktu widzenia każdy program sztucznej inteligencji może być postrzegany jako eksperyment: proponuje pytanie światu przyrody, a wyniki są odpowiedzią natury. Reakcja natury na nasz projekt i zobowiązania programowe kształtuje nasze rozumienie formalizmu, mechanizmu i wreszcie natury samej inteligencji (Newell i Simon 1976). W przeciwieństwie do wielu bardziej tradycyjnych badań ludzkiego poznania, my, jako projektanci inteligentnych artefaktów komputerowych, możemy badać wewnętrzne działanie naszych „badanych". Możemy zatrzymać wykonywanie programu, zbadać stan wewnętrzny i dowolnie modyfikować strukturę. Jak zauważają Newell i Simon, struktura komputerów i ich programów wskazuje na ich potencjalne zachowanie: można je badać, a ich reprezentacje i algorytmy wyszukiwania rozumieć. Wynikiem tego jest moc komputerów jako narzędzi do zrozumienia dwoistości inteligencji. Odpowiednio zaprogramowane komputery są w stanie zarówno osiągnąć poziom złożoności semantycznej i behawioralnej, który aż prosi się o scharakteryzowanie w kategoriach psychologicznych, jak i dają możliwość wglądu w ich stany wewnętrzne, czemu w dużej mierze odmawia się naukowcom badającym większość innych form życia intelektualnego. Na szczęście dla kontynuacji pracy nad sztuczną inteligencją, a także dla stworzenia nauki o inteligentnych systemach, nowocześniejsze techniki psychologiczne, zwłaszcza te związane z fizjologią neuronalną, również rzuciły nowe światło na wiele rodzajów ludzkiej inteligencji. Wiemy teraz na przykład, że inteligentna funkcja człowieka nie jest monolityczna i jednolita. Jest raczej modułowy i rozproszony. Jego moc jest widoczna w narządach zmysłów, takich jak ludzka siatkówka, które mogą wyświetlać i wstępnie przetwarzać informacje wizualne. Podobnie ludzkie uczenie się nie jest jednolitą i jednorodną zdolnością. Uczenie się jest raczej funkcją wielu środowisk i różnych systemów, z których każdy jest przystosowany do osiągnięcia wyspecjalizowanych celów. Analiza fMRI, wraz ze skanami PET, EEG i pokrewnymi procedurami fizycznego obrazowania neuronowego, wszystko to potwierdza zróżnicowany i oparty na współpracy obraz wewnętrznego działania rzeczywistych inteligentnych systemów. Jeśli praca w AI ma osiągnąć poziom nauki, musimy też zająć się ważnymi kwestiami filozoficznymi, zwłaszcza tymi związanymi z epistemologią, czy pytaniem, jak inteligentny system „zna" swój świat. Kwestie te rozciągają się od pytania o to, co jest przedmiotem badań sztucznej inteligencji, do głębszych zagadnień, takich jak kwestionowanie ważności i użyteczności hipotezy o fizycznym systemie symboli. Dalsze pytania dotyczą tego, czym jest „symbol" w podejściu systemu symboli do sztucznej inteligencji i jak symbole mogą odnosić się do zbiorów ważonych węzłów w modelu koneksjonistycznym. Kwestionujemy również rolę racjonalizmu wyrażonego w indukcyjnym uprzedzeniu obserwowanym w większości programów nauczania i jak to się ma do nieskrępowanego braku struktury, często obserwowanego w niekontrolowanych, wzmacniających i pojawiających się

podejściach do uczenia się. Wreszcie, musimy zakwestionować rolę ucieleśnienia, usytuowania i socjologicznych uprzedzeń w rozwiązywaniu problemów. Kończymy naszą dyskusję na temat zagadnień filozoficznych, proponując konstruktywistyczną epistemologię, która pasuje do naszego zaangażowania zarówno w sztuczną inteligencję jako naukę, jak i do sztucznej inteligencji jako badań empirycznych. Dlatego w tym ostatniej części ponownie powrócimy do pytań zadanych w Części 1: Czym jest inteligencja? Czy można to sformalizować? Jak możemy zbudować mechanizmy, które to przejawiają? Jak sztuczna i ludzka inteligencja może wpasować się w szerszy kontekst nauki o inteligentnych systemach? W sekcji 16.1 zaczynamy od poprawionej definicji sztucznej inteligencji, która pokazuje, jak obecna praca w sztucznej inteligencji, choć zakorzeniona w hipotezie Newella i Simona dotyczącej fizycznego systemu symboli, rozszerzyła zarówno jej narzędzia, techniki, jak i badania w znacznie szerszym kontekście. Badamy te alternatywne podejścia do kwestii inteligencji i rozważamy ich możliwości w projektowaniu inteligentnych maszyn oraz jako składnik nauki o systemach inteligencji. W sekcji 16.2 wskazujemy, ile technik współczesnej psychologii poznawczej, neuronauki, a także epistemologii można wykorzystać do lepszego zrozumienia przedsięwzięcia związanego ze sztuczną inteligencją. Wreszcie, w sekcji 16.3 omawiamy niektóre wyzwania, które pozostają zarówno dla współczesnych praktyków sztucznej inteligencji, jak i dla epistemologów. Chociaż tradycyjne podejście do sztucznej inteligencji często było winne racjonalistycznego redukcjonizmu, nowe interdyscyplinarne spostrzeżenia i narzędzia również mają powiązane wady. Na przykład twórcy algorytmu genetycznego i projektanci badań nad życiem definiują świat inteligencji z darwinowskiego punktu widzenia: „Co jest, jest tym, co przetrwa”. Wiedza jest również postrzegana jako „wiedzieć jak”, a nie „wiedzieć co” w złożonym świecie. Dla naukowca odpowiedzi wymagają wyjaśnień, a „sukces” czy „przetrwanie” same w sobie nie wystarczą. W tym ostatnim rozdziale omówimy przyszłość sztucznej inteligencji, badając pytania filozoficzne, którymi należy się zająć, aby stworzyć obliczeniową naukę o inteligencji. Dochodzimy do wniosku, że metodologia empiryczna sztucznej inteligencji jest ważnym i być może jednym z najlepszych dostępnych narzędzi do badania natury inteligencji.

16.1 Sztuczna inteligencja: zmieniona definicja

16.1.1 Inteligencja i hipoteza systemu symboli fizycznych

Opierając się na naszym doświadczeniu z ostatnich 15 Części, oferujemy zmienioną definicję sztucznej inteligencji: sztuczna inteligencja to badanie mechanizmów leżących u podstaw inteligentnego zachowania poprzez konstrukcję i ocenę artefaktów zaprojektowanych w celu wprowadzenia tych mechanizmów.

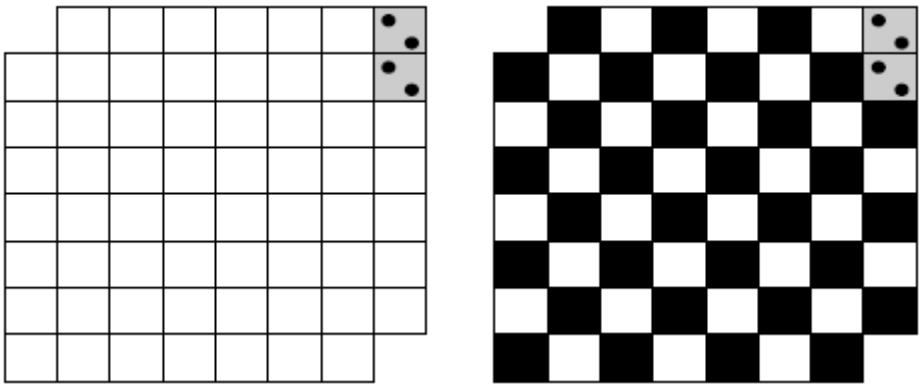
Zgodnie z tą definicją sztuczna inteligencja nie jest teorią dotyczącą mechanizmów leżących u podstaw inteligencji, a bardziej empiryczną metodologią konstruowania i testowania możliwych modeli wspierających taką teorię. Jest to zobowiązanie do naukowej metody projektowania, przeprowadzania i oceny eksperymentów w celu udoskonalenia modelu i dalszego eksperymentowania. Jednak co najważniejsze, ta definicja, podobnie jak sama dziedzina sztucznej inteligencji, bezpośrednio atakuje wieki filozoficznego obskurantyzmu dotyczącego natury umysłu. Daje ludziom, którzy mogliby zrozumieć, co jest być może naszą cechą charakterystyczną jako ludzi, alternatywę dla religii, przesądów, kartezjańskiego dualizmu, new-age placebo lub poszukiwania inteligencji w jeszcze nieodkrytych dziwactwach mechaniki kwantowej. Gdyby nauka wspierająca sztuczną inteligencję w jakikolwiek sposób przyczyniła się do ludzkiej wiedzy, polega na potwierdzeniu, że inteligencja nie jest jakimś mistycznym oparem przenikającym ludzi i anioły, ale raczej efektem zestawu zasad i mechanizmów, które można zrozumieć i stosowane w projektowaniu inteligentnych maszyn. Należy zauważyć, że nasza zmieniona definicja SI nie definiuje inteligencji; proponuje raczej spójną rolę sztucznej inteligencji w badaniu zarówno natury, jak i ekspresji inteligentnych zjawisk. Z perspektywy

historycznej dominujące podejście do sztucznej inteligencji polegało na konstruowaniu formalizmów reprezentacji i związanych z nimi mechanizmów rozumowania opartego na wyszukiwaniu. Wiodącą zasadą wczesnej metodologii sztucznej inteligencji była hipoteza dotycząca fizycznego systemu symboli, po raz pierwszy sformułowana przez Newella i Simona. Ta hipoteza stwierdza:

Warunkiem koniecznym i wystarczającym, aby system fizyczny wykazywał ogólne inteligentne działanie, jest posiadanie fizycznego systemu symboli. Wystarczający oznacza, że inteligencję można osiągnąć za pomocą dowolnego odpowiednio zorganizowanego systemu symboli fizycznych. Niezbędne oznacza, że każdy agent, który wykazuje ogólną inteligencję, musi być instancją fizycznego systemu symboli. Konieczność hipotezy dotyczącej fizycznego systemu symboli wymaga, aby każdy inteligentny agent, czy to człowiek, obcy kosmos czy komputer, osiągnął inteligencję poprzez fizyczną implementację operacji na strukturach symboli. Ogólne inteligentne działanie oznacza ten sam zakres działania, co ludzkie działanie. W granicach fizycznych system zachowuje się odpowiednio do swoich celów i dostosowuje się do wymagań otoczenia.

Newell i Simon podsumowali argumenty przemawiające za koniecznością i wystarczalnością tej hipotezy. W kolejnych latach zarówno sztuczna inteligencja, jak i kognitywistyka badały terytorium wyznaczone przez tę hipotezę. Hipoteza dotycząca fizycznego systemu symboli doprowadziła do czterech istotnych zobowiązań metodologicznych: (a) użycie symboli i systemów symboli jako medium do opisu świata; (b) projektowanie mechanizmów wyszukiwania, zwłaszcza wyszukiwania heurystycznego, w celu zbadania przestrzeni potencjalnych wniosków, które te systemy symboli mogą wspierać; oraz (c) ucieleśnienie architektury poznawczej, przez co rozumiemy, że założono, że odpowiednio zaprojektowany system symboli może zapewnić pełne wyjaśnienie przyczynowe inteligencji, niezależnie od środka jej realizacji. Wreszcie (d), z tego punktu widzenia, sztuczna inteligencja stała się empiryczna i konstruktywistyczna: próbowała zrozumieć inteligencję, budując jej działające modele. W widoku systemu symboli tokeny w języku, zwanym symbolami, były używane do oznaczania lub odniesienia do czegoś innego niż one same. Podobnie jak symbole słowne w języku naturalnym, symbole reprezentowały rzeczy w świecie inteligentnych agentów lub odnosiły się do nich. Tarski (1956, sekcja 2.3) mógłby zaproponować naukę o znaczeniu tych relacji obiekt-odniesienie. Co więcej, użycie symboli przez sztuczną inteligencję wykracza poza kwestie poruszane w semantyce Tarskiana, rozszerzając symbole tak, aby reprezentowały wszystkie formy wiedzy, umiejętności, intencji i przyczynowości. Takie konstruktywne wysiłki polegają na tym, że symbole wraz z ich semantyką, można osadzić w systemach formalnych. Definiują one język reprezentacji. Zdolność do sformalizowania modeli symbolicznych jest niezbędna do modelowania inteligencji jako działającego programu komputerowego. W poprzednich częściach szczegółowo przeanalizowaliśmy kilka reprezentacji: rachunek predykatów, sieci semantyczne, skrypty, grafy pojęciowe, ramki i obiekty. Matematyka systemów formalnych pozwala nam dyskutować o takich kwestiach, jak poprawność, kompletność i złożoność, a także omawiać organizację struktur wiedzy. Ewolucja formalizmów reprezentacyjnych pozwoliła nam ustanowić bardziej złożone (bogatsze) relacje semantyczne. Na przykład systemy dziedziczenia stanowią semantyczną teorię wiedzy taksonomicznej. Formalnie definiując dziedziczenie klas, takie języki uprościć tworzenie inteligentnych programów i dostarcz testowalne modele samej organizacji możliwych kategorii inteligencji. Pojęcie poszukiwania jest ściśle związane ze schematami reprezentacji i ich wykorzystaniem w rozumowaniu. Poszukiwanie stało się krok po kroku badaniem stanów problemowych w ramach przestrzeni stanów (zobowiązanie semantyczne a priori) w poszukiwaniu rozwiązań, celów podproblemu, symetrii problemu lub jakiegokolwiek innego aspektu problemu, który może być rozważany. Reprezentacja i wyszukiwanie są ze sobą powiązane, ponieważ zobowiązanie do określonej reprezentacji określa przestrzeń do przeszukania. Rzeczywiście, niektóre problemy mogą być trudniejsze, lub nawet niemożliwe, z powodu złego wyboru języka reprezentacji. Omówienie błędu indukcyjnego w dalszej części ilustruje ten punkt.

Dramatycznym i często przytaczanym przykładem tej wzajemnej zależności między poszukiwaniem a reprezentacją, a także trudnością w wyborze odpowiedniej reprezentacji (czy ten proces optymalizacji wyboru reprezentacji można zautomatyzować?) Jest problem umieszczania domina na ściętej szachownicy. Załóżmy, że mamy szachownicę i zestaw domina, tak aby każde domino pokryło dokładnie dwa pola szachownicy. Załóżmy również, że na planszy brakuje kilku kwadratów; na Rysunku 1 lewy górny róg i prawy dolny róg zostały usunięte.



Zadanie z szachownicą ściętą polega na pytaniu, czy istnieje sposób na umieszczenie domina na szachownicy tak, aby każde pole szachownicy było obrócone, a każde domino obejmowało dokładnie dwa pola. Możemy spróbować rozwiązać ten problem, wypróbowując wszystkie układy domina na szachownicy; jest to oczywiste podejście oparte na wyszukiwaniu i jest naturalną konsekwencją przedstawiania planszy jako prostej macierzy, pomijając takie pozornie nieistotne cechy, jak kolor kwadratów. Złożoność takiego wyszukiwania jest ogromna i wymagałaby heurystyki dla skutecznego rozwiązania. Na przykład możemy przyciąć częściowe rozwiązania, które pozostawiają pojedyncze kwadraty w izolacji. Moglibyśmy również zacząć od rozwiązania problemu dla mniejszej płytki, takiej jak 2 × 2 i 3 × 3 i spróbuj rozszerzyć rozwiązanie na sytuację 8 × 8. Bardziej wyrafinowane rozwiązanie, opierające się na bardziej złożonym schemacie reprezentacji, zwraca uwagę, że każde umieszczenie domina musi obejmować zarówno czarny, jak i biały kwadrat. Ta ścięta plansza ma 32 czarne kwadraty, ale tylko 30 białych kwadratów; w ten sposób pożądane umieszczenie nie będzie możliwe. Rodzi to poważne pytanie dotyczące wyłącznie symboli rozumując: czy mamy reprezentacje, które pozwalają rozwiązującym problemy dostęp do wiedzy z takim stopniem elastyczności i kreatywności? W jaki sposób dana reprezentacja może automatycznie zmienić swoją strukturę, gdy dowiadujemy się więcej o domenie problemowej? Heurystyki są trzecim, obok reprezentacji i wyszukiwania, składnikiem opartym na symbolach AI. Heurystyka to mechanizm organizowania wyszukiwania w ramach alternatyw oferowanych przez daną reprezentację. Heurystyki mają na celu przezwyciężenie złożoności wyczerpujących poszukiwań, stanowiących barierę dla użytecznych rozwiązań dla wielu klas interesujących problemów. W komputerach, podobnie jak w przypadku ludzi, inteligencja wymaga świadomego wyboru „co dalej robić”. W całej historii badań nad sztuczną inteligencją heurystyki przybierały różne formy. Najwcześniejsze techniki rozwiązywania problemów, takie jak wspinaczka górska w grze w szachownicę Samuela lub analiza średnich i końcowych w Newell, Shaw i Simon’s General Problem Solver , weszły do AI z innych dyscyplin, takich jak badania operacyjne, i stopniowo dojrzewały do ogólnych technik rozwiązywania problemów AI. Właściwości wyszukiwania, w tym dopuszczalność, monotoniczność i poinformowanie, są ważnymi wynikami tych wczesnych badań. Techniki te są często określane jako metody słabe. Słabe metody były ogólnymi strategiami wyszukiwania, które miały być stosowane w całych klasach dziedzin problemowych. Widzieliśmy te metody i ich właściwości w Częściach 2, 3, 4, 6 i 14. Wprowadziliśmy solidne metody rozwiązywania problemów AI z systemem eksperckim opartym na regułach, wnioskach opartych na modelach i

przypadkach oraz uczeniu się opartym na symbolach. W przeciwieństwie do słabych metod rozwiązywania problemów, silne metody koncentrują się na informacjach specyficznych dla każdego obszaru problemowego, takich jak medycyna wewnętrzna lub rachunek całkowy, a nie na projektowaniu metod heurystycznych, które uogólniają obszary problemowe. Silne metody leżą u podstaw systemów eksperckich i innych opartych na wiedzy podejść do rozwiązywania problemów. Silne metody kładą nacisk na takie kwestie, jak ilość wiedzy potrzebnej do rozwiązywania problemów, uczenia się i nabywania wiedzy, syntaktyczna reprezentacja wiedzy, zarządzanie niepewnością oraz kwestie związane z jakością wiedzy. Dlaczego nie stworzyliśmy wielu naprawdę inteligentnych systemów opartych na symbolach? Istnieje wiele zarzutów, które można zrównać z cechą inteligencji w fizycznym systemie symboli. Większość z nich można uchwycić rozważając kwestie znaczenia semantycznego i podstawy symboli inteligentnego agenta. Natura „znaczenia" oczywiście wpływa również na koncepcję inteligencji jako przeszukiwania zinterpretowanych struktur symboli i „użyteczności" ukrytej w stosowaniu heurystyk. Pojęcie znaczenia w tradycyjnej sztucznej inteligencji jest w najlepszym przypadku bardzo słabe. Co więcej, pokusa przejścia w kierunku semantyki bardziej opartej na matematyce, takiej jak podejście Tarskiana do możliwych światów, wydaje się chybiona. Wzmacnia racjonalistyczny projekt zastąpienia elastycznej i ewoluującej inteligencji wcielonego agenta światem, w którym jasne i wyraźne idee są bezpośrednio dostępne. Uziemienie znaczenia to kwestia, która od zawsze frustrowała zarówno zwolenników, jak i krytyków sztucznej inteligencji i przedsiębiorstw kognitywnych. Kwestia uziemienia pyta, jak symbole mogą mieć znaczenie. Właśnie to zwraca uwagę Searle, omawiając tak zwany pokój chiński. Searle umieszcza się w pokoju przeznaczonym do tłumaczenia chińskich zdań na angielski; tam otrzymuje zestaw chińskich symboli, wyszukuje symbole w dużym chińskim systemie katalogowania symboli, a następnie wyświetla odpowiednio połączone zestawy angielskich symboli. Searle twierdzi, że chociaż on sam absolutnie nie zna chińskiego, jego „system" może być postrzegany jako maszyna do tłumaczenia z chińskiego na angielski. Tu jest problem. Chociaż każdy, kto pracował w dziedzinie badań nad tłumaczeniem maszynowym lub rozumieniem języka naturalnego , może argumentować, że „maszyna tłumacząca" Searle'a, ślepo łącząc jeden zestaw symboli z innym zestawem symboli, dawałaby wyniki o minimalnej jakości, Faktem jest, że wiele obecnych inteligentnych systemów ma bardzo ograniczoną zdolność interpretowania zestawów symboli w „znaczący" sposób. Ten problem zbyt słabej semantyki wspierającej przenika również wiele modalności sensorycznych opartych na obliczeniach, czy to wizualnych, kinestetycznych czy werbalnych. W obszarach rozumienia języka ludzkiego Lakoff i Johnson (1999) argumentują, że zdolność tworzenia, używania, wymiany i interpretowania symboli znaczeniowych pochodzi z ludzkiego ucieleśnienia w zmieniającym się kontekście społecznym. Ten kontekst jest fizyczny, społeczny i teraz; wspiera i umożliwia człowiekowi przetrwanie, ewolucję i reprodukcję. To sprawia, że możliwy świat analogicznego rozumowania, używania i doceniania humoru oraz doświadczeń muzycznych i artystycznych. Nasza obecna generacja narzędzi i technik sztucznej inteligencji jest rzeczywiście bardzo daleka od możliwości kodowania i wykorzystywania jakiegokolwiek równoważnego systemu „znaczeń". Bezpośrednim rezultatem tego słabego kodowania semantycznego jest tradycyjne wyszukiwanie / heurystyka sztucznej inteligencji bada stany i konteksty stanów, które są wstępnie interpretowane. Oznacza to, że twórca programu AI „przypisuje" lub „nakłada" na symbole programu różne konteksty znaczenia semantycznego. Bezpośrednim rezultatem tego wstępnie zinterpretowanego kodowania jest to, że zadania bogate w inteligencję, w tym uczenie się i język, mogą wytworzyć tylko pewną obliczoną funkcję tej interpretacji. Tak więc wiele systemów AI ma bardzo ograniczone możliwości ewolucji nowych skojarzeń znaczeniowych podczas eksploracji ich otoczenia. Wreszcie, jako bezpośredni rezultat naszych obecnych ograniczonych możliwości modelowania semantycznego, te aplikacje, w których jesteśmy w stanie oderwać się od bogatego kontekstu ucieleśnionego i społecznego, a jednocześnie uchwycić podstawowe elementy rozwiązywania problemów za pomocą wstępnie zinterpretowanych systemów symboli, są naszymi

najbardziej udane przedsięwzięcia. Wiele z nich zostało omówionych w tej książce. Jednak nawet te obszary pozostają kruche, bez wielu interpretacji i tylko z ograniczoną zdolnością do automatycznego powrotu do zdrowia po awarii. W swojej krótkiej historii społeczność badawcza zajmująca się sztuczną inteligencją badała konsekwencje hipotezy dotyczącej fizycznego systemu symboli i opracowała własne wyzwania dla tego wcześniej dominującego poglądu. Jak zilustrowano w dalszych rozdziałach tej książki, jawny system symboli i wyszukiwanie nie są jedynymi możliwymi reprezentatywnymi mediami do przechwytywania inteligencji. Modele obliczeniowe oparte na architekturze mózgu zwierzęcia, a także na procesach ewolucji biologicznej zapewniają również przydatne ramy do zrozumienia inteligencji w kategoriach procesów poznanych naukowo i odtwarzalnych empirycznie. W następnych sekcjach tego ostatniego rozdziału zbadamy konsekwencje tych podejść. Istotną alternatywą dla hipotezy dotyczącej fizycznego systemu symboli są badania sieci neuronowych i innych biologicznie inspirowanych modeli obliczeń. Na przykład sieci neuronowe są obliczeniowymi i fizycznie utworzonymi modelami poznania, które nie są całkowicie zależne od wyraźnie wymienionych i zinterpretowanych symboli, które charakteryzują świat. Ponieważ „wiedza" w sieci neuronowej jest rozproszona w strukturach tej sieci, często trudno jest, jeśli nie jest to niemożliwe, wyodrębnić poszczególne pojęcia do określonych węzłów i wag sieci. W rzeczywistości każda część sieci może odgrywać zasadniczą rolę w przedstawianiu różnych koncepcji. W konsekwencji sieci neuronowe oferują kontrprzykład, przynajmniej do niezbędnej klauzuli hipotezy fizycznego systemu symboli.

Sieci neuronowe i architektury genetyczne przenoszą nacisk na sztuczną inteligencję z problemów reprezentacji symbolicznej i strategii wnioskowania dźwiękowego na kwestie uczenia się i adaptacji. Sieci neuronowe, podobnie jak ludzie i inne zwierzęta, są mechanizmami dostosowywania się do świata: struktura wyszkolonej sieci neuronowej jest kształtowana zarówno przez uczenie się, jak i przez projekt. Inteligencja sieci neuronowej nie wymaga przekształcenia świata w wyraźny model symboliczny. Sieć jest raczej kształtowana przez jej interakcje ze światem, odzwierciedlone przez ukryte ślady doświadczenia. Podejście to wniosło szereg przyczyn do naszego zrozumienia inteligencji, dając nam wiarygodny model mechanizmów leżących u podstaw fizycznego ucieleśnienia procesów umysłowych, bardziej realistyczny opis uczenia się i rozwoju, demonstrację zdolności do prostych i lokalnych adaptacji kształtować złożony system w odpowiedzi na rzeczywiste zjawiska. Wreszcie, oferują potężne narzędzie badawcze dla neuronauki poznawczej. Właśnie dlatego, że są tak różne, sieci neuronowe mogą odpowiedzieć na wiele pytań, które mogą wykraczać poza możliwości ekspresji sztucznej inteligencji opartej na symbolach. Ważna klasa takich pytań dotyczy percepcji. Natura nie jest tak hojna, aby przekazywać nasze spostrzeżenia do systemu przetwarzania jako zgrabne zestawy wyrażeń rachunku predykatów. Sieci neuronowe oferują model tego, jak możemy rozpoznać „znaczące" wzorce w chaosie bodźców zmysłowych. Ze względu na ich rozproszoną reprezentację sieci neuronowe są często bardziej niezawodne niż ich jawnie symboliczne odpowiedniki. Właściwie wyszkolona sieć neuronowa może skutecznie kategoryzować nowe instancje, wykazując raczej ludzkie postrzeganie podobieństwa niż ścisłą logiczną konieczność. Podobnie utrata kilku neuronów nie musi poważnie wpływać na wydajność dużej sieci neuronowej. Wynika to z często rozległej redundancji właściwej dla modeli sieciowych. Być może najbardziej pociągającym aspektem sieci koneksjonistycznych jest ich zdolność uczenia się. Zamiast próbować zbudować szczegółowy symboliczny model świata, sieci neuronowe polegają na plastyczności własnej struktury, aby dostosować się bezpośrednio do doświadczeń zewnętrznych. Nie tyle konstruują model świata, ile kształtują ich doświadczenia w świecie. Uczenie się jest jednym z najważniejszych aspektów inteligencji. Jest to również problem uczenia się, który rodzi jedne z najtrudniejszych pytań do pracy w komputerach neuronowych.

Dlaczego nie zbudowaliśmy mózgu?

W rzeczywistości obecna generacja skonstruowanych systemów koneksjonistycznych ma bardzo niewielkie podobieństwo do ludzkiego układu neuronowego! Ponieważ temat wiarygodności neuronowej jest krytycznym zagadnieniem badawczym, zaczynamy od tego pytania, a następnie rozważamy rozwój i uczenie się. Badania neuronauki poznawczej (Squire i Kosslyn 1998, Gazzaniga 2000, Hugdahl i Davidson 2003) wnoszą nowy wgląd w zrozumienie ludzkiej architektury poznawczej. Opisujemy pokrótce niektóre ustalenia i komentujemy ich związek z przedsiębiorstwem AI. Rozpatrujemy zagadnienia z trzech poziomów: pierwszy, neuron, drugi, poziom architektury neuronowej, a na końcu omawiamy reprezentację poznawczą lub problem kodowania. Po pierwsze, na poziomie pojedynczego neuronu identyfikują Shephard i Carlson wiele różnych typów architektur neuronowych dla komórek, z których każda jest wyspecjalizowana pod względem funkcji i roli w większym systemie neuronowym. Te typy obejmują komórki receptorów czuciowych, które zwykle znajdują się w skórze i przekazują informacje wejściowe do innych struktur komórkowych, interneurony, których głównym zadaniem jest komunikowanie się w klastrach komórek, podstawowe neurony, których zadaniem jest komunikacja między skupiskami komórek, oraz neurony ruchowe, których zadaniem jest wyjście systemu . Aktywność neuronowa jest elektryczna. Wzory jonów wpływających do i wychodzących z neuronu określają, czy neuron jest aktywny, czy spoczywa. Typowy neuron ma ładunek spoczynkowy -70mV. Kiedy komórka jest aktywna, z zakończenia aksonu uwalniane są określone substancje chemiczne. Te chemikalia, zwane neuroprzekaźnikami, wpływają na błonę postsynaptyczną, zwykle dopasowując się do określonych miejsc receptora, jak klucz do zamka, inicjując dalsze przepływy jonów. Przepływy jonów, gdy osiągną poziom krytyczny, około -50mV, wytwarzają potencjał czynnościowy, całkowicie lub żaden mechanizm wyzwalający wskazujący, że ogniwo zostało uruchomione. W ten sposób neurony komunikują się poprzez sekwencje kodów binarnych. Zmiany postsynaptyczne w zakresie potencjału czynnościowego są dwojakiego rodzaju: hamujące, występujące głównie w strukturach komórek międzyneuronowych lub pobudzające. Te dodatnie i ujemne potencjały są stale generowane w synapsach w układzie dendrytycznym. Ilekroć efektem netto wszystkich tych zdarzeń jest zmiana potencjałów błonowych powiązanych neuronów z -70 mV do około -50 mV, próg zostaje przekroczony i masowe przepływy jonów są ponownie inicjowane do aksonów tych komórek. Po drugie, na poziomie architektury neuronalnej w korze mózgowej znajduje się łącznie około 1010 neuronów, cienka pofałdowana warstwa pokrywająca całą półkulę mózgową. Duża część kory jest zwinięta, co zwiększa całkowitą powierzchnię. Z obliczeniowego punktu widzenia musimy znać nie tylko całkowitą liczbę synaps, ale także parametry wlotu i wylotu. Shephard (1998) szacuje obie te liczby na około 105. Wreszcie, poza różnicami w komórkach i architekturach systemów neuronowych i komputerowych, istnieje głęboki problem reprezentacji poznawczej. Nie wiemy na przykład, jak nawet proste wspomnienia są zakodowane w korze. Na przykład, jak rozpoznaje się twarz i jak rozpoznanie twarzy może łączyć agenta z uczuciem radości lub smutku. Wiemy bardzo dużo o fizycznych / chemicznych aspektach mózgu, ale stosunkowo niewiele o tym, jak system nerwowy koduje i wykorzystuje „wzorce” w swoim kontekście. Jednym z trudniejszych pytań, przed którymi stają badacze, zarówno w społecznościach neuronowych, jak i komputerowych, jest rola wiedzy wrodzonej w uczeniu się: czy efektywne uczenie się może kiedykolwiek nastąpić na tabula rasa lub pustej karcie, zaczynając bez wstępnej wiedzy i ucząc się całkowicie na podstawie doświadczenia ? A może uczenie się musi zaczynać się od uprzedniego uprzedzenia indukcyjnego? Doświadczenie w projektowaniu programów uczenia maszynowego sugeruje, że do uczenia się w złożonych środowiskach niezbędna jest jakaś wcześniejsza wiedza, zwykle wyrażana jako błąd indukcyjny. Zdolność sieci koneksjonistów do konwergencji w sensownym uogólnieniu z zestawu danych szkoleniowych okazała się wrażliwa na liczbę sztucznych neuronów, topologię sieci i określone algorytmy uczenia się. Razem czynniki te stanowią tak silne obciążenie indukcyjne, jakie można znaleźć w dowolnej reprezentacji symbolicznej. Badania nad rozwojem człowieka potwierdzają ten wniosek. Jest na przykład coraz więcej dowodów na to, że ludzkie niemowlęta dziedziczą szereg

„zakodowanych” uprzedzeń poznawczych, które umożliwiają uczenie się takich dziedzin pojęć, jak język i fizyka zdrowego rozsądku. Charakterystyka wrodzonych uprzedzeń w sieciach neuronowych jest aktywnym obszarem badań. Kwestia wrodzonych uprzedzeń staje się jeszcze bardziej zagmatwana, gdy rozważymy bardziej złożone problemy w nauce. Załóżmy na przykład, że opracowujemy obliczenia model odkrycia naukowego i chcemy modelować przejście Kopernika z geocentrycznego do heliocentrycznego widzenia wszechświata. Wymaga to przedstawienia w programie komputerowym zarówno poglądów kopernikańskich, jak i ptolemejskich. Chociaż moglibyśmy przedstawić te poglądy jako wzorce aktywacji w sieci neuronowej, nasze sieci nie powie nam nic o ich zachowaniu jako teoriach. Zamiast tego wolimy wyjaśnienia, takie jak „Kopernik był zaniepokojony złożonością systemu ptolemejskiego i woleliśmy prostszy model pozwalający planetom obracać się wokół Słońca”. Takie wyjaśnienia wymagają symboli. Oczywiście sieci koneksjonistyczne muszą być w stanie wspierać rozumowanie symboliczne; w końcu istoty ludzkie są sieciami neuronowymi i wydają się dość dobrze manipulować symbolami. Jednak neuronowe podstawy rozumowania symbolicznego są ważnym i otwartym problemem badawczym. Kolejnym problemem jest rola rozwoju w nauce. Ludzkie dzieci nie mogą po prostu uczyć się na podstawie dostępnych danych. Ich zdolność uczenia się w określonych dziedzinach pojawia się na dobrze zdefiniowanych etapach rozwojowych. Interesującym pytaniem jest, czy ten postęp rozwojowy jest wyłącznie wynikiem ludzkiej biologii i ucieleśnienia, czy też odzwierciedla pewne logicznie konieczne ograniczenia zdolności inteligencji do uczenia się niezmienności w jej świecie. Czy etapy rozwojowe mogą funkcjonować jako mechanizm dekompozycji problemu poznawania świata na łatwiejsze do opanowania podproblemy? Czy seria sztucznie narzuconych ograniczeń rozwojowych może zapewnić sztucznym sieciom ramy niezbędne do uczenia się o złożonym świecie? Zastosowanie sieci neuronowych do praktycznych problemów rodzi szereg dodatkowych problemów badawczych. Same właściwości sieci neuronowych, które czynią je tak atrakcyjnymi, takie jak zdolność adaptacji i odporność w świetle brakujących lub niejednoznacznych danych, również stwarzają problemy w ich praktycznym zastosowaniu. Ponieważ sieci są uczone, a nie programowane, trudno jest przewidzieć zachowanie. Istnieje kilka wskazówek dotyczących projektowania sieci, które będą odpowiednio zbieżne w danej dziedzinie problemowej. Wreszcie wyjaśnienia, dlaczego sieć doszła do określonego wniosku, są często trudne do skonstruowania i mogą przybrać formę argumentu statystycznego. To są wszystkie obszary obecnych badań. Można więc zapytać, czy sieci koneksjonistyczne i bardziej symboliczna sztuczna inteligencja są tak różne, jak modele inteligencji. Obaj mają wiele wspólnych cech, zwłaszcza że inteligencja jest ostatecznie zakodowana jako obliczenia i ma fundamentalne i formalne ograniczenia, takie jak hipoteza Church / Turinga. Oba podejścia oferują również modele umysłu ukształtowane przez zastosowanie do praktycznych problemów. Co jednak najważniejsze, oba podejścia zaprzeczają filozoficznemu dualizmowi i umieszczają podstawy inteligencji w strukturze i działaniu fizycznie zrealizowanych urządzeń. Uważamy, że pełne pogodzenie tych dwóch bardzo różnych podejść do zdobywania informacji jest nieuniknione. Kiedy to zostanie osiągnięte, teoria, w jaki sposób symbole mogą zredukować się do wzorców w sieci, a co za tym idzie, wpłynąć na przyszłą adaptację tej sieci, będzie niezwykłym wkładem. Wesprze to szereg zmian, takich jak integracja opartych na sieci percepcji i opartych na wiedzy narzędzi wnioskowania w jedną inteligencję. W międzyczasie jednak obie społeczności badawcze mają dużo pracy do wykonania i nie widzimy powodu, dla którego nie miałyby dalej współistnieć. Dla tych, którzy czują się nieswojo z dwoma pozornie niewspółmiernymi modelami inteligencji, nawet fizyka dobrze funkcjonuje z intuicyjnie sprzecznym poglądem, że światło jest czasem najlepiej rozumiane jako fala, a czasami jako cząstka, chociaż oba punkty widzenia można z powodzeniem podciągnąć do teorii strun .

16.1.3 Agenci, pojawienie się i inteligencja

Obliczenia agentowe i modularne teorie poznania rodzą kolejny zestaw interesujących problemów dla badaczy budujących sztuczną inteligencję. Jedna z ważnych szkół kognitywnych głosi, że umysł jest zorganizowany w zestawy wyspecjalizowanych jednostek funkcjonalnych. Moduły te są specjalistami i wykorzystują szereg wrodzonych struktur i funkcji, od „sztywnego" rozwiązywania problemów po indukcyjne uprzedzenia, które odpowiadają za różnorodność problemów, które jako praktyczni agenci muszą rozwiązać. Ma to sens: w jaki sposób pojedyncza sieć neuronowa lub inny system może być wyszkolony do obsługi tak różnych funkcji, jak percepcja, kontrola motoryczna, pamięć i rozumowanie wyższego poziomu? Modułowe teorie inteligencji zapewniają zarówno ramy dla odpowiedzi na te pytania, jak i kierunek dalszych badań nad takimi kwestiami, jak natura wrodzonych uprzedzeń w poszczególnych modułach, a także mechanizmy interakcji modułów. Genetyczne i powstające modele obliczeń oferują jedno z najnowszych i najbardziej ekscytujących podejść do zrozumienia zarówno ludzkiej, jak i sztucznej inteligencji. Wykazując, że globalnie inteligentne zachowanie może wynikać ze współpracy dużej liczby ograniczonych, niezależnych i indywidualnych agentów, teorie genetyczne i emergentne postrzegają złożone wyniki poprzez wzajemne powiązania stosunkowo prostych struktur. Na przykładzie Holandii mechanizmy utrzymujące duże miasto, takie jak Nowy Jork, zaopatrywane w chleb, pokazują podstawowe procesy leżące u podstaw pojawienia się inteligencji w systemie agentowym. Jest mało prawdopodobne, abyśmy mogli napisać scentralizowany planista, który z powodzeniem zaopatrywałby nowojorczyków w bogatą gamę pieczywa codziennego, do którego są przyzwyczajeni. Rzeczywiście, niefortunny eksperyment komunistycznego świata z centralnym planowaniem ujawnił ograniczenia takiego podejścia! Jednak pomimo praktycznych trudności związanych z napisaniem algorytmu scentralizowanego planowania, który zapewni zaopatrzenie Nowego Jorku w chleb, luźno skoordynowane wysiłki wielu piekarzy, kierowców ciężarówek, dostawców surowców, a także sprzedawców detalicznych w mieście rozwiązują problem całkowicie. ładnie. Podobnie jak we wszystkich nowych systemach opartych na agentach, nie ma centralnego planu. Żaden piekarz nie ma bardzo ograniczonej wiedzy na temat zapotrzebowania miasta na chleb; każdy piekarz stara się po prostu zoptymalizować swoje własne możliwości biznesowe. Rozwiązanie problemu globalnego wyłania się ze zbiorowych działań tych niezależnych i lokalnych agentów. Pokazując, jak bardzo ukierunkowane na cel, solidne, prawie optymalne zachowania mogą wynikać z interakcji lokalnych, indywidualnych agentów, modele te dostarczają kolejnej odpowiedzi na stare filozoficzne pytania dotyczące pochodzenia umysłu.

Podejście do inteligencji jest takie, że pełna inteligencja może i rzeczywiście powstaje w wyniku interakcji wielu prostych, indywidualnych, lokalnych i wcielonych inteligencji agentowych. Drugą główną cechą wyłaniających się modeli jest ich poleganie na doborze darwinowskim jako podstawowym mechanizmie kształtującym zachowanie poszczególnych czynników. Na przykładzie piekarni wydaje się, że każdy piekarz nie zachowuje się w sposób, który jest w pewnym sensie optymalny globalnie. Raczej źródłem ich optymalności nie jest centralny projekt; to prosty fakt, że piekarze, którzy słabo radzą sobie z zaspokajaniem potrzeb lokalnych klientów, generalnie zawodzą. To dzięki niestrudzonym, wytrwałym działaniom tych wybiórczych nacisków poszczególni piekarze dochodzą do zachowań, które prowadzą do ich indywidualnego przetrwania, a także do pożytecznych, wyłaniających się zbiorowych zachowań. Połączenie rozproszonej architektury opartej na agentach i adaptacyjnych presji doboru naturalnego to potężny model pochodzenia i działania umysłu. Psychologowie ewolucyjni dostarczyli modelu sposobu, w jaki dobór naturalny ukształtował ewolucję wrodzonej struktury i uprzedzeń w ludzkim umyśle. Podstawą psychologii ewolucyjnej jest pogląd na umysł jako wysoce modułowy, jako system oddziałujących, wysoce wyspecjalizowanych czynników. Rzeczywiście, dyskusje o psychologii ewolucyjnej często porównują umysł do szwajcarskiego scyzoryka, zbioru specjalistycznych narzędzi, które można zastosować do rozwiązywania różnych problemów. Istnieje coraz więcej dowodów na to, że ludzkie umysły są rzeczywiście wysoce modułowe. Fodor

przedstawia filozoficzny argument na rzecz modułowej struktury umysłu. Minsky bada konsekwencje modularnych teorii sztucznej inteligencji. Ta architektura jest ważna dla teorii ewolucji umysłu. Trudno byłoby sobie wyobrazić, jak ewolucja mogłaby ukształtować pojedynczy system tak złożony jak umysł. Jest jednak prawdopodobne, że ewolucja, działająca przez miliony lat, może sukcesywnie kształtować indywidualne, wyspecjalizowane zdolności poznawcze. Wraz z postępem ewolucji mózgu może on również pracować nad kombinacjami modułów, tworząc mechanizmy umożliwiające modułom interakcję, wymianę informacji i współpracę przy wykonywaniu coraz bardziej złożonych zadań poznawczych . Teorie selekcji neuronalnej pokazują, w jaki sposób te same procesy mogą odpowiadać za adaptację indywidualnego układu nerwowego. Darwinizm neuronowy modeluje adaptację systemów neuronowych w kategoriach darwinowskich: wzmacnianie określonych obwodów w mózgu i osłabianie innych jest procesem selekcji w odpowiedzi na świat. W przeciwieństwie do symbolicznych metod uczenia się, które próbują wydobyć informacje z danych treningowych i wykorzystać te informacje do budowy modeli świata, teorie selekcji neuronalnej badają wpływ presji selektywnych na populacje neuronów i ich interakcje. Edelman stwierdza: Rozważając naukę o mózgu jako naukę o rozpoznawaniu, sugeruję, że rozpoznawanie nie jest pouczającym procesem. Nie zachodzi żaden bezpośredni transfer informacji, podobnie jak żaden w procesach ewolucyjnych lub odpornościowych. Zamiast tego rozpoznawanie jest selektywne. Technologie agentowe oferują również modele współpracy społecznej. Korzystając z podejścia opartego na agentach, ekonomiści zbudowali informacyjne (jeśli nie całkowicie predykcyjne) modele rynków ekonomicznych. Technologie agentowe wywierają coraz większy wpływ na projektowanie rozproszonych systemów obliczeniowych, budowę narzędzi wyszukiwania internetowego oraz wdrażanie kooperacyjnych środowisk pracy. Wreszcie modele oparte na agentach wywarły wpływ na teorie świadomości. Na przykład Daniel Dennett oparł opis funkcji i struktury świadomości na agentowej architekturze umysłu. Zaczyna od argumentacji, że niewłaściwe jest pytanie, gdzie w umyśle / mózgu znajduje się świadomość. Zamiast tego, jego wielokrotna szkicowa teoria świadomości skupia się na roli świadomości w interakcjach agentów w rozproszonej architekturze mentalnej. W toku percepcji, kontroli motorycznej, rozwiązywania problemów, uczenia się i innych czynności umysłowych tworzymy koalicje oddziałujących agentów. Koalicje te są bardzo dynamiczne, zmieniające się w odpowiedzi na potrzeby różnych sytuacji. Świadomość, dla Dennetta, służy jako wiążący mechanizm dla tych koalicji, wspierając interakcję agentów i podnosząc krytyczne koalicje oddziałujących agentów na pierwszy plan przetwarzania poznawczego.

Jakie kwestie ograniczają przybliżenie inteligencji oparte na agentach? Podejścia oparte na agentach i „emergentne" otworzyły szereg problemów, które muszą zostać rozwiązane, jeśli ich obietnica ma zostać zrealizowana. Na przykład jeszcze nie wykonaliśmy wszystkich kroków, które umożliwiły ewolucję zdolności poznawczych wyższego poziomu, takich jak język. Podobnie jak wysiłki paleontologów mające na celu odtworzenie ewolucji gatunków, śledzenie rozwoju tych problemów wyższego poziomu będzie wymagało wielu dodatkowych, szczegółowych prac. Musimy oboje wyliczyć czynniki leżące u podstaw architektury umysłu i prześledzić ich ewolucję w czasie.

Innym ważnym problemem związanym z teoriami opartymi na agentach jest wyjaśnienie interakcji między modułami. Chociaż model umysłu „szwajcarskiego scyzoryka" jest użytecznym konstruktorem intuicji, moduły składające się na umysł nie są tak niezależne jak ostrza scyzoryka. Umysły wykazują rozległe, bardzo płynne interakcje między domenami poznawczymi: możemy mówić o rzeczach, które widzimy, wskazując na interakcję między modułami wizualnymi i językowymi. Możemy budować budynki, które umożliwiają określony cel społeczny, wskazując na interakcję między inteligencją techniczną a społeczną. Poeci mogą konstruować dotykowe metafory scen wizualnych, wskazujące na płynną interakcję między wizualnymi i dotykowymi zapachami. Definiowanie reprezentacji i procesów, które umożliwiają takie interakcje międzymodułowe, jest aktywnym obszarem badań. Coraz większe

znaczenie zyskują również praktyczne zastosowania technologii agentowych. Korzystając z symulacji komputerowych opartych na agentach, możliwe jest modelowanie złożonych systemów, które nie mają opisu matematycznego w postaci zamkniętej i których dotychczas nie można było zbadać tak szczegółowo. Techniki oparte na symulacji zastosowano do szeregu zjawisk, takich jak adaptacja ludzkiego układu odpornościowego i kontrola złożonych procesów, w tym akceleratorów cząstek, zachowanie globalnych rynków walutowych oraz badanie systemów pogodowych. Problemy reprezentacyjne i obliczeniowe, które należy rozwiązać, aby zaimplementować takie symulacje, nadal napędzają badania w zakresie reprezentacji wiedzy, algorytmów, a nawet projektowania sprzętu komputerowego. Dalsze praktyczne problemy, z którymi muszą sobie radzić architektury agentów, obejmują protokoły komunikacji między agentami, zwłaszcza gdy lokalni agenci często mają ograniczoną wiedzę o ogólnym problemie lub o wiedzy, którą mogą już posiadać inni agenci. Co więcej, istnieje niewiele algorytmów do dekompozycji większych problemów na spójne podproblemy zorientowane na agentów, czy też faktycznie, jak ograniczone zasoby mogą być rozdzielone między agentów. Te i inne kwestie związane z agentami zostały przedstawione w sekcji 7.4. Być może najbardziej ekscytującym aspektem wyłaniających się teorii umysłu jest ich potencjał do umieszczania czynności umysłowych w jednolitym modelu wyłaniania się porządku z chaosu. Nawet w krótkim przeglądzie przedstawionym w tej sekcji zacytowano prace wykorzystujące wyłaniające się teorie do modelowania szeregu procesów, od ewolucji mózgu w czasie, po siły, które umożliwiają jednostkom uczenie się, budowanie ekonomicznych i społecznych modeli zachowań. Jest coś niezwykle pociągającego w koncepcji, że te same procesy wyłaniającego się porządku, kształtowane przez procesy darwinowskie, mogą wyjaśniać inteligentne zachowanie przy różnych rozdzielczościach, od interakcji poszczególnych neuronów, przez kształtowanie modułowej struktury mózgu, po funkcjonowanie rynków gospodarczych i systemów społecznych. Może się zdarzyć, że inteligencja ma geometrię fraktalną, w której te same wyłaniające się procesy pojawiają się na dowolnym poziomie rozdzielczości, w jakim postrzegamy system jako całość.

16.1.4 Modele probabilistyczne i technologia stochastyczna

Już w latach pięćdziesiątych XX wieku do zrozumienia i generowania wyrażeń w języku naturalnym stosowano techniki stochastyczne. Claude Shannon zastosował modele probabilistyczne, w tym dyskretne łańcuchy Markowa, do zadania przetwarzania języka. Shannon również zapożyczył pojęcie entropii z termodynamiki jako sposób pomiaru pojemności informacyjnej wiadomości. Mniej więcej w tym czasie firma Bell Labs stworzyła pierwszy system statystyczny zdolny do rozpoznawania dziesięciu cyfr, 0, ..., 9, używanych przez jednego mówcę. Działał z dokładnością 97-99%. W latach sześćdziesiątych i siedemdziesiątych bayesowskie podejście do rozumowania było nadal obecne w kontekście działalności badawczej w dziedzinie sztucznej inteligencji. Chociaż wiele systemów eksperckich, na przykład MYCIN, stworzyło własne „algebry czynników pewności", jak widać, kilka, w tym PROSPECTOR, przyjęło podejście bayesowskie. Jednak złożoność takich systemów szybko staje się nie do rozwiązania. Jak wskazaliśmy, pełne wykorzystanie reguły Bayesa do realistycznego programu diagnostyki medycznej obejmującego 200 chorób i 2000 objawów wymagałoby zebrania i zintegrowania ośmiuset milionów informacji. Pod koniec lat osiemdziesiątych Judea Pearl zaproponował obliczeniowy model wnioskowania diagnostycznego w kontekście związków przyczynowych w dziedzinie problemowej: sieci przekonań bayesowskich. BBN rozluźniają dwa ograniczenia pełnego modelu bayesowskiego. Po pierwsze, w wnioskowaniu stochastycznym zakłada się niejawną „przyczynowość"; to znaczy, rozumowanie przechodzi od przyczyny do skutku i nie jest koliste, tj. skutek nie może zawrócić, by spowodować sam siebie. To wspiera reprezentowanie BBN jako skierowanego acyklicznego grafu. Po drugie, BBN zakładają, że bezpośredni rodzic węzła wspiera pełny wpływ przyczynowy na ten węzeł. Zakłada się, że wszystkie inne węzły są warunkowo niezależne lub mają wpływ na tyle mały, że można je zignorować. Badania Pearl odnowiły zainteresowanie

stochastycznymi podejściami do modelowania świata. Jak widzieliśmy , BBN stanowiły bardzo potężne narzędzie reprezentacyjne do wnioskowania diagnostycznego (abdukcyjnego). Jest to prawdziwe zwłaszcza w przypadku dynamicznej natury zarówno systemów ludzkich, jak i stochastycznych: gdy świat zmienia się w czasie, nasze rozumienie jest wzbogacane: niektóre przyczyny okazują się bardziej wyjaśniać to, co widzimy, podczas gdy inne potencjalne przyczyny są „wyjaśniane". Badania nad projektowaniem systemów stochastycznych, a także wspomagające je schematy wnioskowania są dopiero w powijakach. Pod koniec lat 80. nowa energia badawcza została również zastosowana do zagadnień przetwarzania języka naturalnego. Jak wspomniano w sekcji 15.4, te stochastyczne podejścia obejmowały nowe analizowanie, tagowanie i wiele innych technik ujednoznaczniania wyrażeń językowych. Pełen zakres tych podejść można znaleźć w książkach o rozpoznawaniu mowy i zadaniach w przetwarzaniu języka. Wraz z odnowionym zainteresowaniem i sukcesami w stochastycznym podejściu do charakteryzowania inteligentnego zachowania, osoba może w naturalny sposób zastanawiać się, jakie mogą być jego ograniczenia. Czy inteligencja jest zasadniczo stochastyczna?

Istnieje ogromna atrakcyjność w kierunku stochastycznego punktu widzenia interakcji agentów w zmieniającym się świecie. Wielu może argumentować, że „system reprezentacji" człowieka jest zasadniczo stochastyczny, tj. Uwarunkowany światem percepcji i przyczyn, w którym jest pogrążony. Z pewnością behawiorystyczno-empiryczny punkt widzenia uznałby to przypuszczenie za atrakcyjne. Teoretycy usytuowanego i osadzonego działania mogą pójść jeszcze dalej i postawić hipotezę, że uwarunkowane relacje agenta z jego fizycznym i społecznym otoczeniem stanowią wystarczające wyjaśnienie dostosowania się agenta do tego świata. Niektórzy współcześni badacze twierdzą nawet, że aspekty funkcji nerwowych są zasadniczo stochastyczne. Rozważmy język. Jedną z mocnych stron wypowiedzi ustnej i pisemnej, jak wskazali Chomsky i inni, jest jej generatywny charakter. Oznacza to, że w zestawie dostępnych form słownictwa i języka w naturalny sposób pojawiają się nowe i wcześniej niedoświadczone wyrażenia. Dzieje się tak zarówno na poziomie tworzenia nowych zdań, jak i pojedynczych słów, werbalizując np. „Google it!". Stochastyczny opis języka musi wykazać tę generującą moc. Obecnie ograniczenia gromadzonych informacji językowych, czy to banki drzew, czy inne kopory danych, mogą radykalnie ograniczyć wykorzystanie technologii stochastycznej. Dzieje się tak, ponieważ zebrane informacje muszą oferować odpowiednie warunki (lub wcześniejsze) do interpretacji obecnej nowej sytuacji. Stochastyczne modele dziedzin zastosowań, powiedzmy, dla silnika lotniczego lub systemu przekładniowego mogą mieć podobne ograniczenia. Tam z konieczności zawsze będzie istniał zamknięty świat lub minimalne założenia modelu, dla realistycznie złożonego systemu. Jednak dynamiczne sieci bayesowskie dają obecnie nadzieję na interpretację stanu systemu . Jednak nadal istnieją ograniczone możliwości przewidywania nowych sytuacji w czasie. Na wyższym poziomie wyjaśniającym modelowi probabilistycznemu może być trudno wyjaśnić odejście od własnego systemu wyjaśniającego lub paradygmatu. W jakim sensie, model stochastyczny może ewentualnie wyjaśniać teorie lub przeorganizowanie poglądów pojęciowych wyższego poziomu, to znaczy ponownie oceniać kwestie związane z adekwatnością samego modelu, być może z potrzebą przejść do różnych punktów widzenia? Tematy te pozostają ważnymi kwestiami badawczymi i ograniczeniami stochastycznego podejścia do zrozumienia niepewności. Faktem jest jednak, że często bez wyraźnych instrukcji agenci „tworzą" całkiem udane modele. Jak widzimy z konstruktywistycznego punktu widzenia, w sekcji 16.2, pewien model jest warunkiem sine qua non dla podmiotu, aby zrozumieć swój świat, tj. Jeśli nie ma a priori zaangażowania w to, o czym świat „chodzi", zjawiska nie są ani postrzegane, ani rozumiane. ! Oprócz przedstawionych właśnie argumentów filozoficznych, istnieje szereg praktycznych ograniczeń stosowania systemów stochastycznych. Obecna generacja BBN ma charakter propozycyjny. Dopiero niedawno było możliwe tworzenie ogólnych praw lub relacji, takich jak ∀X mężczyzna (X) → inteligentny (X) z dołączoną dystrybucją. Ponadto pożądane jest, aby

BBN zawierały relacje rekurencyjne, zwłaszcza w przypadku analizy szeregów czasowych. Badania mające na celu opracowanie stochastycznych systemów reprezentacji pierwszego rzędu, odnoszących się do tych ogólnych problemów, są ważne dla kontynuowania badań. Jak zauważono w sekcji 13.3, dostępne są teraz narzędzia do modelowania stochastycznego pierwszego rzędu i pełnego Turinga. Ważne jest również zbadanie zastosowania tych ogólnych modeli stochastycznych do danych neuro / psychologicznych, tam gdzie zostały one ostatnio zastosowane. Następnie omówimy te psychologiczne i filozoficzne aspekty ludzkiej inteligencji, które mają wpływ na tworzenie, wdrażanie i ocenę sztucznej inteligencji

16.2 Nauka o inteligentnych systemach

To nie przypadek, że duża podgrupa społeczności zajmującej się sztuczną inteligencją skupiła się w swoich badaniach na zrozumieniu ludzkiej inteligencji. Ludzie dostarczają prototypowych przykładów inteligentnej aktywności, a inżynierowie sztucznej inteligencji, chociaż zwykle tak nie jest zaangażowani w „tworzenie programów, które zachowują się jak ludzie", rzadko ignorują ludzkie rozwiązania. Niektóre zastosowania, takie jak rozumowanie diagnostyczne, są często celowo wzorowane na procesach rozwiązywania problemów ekspertów ludzkich pracujących w tej dziedzinie. Co ważniejsze, zrozumienie ludzkiej inteligencji jest samo w sobie fascynującym i otwartym wyzwaniem naukowym. Współczesna kognitywistyka, czyli nauka o systemach inteligentnych (Luger 1994), rozpoczęła się wraz z pojawieniem się komputera cyfrowego, mimo że, jak widzieliśmy w rozdziale 1, było wielu intelektualnych przodków tej dyscypliny, od Arystotelesa po Kartezjusza i Boole'a, bardziej współczesnym teoretykom, takim jak Turing, McCulloch i Pitts, twórcy modelu sieci neuronowej oraz John von Neumann, wczesny zwolennik a-life. Badanie stało się jednak nauką ze zdolnością do projektowania i przeprowadzania eksperymentów w oparciu o te teoretyczne pojęcia, i stało się to w istotnym stopniu wraz z pojawieniem się komputera. Na koniec musimy zapytać: „Czy istnieje wszechstronna nauka o inteligencji?" Możemy dalej zapytać: „Czy nauka o inteligentnych systemach może wspierać konstrukcję sztucznych inteligencji?" W kolejnych sekcjach omówimy pokrótce, w jaki sposób nauki psychologiczne, epistemologiczne i socjologiczne wspierają badania i rozwój w dziedzinie sztucznej inteligencji.

16.2.1 Ograniczenia psychologiczne

Wczesne badania kognitywne badały ludzkie rozwiązania problemów logicznych, prostych gier, planowania i uczenia się koncepcji. Zbiegając się z ich pracą nad teoretykiem logiki, Newell i Simon zaczęli porównywać swoje podejścia obliczeniowe z wyszukiwaniem strategie stosowane przez ludzi. Ich dane składały się z protokołów „myśl na głos", opisów ich myśli przez ludzi podczas procesu opracowywania rozwiązania problemu, na przykład dowodu logicznego. Newell i Simon następnie porównali te protokoły z zachowaniem programu komputerowego rozwiązującego ten sam problem. Naukowcy odkryli niezwykłe podobieństwa i interesujące różnice zarówno między problemami, jak i przedmiotami. Te wczesne projekty ustanowiły metodologię, którą dyscyplina kognitywna miałaby stosować w następnych dziesięcioleciach:

1. Na podstawie danych od ludzi rozwiązujących poszczególne klasy problemów, zaprojektuj schemat reprezentacyjny i powiązaną strategię wyszukiwania w celu rozwiązania problemu.

2. Uruchom model komputerowy, aby prześledzić zachowanie rozwiązania.

3. Obserwuj ludzi pracujących nad tymi samymi problemami i śledź mierzalne parametry ich procesu rozwiązywania, takie jak te znalezione w protokołach myślenia na głos, ruchy oczu i pisemne wyniki cząstkowe.

4. Analizować i porównywać rozwiązania ludzkie i komputerowe.

5. Popraw model komputerowy dla następnej rundy testów i porównań z ludźmi.

Ta empiryczna metodologia jest opisana w wykładzie Newell and Simon's Turing Award, cytowanym na początku tego rozdziału. Ważnym aspektem kognitywistyki jest wykorzystywanie eksperymentów do walidacji architektury rozwiązywania problemów, niezależnie od tego, czy będzie to system produkcyjny, łącznikowy, emergentny, czy też architektura oparta na interakcji rozproszonych agentów. W ostatnich latach do tego paradygmatu nadano zupełnie nowy wymiar. Teraz nie tylko programy można dekonstruować i obserwować podczas rozwiązywania problemów, ale także ludzi i inne formy życia. Szereg nowych technik obrazowania zostało włączonych do narzędzi dostępnych do obserwacji aktywności kory. Należą do nich magnetoencefalografia (MEG), która wykrywa pola magnetyczne generowane przez populacje neuronów. W przeciwieństwie do potencjałów elektrycznych generowanych przez te populacje, pole magnetyczne nie jest rozmazane przez czaszkę i skórę głowy, dzięki czemu możliwa jest znacznie większa rozdzielczość. Drugą technologią obrazowania jest pozytonowa tomografia emisyjna, czyli PET. Substancja radioaktywna, zwykle O15, jest wstrzykiwana do krwiobiegu. Kiedy dany obszar mózgu jest aktywny, więcej tego czynnika przechodzi przez czułe detektory niż w stanie spoczynku. Porównanie obrazów spoczynkowych i aktywnych może potencjalnie ujawnić funkcjonalną lokalizację przy rozdzielczości około 1 cm (patrz Stytz i Frieder 1990). Inną techniką analizy neuronowej jest funkcjonalne obrazowanie metodą rezonansu magnetycznego lub fMRI. Podejście to wyłoniło się z bardziej standardowego obrazowania strukturalnego opartego na magnetycznym rezonansie jądrowym (NMR). Podobnie jak w przypadku PET, podejście to porównuje odpoczynek z aktywnymi stanami neuronalnymi w celu ujawnienia lokalizacji funkcjonalnej. Dalszym wkładem w lokalizację funkcji mózgu, z ważnym powiązaniem z właśnie wspomnianymi technikami obrazowania, są algorytmy oprogramowania opracowane przez Baraka Pearlmuttera i jego współpracowników. Badacze ci są w stanie przyjąć złożone wzorce szumu, często postrzegane jako wynik różnych technik obrazowania neuronowego, i podzielić je na oddzielne komponenty. Jest to niezbędny krok w analizie, ponieważ wzorce normalnego, ustalonego istnienia, takie jak ruchy oczu, oddychanie i bicie serca, przeplatają się z innymi wzorcami odpalania neuronów, które chcemy zrozumieć. Wyniki badań neuronauki poznawczej znacznie poszerzyły naszą wiedzę na temat komponentów neuronalnych zaangażowanych w inteligentną aktywność. Chociaż analiza i krytyka tych wyników wykracza poza zakres tej książki, wymieniamy i odnosimy się do kilku ważnych kwestii: W obszarze percepcji i uwagi istnieje wiążący problem. Badacze tacy jak Anne Triesman i Jeff Hawkins zauważają, że reprezentacje percepcyjne zależą od rozproszonych kodów neuronowych, aby powiązać ze sobą części i właściwości obiektów, i pytają, jaki mechanizm jest potrzebny do „wiązania" informacji dotyczących do każdego przedmiotu i odróżnić go od innych. Jakie mechanizmy neuronowe wspomagają percepcję obiektów osadzonych w dużych, złożonych scenach w obszarze poszukiwań wizualnych? Niektóre eksperymenty pokazują, że tłumienie informacji z nieistotnych obiektów odgrywa ważną rolę w wyborze celów wyszukiwania. Co więcej, jak „uczymy się" widzieć? W obszarze plastyczności w percepcji Gilbert twierdzi, że to, co widzimy, nie jest ściśle odzwierciedleniem fizycznych cech sceny, ale raczej jest wysoce zależne od procesów, za pomocą których nasz mózg próbuje zinterpretować tę scenę. W jaki sposób układ korowy przedstawia i indeksuje informacje związane w czasie, w tym interpretację spostrzeżeń i produkcję aktywności motorycznej? W badaniach nad pamięcią hormony stresu uwalniane w sytuacjach pobudzenia emocjonalnego modulują procesy pamięci. Odnosi się to do problemu uziemienia: w jaki sposób myśli, słowa, spostrzeżenia mają znaczenie dla agenta? W jakim sensie może istnieć „smutek (lub łzy) w rzeczach", lacremae rerum Wergiliusza? Akustyczno-fonetyczne aspekty mowy dostarczają ważnych zasad organizujących łączenie badań neuronauki z teoriami poznawczymi i językowymi. W jaki sposób integruje się składniowe i semantyczne składniki kory?

W jaki sposób jednostka nabywa określony język i jakie etapy neurofizjologiczne wspierają ten rozwój? Jak rozumiany jest rozwój, czym jest plastyczność okresu krytycznego i reorganizacje dorosłych obserwowane w układach somatosensorycznych ssaków? Czy etapy rozwoju są krytyczne dla „budowania" inteligencji? Dalsza dyskusja - patrz Karmiloff-Smith i Gazzaniga. Praktyka sztucznej inteligencji z pewnością nie wymaga rozległej wiedzy na temat tych i pokrewnych dziedzin neuro / psychologicznych. Jednak ten rodzaj wiedzy może wspierać inżynierię inteligentnych artefaktów, a także pomóc zlokalizować badania i rozwój w dziedzinie sztucznej inteligencji w kontekście szerszej nauki o systemach inteligencji. Wreszcie tworzenie psychologicznej, neurofizjologicznej i obliczeniowej syntezy jest naprawdę ekscytujące. Ale to wymaga dojrzałej epistemologii, którą omówimy dalej.

16.2.2 Zagadnienia epistemologiczne

Rozwój sztucznej inteligencji został ukształtowany przez szereg ważnych wyzwań i pytań. Rozumienie języka naturalnego, planowanie, rozumowanie w niepewnych sytuacjach i uczenie maszynowe są typowe dla tych typów problemów, które obejmują pewien istotny aspekt inteligentnego zachowania. Co ważniejsze, inteligentne systemy działające w każdej z tych dziedzin wymagają znajomości celu, praktyki i działania w kontekstach usytuowanych i osadzonych społecznie. Aby lepiej zrozumieć te kwestie, przeanalizujemy epistemologiczne zaangażowanie programu, który ma być „inteligentny". Zaangażowanie epistemologiczne odzwierciedla zarówno semantykę wspierającą użycie symbolu, jak i strukturę użytych symboli. Zadaniem w takich sytuacjach jest odkrywanie i wykorzystanie niezmienności istniejące w domenie problemowej. „Niezmienny" to termin używany do opisania prawidłowości lub znaczących aspektów złożonych środowisk, które można manipulować. W niniejszej dyskusji terminy symbole i systemy symboli są używane ogólnie, od wyraźnych symboli tradycji Newella i Simona, węzły i architektura sieci systemu łącznikowego, wyłaniające się symbole genetycznego i sztucznego życia. Chociaż punkty, które omówimy w następnej kolejności, dotyczą większości sztucznej inteligencji, skupimy się na naszej dyskusji na temat zagadnień związanych z uczeniem maszynowym, ponieważ w tej książce stworzyliśmy wiele przykładów i algorytmów uczenia się. Mimo postępów w uczeniu maszynowym pozostaje jednym z najtrudniejszych problemów w obliczu sztucznej inteligencji. Istnieją trzy kwestie ograniczające nasze obecne rozumienie i postęp w badaniach: po pierwsze, problem uogólnienia i przeuczenia, po drugie, rola indukcyjnego uprzedzenia w uczeniu się, a po trzecie, dylemat empirysta lub zajęcie się ideą nauka bez ograniczeń. Ostatnie dwa problemy są ze sobą powiązane: ukryte indukcyjne nastawienie wielu algorytmów uczenia się jest wyrazem problemu racjonalistów polegającego na tym, że jesteśmy uprzedzeni przez oczekiwania, to znaczy to, czego się uczymy, często wydaje się być bezpośrednią funkcją tego, czego się spodziewamy się nauczyć. Z przeciwnego punktu widzenia, jak widzieliśmy w badaniach a-life, gdzie istnieje bardzo niewiele oczekiwań a priori co do tego, czego należy się nauczyć, czy naprawdę wystarczy powiedzieć: „Zbuduj to, a to się stanie"? Zgodnie z przypisem Yogi Berra na początku tej sekcji, prawdopodobnie nie będzie! W następnych sekcjach pokrótce omówiono te kwestie.

Problem uogólnienia

Przykłady, których użyliśmy do wprowadzenia różnych modeli uczenia się - opartych na symbolach, koneksjonistycznych i emergentnych - były zwykle bardzo ograniczone. Na przykład architektury koneksjonistyczne często zawierały tylko kilka węzłów lub jedną częściowo ukrytą warstwę. Jest to właściwe, ponieważ główne prawa uczenia się można odpowiednio wyjaśnić w kontekście kilku neuronów i częściowych warstw. Może to być bardzo mylące, ponieważ aplikacje sieci neuronowych są zwykle znacznie większe, a problem skali JEST ważny. Na przykład, w przypadku uczenia się z propagacją wsteczną, do rozwiązywania problemów o istotnym znaczeniu praktycznym wymagana jest duża liczba przykładów szkoleniowych i większe sieci. Wielu badaczy obszernie komentuje kwestię doboru odpowiedniej liczby wartości wejściowych, stosunku między parametrami wejściowymi a

ukrytymi węzłami oraz prób treningowych niezbędnych do osiągnięcia konwergencji. W rzeczywistości jakość i ilość danych szkoleniowych są ważnymi kwestiami dla każdego algorytmu uczenia się. Bez odpowiedniej stronniczości lub obszernej wbudowanej wiedzy algorytm uczenia się może zostać całkowicie wprowadzony w błąd, próbując znaleźć wzorce w hałaśliwych, rzadkich lub nawet złych danych.

Podobnym problemem jest kwestia „wystarczalności" w nauce. Kiedy możemy powiedzieć, że nasze algorytmy są wystarczające do wychwycenia ważnych ograniczeń lub niezmienników domeny problemowej? Czy rezerwujemy część naszych oryginalnych danych do testowania naszych algorytmów uczenia się? Czy ilość danych, które posiadamy, ma związek z jakością uczenia się? Być może ocena wystarczalności jest heurystyczna lub estetyczna: my, ludzie, często postrzegamy nasze algorytmy jako „wystarczająco dobre". Zilustrujmy ten problem uogólnienia na przykładzie, używając formy propagacji wstecznej w celu wywołania funkcji ogólnej ze zbioru punktów danych. Rysunek 16.2 może przedstawiać punkty danych, o których uogólnienie prosimy nasz algorytm. Linie w tym zestawie punktów reprezentują funkcje wywołane przez algorytm uczący się. Pamiętaj, że po przeszkoleniu algorytmu będziemy chcieli zaoferować mu nowe punkty danych i poprosić algorytm o wygenerowanie dobrego uogólnienia również dla tych danych. Indukowana funkcja f1 może reprezentować dość dokładne dopasowanie najmniejszych średnich kwadratów. Przy dalszym szkoleniu system może wytworzyć f2, co wydaje się dość „dobre" dopasować do zbioru punktów danych; ale nadal f2 nie przechwytuje dokładnie punktów danych. Dalsze szkolenie może wytworzyć funkcje, które dokładnie pasują do danych, ale mogą dać straszne uogólnienia dla dalszych danych wejściowych. Zjawisko to określa się jako przetrenowanie sieci. Jedną z mocnych stron uczenia się z propagacją wsteczną jest to, że w wielu dziedzinach aplikacji znane jest tworzenie skutecznych uogólnień, to jest przybliżeń funkcjonalnych, które dobrze pasują do danych uczących, a także odpowiednio obsługują nowe dane. Jednak określenie punktu, w którym sieć przechodzi ze stanu niedoświadczonego do przetrenowanego, jest nietrywialne. Naiwnością jest myślenie, że można przedstawić sieć neuronową lub jakiekolwiek inne narzędzie edukacyjne z surowymi danymi, a następnie po prostu odejść na bok i obserwować, jak generuje najbardziej efektywne i użyteczne uogólnienia dotyczące rozwiązywania nowych podobnych problemów. Kończymy, umieszczając tę kwestię uogólnienia z powrotem w jej epistemologicznym kontekście.

Kiedy osoby rozwiązujące problemy tworzą i wykorzystują reprezentacje (symbole, węzły sieci lub cokolwiek innego) w procesie rozwiązywania, tworzą niezmienniki i najprawdopodobniej systemy niezmienników do badania domeny problemu / rozwiązania i tworzenia powiązanych uogólnień. Ten właśnie punkt widzenia wniesiony do procesu rozwiązywania problemów wpływa na ostateczny sukces przedsięwzięcia. W następnym podrozdziale zajmiemy się dalej tą kwestią. Indukcyjne nastawienie, a priori racjonalistów Zautomatyzowane uczenie się w rozdziałach od 10 do 13, a także w większości technik sztucznej inteligencji, odzwierciedlało uprzedzenia a priori ich twórców. Problem błędu indukcyjnego polega na tym, że wynikowe reprezentacje i strategie wyszukiwania oferują medium do kodowania już zinterpretowanego świata. Rzadko oferują mechanizmy kwestionowania naszych interpretacji, generowania nowych punktów widzenia lub cofania się i zmieniania perspektyw, gdy są nieproduktywne. Ta ukryta stronniczość prowadzi do racjonalistycznej epistemologicznej pułapki widzenia świata dokładnie i tylko tego, czego oczekujemy lub jesteśmy przygotowani, aby zobaczyć. W każdym paradygmacie uczenia się należy jasno określić rolę błędu indukcyjnego. Co więcej, tylko dlatego, że nie uznaje się błędu indukcyjnego, nie oznacza to, że nie istnieje i ma krytyczny wpływ na parametry uczenia się. W uczeniu się opartym na symbolach odchylenie indukcyjne jest zwykle oczywiste, na przykład użycie sieci semantycznej do uczenia się koncepcji. W algorytmach uczenia Winstona, uprzedzenia obejmują reprezentację związku koniunkcyjnego oraz znaczenie używania „bliskich trafień" dla udoskonalenia ograniczeń. Widzimy podobne błędy w używaniu określonych

predykatów do przeszukiwania przestrzeni wersji, drzew decyzyjnych w ID3 .Na przykład ograniczenia sieci perceptronowych doprowadziło do wprowadzenia ukrytych węzłów. Możemy zapytać, jaki wkład mają ukryte węzły w generowanie rozwiązania. Jednym ze sposobów zrozumienia roli ukrytych węzłów jest to, że dodają one wymiary do przestrzeni reprezentacji. Jako prosty przykład widzieliśmy w sekcji 11.3.3, że punkty danych dla problemu wykluczającego lub problemu nie można było rozdzielić liniowo w dwóch wymiarach. Wyuczona waga ukrytego węzła nadaje reprezentacji inny wymiar. W trzech wymiarach punkty można rozdzielić za pomocą dwuwymiarowej płaszczyzny. Biorąc pod uwagę dwa wymiary przestrzeni wejściowej i ukrytego węzła, warstwę wyjściową tej sieci można następnie postrzegać jako zwykły perceptron, który znajduje płaszczyznę oddzielającą punkty w trójwymiarowej przestrzeni. Uzupełniająca się perspektywa polega na tym, że wiele „różnych" paradygmatów uczenia się ma wspólne (czasami nieoczywiste) wspólne uprzedzenia indukcyjne. Wskazaliśmy na wiele z nich: związek między grupowaniem z CLUSTER / 2 w sekcji 10.5, perceptronem w sekcji 11.2 i sieciami prototypowymi w sekcji 11.3. Zauważyliśmy, że kontrpropagacja, sprzężona sieć, która wykorzystuje nienadzorowane konkurencyjne uczenie się w warstwie Kohonena wraz z nadzorowanym uczeniem się języka Hebba w warstwie Grossberga, jest pod wieloma względami podobna do uczenia się wstecznego. W kontrpropagacji skupione dane na warstwie Kohonena odgrywają rolę podobną do uogólnień poznanych przez ukryte węzły, które używają propagacji wstecznej. Pod wieloma ważnymi względami prezentowane przez nas narzędzia są podobne. W rzeczywistości nawet odkrycie prototypów reprezentujących klastry danych stanowi uzupełniający przypadek przybliżenia funkcji. W pierwszej sytuacji próbujemy sklasyfikować zbiory danych; w drugiej generujemy funkcje, które jawnie dzielą od siebie skupiska danych. Widzieliśmy to, gdy algorytm klasyfikacji minimalnej odległości użyty przez perceptron podał również parametry określające liniową separację danych. Nawet uogólnienia, które tworzą funkcje, można postrzegać z wielu różnych punktów widzenia. Na przykład techniki statystyczne od dawna umożliwiają odkrywanie korelacji danych. Do przybliżenia większości funkcji można użyć iteracyjnego rozwinięcia szeregu Taylora. Algorytmy aproksymacji wielomianów są używane od ponad wieku do aproksymacji funkcji dopasowywania punktów danych. Podsumowując, zobowiązania podjęte w ramach schematu uczenia się, niezależnie od tego, czy są oparte na symbolach, koneksjonistyczne, emergentne czy stochastyczne, w bardzo dużym stopniu wpływają na wyniki, można oczekiwać po wysiłku rozwiązywania problemów. Kiedy doceniamy ten efekt synergii w całym procesie projektowania komputerowych rozwiązań do rozwiązywania problemów, często możemy zwiększyć nasze szanse na sukces, a także dokładniej zinterpretować nasze wyniki.

Dylemat empirysty

Jeśli obecne podejścia do uczenia maszynowego, zwłaszcza uczenia się nadzorowanego, mają dominującą tendencję indukcyjną, uczenie się nienadzorowane, w tym wiele podejść genetycznych i ewolucyjnych, musi zmierzyć się z przeciwnym problemem, czasami nazywanym dylematem empirysta. Tematy tych obszarów badawczych obejmują: pojawią się rozwiązania, ewoluują alternatywy, populacje odzwierciedlają przetrwanie najlepiej przystosowanych. Jest to potężna rzecz, szczególnie umieszczona w kontekście równoległych i rozproszonych możliwości wyszukiwania. Ale jest problem: skąd możemy wiedzieć, że jesteśmy gdzieś, skoro nie jesteśmy pewni, dokąd zmierzamy? Platon, ponad 2000 lat temu, postawił ten problem słowami niewolnika Menona: A jak możesz dociekać, Sokratesie, o to, czego jeszcze nie wiesz? Co podasz jako przedmiot zapytania? A jeśli dowiesz się, czego chcesz, skąd będziesz wiedzieć, że tego nie wiedziałeś? Kilku badaczy wykazało, że Meno miał rację, i twierdzenia Wolperta i Macready'eg o zakazie obiadu. W rzeczywistości empirysta potrzebuje resztek a priori racjonalistów, aby ocalić naukę! Niemniej jednak istnieje wielkie podekscytowanie nienadzorowanymi i ewolucyjnymi modelami uczenia się; na przykład w tworzeniu sieci opartych na wzorach lub minimalizacji energii, które można postrzegać jako atraktory punktów stałych lub baseny dla złożonych niezmienności relacyjnych. Obserwujemy, jak punkty danych

„osiedlają się” w kierunku atraktorów i mamy pokusę, by postrzegać te nowe architektury jako narzędzia do modelowania zjawisk dynamicznych. Jakie, możemy zapytać, są granice obliczeń w tych paradygmatach?

W rzeczywistości badacze wykazali, że sieci rekurencyjne są obliczeniowo kompletne, to znaczy równoważne klasie maszyn Turinga. Ta równoważność Turinga rozszerza wcześniejsze wyniki: Kołmogorow wykazał, że dla każdej funkcji ciągłej istnieje sieć neuronowa, która oblicza tę funkcję. Wykazano również, że sieć propagacji wstecznej z jedną warstwą ukrytą może przybliżyć dowolną z bardziej ograniczonej klasy funkcji ciągłych. Podobnie, widzieliśmy w sekcji 11.3, że von Neumann stworzył automaty skończone, które były kompletne według Turinga. Tak więc sieci łącznikowe i automaty skończone wydają się być tylko dwiema innymi klasami algorytmów zdolnych do obliczania praktycznie dowolnej funkcji obliczalnej. Co więcej, biasy indukcyjne DOTYCZY mają zastosowanie do nienadzorowanych, a także genetycznych i nowych modeli uczenia się; reprezentacyjne odchylenia dotyczą projektowania węzłów, sieci i genomów, a algorytmiczne błędy dotyczą operatorów wyszukiwania, nagrody i selekcji. Co zatem mogą zaoferować uczący się bez nadzoru, czy to koneksjoniści, genetycy, czy też ewoluujące maszyny skończone w różnych formach?

1. Jedną z najbardziej atrakcyjnych cech uczenia się koneksjonistów jest to, że większość modeli opiera się na danych lub przykładach. Oznacza to, że nawet jeśli ich architektury są wyraźnie zaprojektowane, uczą się na przykładzie, generalizując dane z określonej domeny problemowej. Jednak wciąż pojawia się pytanie, czy dane są wystarczające lub wystarczająco czyste, aby nie zakłócać procesu rozwiązywania. A skąd projektant może wiedzieć?

2. Algorytmy genetyczne wspierają również potężne i elastyczne przeszukiwanie przestrzeni problemowej. Wyszukiwanie genetyczne jest napędzane zarówno przez różnorodność wymuszoną przez mutację, jak i przez operatory, takie jak krzyżowanie i inwersja, które zachowują ważne aspekty informacji rodziców dla przyszłych pokoleń. W jaki sposób projektant programu może zachować i pielęgnować ten kompromis między różnorodnością a ochroną?

3. Algorytmy genetyczne i architektury koneksjonistyczne można postrzegać jako przykłady przetwarzania równoległego i asynchronicznego. Czy rzeczywiście zapewniają wyniki poprzez równoległe asynchroniczne wysiłki, które nie są możliwe w przypadku jawnego programowania sekwencyjnego?

4. Chociaż inspiracja neuronalna i socjologiczna nie jest ważna dla wielu współczesnych praktyków uczenia się koneksjonizmu i genetyki, techniki te odzwierciedlają wiele ważnych aspektów naturalnej ewolucji i selekcji. Widzieliśmy modele uczenia się z redukcją błędów z perceptronem, wsteczną propagacją i modelami Hebbian. W sekcji 11.3.4 widzieliśmy również autosocjatywne sieci Hopfielda. Różne modele ewolucji znalazły odzwierciedlenie w paradygmatach Części 12.

5. Wreszcie, wszystkie paradygmaty uczenia się są narzędziami do badania empirycznego. Gdy wychwytujemy niezmienniki naszego świata, czy nasze narzędzia są wystarczająco potężne i ekspresyjne, aby zadawać dalsze pytania dotyczące natury percepcji, uczenia się i rozumienia?

W następnej części proponujemy, aby konstruktywistyczna epistemologia, w połączeniu z eksperymentalnymi metodami współczesnej sztucznej inteligencji.

Zbliżenie konstruktywisty

Konstruktywiści stawiają hipotezę, że wszelkie rozumienie jest wynikiem interakcji między wzorcami energii w świecie a kategoriami umysłowymi narzuconymi światu przez inteligentnego agenta. Korzystając z opisów Piageta, asymilujemy zjawiska zewnętrzne zgodnie z naszym obecnym

rozumieniem i dostosowujemy nasze rozumienie do „wymagań" zjawisk. Konstruktywiści często używają terminu schemata do opisania struktury a priori używanej do organizowania doświadczenia świata zewnętrznego. Termin ten pochodzi od brytyjskiego psychologa Bartletta, a jego filozoficzne korzenie sięgają Kanta. Z tego punktu widzenia obserwacja nie jest bierna i neutralna, ale aktywna i interpretująca. Postrzegana informacja, wiedza Kanta a posteriori, nigdy nie pasuje dokładnie do naszych z góry przyjętych, a priori, schematów. W wyniku tego napięcia uprzedzenia oparte na schemacie, których podmiot używa do organizowania doświadczenia, są modyfikowane lub zastępowane. Potrzeba akomodacji w obliczu nieudanych interakcji z otoczeniem napędza proces równowagi poznawczej. Zatem konstruktywistyczna epistemologia jest zasadniczo ewolucją i udoskonalaniem poznawczym. Ważną konsekwencją konstruktywizmu jest to, że interpretacja każdej sytuacji wiąże się z narzuceniem pojęć i kategorii obserwatora rzeczywistości (uprzedzenie indukcyjne). Kiedy Piaget zaproponował konstruktywistyczne podejście do rozumienia, nazwał je epistemologią genetyczną. Brak wygodnego dopasowania aktualnych schematów do świata „takiego, jakim jest", stwarza napięcie poznawcze. To napięcie napędza proces rewizji schematu. Rewizja schematu, akomodacja Piageta, jest ciągłą ewolucją rozumienia agenta w kierunku równowagi. Rewizja schematu i ciągły ruch w kierunku równowagi jest genetyczną predyspozycją czynnika dostosowującego się do struktur społeczeństwa i świata. Łączy obie te siły i reprezentuje ucieleśnioną predyspozycję do przetrwania. Modyfikacja schematu jest zarówno a priori wynikiem naszej genetyki, jak i funkcją a posteriori społeczeństwo i świat. Odzwierciedla ucieleśnienie agenta nastawionego na przetrwanie, istoty w czasie i przestrzeni. Występuje tu mieszanka tradycji empirystycznej i racjonalistycznej, w której pośredniczy cel przetrwania agentów. Wcielone podmioty nie mogą pojąć niczego poza tym, co najpierw przechodzi przez ich zmysły. Jako przychylni agenci przetrwają dzięki poznaniu ogólnych wzorców świata zewnętrznego. To, co jest postrzegane, jest zapośredniczane przez to, czego się oczekuje; to, czego się oczekuje, zależy od tego, co jest postrzegane: te dwie funkcje można zrozumieć tylko w kategoriach siebie. W tym sensie modele stochastyczne - zarówno bayesowskie, jak i markowskie - są właściwe, ponieważ wcześniejsze doświadczenie warunkuje obecne interpretacje. Wreszcie, my, jako agenci, rzadko zdajemy sobie sprawę ze schematów, które wspierają nasze interakcje ze światem. Jako źródła uprzedzeń i uprzedzeń zarówno w nauce, jak iw społeczeństwie, często jesteśmy świadomi a priori schemata. Są one konstytutywne dla naszej równowagi ze światem, a nie (zwykle) dostrzegalnym elementem świadomego życia psychicznego. Wreszcie, dlaczego konstruktywistyczna epistemologia jest szczególnie użyteczna w rozwiązywaniu problemów związanych ze zrozumieniem inteligencji? W jaki sposób agent w środowisku może zrozumieć swoje własne rozumienie tej sytuacji? Uważamy, że konstruktywizm odnosi się również do problemu epistemologicznego dostępu zarówno w filozofii, jak i psychologii. Od ponad wieku toczy się walka w obu tych dyscyplinach między dwiema frakcjami, pozytywistą, który proponuje wnioskowanie o zjawiskach psychicznych na podstawie obserwowalnych zachowań fizycznych, oraz podejściem bardziej fenomenologicznym, które pozwala na wykorzystanie raportów z pierwszej osoby w celu uzyskania dostępu do zjawisk poznawczych. Ten frakcyjność istnieje, ponieważ oba sposoby dostępu do zjawisk psychologicznych wymagają jakiejś formy konstrukcji modelu i wnioskowania. W porównaniu z przedmiotami fizycznymi, takimi jak krzesła i drzwi, które często naiwnie wydają się być bezpośrednio dostępne, stany psychiczne i dyspozycje agenta wydają się szczególnie trudne do scharakteryzowania. W istocie twierdzimy, że ta dychotomia między bezpośrednim dostępem do zjawisk fizycznych a pośrednim dostępem do mentalności jest iluzoryczna. Analiza konstruktywistyczna sugeruje, że żadne doświadczenie rzeczy nie jest możliwe bez użycia jakiegoś modelu lub schematu organizacji tego doświadczenia. W badaniach naukowych, jak również w naszych normalnych ludzkich doświadczeniach, sugeruje to, że wszelki dostęp do zjawisk odbywa się poprzez eksplorację, przybliżanie i ciągłe udoskonalanie modelu.ferowała narzędzia i techniki do kontynuowania eksploracji nauki o inteligentnych systemach.

Jaki jest więc projekt praktyka AI?

Jako praktycy AI jesteśmy konstruktywistami. Budujemy, testujemy i udoskonalamy modele. Ale co przybliżamy w naszym budowaniu modeli? Omówimy tę kwestię w następnych akapitach, ale najpierw dokonamy obserwacji epistemologicznej: Zamiast próbować uchwycić istotę „rzeczy poza nami", rozwiązaniu problemów AI najlepiej służy próba naśladowania budowy modelu, udoskonalania i heurystyki równoważenia samego inteligentnego agenta. Tylko skrajny solipsysta (lub umysłowo upośledzony) zaprzeczyłby „rzeczywistości" świata pozaprzedmiotowego. Ale co to jest tak zwany „prawdziwy świat"? Oprócz tego, że jest złożoną kombinacją „rzeczy twardych" i „rzeczy miękkich", jest to także układ atomów, cząsteczek, kwarków, grawitacji, teorii względności, komórek, DNA i (być może nawet) superstrun. Wszystkie te koncepcje są jedynie modelami eksploracyjnymi napędzanymi przez wyjaśniające wymagania czynników opartych na równościach. Ponownie, te modele eksploracyjne nie dotyczą świata zewnętrznego. Raczej chwytają dynamiczne, równoważące napięcia inteligentnego i społecznego sprawcy, materialnej inteligencji ewoluującej i nieustannie kalibrującej się w przestrzeni i czasie. Ale dostęp do i tworzenie „rzeczywistości" jest również osiągane poprzez zaangażowanie agentów. Wcielony podmiot tworzy rzeczywistość poprzez egzystencjalną afirmację, że postrzegany model jego oczekiwań jest wystarczająco dobry, aby zaspokoić niektóre z jego praktycznych potrzeb i celów. Ten akt zaangażowania ugruntowuje symbole i systemy symboli, których agent używa w swoim materialnym i społecznym kontekście. Konstrukty te są ugruntowane, ponieważ są uznane za wystarczająco dobre, aby osiągnąć pewne aspekty swojego celu. To uziemienie jest również widoczne w używaniu języka agentów. Searle ma rację w swoim pojmowaniu zjawisk mowy jako aktów. Ta kwestia uziemienia jest jednym z powodów, dla których komputery mają fundamentalne problemy z przejawami inteligencji, w tym z demonstracjami języka i uczenia się. Jaką dyspozycję można dać komputerowi, który zapewni mu odpowiednie cele i cele? Chociaż Dennett przypisywał uziemienie komputerowi rozwiązującemu problemy wymagające i używające „inteligencji", brak wystarczającego uziemienia jest łatwo dostrzegalny w uproszczeniach komputera, kruchości i często ograniczonej ocenie kontekstu. Użycie i uziemienie symboli przez ożywionych agentów oznacza jeszcze więcej. Szczegóły dotyczące ucieleśnienia i kontekstów społecznych agenta ludzkiego pośredniczą w jego interakcjach ze światem. Systemy słuchowe i wizualne wrażliwe na określone pasmo; postrzeganie świata jako wyprostowanego dwunożnego, mającego ręce, nogi, dłonie; przebywanie w świecie z pogodą, porami roku, słońcem i ciemnością; część społeczeństwa o zmieniających się celach i celach; osoba, która rodzi się, rozmnaża i umiera: są to krytyczne elementy wspierające metafory rozumienia, uczenia się i języka; one pośredniczą w naszym pojmowaniu sztuki, życia i miłości.

Czy mam cię porównać do letniego dnia?

Jesteś piękniejszy i bardziej umiarkowany:

Ostry wiatr wstrząsa ukochanymi pąkami maja,

A letnia dzierżawa ma zbyt krótką datę ...

Szekspirowski Sonet XVIII Kończymy podsumowaniem krytycznych zagadnień, które wspierają i ograniczają nasze wysiłki w tworzeniu nauki o inteligentnych systemach.

16.3 AI: obecne wyzwania i przyszłe kierunki

Chociaż wykorzystanie technik sztucznej inteligencji do rozwiązywania problemów praktycznych dowiodło swojej użyteczności, wykorzystanie tych technik do stworzenia ogólnej nauki o inteligencji jest trudnym i ciągłym problemem. W tej ostatniej części wracamy do pytań, które doprowadziły nas

do wejścia w dziedzinę sztucznej inteligencji i do napisania tej książki: czy możliwe jest formalne, obliczeniowe ujęcie procesów, które umożliwiają inteligencję? Obliczeniowa charakterystyka inteligencji rozpoczyna się od abstrakcyjnej specyfikacji urządzeń obliczeniowych. Badania w latach trzydziestych, czterdziestych i pięćdziesiątych XX wieku zapoczątkowały to zadanie, a Turing, Post, Markov i Church wnieśli wkład w formalizm opisujący obliczenia. Celem tych badań było nie tylko sprecyzowanie, co oznacza obliczenie, ale raczej określenie granic tego, co można obliczyć. Universal Turing Machine jest najczęściej badaną specyfikacją, chociaż reguły przepisywania Posta, podstawa obliczania systemu produkcyjnego (po 1943), są również ważnym wkładem. Model Churcha, oparty na funkcjach częściowo rekurencyjnych, oferuje obsługę nowoczesnych języków funkcjonalnych wysokiego poziomu, takich jak Scheme, Ocaml i Standard ML. Teoretycy udowodnili, że wszystkie te formalizmy mają równoważną moc obliczeniową w tym sensie, że każda funkcja obliczalna przez jednego jest obliczalna przez inne. W rzeczywistości można wykazać, że uniwersalna maszyna Turinga jest odpowiednikiem każdej nowoczesnej maszyny obliczeniowej. Opierając się na tych wynikach, hipoteza Church-Turinga wysuwa jeszcze silniejszy argument: nie można zdefiniować żadnego modelu obliczeń, który byłby silniejszy niż te znane modele. Po ustaleniu równoważności specyfikacji obliczeniowych uwolniliśmy się od środka mechanizacji tych specyfikacji: możemy zaimplementować nasze algorytmy za pomocą lamp próżniowych, krzemu, protoplazmy lub zabawek druciarza. Zautomatyzowany projekt w jednym medium może być postrzegany jako odpowiednik mechanizmów w innym. To sprawia, że metoda badania empirycznego jest jeszcze bardziej krytyczna, ponieważ eksperymentujemy na jednym medium aby sprawdzić nasze zrozumienie mechanizmów zaimplementowanych w innym. Jedną z możliwości jest to, że uniwersalna maszyna Turinga i Posta może być zbyt ogólna. Paradoksalnie, inteligencja może wymagać mniej wydajnego mechanizmu obliczeniowego z bardziej skoncentrowaną kontrolą. Levesque i Brachman zasugerowali ,że inteligencja może wymagać bardziej wydajnych obliczeniowo (chociaż mniej wyrazistych) reprezentacji, takich jak klauzule Horn dla rozumowania, ograniczenie wiedzy faktycznej do literałów naziemnych oraz wykorzystanie obliczeniowo wykonalnych systemów utrzymywania prawdy. Oparte na agentach i wyłaniające się modele inteligencji również wydają się wspierać tę filozofię. Kolejną kwestią, do której odnosi się formalna równoważność naszych modeli mechanizmów, jest kwestia dwoistości i problem ciała i umysłu. Przynajmniej od czasów Kartezjusza , filozofowie postawili pytanie o interakcję i integrację umysłu, świadomości i ciała fizycznego. Filozofowie zaproponowali każdą możliwą odpowiedź, od całkowitego materializmu po zaprzeczenie egzystencji materialnej, a nawet wspierającą interwencję łagodnego boga! Badania nad sztuczną inteligencją i kognitywistyką odrzucają kartezjański dualizm na rzecz materialnego modelu umysłu opartego na fizycznej implementacji lub konkretyzacji symboli, formalnej specyfikacji mechanizmów obliczeniowych służących do manipulacji tymi symbolami, równoważności paradygmatów reprezentacji oraz mechanizacji wiedzy i umiejętności w modelach ucieleśnionych. Sukces tych badań wskazuje na słuszność tego modelu . Wiele dalszych pytań pozostaje jednak w epistemologicznych podstawach inteligencji w systemie fizycznym. Podsumowując: po raz ostatni kilka z tych krytycznych kwestii.

1. Problem reprezentacji. Newell i Simon postawili hipotezę, że fizyczny system symboli i wyszukiwanie są niezbędnymi i wystarczającymi cechami inteligencji (patrz Rozdział 16.1). Czy sukcesy modeli neuronowych lub sub-symbolicznych oraz genetycznych i wyłaniających się podejść do inteligencji obalają hipotezę dotyczącą fizycznego symbolu, czy są to po prostu inne jej przykłady? Nawet słaba interpretacja tej hipotezy - że symbol fizyczny system jest wystarczającym modelem dla inteligencji - przyniósł wiele potężnych i użytecznych wyników w nowoczesnej dziedzinie kognitywistyki. To dowodzi, że możemy zaimplementować fizyczne systemy symboli, które zademonstrują inteligentne zachowanie. Wystarczalność pozwala na tworzenie i testowanie modeli opartych na symbolach dla

wielu aspektów ludzkiej wydajności (Pylyshyn 1984, Posner 1989). Ale silna interpretacja - że fizyczny system symboli i poszukiwania są niezbędne dla inteligentnej aktywności - pozostaje otwarta

2. Rola ucieleśnienia w poznaniu. Jednym z głównych założeń hipotezy systemu symboli fizycznych jest to, że konkretna instancja fizycznego systemu symboli nie ma znaczenia dla jego działania; liczy się tylko jego struktura formalna. Zostało to zakwestionowane przez wielu myślicieli, którzy zasadniczo argumentują, że wymagania inteligentnego działania na świecie wymagają fizycznego ucieleśnienia, które pozwala agentowi na pełną integrację z tym światem. Architektura współczesnych komputerów nie wspiera tego stopnia usytuowania, wymagając od sztucznej inteligencji interakcji ze swoim światem przez niezwykle ograniczone okno współczesnych urządzeń wejścia / wyjścia. Jeśli to wyzwanie jest słuszne, to chociaż jakaś forma inteligencji maszynowej może być możliwa, będzie wymagała zupełnie innego interfejsu niż ten oferowany przez współczesne komputery.

3. Kultura i inteligencja. Tradycyjnie sztuczna inteligencja skupiała się na indywidualnym umyśle jako jedynym źródle inteligencji; zachowywaliśmy się tak, jakby wyjaśnienie sposobu, w jaki mózg koduje wiedzę i manipuluje nią, byłoby całkowitym wyjaśnieniem pochodzenia inteligencji. Moglibyśmy jednak również argumentować, że wiedzę najlepiej postrzegać jako konstrukcję społeczną, a nie indywidualną. W opartej na memach teorii inteligencji (Edelman 1992) samo społeczeństwo posiada podstawowe składniki inteligencji. Możliwe, że zrozumienie społecznego kontekstu wiedzy i ludzkich zachowań jest tak samo ważne dla teorii inteligencji, jak zrozumienie dynamiki indywidualnego umysłu / mózgu.

4. Charakterystyka charakteru interpretacji. Większość modeli obliczeniowych w tradycji reprezentacji działa z już zinterpretowaną domeną: to znaczy, istnieje ukryte i a priori zaangażowanie projektantów systemu w kontekst interpretacyjny. W ramach tego zobowiązania istnieje niewielka możliwość zmiany kontekstów, celów, lub reprezentacje w miarę ewolucji rozwiązywania problemu. Obecnie niewiele jest wysiłku, aby naświetlić proces, za pomocą którego ludzie konstruują interpretacje. Tarskianowski pogląd na semantykę jako odwzorowanie między symbolami i przedmiotami w dziedzinie dyskursu jest z pewnością zbyt słaby i nie wyjaśnia na przykład faktu, że jedna dziedzina może mieć różne interpretacje w świetle różnych celów praktycznych. Lingwiści próbowali zaradzić ograniczeniom semantyki Tarskiana, dodając teorię pragmatyki (Austin 1962). Analiza dyskursu, z jej fundamentalną zależnością od użycia symboli w kontekście, zajmowała się tymi kwestiami w ostatnich latach. Problem jest jednak szerszy, ponieważ dotyczy ogólnej awarii narzędzi referencyjnych. Tradycja semiotyczna zapoczątkowana przez C. S. Peirce'a i kontynuowana przez Eco, Seboek i innych przyjmuje bardziej radykalne podejście do języka. Umieszcza symboliczne wyrażenia w szerszym kontekście znaków i interpretacja znaków. Sugeruje to, że znaczenie symbolu można zrozumieć jedynie w kontekście jego roli interpretatora, czyli w kontekście interpretacji i interakcji z otoczeniem

5. Nieokreśloność reprezentacyjna. Przypuszczenie Andersona o nieokreśloności reprezentacji sugeruje, że w zasadzie niemożliwe może być ustalenie, który schemat reprezentacji najlepiej przybliża człowieka rozwiązującego problemy w kontekście konkretnego aktu umiejętnego wykonania. To przypuszczenie opiera się na fakcie, że każdy schemat reprezentacji jest nierozerwalnie związany z większą architekturą obliczeniową, a także strategiami wyszukiwania. W szczegółowej analizie umiejętności ludzkich może być niemożliwe kontrolowanie procesu na tyle, abyśmy mogli określić reprezentację; lub ustal reprezentację do punktu, w którym proces może być określony w sposób unikalny. Podobnie jak w przypadku zasady nieoznaczoności w fizyce, gdzie zjawiska mogą być zmieniane przez sam proces ich pomiaru, jest to ważna kwestia przy konstruowaniu modeli inteligencji, ale nie musi ograniczać ich użyteczności. Ale co ważniejsze, tę samą krytykę można skierować na sam model obliczeniowy, w którym indukcyjne uprzedzenia symboli i poszukiwań w kontekście hipotezy Churcha-Turinga wciąż ograniczają system. Postrzegana potrzeba jakiegoś optymalnego schematu

reprezentacji może równie dobrze być pozostałością marzenia racjonalisty, podczas gdy naukowiec po prostu potrzebuje modeli dostatecznie solidnych, aby ograniczyć pytania empiryczne. Dowodem jakości modelu jest jego zdolność do interpretacji, przewidywania i rewizji.

6. Konieczność projektowania modeli obliczeniowych, które są falsyfikowalne. Popper i inni argumentowali, że teorie naukowe muszą być falsyfikowalne. Oznacza to, że muszą zaistnieć okoliczności, w których model nie będzie skutecznym przybliżeniem zjawiska. Oczywistym powodem jest to, że jakakolwiek liczba potwierdzających instancji eksperymentalnych nie jest wystarczająca do potwierdzenia modelu. Ponadto wiele nowych badań jest podejmowanych w bezpośredniej odpowiedzi na niepowodzenie istniejących teorii. Ogólny charakter hipotezy fizycznego systemu symboli, a także umiejscowienia a pojawiające się modele inteligencji mogą uniemożliwić ich sfałszowanie, a tym samym ich ograniczone wykorzystanie jako modeli. Ta sama krytyka może dotyczyć przypuszczeń tradycji fenomenologicznej (patrz punkt 7). Niektóre struktury danych AI, takie jak sieć semantyczna, są tak ogólne, że mogą modelować prawie wszystko, co można opisać lub, jak w przypadku uniwersalnej maszyny Turinga, dowolną obliczalną funkcję. Tak więc, gdy badacz AI lub kognitywisty zostanie zapytany, w jakich warunkach jego model inteligencji nie zadziała, odpowiedź może być trudna.

7. Ograniczenia metody naukowej. Wielu badaczy twierdzi, że najważniejsze aspekty inteligencji nie są i w zasadzie nie mogą być modelowane, a w szczególności nie mają reprezentacji symbolicznej. Obszary te obejmują uczenie się, rozumienie języka naturalnego i tworzenie aktów mowy. Kwestie te mają głębokie korzenie w naszej tradycji filozoficznej. Na przykład krytyka Winograda i Floresa opiera się na zagadnieniach fenomenologii .

Większość założeń współczesnej sztucznej inteligencji ma swoje korzenie od Carnapa, Frege i Leibniza, poprzez Hobbesa, Locke'a i Hume'a do Arystotelesa. Ta tradycja twierdzi, że inteligentne procesy są zgodne z uniwersalnymi prawami i są w zasadzie zrozumiałe. Heidegger i jego zwolennicy reprezentują alternatywne podejście do rozumienia inteligencji. Heidegger uważa, że refleksyjna świadomość opiera się na świecie ucieleśnionego doświadczenia (świat życia). To stanowisko, podzielane przez Winograda i Floresa, Dreyfusa i innych, dowodzi, że ludzkie rozumienie rzeczy jest zakorzenione w praktycznej działalności polegającej na „używaniu" ich w radzeniu sobie z codziennym światem. Ten świat jest zasadniczo kontekstem społecznie zorganizowanych ról i celów. Ten kontekst i ludzkie w nim funkcjonowanie nie jest czymś wyjaśnianym przez zdania i rozumianym przez twierdzenia. Jest to raczej strumień, który kształtuje i sam jest nieustannie tworzony. W podstawowym sensie, ludzka wiedza nie polega na wiedzy, ale raczej w świecie ewoluujących norm społecznych i ukrytych celów, wiedząc, jak to zrobić. Z natury nie jesteśmy w stanie umieścić naszej wiedzy i większości naszych inteligentnych zachowań w języku, ani formalnym, ani naturalnym. Rozważmy ten punkt widzenia. Po pierwsze, jako krytyka czystej tradycji racjonalistycznej jest słuszna. Racjonalizm twierdzi, że wszelka ludzka działalność, inteligencja i odpowiedzialność mogą, przynajmniej w zasadzie, być reprezentowane, sformalizowane i zrozumiane. Większość refleksyjnych ludzi nie wierzy, że tak jest, rezerwując ważne role dla emocji, autoafirmacji i odpowiedzialnego zaangażowania (przynajmniej!). Sam Arystoteles powiedział w swoim eseju o racjonalnym działaniu: „Dlaczego nie czuję się zmuszony do wykonania tego, co się z tym wiąże?" Istnieje wiele działań człowieka poza dziedziną nauki, które odgrywają zasadniczą rolę w odpowiedzialnych interakcjach międzyludzkich; nie można ich powielać ani przenosić na maszyny. Mając to jednak na uwadze, naukowa tradycja badania danych, konstruowania modeli, przeprowadzanie eksperymentów i badanie wyników wraz z udoskonalaniem modeli do dalszych eksperymentów przyniosło społeczności ludzkiej istotny poziom zrozumienia, wyjaśnienia i zdolności przewidywania. Metoda naukowa to potężne narzędzie zwiększające ludzkie zrozumienie. Niemniej jednak istnieje wiele zastrzeżeń dotyczących tego podejścia, które naukowcy

muszą zrozumieć. Po pierwsze, naukowcom nie wolno mylić modelu z modelowanym zjawiskiem. Model pozwala nam stopniowo przybliżać zjawisko: z konieczności zawsze będzie istnieć „pozostałość", której nie da się empirycznie wyjaśnić. W tym sensie również nieokreśloność reprezentacji nie jest problemem. Model służy do eksploracji, wyjaśniania i przewidywania; a jeśli pozwoli to naukowcom to osiągnąć, to się powiedzie. Rzeczywiście, różne modele mogą z powodzeniem wyjaśniać różne aspekty zjawiska, takie jak teorie fal i cząstek światła. Co więcej, kiedy badacze twierdzą, że aspekty inteligentnych zjawisk są poza zakresem i metodami tradycji naukowej, samo to stwierdzenie można zweryfikować tylko przy użyciu tej właśnie tradycji. Metoda naukowa jest jedynym narzędziem, jakim dysponujemy, aby wyjaśnić, w jakim sensie kwestie mogą nadal pozostawać poza naszym obecnym zrozumieniem. Każdy punkt widzenia, nawet ten z tradycji fenomenologicznej, jeśli ma mieć jakiekolwiek znaczenie, musi odnosić się do naszych aktualnych pojęć wyjaśniania - a nawet być spójny co do zakresu, w jakim zjawiska nie mogą być wyjaśnione. Najbardziej ekscytującym aspektem pracy nad sztuczną inteligencją jest to, że aby zachować spójność i wnieść wkład w przedsięwzięcie, musimy zająć się tymi kwestiami. Aby zrozumieć rozwiązywanie problemów, uczenie się i język, musimy zrozumieć filozoficzny poziom reprezentacji i wiedzy. W pokorny sposób jesteśmy proszeni o rozwiązanie napięcia Arystotelesa między teoriami i praktyki, aby stworzyć związek zrozumienia i praktyki, teorii i praktyki, by żyć między nauką a sztuką. Praktycy sztucznej inteligencji są twórcami narzędzi. Nasze reprezentacje, algorytmy i języki są narzędziami do projektowania i budowania mechanizmów wykazujących inteligentne zachowanie. Poprzez eksperyment sprawdzamy zarówno ich obliczeniową adekwatność do rozwiązywania problemów, jak i nasze własne rozumienie inteligentnych zjawisk.