SZTUCZNA INTELIGENCJA

(Materiały do wykładu)

dr hab. inż. Wojciech Jędruch
KASK, WETI, Politechnika Gdańska
p. 520, tel. 347 20 03
email: wjed@eti.pg.gda.pl

GDAŃSK marzec 2009

Rozdział 1

Wiadomości wstępne

Skrypt zawiera przegląd wiedzy teoretycznej i metod praktycznych dotyczących konstrukcji i zasad działania komputerowych systemów ze sztuczną inteligencją, a także przedstawia wgląd w dyskusje filozoficzne dotyczące relacji pomiędzy działaniem komputerów i działaniem umysłu.

Dla pełnego zrozumienia prezentowanego materiału wystarcza elementarna znajomość algebry Boole'a, podstaw matematyki wyższej na poziomie kursów na pierwszym roku studiów technicznych, oraz ogólne umiejetności programowania komputerów.

1.1. Zakres materiału

Podstawowe pojęcia Sztuczna inteligencja: definicje, metody i zastosowania, historia rozwoju, filozofia (test Turinga, hipoteza silnej SI, chiński pokój Searlego).

Metody szukania na grafach Metody szukania: wszerz, w głąb, równych kosztów, heurystyczne, na grafach AND-OR, minimax, alfa-beta.

Logika w sztucznej inteligencji Metody automatycznego wnioskowania w rachunku predykatów, zasada rezolucji. Język Prolog jako system wnioskowania.

Wnioskowanie w warunkach niepewności Metody rozmyte i sieci probabilistyczne.

Metody uczenia Uczenie z nauczycielem. Uczenie bez nauczyciela i samoorganizacja. Metody szukanie ekstremum jako algorytmy uczenia. Algorytmy genetyczne. Metody symulowanego wyżarzania. Sieci neuronowe i algorytm propagacji wstecznej. Systemy rozmyto-neuronowe. Uczenie drzew decyzyjnych. Uczenie ze wzmocnieniem (z krytykiem). Metody wydobywania wiedzy.

Systemy z bazą wiedzy Architektura. Metody reprezentacji wiedzy. Pozyskiwanie wiedzy. Strategie wnioskowania. Wiedza rozproszona – neuronowe systemy ekspertowe.

Inteligencja zespołowa Metody indywiduowe modelowania: automaty komórkowe i dynamika molekularna. Sztuczne życie. Inteligencja zespołowa.

Języki programowania Języki stosowane w sztucznej inteligencji i do tworzenia systemów ekspertowych: Prolog, Lisp, Clips

1.1.1. Literatura zalecana

Wymieniono tu pozycje wyróżniające się tym że pokrywają większe fragmenty wykładanego materiału lub są dostępne w języku polskim lub są ogólnie znane i często cytowane. Na końcu opracowania znajduje się obszerniejsza bibliografia zawierająca pozycje cytowane w opracowaniu.

Adami Ch., Introduction to artifial life, Springer, New York 1998.

Arabas J., Wykłady z algorytmów ewolucyjnych. WNT. Warszawa 2001.

Bishop M.C. Neural Networks for Pattern Recognition, Clarendon Press, Oxford, 1996.

Bolc L, Cytowski J., Metody przeszukiwania heurystycznego, t.I i t.II, PWN, Warszawa 1989 i 1991.

Bolc L, Zaremba J., Wprowadzenie do uczenia się maszyn, Akademicka Oficyna Wydawnicza RM, Warszawa 1992.

Bonabeau E., Dorigo M., Theraulaz G., Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York 1999.

Brachman R.J., Levesque H.J., Knowledge representation, Elsevier, Morgan Kaufmann, Amsterdam, 2004.

Buller A., Sztuczny mózg, Prószyński i S-ka, Warszawa 1998.

Casti J.L., Kwintet z Cambridge. Owoc naukowej wyobraźni, Prószyński iSka, W-wa, 2005.

Cichosz P., Systemy uczące się, WNT, Warszawa, 2000.

Engelbrecht A.P., Fundamentals of Computational Swarm Intelligence, J. Wiley & Sons, Chichester, 2005

Franchi S., Guzeldere G., (ed.), Construction of the Mind: Artificial Intelligence and the Humanities, Stanford Humanities Review. v.4. issue 2, 1995, internet: `http://www.stanford.edu/group/SHR/4-2/text/toc.html`

Goldberg D.E., Algorytmy genetyczne i ich zastosowania, WNT, Warszawa 1995.

Hippe Z., Zastosowanie metod sztucznej inteligencji w chemii, PWN, W-wa 1993.

Jang J-S.R., Sun C-T., Mizutani E., Neuro-Fuzzy and Soft Computing, Prentice Hall, Upper Saddle River 1997.

Jędruch W., Turbo Prolog, Wyd. Pol. Gd., Gdańsk 1989.

Jędruch W., Środowisko programowe dla modelowania cząsteczkowego systemów złożonych, Zeszyty Naukowe Politechniki Gdańskiej – monografie, Elektronika, Nr 84, Gdańsk 1997.

Kennedy J., Eberhardt R.C.: Swarm intelligence, Morgan Kaufmann Publishers, San Francisco, 2001.

Kosiński R.A., Sztuczne sieci neuronowe, dynamika nieliniowa i chaos, WNT, Warszawa, 2002.

Kowalski R., Logika w rozwiązywaniu zadań, WNT, Warszawa 1989.

Koza J., Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, Massachusetts, 1992.

Koza J., Genetic Programming II: Automatic Discovery of Reusable Programs (Complex Adaptive Systems), MIT Press, Cambridge, Massachusetts, 1994.

Koza J., Bennett F.H.III, Bennett F.H., Keane M., Andre D., Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis, Morgan Kaufmann Publishers, 1999.

Koza J.R., Keane M.A., Streeter M.J., Mydlowec W., Yu J., Lanza G., Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer Academic Publishers, 2003.

Lem S., Tajemnica chińskiego pokoju, Universitas, Kraków 1996.

Michalewicz Z., Algorytmy genetyczne + struktury danych = programy ewolucyjne, WNT, Warszawa 1996.

Michalski R.S., Kubat M., Bratko I., Machine Learning & Data Mining: Methods & Applications, J. Wiley & Sons, Chichester 1998.

Mulawka J.J., Systemy ekspertowe, WNT, Warszawa 1996.

Nilsson N.J., Principles of Artificial Intelligence, Tioga, Palo Alto, Cal. 1980.

Penrose R., Nowy umysł cesarza, PWN, Warszawa 1995.

Penrose R., Cienie umysłu. Poszukiwanie naukowej teorii świadomości. Zysk S-ka. Poznań. 2000.

Preston J., Bishop M., (eds.), Views into the chinese room. New essays on Searle and artificial intelligence, Clarendon Press, Oxford, 2002.

Russel S., Norvig P., Artificial Intelligence, Prentice-Hall, London 2003.

Rutkowski L.: Metody i techniki sztucznej inteligencji, Wydawnictwo Naukowe PWN, Warszawa 2006.

Sutton R.S., Barto A.G., (1998), Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, A Bradford Book.

Wolfram S., A new kind of science, Wolfram Media, 2002.

Wójcik M., Zasada rezolucji, PWN, Warszawa 1991.

Żurada J., Barski M., Jędruch W., Sztuczne sieci neuronowe, PWN, Warszawa 1996.

1.2. Definicje, metody, zastosowania, historia

1.2.1. Definicje sztucznej inteligencji

Interesujące wysiłki uczynienia komputery myślącymi . . . „maszynami posiadającymi umysł“ w pełnym i dosłownym sensie (Haugeland).

Sztuczna inteligencja jest częścią informatyki dotyczącą projektowania inteligentnych systemów komputerowych, to jest systemów, które przejawiają własności, które wiążemy z inteligencją w zachowaniu ludzkim – zrozumienie języka, uczenie się, rozwiązywanie zadań, itp. (Barr, Feigenbaum).

(Automatyzacja) aktywności, które wiążemy z myśleniem ludzkim, takich jak podejmowanie decyzji, rozwiązywanie zadań, uczenie się (Bellman).

Sztuka budowy maszyn, które wykonują zadania wymagające inteligencji gdyby wykonywane były przez ludzi (Kurzweil i podobnie Minsky).

Studia nad budową komputerów, które wykonywałyby rzeczy, które obecnie ludzie wykonują lepiej (Rich, Knight).

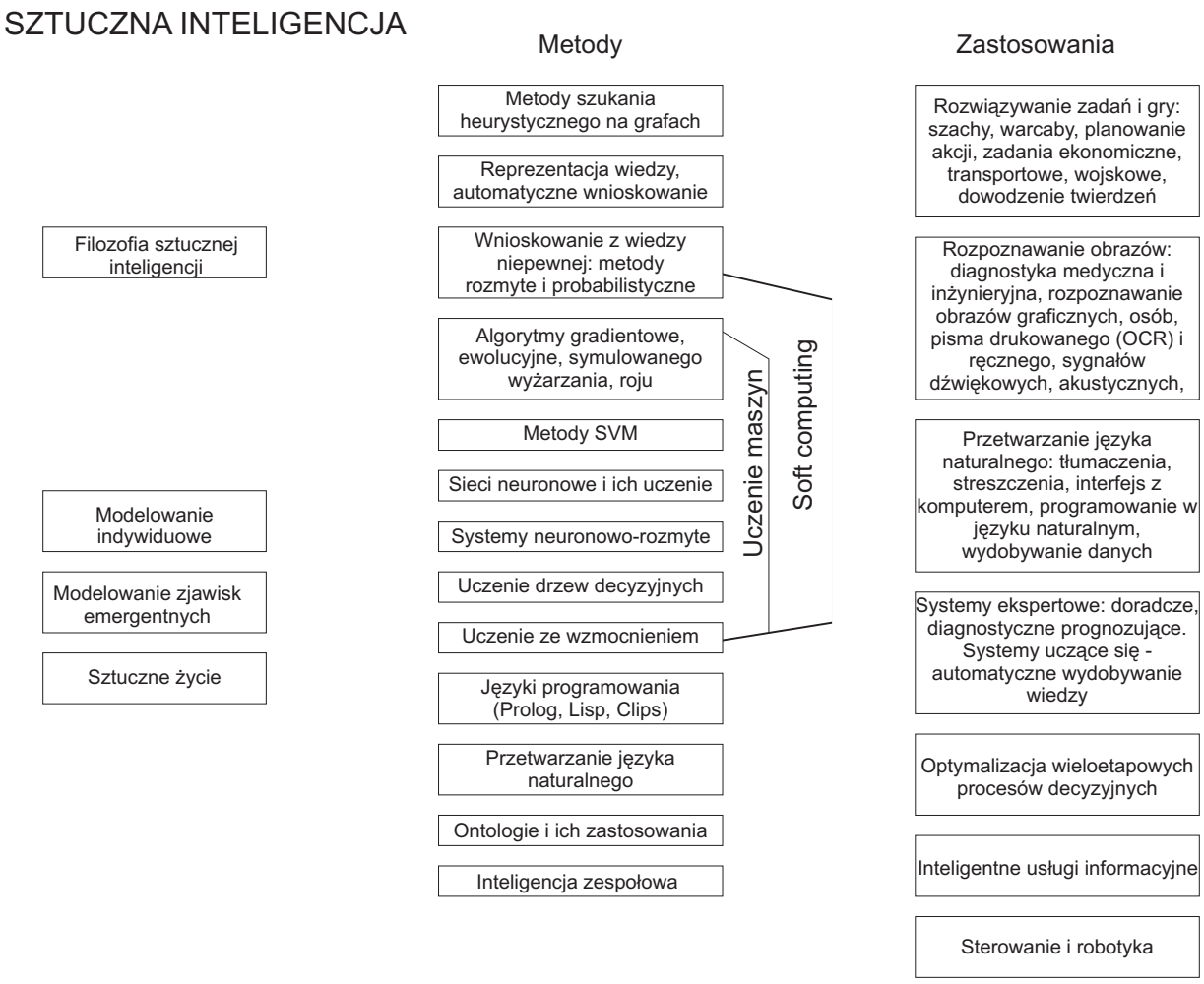
Studia nad możliwościami umysłu poprzez stosowanie modeli komputerowych (Charniak, McDermott).

Studia nad metodami obliczeń, które mogłyby postrzegać, wnioskować i działać (Winston).

Dziedzina badań dążąca do wyjaśnienia i naśladowania inteligentnego zachowania przy pomocy procesów obliczeniowych (Schalkoff).

Dziedzina informatyki dotycząca automatyzacji inteligentnego zachowania się (Luger, Stubblefield).

Historia Gry, tłumaczenia, rozpoznawanie obrazów, sieci neuronowe, systemy ekspertowe – obiecujące poczatki w latach 50-tych, późniejsze porażki i zahamowanie (np. tłumaczenia). Stopniowy rozwój na uniwersytetach, nowe nadzieje i komercjalizacja od połowy lat 80-tych spowodowane tanimi szybkimi komputerami i nagromadzeniem doświadczeń. Obecnie coraz więcej cząstkowych zastosowań (często spektakularnych jak np. mecz szachowy Kasparow – komputer IBM Deep Blue) oraz coraz większa specjalizacja i oddzielanie się poszczególnych metod. Bardzo obiecujące projekty realizacji uczenia za pomocą specjalizowanych równoległych struktur sprzętowych.



Rys. 1.1: Główne metody i zastosowania sztucznej inteligencji

1.2.2. Obszary zastosowań (patrz rys. 1.1)

- Rozwiązywanie zadań, gry, planowanie i dowodzenie twierdzeń – Szachy, warcaby, otello, itp. Planowanie akcji. Zadania ekonomiczne, transportowe, wojskowe.
- Przetwarzanie języka naturalnego – Tłumaczenia. Streszczenia. Interfejs z komputerem (chatboty). Programowanie w języku naturalnym. W różnych zadaniach przetwarzania języka naturalnego wystepuje takie samo zadanie ogólne: zamiana wypowiedzi słownej na opis w języku formalnym.
- Rozpoznawanie obrazów – rozpoznawanie obrazów grficznych, Rozpoznawanie pisma drukowanego (OCR) i ręcznego, rozpoznawanie dźwięków. Diagnostyka: medyczna i urządzeń. Przykład: odczytywanie numerów rejestracyjnych samochodów z obrazów scen ulicznych. Analiza scen – zamiana obrazu graficznego na opis formalny.
- Systemy ekspertowe – Systemy doradcze. Diagnostyka.
- Wydobywanie wiedzy z danych doswiadczalnych.
- Robotyka. Mogą tu być wykorzystywane wszystkie powyżej wymienione zastosowania. Przykład: automatyczne prowadzenie pojazdu terenowego (projekty realizowane), robot kroczący, rozpoznający obrazy i porozumiewający się w języku naturalnym (projekty w pełni jeszcze nie realizowane).

1.2.3. Charakterystyczne narzędzia i metody

Pomimo postępującego rozdzielania się SI na poszczególne kierunki można wciąż wyróżnić wspólne narzędzia i metody:

- Metody szukania.
- Metody logiczne, w tym wnioskowanie przy wiedzy niepełnej.
- Metody uczenia.
- Języki Lisp, Prolog, Clips — do przedstawiania idei i prototypowania, ale także do budowy rzeczywistych systemów.

1.2.4. Sprzęt

Maszyny szeregowe. Maszyny równoległe. Maszyny specjalizowane: automaty komórkowe, sieci neuronowe, ewolujący hardware.

We wszystkich kierunkach SI wyróżnić można nurt poznawczy i inżynierski.

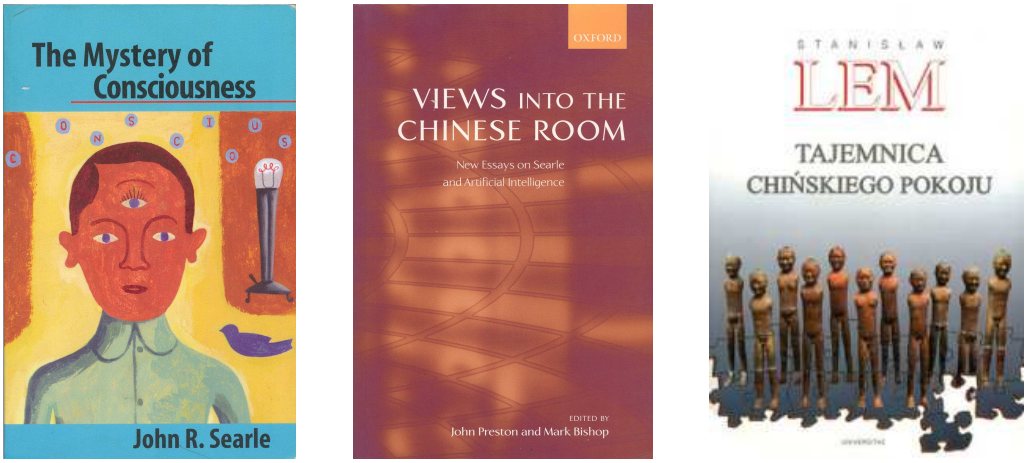
Rozdział 2

Filozofia sztucznej inteligencji

Jako dyscyplina naukowa sztuczna inteligencja jest bardzo młoda, ale podstawowe problemy filozoficzne leżące u jej podstaw budziły zainteresowanie przez wiele wieków. Chodzi tu o pytania czym jest myślenie, czucie i świadomość, jaki jest związek umysłu z ciałem i czy możliwa jest budowa urządzenia, które by posiadało te cechy umysłu człowieka.

Kartezjusz w XVII wieku jako jeden z pierwszych myślicieli uważał, że funkcjonowanie ciała człowieka powinno dać się wyjaśnić mechanistycznie, tzn. że człowiek jest złożoną maszyną, której działanie daje się opisać przy pomocy zasad fizyki i chemii. Z drugiej strony jednak twierdził, że zjawiska psychiczne są innej natury i nie dają się wyprowadzić ze zjawisk cielesnych. Kartezjusz dzięki tym poglądom uważany jest za twórce nowoczesnej doktryny dualizmu czyli istnienia dwóch niesprowadzalnych do siebie czynników: ciała i ducha. W okresie Oświecenia w XVIII wieku pojawiły się idee materialistyczne głoszące, że także działanie umysłowe powinny dać się wyjaśnić mechanistycznie (wydana w 1747 roku słynna książka La Mettriego (1984)).

Obecnie, wśród części badaczy SI panuje przekonanie, że zachodzi prosta analo-



Rys. 2.1: Okładki przykładowych książek o filozofii sztucznej inteligencji: (a) książka Searla twórcy koncepcji „chińskiego pokoju” [?], (b) zbiór klasycznych artykułów [55] i (c) zbiór esejów Lema [37] o filozofii sztucznej inteligencji

gia pomiędzy zachowaniem człowieka, a wykonywaniem programu komputerowego. Uważają oni, że myślenie ludzkie opiera się na takich samych mechanizmach jakie występują w maszynach cyfrowych, albo ostrożniej – że wszelkie ludzkie działanie da się opisać i sformalizować przy pomocy reguł mechanistycznych (Gurba 1997).

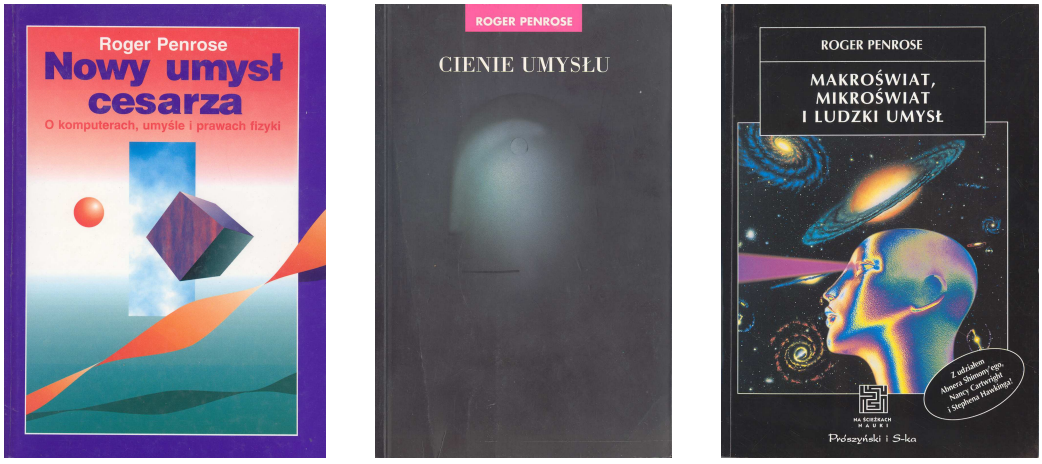
Operacyjny pogląd na myślenie i świadomość polega na przyjęciu, że coś myśli, jeżeli działa w sposób nieodróżnialny od działania myślącej osoby. Pogląd taki wyrażał Alan Turing (1950) proponując kryterium zwane obecnie *testem Turinga* rozstrzygające czy można twierdzić, że dana maszyna (komputer) myśli. Kryterium to mówi, że jeżeli człowiek prowadząc dialog w języku naturalnym z komputerem nie będzie w stanie odróżnić czy ma do czynienia z człowiekiem czy komputerem, to należy przyjąć, że komputer ten myśli.

Newell i Simon, znani badacze z Carnegie Mellon University w Pittsburghu opublikowali głośny artykuł (1976), w którym stwierdzają, że wszystko co wykonuje człowiek fizycznie i umysłowo jest wynikiem przetwarzania informacji analogicznego do przetwarzania realizowanego w maszynie cyfrowej przez nadzwyczaj złożony program – ale tylko program. Stanowisko takie nazywa się *hipoteza silnej SI*. Jego zwolennicy uważają ponadto, że tworząc i realizując w komputerze program przejawiający działanie analogiczne do działania mózgu ludzkiego, komputer ten będzie posiadał cechy umysłu człowieka łącznie z myśleniem, czuciem, inteligencją, rozumieniem i świadomością. Atrybuty te są tylko cechami algorytmu wykonywanego przez mózg.

Polemikę z hipotezą silnej SI podjął Searle (Searle 1991, Churchland, Churchland 1991, Penrose 1995, Lem 1996). Przedstawił hipotetyczne przykłady działania komputerów prowadzących dialog z użytkownikiem w uproszczonym języku naturalnym, o których można powiedzieć, że przechodzą uproszczony test Turinga, czyli przejawiają dla zewnętrznego obserwatora pewne cechy inteligencji. Następnie Searle przedstawia rozumowanie, w którym wyobraża siebie w roli procesora znajdującego się w zamkniętym pokoju. Komunikacja ze światem zewnętrznym odbywa się przez wąską szczelinę, przez którą wprowadzane i wyprowadzane są zapytania i odpowiedzi w języku naturalnym. Po otrzymaniu zapytania procesor wykonuje program pobierając instrukcje znajdujące się w regałach wewnątrz pokoju i wytwarza odpowiedź. Jeżeli językiem naturalnym jest np. język chiński, którego Searle nie zna, to w dalszym ciągu wykonując instrukcje będzie prowadził poprawny dialog w języku chińskim kompletnie go nie rozumiejąc. Argumentacja Searlego sprowadza się do wykazania, że wykonanie algorytmu nie implikuje, że mamy do czynienia z prawdziwym zrozumieniem.

W dyskusjach dotyczących silnej SI często przytaczany jest hipotetyczny przykład podróży poprzez przesyłanie informacji o strukturze człowieka, a następnie w miejscu docelowym odtworzenie go na podstawie tej informacji (teleportacja). Powstaje pytanie czy nowo utworzony osobnik będzie miał tą samą świadomość? A co się stanie jeżeli nie zniszczono jednocześnie oryginału?

Kiedyś idee mechanistyczne, a współcześnie idee silnej SI spowodowały postawienie pytania o takie atrybuty umysłu człowieka, które odróżniałyby go od realizacji mechanicznych czy komputerowych. Do takich atrybutów zalicza sie *intencjonalność* czyli cechę zjawisk psychicznych polegająca na kierowaniu się ich ku jakiemuś przedmiotowi lub cechę świadomości kierującej się na przedmiot, konstytuującej sens przez



Rys. 2.2: Okładki książek Penrosa [53, 54] – twórcy oryginalnej hipotezy o niemożności modelowania procesów umysłowych przez komputer

„zdawanie sobie z czegoś sprawy" (NEP 1995, Brit 1998). Myślenie, wiara, pożądanie i inne tego rodzaju odczucia uważa się za podobne do siebie w tym znaczeniu, że można powiedzieć że obejmują obiekt lub kierują się ku obiektowi, w sposób zasadniczo różny od oddziaływań spotykanych w świecie nieożywionym i ten sposób zwracania się nazywa się intencjonalnym. Podkreśla się, że obiekty zainteresowania intencjonalnego mogą nawet nie istnieć. Przeciwnicy poglądu o możliwości budowy sztucznego umysłu uważają, że intencjonalność nie jest możliwa do sztucznego odtworzenia.

Niezależnie od rozstrzygnięć dyskusji filozoficznych czy możliwa jest komputerowa realizacja umysłu, wciąż wielkie trudności napotyka komputerowa realizacja takich atrybutów inteligencji jak adaptowanie się do nowych warunków, rozwiązywanie nowych zadań czy działalność twórcza.

Rozważając w swoich książkach problemy modelowania komputerowego umysłu Roger Penrose (Penrose 1995, 2000) wyróżnia cztery stanowiska, które można zająć w tej sprawie:

A Myślenie zawsze polega na obliczeniach, a w szczególności świadome doznania powstają wskutek realizacji odpowiedniego procesu obliczeniowego.

B Świadomość jest cechą fizyczną działającego mózgu; wprawdzie wszystkie fizyczne procesy można symulować obliczeniowo, symulacjom obliczeniowym nie towarzyszy jednak świadomość.

C Odpowiednie procesy fizyczne w mózgu powodują powstanie świadomości, ale tych procesów nie można nawet symulować obliczeniowo.

D Świadomości nie można wyjaśnić w żaden fizyczny, obliczeniowy czy inny naukowy sposób.

Stanowisko **A** jest zgodne z hipotezą silnej sztucznej inteligencji, zaś stanowisko **B** jest zbieżne z poglądami Searlego.

Penrose opowiada się za stanowiskiem **C**. Główną linią jego wywodu jest to, że w przyrodzie, a tym samym w mózgu, występują procesy, które nie dają się modelować

przy pomocy uniwersalnej maszyny Turinga (komputera). Jest to pogląd sprzeczny z powszechnie uznawaną hipotezą Churcha-Turinga mówiącą, że wszystkie procesy fizyczne są obliczalne, a zatem dają się modelować za pomocą maszyny Turinga.

Stanowiska **A** i **B** implikują możliwość posiadania przez maszyny inteligencji. Przyjmując jedno z tych stanowisk i biorąc pod uwagę wielkie i stale rosnące moce obliczeniowe komputerów mozna spodziewać się powstania maszyn o inteligencji znacznie przewyższającej inteligencję człowieka i w konsekwencji wyparcie przez nie i zmarginalizowanie roli człowieka. Paradoksalnie stanowisko **B** jest bardziej pesymistyczne niż **A**, bo wynika z niego, że światem rządzić będą automaty nie posiadające umysłu (świadomości), gdy z **A** wynikałoby, że atomaty te posiadałyby umysł, a tym samym możnaby powiedzieć, że w pewnym sensie dziedziczyłyby cechy człowieka.

Rozdział 3

Metody szukania

Omawiane będą następujące metody szukania: metoda szukania wszerz (ang. breadth first search), metoda szukania w głąb (ang. depth first search), metoda równomiernych kosztów (metoda Dijkstry)(ang. uniform cost search), metody heurystyczne, metody szukania na grafach AND/OR, metoda minimax oraz metoda $\alpha - \beta$.

3.1. Przykłady zastosowań

Przykłady szkolne (ang. *toy problems*)

1. Układanka 3×3.
2. Kryptogramy. Np.:

```
 SEND
 MORE
-----
MONEY
```

3. Zadanie o 8 hetmanach na szachownicy.
4. Zadanie o znalezieniu trasy konika szachowego przechodzącego wszystkie pola szachownicy.
5. Zadanie o misjonarzach i kanibalach.

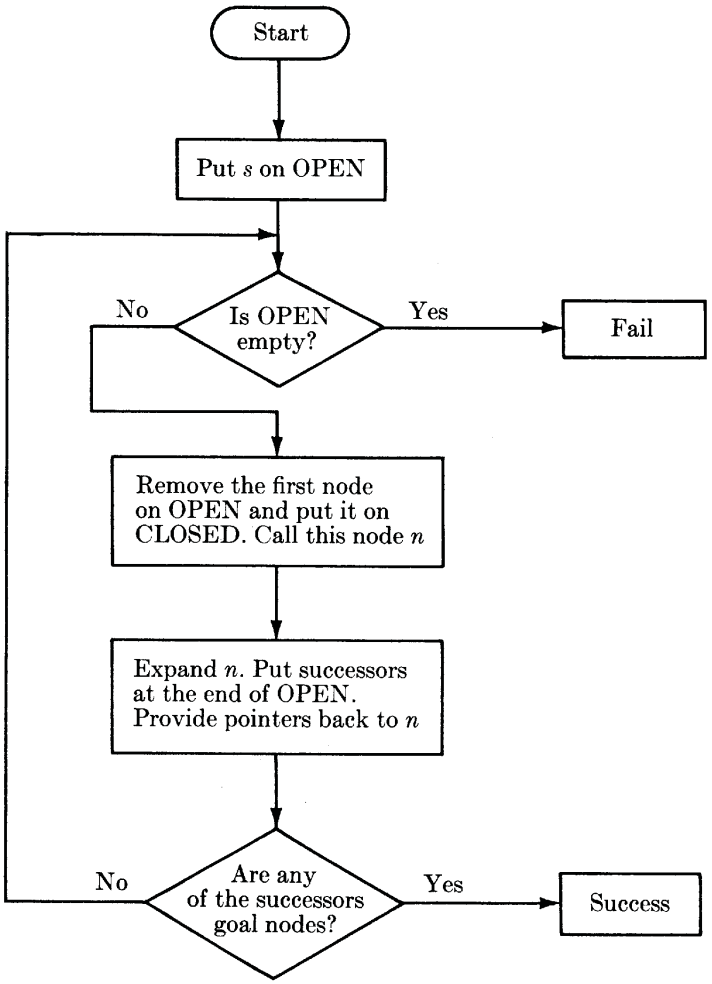
Zadania o znaczeniu praktycznym

1. Znajdywanie drogi (najkrótszej lub optymalizującej inne kryteria).
2. Zadanie o komiwojażerze – np. droga ramienia wiertarki.
3. Planowanie akcji – konstrukcja układów VLSI, asemblacja podzespołów maszyn, układanie rozkładów zajęć lub rozkładów jazdy, rozgrywanie gier – np. szachów.
4. Nawigacja robotów.

3.2. Metoda szukania wszerz

Metoda szukania wszerz polega na pobieraniu i rozmanażaniu węzłów warstwa po warstwie aż do znalezienia węzła stanowiącego rozwiązanie. Na rys. 3.1 przedstawiona jest sieć działań metody, a na rys. 3.2 przykład zastosowania metody do

rozwiązania zadania polegającego na przeprowadzeniu układanki 3×3 z zadanego stanu początkowego do zadanego stanu końcowego.



Rys. 3.1: Sieć działań metody szukania wszerz dla drzew (z książki [**?**])

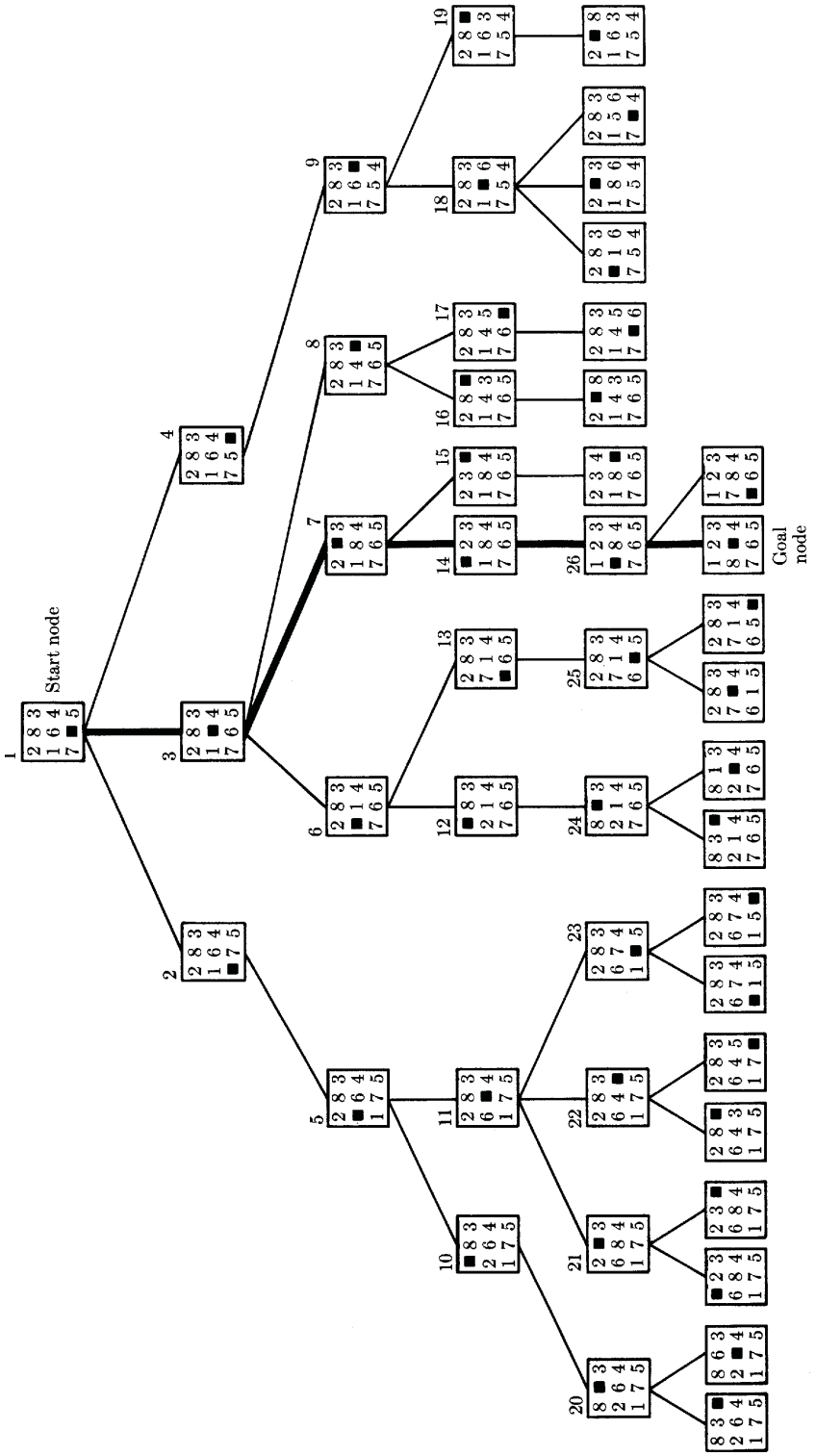
3.2.1. Złożoność obliczeniowa metody szukania wszerz

Liczba węzłów rozmnażanych przed znalezieniem rozwiązania wynosi:

$$1 + b + b^2 + b^3 + \ldots + b^d$$

gdzie b jest liczbą węzłów potomnych generowanych z jednego węzła, a d jest liczbą przebytych warstw. Stąd złożoność obliczeniowa metody szukania wszerz wynosi $O(b^d)$. Złożoność tą ilustruje tablica 3.1 przedstawiająca czasy obliczeń i zajętość pamięci metody w zależności od liczby przejrzanych warstw. Tablica została utworzona przy założeniu, że czas generacji jednego węzła wynosi 1ms, zapamiętanie jednego węzła wymaga 100 bajtów pamięci oraz $b = 10$.

Biorąc pod uwagę liczbę węzłów grafu n i krawędzi m złożoność obliczeniowa metody szukania wszerz wynosi $O(m + n)$ (Kubale 1994).



Rys. 3.2: Układanka 3×3. Drzewo stanów uzyskane metodą szukania wszerz (z książki [**?**])

Głębokość	Węzły	Czas		Pamięć	
0	1	1	ms	100	bajtów
2	111	0.1	s	11	kB
4	11111	11	s	1	MB
6	10^6	18	m	111	MB
8	10^8	31	godz	11	GB
10	10^{10}	128	dni	1	TB
12	10^{12}	35	lat	111	TB
14	10^{14}	3500	lat	11111	TB

Tablica 3.1: Wymagany czas i pamięć przy szukaniu metodą wszerz. Przykład przedstawiony dla $b = 10$, prędkości generacji: 1000 węzłów/śekundę oraz 100 bajtach wymaganych dla zapisu jednego węzła. (Russel, Norvig, 1995)

3.3. Metoda szukania w głąb

Przy stosowaniu metody szukania w głąb ustalana jest głębokość na jaką odbywać się będzie szukanie. Na rys. 3.3 przedstawiona jest sieć działań metody, a na rys. 3.4 przykład zastosowania metody do przeprowadzenia układanki 3×3 z zadanego stanu początkowego do zadanego stanu końcowego.

Złożoność obliczeniowa metody wynosi $O(b^l)$ gdzie l jest przyjętą głębokością szukania. Biorąc pod uwagę liczbę węzłów grafu n i krawędzi m złożoność obliczeniowa metody szukania w głąb wynosi $O(m + n)$ (Kubale 1994).

W przypadku nieznajomości głębokości, na której znajduje się rozwiązanie często stosowaną odmianą jest metoda szukania z *iteracyjnym pogłębianiem* (ang. iterative deepening search). Metoda ta rozpoczyna się od głębokości jeden i gdy rozwiązanie nie zostało znalezione, to przyjmowana jest głębokość 2 i ponownie przeprowadzone jest szukanie (od początku) i analogicznie za każdym razem gdy rozwiązanie nie zostało znalezione, głębokość jest zwiększana o 1. Liczba węzłów rozmnażanych przed znalezieniem rozwiązania wynosi:

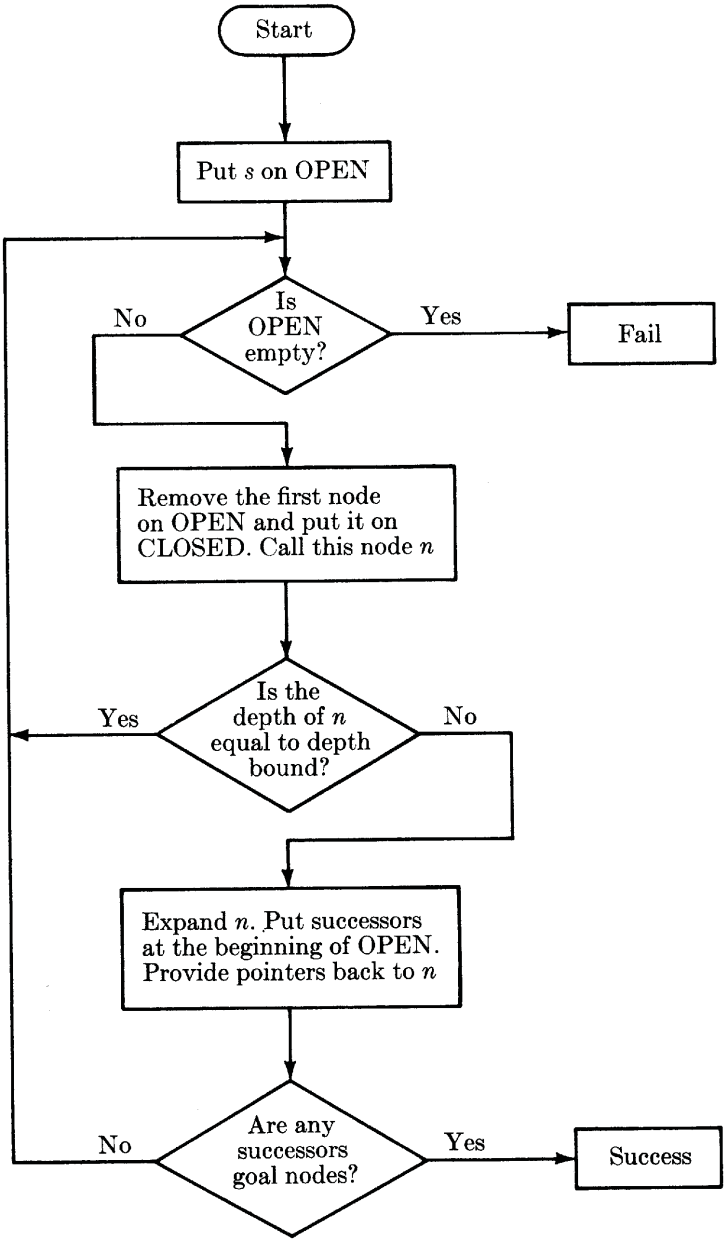
$$(d+1)1 + (d)b + (d-1)b^2 + (d-2)b^3 + \ldots 3b^{d-2} + 2b^{d-1} + 1b^d$$

gdzie b jest liczbą węzłów potomnych generowanych z jednego węzła, a d jest liczbą przebytych warstw. Pomimo wielokrotnego powtarzania szukania liczba rozmnażanych węzłow nie jest wiele większa niż w przypadku stosowania metod szukania wszerz lub w głąb. Np. dla $b = 10$ i $d = 5$ dla metody iteracyjnego pogłębiania otrzymujemy liczbę węzłów 123456 podczas gdy dla metody szukania wszerz mamy 111111.

Często wygodnie jest zrealizować metodę szukania w głąb przy pomocy funkcji rekurencyjnej (np. w zadaniu sprawdzania istnienia drogi pomiędzy węzłami grafu).

3.4. Metoda z powracaniem

Metoda szukania z powracaniem jest podobna do metody szukania w głąb. Różni się od niej jedynie tym, że w jednym kroku metody generowany jest tylko jeden węzeł potomny, a nie wszystkie jak w metodzie szukania w głąb.

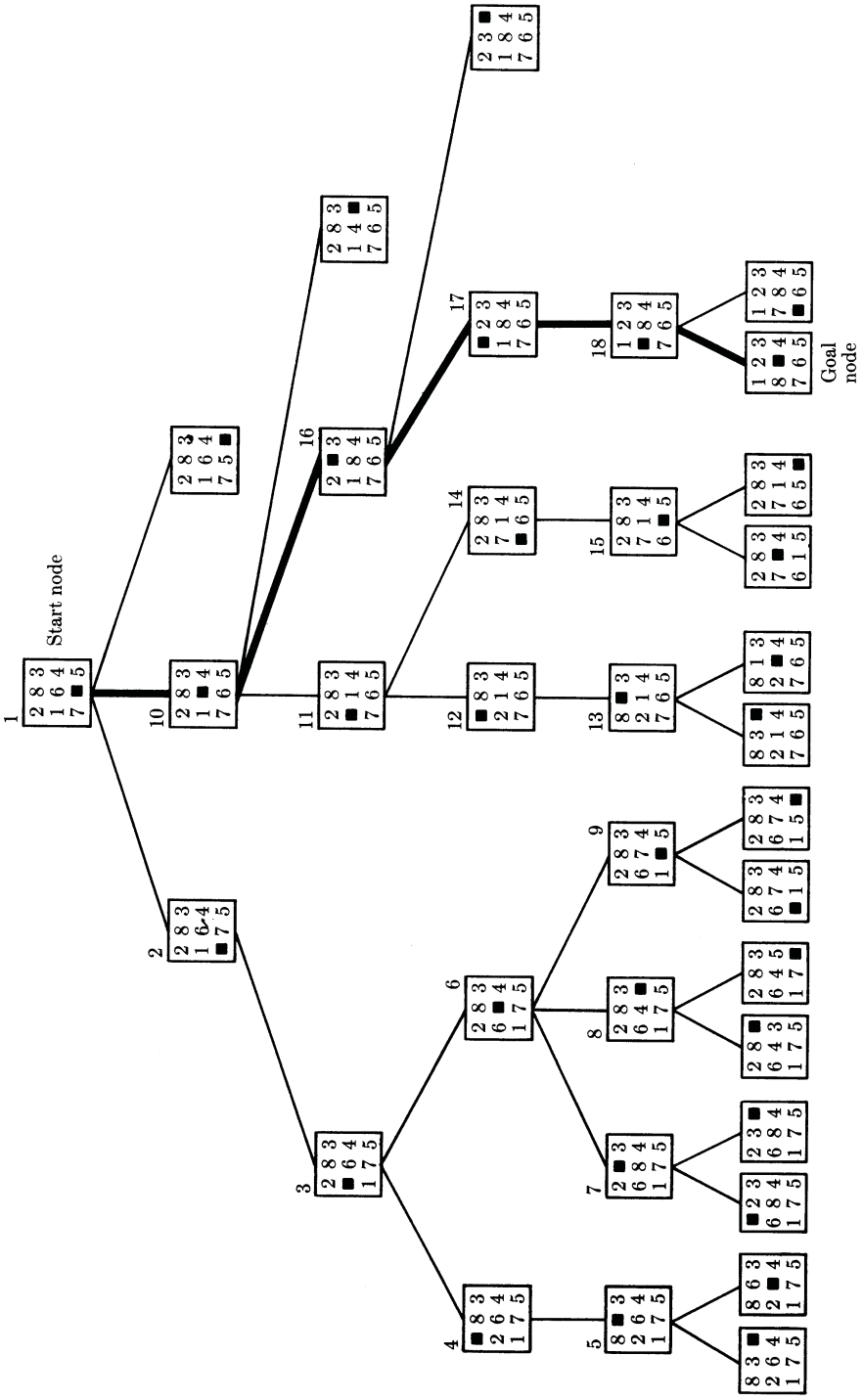


Rys. 3.3: Sieć działań metody szukania w głąb dla drzew (z książki [?])

3.5. Metoda równomiernych kosztów

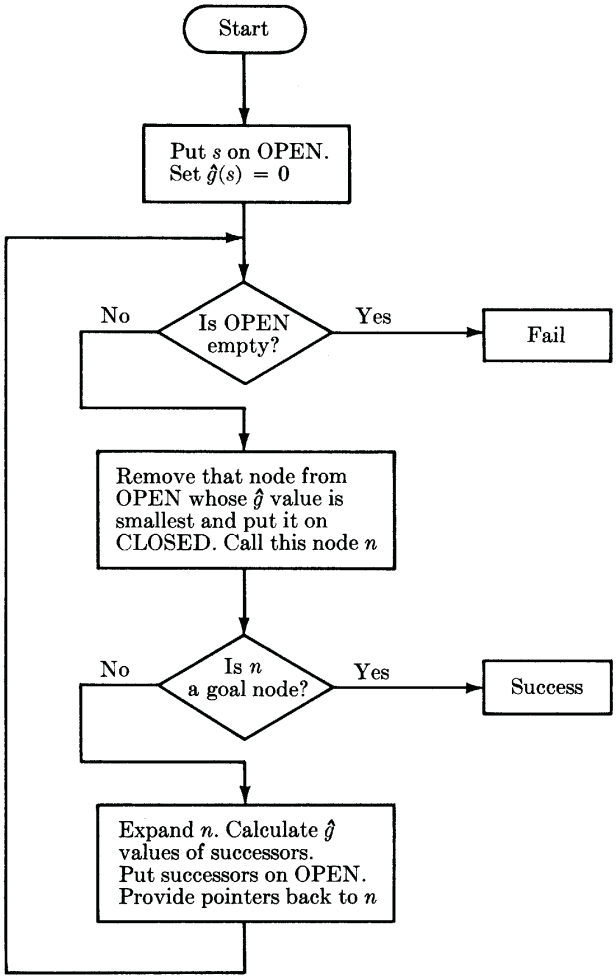
W sytuacjach gdy oprócz znalezienia węzła końcowego należy wyznaczyć najtańszą drogę jego osiągnięcia (od węzła startowego s) stosuje się metodę równomiernych kosztów nazywaną też algorytmem Dijkstry. Przy jej stosowaniu wykorzystuje się funkcję $\hat{g}(n)$ oznaczającą znaleziony dotychczas koszt najtańszej drogi od węzła s do węzła n (koszt estymowany) ($g(n)$ oznacza nieznany w czasie działania metody koszt minimalny). Na rys. 3.5 przedstawiona jest sieć działań metody.

Typowymi zadaniami, przy których rozwiązaniu można stosować metodę równomiernych kosztów są zadania poszukiwania najkrótszej drogi pomiędzy węzłami grafu (np. drogi łączącej dwa miasta) oraz zadania planowania akcji. Metoda za-



Rys. 3.4: Układanka 3×3. Drzewo stanów uzyskane metodą szukania w głąb (z książki [**?**])

pewnia znalezienie rozwiązania optymalnego.



Rys. 3.5: Sieć działań metody równomiernych kosztów (z książki [?])

3.6. Modyfikacje wymagane przy szukaniu na grafach

W metodzie szukania wszerz należy tylko stwierdzić, czy nowo wygenerowany stan jest już na listach *OTWARTE* lub *ZAMKNIĘTE*. Jeżeli tak, to nie potrzeba go umieszczać ponownie na liście *OTWARTE*.

W metodzie szukania w głąb i w metodzie z nawracaniem należy modyfikować zapamiętywane głębokości węzłów, gdy okazuje się, że istnieje do nich krótsza droga; w takim przypadku węzły znajdujące się na liście *ZAMKNIĘTE* należy przenieść na listę *OTWARTE*.

W metodzie równomiernych kosztów postępuje się następująco:

1. Jeżeli nowo wygenerowany węzeł jest już na liście *OTWARTE*, nie potrzebuje być do niej dopisany, ale należy mu przypisać mniejszą z wartości kryterium

$\hat{g}$. W przypadku zmiany wartości kryterium należy uaktualnić wskaźnik wskazujący najtańszą drogę.

2. Jeżeli nowo wygenerowany węzeł jest już na liście *ZAMKNIĘTE*, to możnaby przypuszczać, że jego wartość $\hat{g}$ mogła by być mniejsza niż do tej pory. Tak jednak nie jest. Poniżej pokazano, że gdy algorytm równomiernych kosztów umieszcza węzeł na liście *ZAMKNIĘTE*, to najmniejsza wartość $\hat{g}$ została już znaleziona, czyli $\hat{g} = g$.

Lemat
Na każdym etapie działania algorytmu równomiernych kosztów istnieje na liście O węzeł n' z P oraz $\hat{g}(n') = g(n')$, gdzie P oznacza ścieżkę rozwiązania (od węzła startowego do rozwiązania).
Dowód
Przypuśćmy, że n' jest pierwszym z ciągu P węzłem znajdującym się na O. Z zasady działania algorytmu przynajmniej jeden z węzłów z P musi być na O. Najpierw jest to s, a później jeden z jego potomków.

Na to żeby $\hat{g}(n') = g(n')$ potrzeba i wystarcza aby na liście Z były wszystkie węzły z P, od węzła s do węzła bezpośrednio poprzedzającego n'.

Jeżeli na Z nie ma jakiegoś węzła m z P poprzedzającego n', to węzeł ten:

1. jest na O,

2. nie jest na O.

Gdy zachodzi:

1. to węzeł m będzie pierwszym z P na O zamiast n',

2. to pewien węzeł z P poprzedzający m będzie na O i n' nie będzie pierwszym.

Ponieważ przypadki 1) i 2) przeczą temu, że n' jest pierwszym z P na liście O, to na Z muszą być wszystkie węzły poprzedzające n'. Wobec tego $\hat{g}(n') = g(n')$

Przykład 3.1

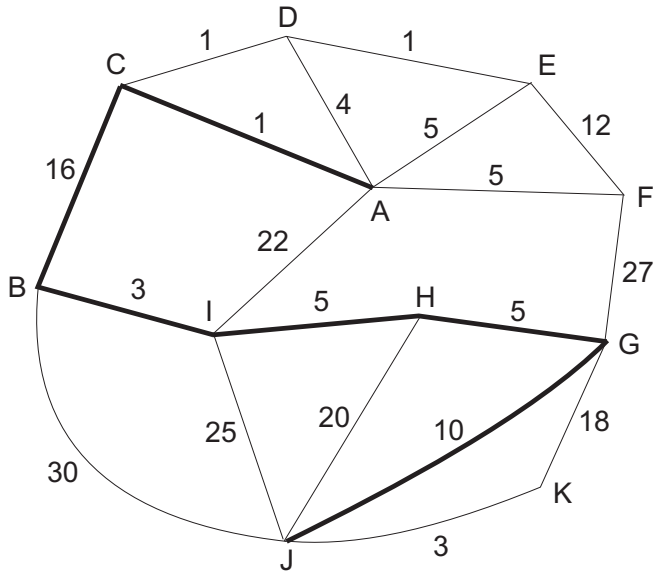
Na rys. 3.6 przedstawiono przykład zastosowania metody równomiernych kosztów do znalezienia najkrótszej drogi łączącej dwa węzły grafu.

3.7. Inne metody

3.7.1. Metody spełniania warunków

Jedna z często stosowanych, oczywistych odmian metod szukania nazywa się metodą spełniania warunków (ang. constraint satisfaction search). Polega ona na generacji tylko takich stanów następnych, które spełniają dodatkowe warunki nałożone na rozwiązanie zadania, a nie na generacji wszystkich możliwych stanów następnych.

Typowym zadaniem rozwiązywanym tą metodą jest zadanie o rozmieszczeniu 8 figur hetmanów na szachownicy – należy je tak rozmieścić aby sie wzjemnie nie



			2	3		20		30	40 45			
Otwarte	0 A	1 C	4 D	5 E	5 F	22 I	17 B	32 G	47 J	25 H	48 K	
Zamknięte	A	C	D	E	F	B	I	H	G	J		

Rys. 3.6: Przykład zastosowanie metody Dijkstry do wyznaczenia minimalnej drogi łączącej dwa węzły grafu (od węzła A do węzła J). W tablicy pokazano kolejność umieszczania i usuwania węzłow z list *OTWARTE* i *ZAMKNIĘTE*. Linią pogrubioną zaznaczono najkrótszą drogę

biły. Stosowanie tej metody rozpoczyna się od postawienia pierwszego hetmana, a następnie kolejni hetmani stawiani są tylko na te pola, na których nie są bici przez figury już postawione.

3.7.2. Metody dwukierunkowe

W niektórych zadań można zastosować metode dwukierunkową polegającą na generowaniu stanów następnych idąc od stanu początkowego i jednocześnie generowaniu stanów wcześniejszych idąc od stanu końcowego. Metoda się kończy, gdy w obu procedurach uzyska się identyczny stan.

3.8. Metody heurystyczne

W wielu zadaniach można wykorzystać dodatkowe informacje o nich redukując w ten sposób liczbę generowanych węzłów grafu. Metody wykorzystujące przy szukaniu dodatkowe informacje noszą nazwę metod szukania heurystycznego.

3.8.1. Algorytm A^*

Przypuśćmy, że $\hat{f}(n)$ jest estymatorem kosztów najtańszej drogi od węzła startowego do końcowego przechodzącej przez węzeł n. Rzeczywisty koszt tej drogi można wyrazić jako

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

gdzie $g(n)$ oznacza koszt najtańszej drogi od węzła startowego s do węzła n, a $h(n)$ jest kosztem najtańszej drogi od węzła n do rozwiązania.

Algorytm A^* spośród węzłów listy O wybiera do rozwinięcia węzeł, dla którego wyrażenie

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$$

ma wartość najmniejszą, gdzie $\hat{g}(n)$ oznacza wyznaczony do tej pory (estymowany) koszt najtańszej drogi od węzła startowego s do węzła n, a $\hat{h}(n)$ jest nadanym przez użytkownika oszacowniem kosztu najtańszej drogi od węzła n do rozwiązania.

Jeżeli dla każdego węzła n zachodzi

$$\hat{h}(n) \leq h(n) \tag{3.1}$$

to algorytm A^* znajduje najtańszą drogę (algorytm jest dopuszczalny).

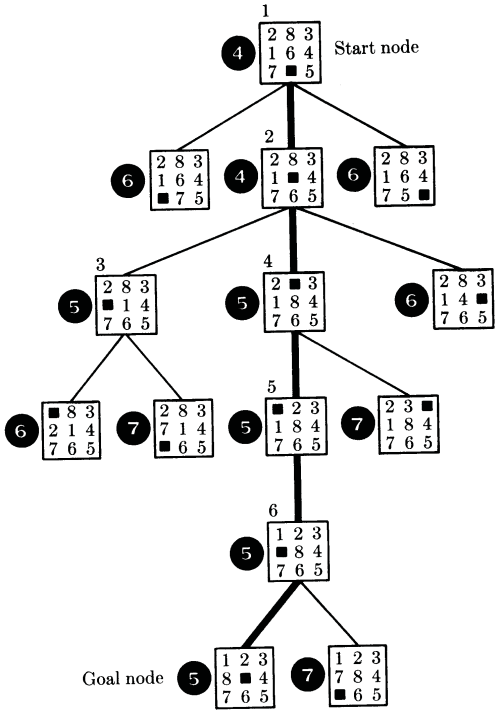
Algorytm szukania na grafach składa się z następujących kroków:

1. Umieść s na liście O i oblicz $\hat{f}(s)$.
2. Jeżeli O jest pusta, to zakończ program z sygnałem niepowodzenia.
3. Usuń z O ten węzeł, dla którego wartość $\hat{f}$ jest najmniejsza i umieść go na liście Z. Oznacz ten węzeł przez n. Spośród węzłów o równych wartościach $\hat{f}$ wybierz dowolny ale zawsze końcowy (jeżeli jest taki).
4. Jeżeli węzeł n jest końcowym, to zakończ program z sygnałem powodzenia i podaj ścieżkę rozwiązania.
5. Rozmnóż n. Dla każdego potomka oblicz $\hat{f}$. Jeżeli n nie ma następników idź do 2.
6. Przyporządkuj potomkom nie znajdującym się ani na O ani na Z obliczoną właśnie wartość $\hat{f}$. Umieść te potomki na O i zapamiętaj wskaźniki wskazujące drogę do n.
7. Przyporządkuj tym potomkom, które już znajdują się na O lub Z mniejszą z wartości $\hat{f}$ spośród ostatnio obliczonej i tej którą dotychczas posiadała. Umieść na O te potomki na Z, których wartości $\hat{f}$ zostały obniżone.
8. Idź do 2.

Przykład 3.2

Na rysunkach 3.7 i 3.8 przedstawiono przebiegi szukania rozwiązania układanki 3×3 przy zastosowaniu metody A^* i dwóch różnych sposobach wyznaczania funkcji h.

■



Rys. 3.7: Układanka 3×3. Drzewo stanów uzyskane metodą szukania heurystycznego dla $\hat{g}$ będącego liczbą kroków od węzła startowego do danego, a $\hat{h}$ będącego liczbą klocków nie znajdujących się na właściwych pozycjach (z książki [**?**])

3.8.2. Dowód dopuszczalności algorytmu A^*

Lemat
Jeżeli $\hat{h}(n) \leq h(n)$, a P oznacza ścieżkę optymalną (ciąg optymalny), to na każdym etapie działania algorytmu istnieje na liście O węzeł n' z P oraz $\hat{f}(n') \leq f(s)$, gdzie $f(s)$ jest minimalnym kosztem całej drogi (kosztem otymalnym).
Dowód
Przypuśćmy, że n' jest pierwszym, z ciągu P, węzłem na O. Z zasady działania algorytmu przynajmniej jeden z węzłów z P musi być na O. Najpierw jest to s, a później jeden z jego potomków.

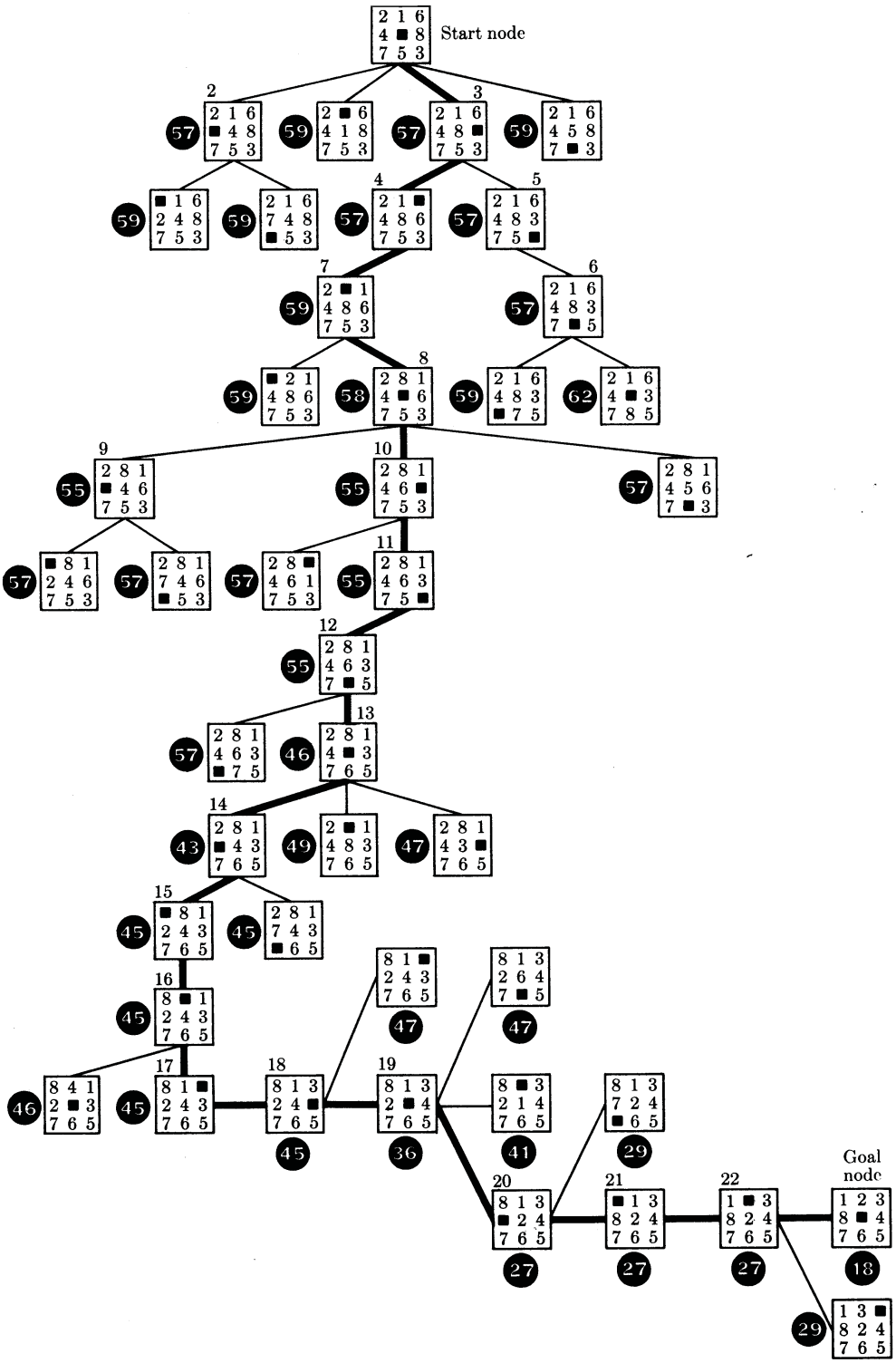
Na to żeby pokazać, że $f(n') \leq f(s)$ należy pokazać, że $\hat{g}(n') = g(n')$. Na to żeby $\hat{g}(n') = g(n')$ potrzeba i wystarcza aby na liście Z były wszystkie węzły z P, od węzła s do węzła bezpośrednio poprzedzającego n'.

Jeżeli na Z nie ma jakiegoś węzła m z P poprzedzającego n', to węzeł ten:

1. jest na O,
2. nie jest na O.

Gdy zachodzi:

1. to węzeł m będzie pierwszym z P na O zamiast n',
2. to pewien węzeł z P poprzedzający m będzie na O i n' nie będzie pierwszym.



Rys. 3.8: Układanka 3 × 3. Drzewo stanów uzyskane metodą szukania heurystycznego, gdzie $\hat{g}$ jest liczbą kroków od węzła startowego do danego, a $\hat{h}$ jest sumą odległości każdego klocka od swojego położenia końcowego (wyrażanych liczbą przesunięc) zwiększoną o liczbę punktów dla każdego klocka na obwodzie, któremu przyznane zostaje 6 punktów, gdy nie następuje za nim właściwy klocek, 0 punktów w przeciwnym przypadku oraz zwiększoną o 3 punkty za klocek na środku tablicy (z książki [?])

Ponieważ przypadki 1) i 2) przeczą temu, że n' jest pierwszym z P na liście O, to na Z muszą być wszystkie węzły poprzedzające n'. Wobec tego

$$\begin{aligned} \hat{g}(n') &= g(n') \\ \hat{f}(n') &= \hat{g}(n') + \hat{h}(n') \\ \hat{f}(n') &= g(n') + \hat{h}(n') \end{aligned}$$

Ponieważ $\hat{h}(n') \leq h(n')$ mamy ostatecznie

$$\hat{f}(n') \leq g(n') + h(n') = f(n') = f(s)$$

Jeżeli $\hat{f}(n') \leq f(s)$, to łatwo widać, że algorytm nie zatrzyma się wybierając drogę nieoptymalną. Bowiem nawet jeżeli na O jest węzeł końcowy k uzyskany z drogi nieoptymalnej, to nie będzie zabrany z O przed węzłem n' ponieważ $\hat{f}(k) > \hat{f}(n')$. Ponieważ sprawdzenie, czy węzeł jest końcowym odbywa się po zabraniu go z O, wobec tego z O zabrany będzie węzeł k dopiero wtedy, gdy będzie on znaleziony na drodze optymalnej.

3.8.3. Dobór funkcji $\hat{h}(n)$

W przypadku posiadania wielu funkcji $\hat{h}(n)$ spełniających warunek (3.1) i braku globalnej dominacji jednej z nich nad pozostałymi, najlepszą funkcją $\hat{h}(n)$ dla węzła n jest ta, która przyjmuje dla tego węzła wartość największą.

3.8.4. Złożoność obliczeniowa algorytmów heurystycznyh

Pomimo znacznego zmniejszania liczby rozmnażanych węzłów algorytmy heurystyczne nadal mają złożoność wykładniczą – taką samą jak algorytmy nie wykorzystujące informacji o rozwiązywanym zadaniu.

Jednym ze sposobów oceny jakości algorytmu heurystycznego jest określenie *efektywnego współczynnika rozgałęzienia* $b^\star$ obliczanego na podstawie zadanej liczby przeglądanych węzłów N i liczby przebytych warstw d według poniższego wzoru

$$N = 1 + b^\star + (b^\star)^2 + (b^\star)^3 + \ldots + (b^\star)^d$$

Zastosowanie zwykłego algorytmu szukania (nie korzystającego z informacji heurystycznej) dla współczynnika rozgałęzienia $b^\star$ spowodowałoby przejrzenie tej samej liczby węzłów N. Np. gdyby algorytm heurystyczny znjadował rozwiązanie na głębokości $d = 7$ rozmnażając $N = 24$ węzły, wtedy jego efektywny współczynnik rozgałęzienia wynosiłby $b = 1.3$.

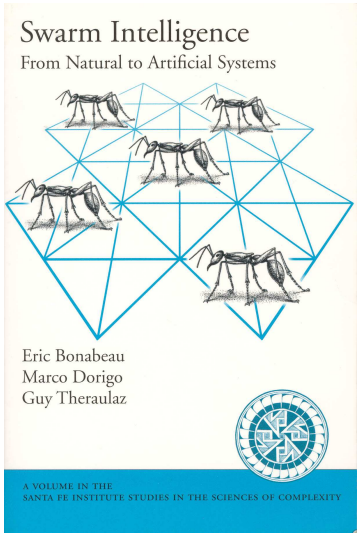
Przykładowe efektywne współczynniki rozgałęzienia przy rozwiązywaniu układanki 3×3 uzyskane dla metody iteracyjnego pogłębiania i metody $A^\star$ dla dwóch różnych funkcji oceniających h_1 i h_2 przedstawione zostało w tablicy 3.2.

3.8.5. Algorytmy mrówkowe

Algorytmy mrówkowe są jeszcze innymi oprócz algorytmów Dijkstry i A^* algorytmami znajdujacymi najtańszą drogę w grafach. Autorami tej metody są Bonebeau i współpracownicy (patrz rys. 3.9).

	Koszt szukania			Efekt. wsp. rozg.		
d	IP	$A^*(\hat{h}_1)$	$A^*(\hat{h}_2)$	IP	$A^*(\hat{h}_1)$	$A^*(\hat{h}_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Tablica 3.2: Przykładowy koszt szukania (liczba przeglądanych węzłów) i efektywny współczynnik rozgałęzienia przy rozwiązywaniu układanki 3×3 dla metody iteracyjnego pogłębiania (IP) i metody A^* z dwoma różnymi funkcjami heurystycznymi $\hat{h}_1$ i $\hat{h}_2$, gdzie $\hat{h}_1 =$ liczba klocków znajdujących się na niewłaściwej pozycji, $\hat{h}_2 =$ suma odległości klocków od swoich oczekiwanych pozycji końcowych (suma odległości pionowych i poziomych) (Russel, Norvig, 1995)



Rys. 3.9: Okładka monografii autorów algorytmów meówkowych [7]

Algorytmy te należa do grupy algorytmów naśladujących zachowanie się prostych organizmów żyjacych w wielkich społecznosciach, W szczególności algorytmy mrówkowe naśladują zachowanie mrówek lub termitów poszukujących pożywienia i następnie znajdujących najkrótsze drogi do pożywienia.

Mrówki – poszukiwanie pożywienia

Szacuje się, że na świecie żyje około 10^{16} mrówek w koloniach liczących do 30 milionów osobników. Złozone zachowanie (patrz rys. 3.10) wyłaniające się ze współdziałania owadów w koloniach zawsze bardzo intrygowało ludzi.

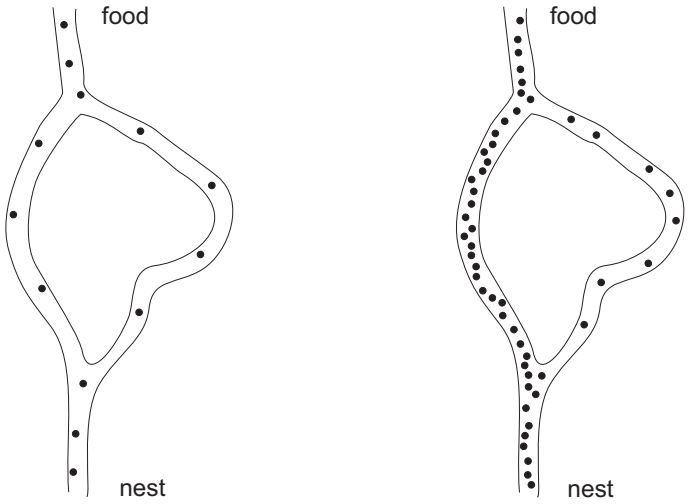
Rys. 3.10: Termitiery. http://upload.wikimedia.org/wikipedia/ćommons/7/73/Termite_Cathedral_DSC03570.jpg oraz http://upload.wikimedia.org/wikipedia/ćommons/ę/ęb/Termite_cathedral_mounds.jpg

Obserwując zachowanie się zbiorowiska mrówek zauważono, że mrówki początkowo poruszają się chaotycznie. Po znalezieniu pożywienia coraz więcej mrówek porusza się po tej samej drodzr do pożywienia, aby w końcu wszystkie mrówki poruszają się po jednej, najkrótszej, drodze. W rzeczywistych eksperymentach an mrówkach obserwowano sytuację przedstawioną na rys. 3.11. Stosunkowo niedawno wyjaśniono przyczyny takiego zachowania się mrówek. Okazało się, że mrówki poruszając się wydzielają i pozostawiają na drodze specyficzny zapachowy związek chemiczny, feromon. zostawiają w ten spsób ślad swojej drogi. Inne mrówki z kolei poruszając się wybierają z większym prawdopodobieństwem drogę na której jest silniejszy zapach feromonu. W ten sposób, jeżeli do pożywienia są możliwe drogi o różnej długości, to mrówki poruszając się początkowo losowo, stopniowo pozostawiają silniejszy ślad na drodze krótszej ponieważ przebywają na niej krócej. Początkowo niewielkie różnice w koncentracji zapachu szybko się powiększają dzięki temu, że następne mrówki wybierają krótszą drogę, bo na niej jest silniejszy zapach (dodatnie sprzężenie zwrotne).

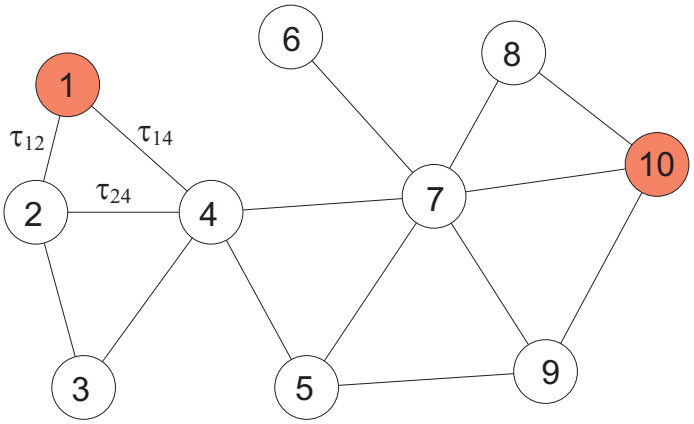
Algorytm mrówkowy

Powyżej omówione zachowanie mrówek nasladują algorytmy mrówkowe (ang ACO – ant colony optimization) poszukujące najtańszej drogi w grafie. Omówiony tu będzie najprostszy algorytm (ang. SACO – simple ACO).

Mrówka rozpoczyna swoją drogę od węzła startowego (patrz np. rys. 3.12). W każdym węźle mrówka decyduje którą krawędź wybrać. Jeżeli mrówka k znajduje



Rys. 3.11: Ilustracja zachowania się mrówek mających dwie możliwe drogi do pożywienia: (a) początkowo nieliczne mrówki znajdują się na drogach nie preferując wyraźnie żadnej z dróg, (b) po upływie pewnego czasu na drogach znajduje się wiele mrówek, jednak zdecydowanie wybierających drogę krótszą



Rys. 3.12: Algorytm mrówkowy – ilustracja działania. Zadanie znalezienia najkrótszej drogi między węzłami 1 i 10, τ_{ij} – całkowita ilość feromonu na krawędzi (ij)

się w węźle i w epoce t, wybiera jako następny węzeł j z prawdopodobieństwem

$$p_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}^\alpha(t)}{\sum\limits_{j \in \mathcal{N}_i^k} \tau_{ij}^\alpha(t)} & \text{jeżeli } j \in \mathcal{N}_i^k \\ 0 & \text{w przeciwnym razie} \end{cases} \tag{3.2}$$

gdzie $\mathcal{N}_i^k$ jest zbiorem dopuszczalnych, w odniesieniu do mrówki k, węzłów połączonych krawędziami z węzłem i (zakłada się, że mrówka nie zawraca), α jest dodatnią stałą, τ_{ij} jest całkowitą ilością feromonu na krawędzi (ij)

Gdy wszystkie mrówki przejdą drogę od węzła startowego do końcowego usuwane są wszystkie pętle które mrówki mogły przebyć i na wszystkich krawędziach (ij) na

drodze przebytej przez każdą mrówkę umieszcza się feromon w ilości

$$\Delta\tau_{ij}^k(t) \propto \frac{1}{L^k(t)}$$

gdzie $L^k(t)$ jest długość trasy przebytej przez mrówkę k w epoce t.

Całkowita ilość feromonu na krawędzi ij, na początku epoki $t+1$ wynosi

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t).$$

gdzie n_k jest liczbą mrówek.

Ilość feromonu może się zmniejszać (parowanie) przed dodaniem nowej porcji według wzoru

$$\tau_{ij}(t) = \rho\tau_{ij}(t)$$

gdzie $\rho \in [0,1]$.

Heurystyczny algorytm mrówkowy

Podobnie jak algorytm szukania A^*, algorytm mrówkowy może wykorzystywać dodatkowe informacje o jakości wezłów do których mrówka ma przejść. Można to np. zrealizować wybierając następujace prawdopodobieństwo przejścia mrówki do następnego węzła:

$$p_{ij}^k(t) = \begin{cases} \dfrac{\tau_{ij}^\alpha(t)\eta_{ij}^\beta(t)}{\sum\limits_{j\in\mathcal{N}_i^k} \tau_{ij}^\alpha(t)\eta_{ij}^\beta(t)} & \text{if } j \in \mathcal{N}_i^k \\ 0 & \text{w przeciwnym razie} \end{cases} \tag{3.3}$$

gdzie $\mathcal{N}_i^k$ jest zbiorem dopuszczalnych, w odniesieniu do mrówki k, węzłów połączonych krawędziami z węzłem i (zakłada się, że mrówka nie zawraca), α jest dodatnią stałą, τ_{ij} jest całkowitą ilością feromonu na krawędzi (ij), zaś $\eta_{ij}^\beta(t)$ reprezentuje atrakcyjność przejścia mrówki z węzła i do j (informacja heurystyczna).

Porównując wzory (3.2) i (3.3) widać, że wykorzystanie informacji heurystycznej zwiększa prawdopodobieństwo przejscia do węzła o wiekszej atrakcyjności. Miarą atrakcyjnością może tu być np. szacowany najtańszy koszt przejścia drogi od danego węzła do węzła końcowego.

3.9. Szukanie na grafach AND/OR

3.9.1. Systemy dekomponowalne – grafy AND/OR

Niektóre zadania daje się zdekomponować do zespołów zadań prostszych. Schemat dekompozycji wygodnie jest przedstawić w postaci grafu AND/OR.

Przykłady

1. Zadanie zarobienia dużej ilości pieniędzy – dobry przykład problemu dającego się przedstawić w postaci grafu AND/OR

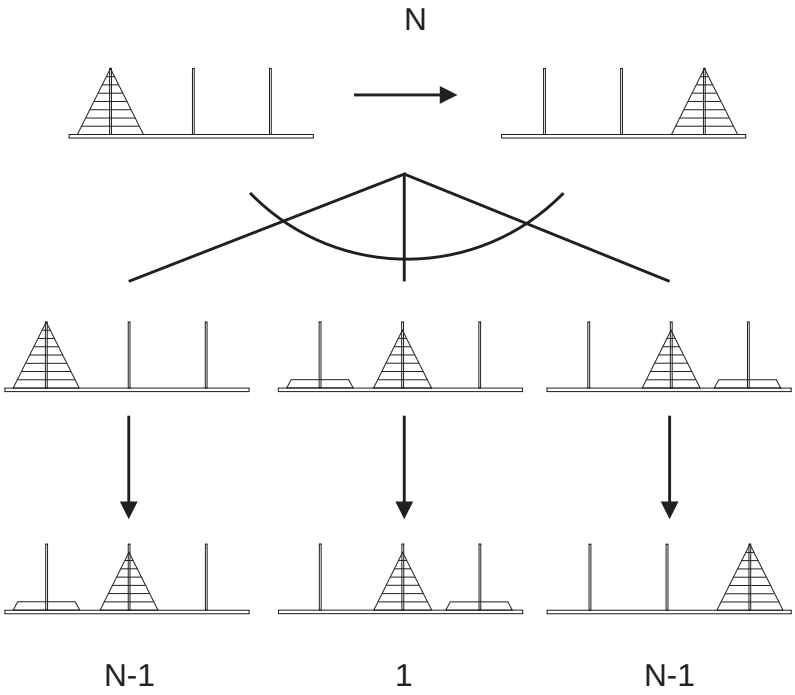
2. Wieże z Hanoi (patrz rys. 3.13)

$$move(N, lewy, środkowy, prawy) \leftarrow move(N-1, lewy, prawy, środkowy), \\ move(1, lewy, \_, prawy), \\ move(N-1, środkowy, lewy, prawy)$$

3. Gry (np. szachy, warcaby, gra w podział liczb – patrz przykład 3.3)

4. Dowodzenie twierdzeń

5. Całkowanie symboliczne (patrz rys. 3.14)

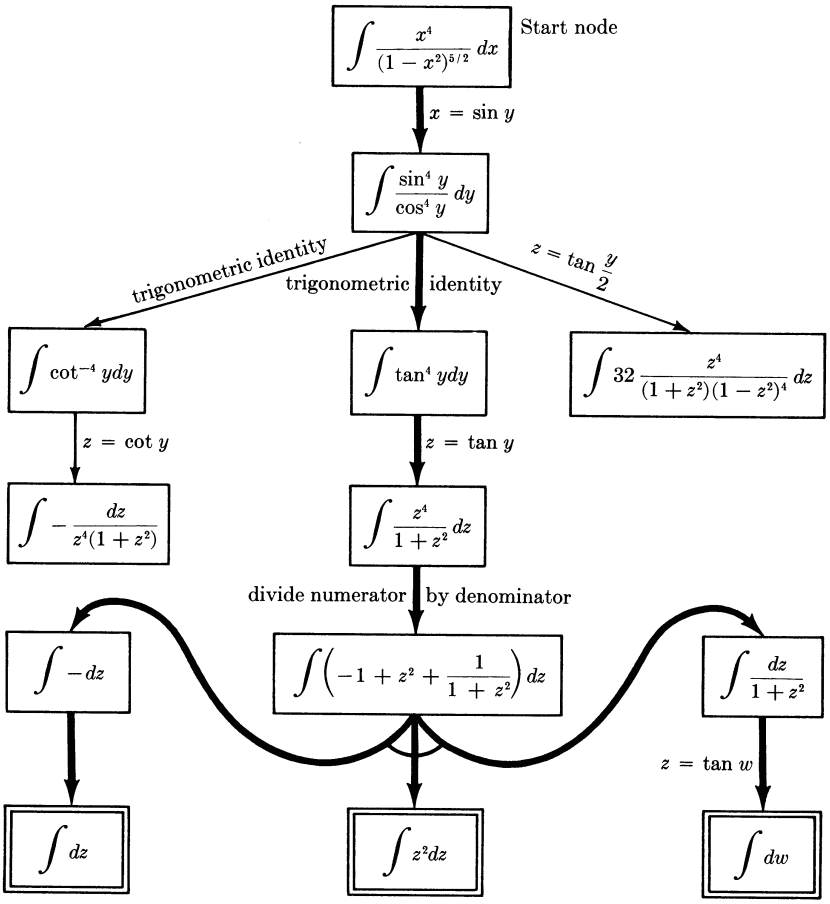


Rys. 3.13: Układanka „Wieże Hanoi". Dekompozycja zadania początkowego z N kręgami do dwóch zadań z $N-1$ kręgami i jednego zadania z jednym kręgiem.

3.9.2. Rodzaje węzłów

Będziemy dalej rozpatrywać grafy AND/OR jednorodne, to znaczy takie, których węzły niekońcowe są albo AND albo OR, natomiast nie są jednocześnie węzłami AND i OR (za wyjątkiem węzłów mających pojedyncze krawędzie potomne). Graf niejednorodny (zawierający węzły, będące jednocześnie węzłami AND i OR) może zawsze zostać zamieniony na graf jednorodny, poprzez wprowadzenie dodatkowych węzłow.

W jednorodnym grafie AND/OR istnieją następujące typy węzłów:



Rys. 3.14: Całkowanie symboliczne jako zadanie szukania na grafie AND/OR (Nilsson 1980)

- węzły końcowe spełniające kryterium celu (wygrywające),
- węzły niekońcowe typu OR lub AND,
- węzły końcowe nie spełniające kryterium celu (przegrywające).

W czasie poszukiwania rozwiązania węzłom nadaje się etykiety: „rozwiązywalny“ lub „nierozwiązywalny“ według następujących zasad:

Węzły rozwiązywalne

- węzły wygrywające są rozwiązywalne.
- węzeł niekońcowy typu OR jest rozwiązywalny wtedy i tylko wtedy, gdy przynajmniej jeden z jego potomków jest rozwiązywalny.
- węzeł niekońcowy typu AND jest rozwiązywalny wtedy i tylko wtedy, gdy wszystkie jego potomki są rozwiązywalne.

Węzły nierozwiązywalne

- węzły przegrywające są nierozwiązywalne.

- węzeł niekońcowy typu OR jest nierozwiązywalny wtedy i tylko wtedy, gdy wszystkie jego potomki są nierozwiązywalne.
- węzeł niekońcowy typu AND jest nierozwiązywalny wtedy i tylko wtedy, gdy przynajmniej jeden z jego potomków jest nierozwiązywalny.

Zadaniem metod szukania na grafach AND/OR jest stwierdzenie czy węzeł startowy jest rozwiązywalny i pokazanie drogi (dróg) wnioskowania to wykazującej (od węzła (węzłów) wygrywającego do startowego). Bardziej formalnie (Bolc, Cytowski 1989): wyznaczenie rozwiązania w grafie AND/OR oznacza znalezienie grafu rozwiązania (podgrafu AND/OR), spełniającego następujące własności:

- zawiera węzeł początkowy,
- wszystkie jego węzły końcowe są wygrywającymi,
- zawierając krawędź wywodzącą się z węzła AND zawiera wszystkie inne krawędzie wywodzące się z tego węzła.

Wiele gier dwuosobowych z pełną informacją może być reprezentowane przez grafy AND/OR z następującymi po sobie węzłami typu OR i AND.

Przykład 3.3

Gra w podział liczb. Grający kolejno rozdzielają jedną z istniejących liczb na dwie różne liczby. Grę przegrywa grający, który nie może wykonać ruchu (wszystkie liczby są liczbami jeden lub dwa). Gra rozpoczyna się od jednej liczby równej 7. Graf stanów gry przedstawiony jest na rys. 3.15. ■

3.9.3. Metoda szukania z powracaniem na grafach AND/OR

Do szukania na grafach AND/OR mogą być zastosowane, po odpowiedniej modyfikacji, wszystkie metody szukania omówione wcześniej.

W poniżej podanym algorytmie szukania z powracaniem wykorzystywane są następujące procedury:

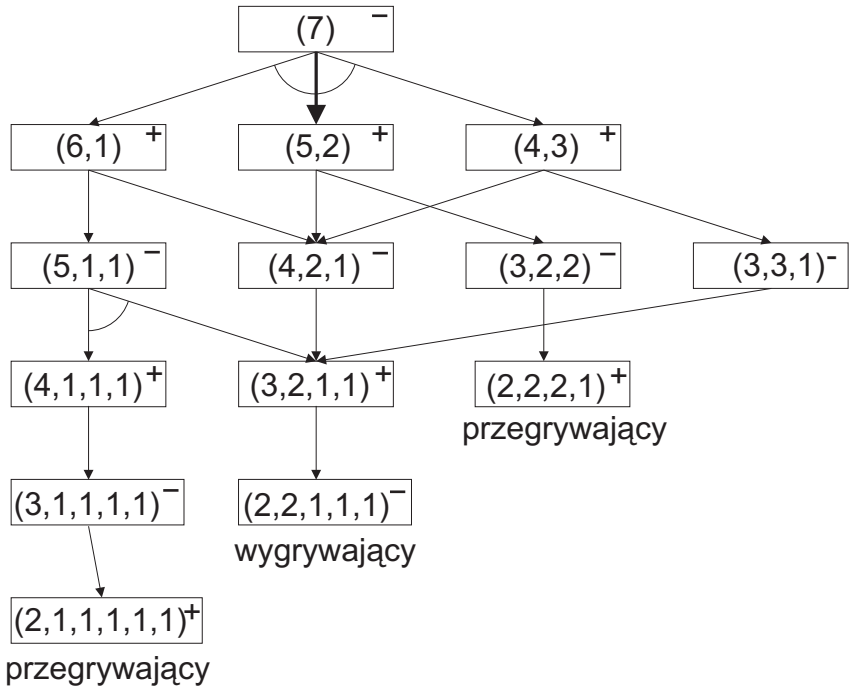
procedure ROZSZERZENIE_LISTY(O: lista, w: węzeł);
{procedura umieszcza węzeł w na początku listy O}

procedure ROZSZERZENIE_GRAFU(G: graf, w: węzeł);
{procedura umieszcza w grafie G węzeł w};

function WYBÓR_Z_LISTY(O: lista, G: graf): wezeł;
{funkcja zwraca pierwszy element z listy O należący do G};

procedure USUNIĘCIE_Z_LISTY(a: węzeł, O: lista);
{procedura usuwa węzeł a z listy O};

function TEST_KRAWĘDZI(a: węzeł): Boolean;
{funkcja zwraca wartość TRUE, gdy węzeł a ma wykorzystane wszystkie krawędzie lub wartość FALSE w przeciwnym przypadku};



Rys. 3.15: Graf stanów gry w podział liczb. Grę rozpoczyna gracz "−". Znak "+" w węźle grafu oznacza stan zastany przez gracza "+", zaś znak "−" stan zastany przez gracza "−". Pogrubione linie wskazują drzewo rozwiązania (strategię wygrywającą dla gracza "+")

function GENERACJA_WĘZŁA(w: węzeł): węzeł;
{funkcja generuje jeden węzeł potomny węzła w wzdłuż niewykorzystanej krawędzi};

procedure OZNACZENIE_KRAWĘDZI(a,b: węzeł);
{procedura zaznacza krawędź ab jako wykorzystaną}

procedure ZMIANA_ETYKIET(G: graf, w: węzeł);
{procedura modyfikuje etykiety wezłów grafu (zgodnie z zasadami podanymi powyżej) w zależności od etykiety węzła w};

procedure USUWANIE_WĘZŁÓW_GRAFU(G: graf);
{procedura usuwa z grafu G wszystkie węzły mające etykiety N};

function TEST(w: węzeł, W: własność): Boolean;
{funkcja zwraca wartość TRUE, gdy węzeł w ma własność W lub wartość FALSE w przeciwnym przypadku};

W trakcie wykonywania algorytmu badanym węzłom w nadawane są etykiety przechowywane w tablicy E(w). Etykiety nadawane są zgodnie z zasadą:

$$E[w] := \begin{cases} R & \text{gdy węzeł jest rozwiązywalny} \\ N & \text{gdy węzeł jest nierozwiązywalny} \\ P & \text{gdy nie ustalono jeszcze rozwiązywalności węzła} \end{cases}$$

Algorytm szukania z powracaniem (Bolc, Cytowski 1989):

procedure POWRACANIE_NA_AND_OR(p: węzeł początkowy;

```
                C:  własność spełnienia kryterium celu (wygrana),
                K:  własność nie spełnienia kryterium celu (przegrana));
begin
  ROZSZERZENIE_LISTY(O,p);
  ROZSZERZENIE_GRAFU(G,p);
  E[p]:=P;
  while E[p]=P do
  begin
    a:=WYBÓR_Z_LISTY(O,G);
    if TEST_KRAWĘDZI(a) then
    USUNIĘCIE_Z_LISTY(a,O)
    else
    begin
      b:=GENERACJA_WĘZŁA(a);
      E[b]:=P;
      ROZSZERZENIE_LISTY(O,b);
      ROZSZERZENIE_GRAFU(G,b);
      OZNACZENIE(a,b);
      if TEST(b,C) then E[b]:=R;
      if TEST(b,K) then E[b]:=N;
      if E[b]<>P then
      begin
        ZMIANA_ETYKIET(G,b);
        USUWANIE_WĘZŁÓW_GRAFU(G);
      end;
    end;
  end;
  if E[p]=R then SUKCES;
  if E[p]=N then NIEPOWODZENIE
end;
```

3.10. Metody minimax

Dla złożonych gier dwuosobowych liczba węzłów grafu gry jest bardzo duża. Na przykład ocenia się, że w szachach średni współczynnik rozgałęzienia wynosi 35, co przy grach trwających 50 posunięć wykonanych przez każdego gracza daje około $35^{100} \approx 10^{120}$ węzłów. Liczba możliwych ustawień 32 figur na szachownicy o 64 polach (liczba wariacji bez powtórzeń 32 elementów z 64) wynosi $4,82 \cdot 10^{53}$, a po uwzględnieniu ilości ustawień mniejszych liczb figur wynosi $4,97 \cdot 10^{53}$. Skądinąd liczba dopuszczalnych stanów gry wynosi około 10^{40}. Z powodu tak dużej liczby stanów niemożliwe jest stwierdzenie istnienia strategii wygrywającej i jej wyznaczenie.

Zamiast poszukiwania strategii wygrywającej trzeba się ograniczyć do znalezienia „dobrego“ ruchu w danej sytuacji. Ruch ten jest wykonywany, czeka się na odpowiedź przeciwnika i ponownie wyznaczany jest dobry ruch. Proces szukania może odbywać się jedną z omawianych wcześniej metod, natomiast zmienić należy kryterium zakończenia szukania (nie jest to teraz znalezienie strategii wygrywającej lub wykrycie

jej nieistnienia).

Powszechnie stosowanym sposobem wyznaczania najlepszego ruchu jest metoda *minimax*. Polega ona na wyznaczeniu w specyficzny sposób oceny danego węzła na podstawie ocen węzłów znajdujących na określonej głębokości w stosunku do niego.

Jakość prowadzonej gry zależy od jakości funkcji oceniającej stan gry i od głębokości szukania.

Funkcja oceniająca stan gry ma decydujący wpływ na jakość prowadzonej gry. Gdyby pozwalała bezbłędnie wskazywać stany lepsze, to zniknełaby potrzeba prowadzenia procesu szukania. W praktyce stosuje się *funkcje ważone* będące kombinacją liniową wartości różnych częściowych ocen f stanu gry:

$$w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

gdzie $w_i, i = 1, 2, \ldots, n$ są arbitralnie dobranymi współczynnikami.

Przykład 3.4

Gra w kółko i krzyżyk. Dwóch graczy: X (my – stawiamy znaki $\times$) i O (znaki $\circ$). Przyjmujemy następujące kryterium oceny stanu gry:

$e(p)$ = (liczba wierszy, kolumn i przekątnych ciągle dostępnych dla X) − (liczba wierszy, kolumn i przekątnych ciągle dostępnych dla O)

Na rysunkach 3.16, 3.17 i 3.18 przedstawione są kolejne etapy gry w kółko i krzyżyk przy zastosowaniu strategii minimax o głębokości 2. ∎

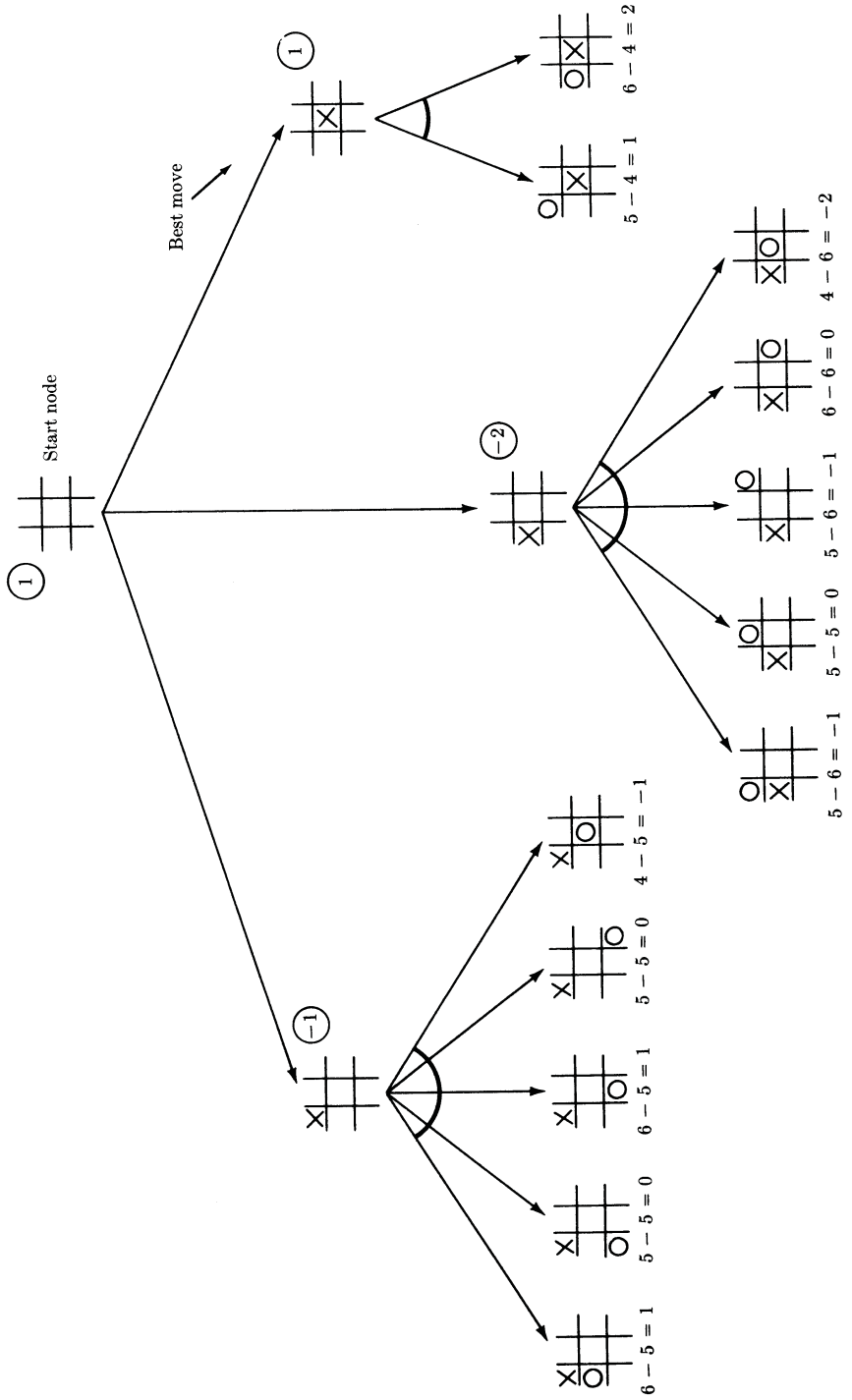
3.10.1. Szachy komputerowe

Szachy powszechnie są uważane za grę wymagająca inteligencji i jak to ujął Goethe – „szachy to kamień probierczy umysłu". Uważano, że zbudowanie maszyny wybitnie grającej w szachy będzie dowodem, że myślenie można modelować lub przeciwnie, że szachy nie wymagają myslenia. Każda z tych konkluzji zmienia powszechnie przyjętą koncepcje inteligencji.

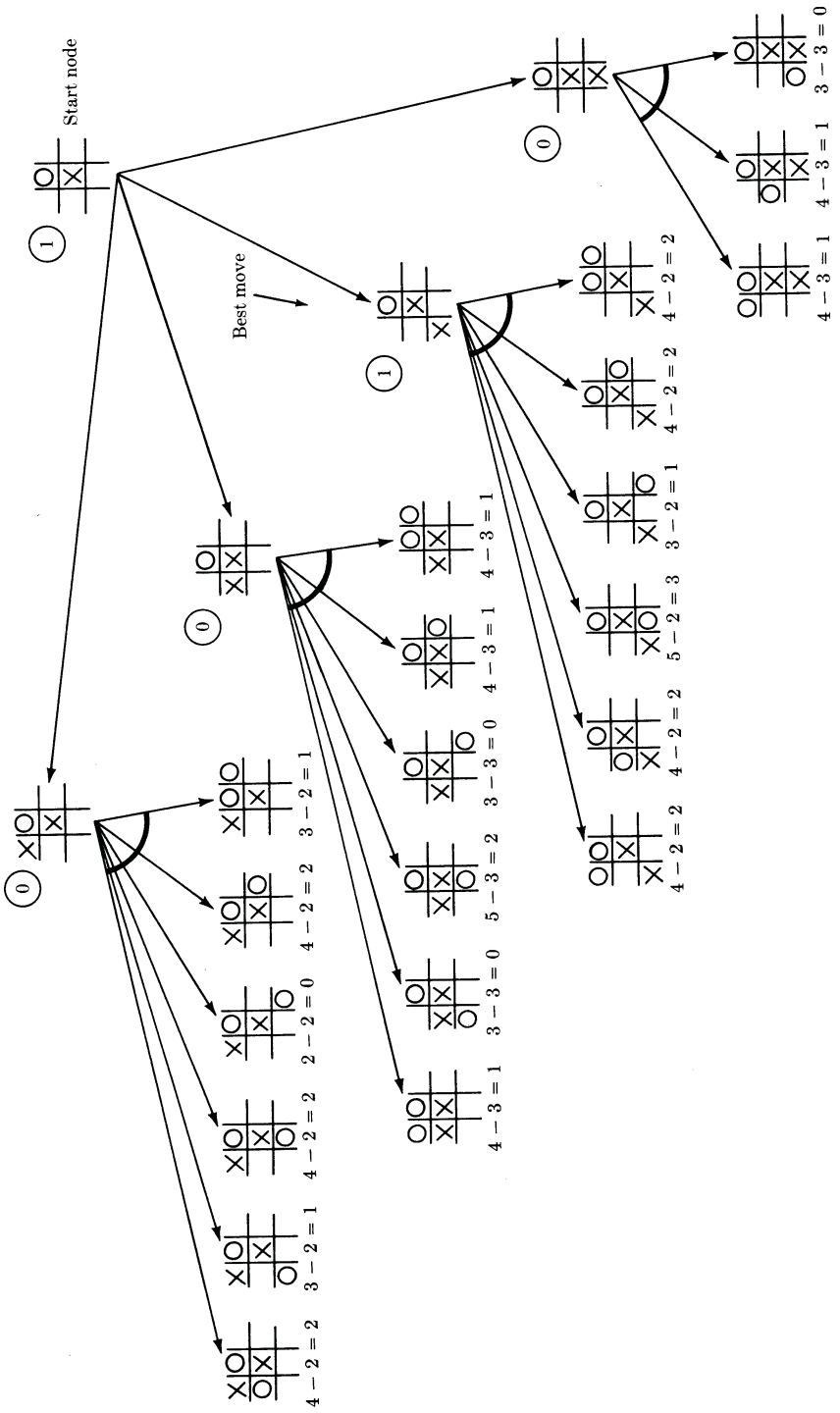
Kryteriami oceny stanu gry w szachach mogą być: liczba figur posiadanych przez graczy, liczba możliwych ruchów w danym stanie, otwarcie głownych przekątnych otwarte dla własnych figur, wzajemnie bronione piony i opanowanie przez nie środka pola itp.

Kilka sytuacji na szachownicy wraz ze słownymi ich ocenami jest przedstawionych na rys. 3.19.

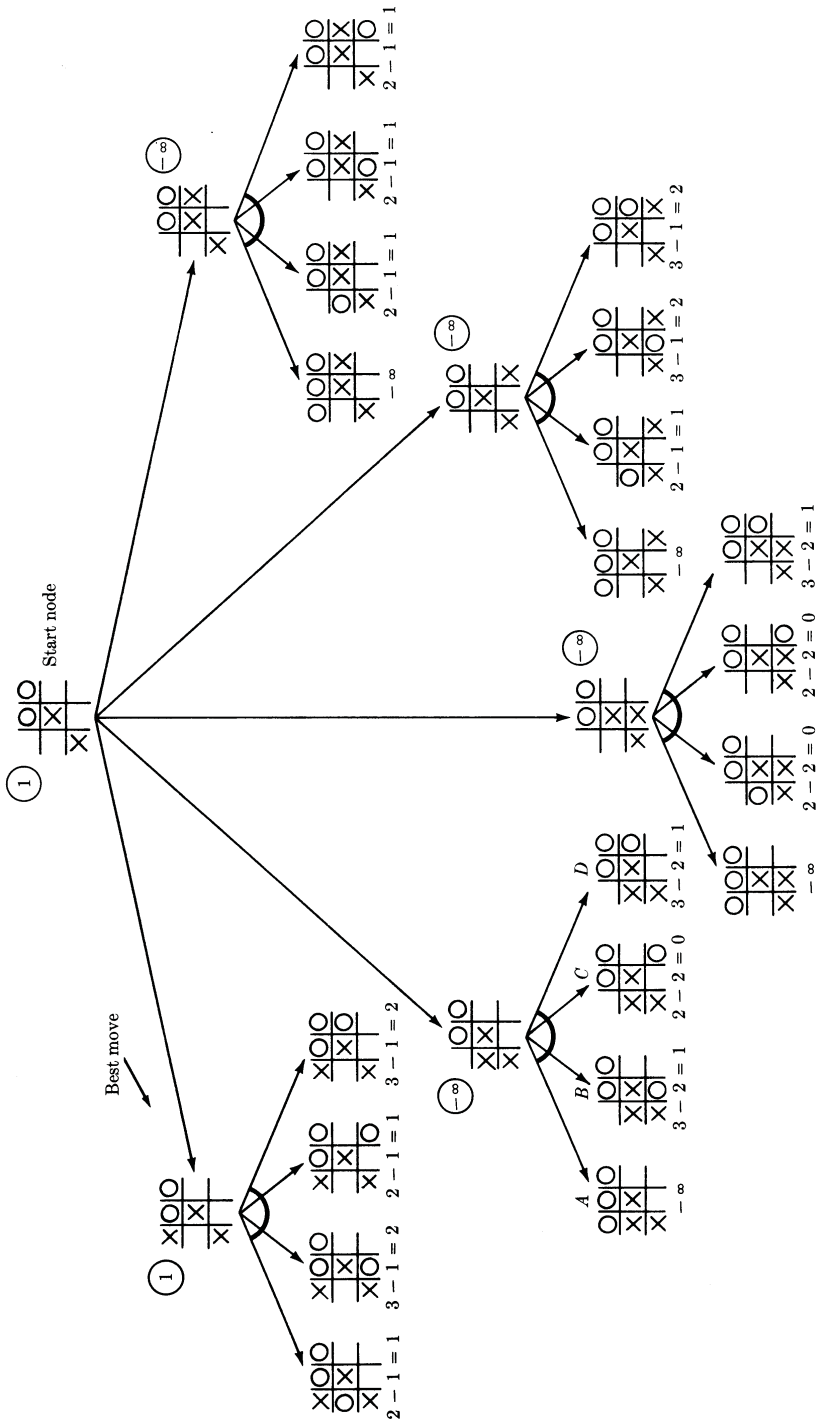
Oddzielnym problemem jest dobranie głębokości szukania. Naturalnym jest przyjęcie stałej głębokości. Może to jednak prowadzić do niebezpiecznych sytuacji wynikających z niedoskonałości funkcji oceny stana gry. Np. przyjmijmy dla gry prowadzonej przez białe, że stan przedstawiony na rys. 3.19d znajduje się na maksymalnej głębokości oraz, że następny ruch mają czarne. Ponieważ białe mają przewagę w sile figur, ocena stanu jest wysoka przy zastosowaniu typowej funkcji oceniającej preferującej siłę figur. Jednak w następnym ruchu czarne mogą zbić hetmana i uzyskać pozycję wygraną, ale program to uwzględniający musiałby prowadzić szukanie o jeden krok głębiej. W związku z tym stosuje się zwykle zmienną głębokość szukania.



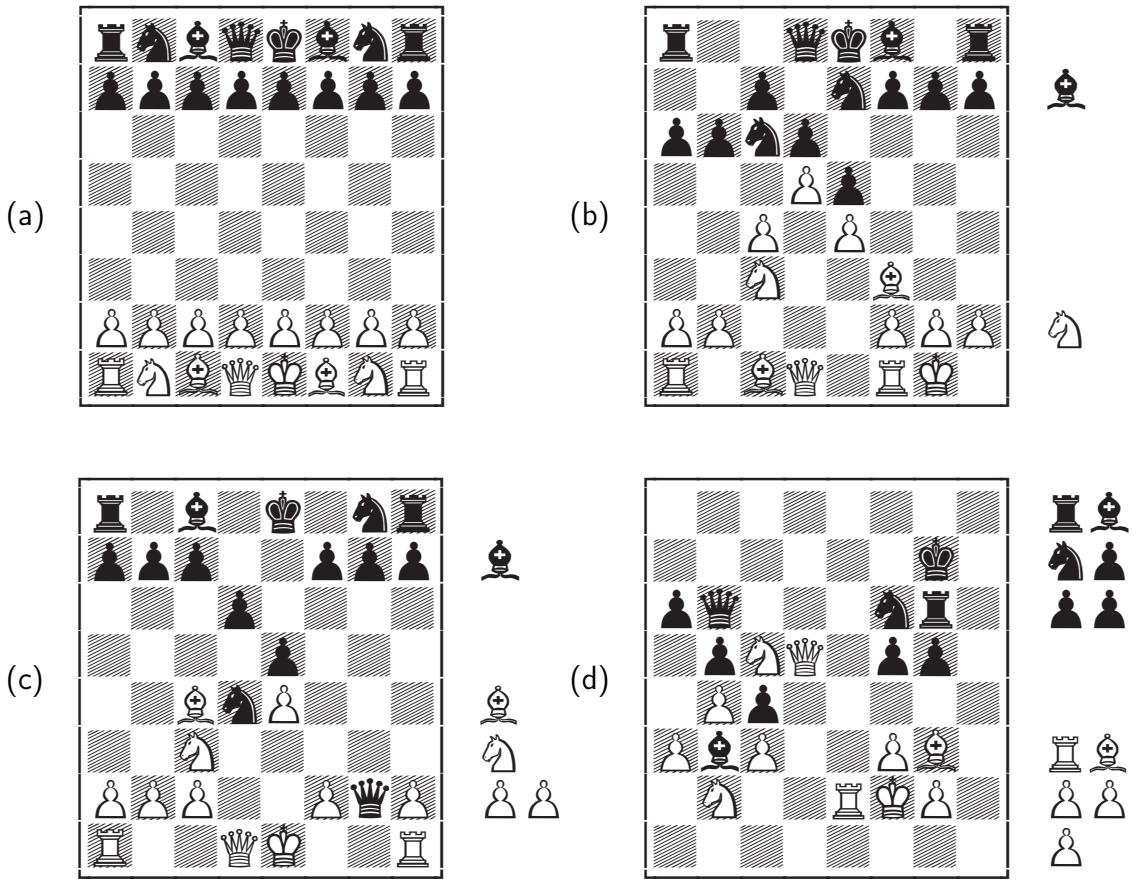
Rys. 3.16: Gra w kółko i krzyżyk — etap 1. Wyznaczenie najlepszego ruchu metodą minimax



Rys. 3.17: Gra w kółko i krzyżyk — etap 2. Wyznaczenie najlepszego ruchu metodą minimax



Rys. 3.18: Gra w kółko i krzyżyk — etap 3. Wyznaczenie najlepszego ruchu metodą minimax



Rys. 3.19: Kilka sytuacji na szachownicy i ich ocena: (a) Początek gry – ruch białych. Stan wyrównany. (b) Ruch czarnych. Białe stoją nieznacznie lepiej. (c) Ruch białych. Czarne wygrywają. (d) Ruch czarnych. Białe blisko przegranej

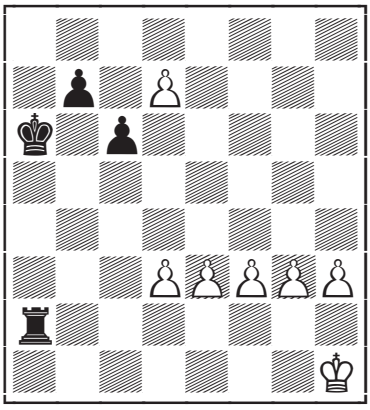
Dopuszcza się zakończenie szukania, gdy widać na podstawie dodatkowych testów, że stan gry nie ulegnie gwałtownym zmianom w najbliższej przyszłości.

Innym problemem przy stosowaniu metody minimax jest trudność dostrzeżenia oczywistego (dla człowieka) rozwiązania znajdującego się poza maksymalną głębokością szukania. Rozpatrzmy sytuację szachową przedstawioną na rys. 3.20a. Czarne mają nieznaczną przewagę materialną ale białym wystarczy jeden ruch pionem do uzyskania hetmana. Jeżeli czarne rozpoczną szachowanie białego króla, to mogą przez szereg ruchów powstrzymać białe od zdobycia hetmana ale po określonej liczbie posunięć białe go uzyskają po odpowiedniej grze. Problemem jest to, że odpowiednia w tym przypadku gra białych może nie być znaleziona, jeżeli rozwiązanie znajduje się poza maksymalną głębokością szukania.

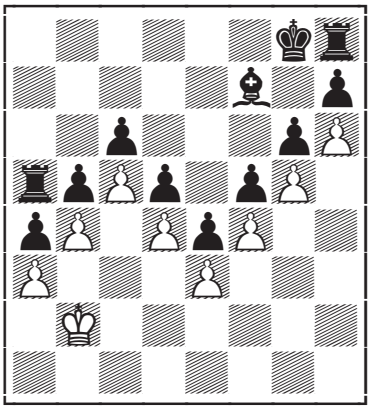
Podobnie, w sytuacji przedstawionej na rys. 3.20b każdy człowiek znający zaledwie zasady gry widzi, że białe mogą łatwo utrzymać remis poruszając się wyłącznie królem. Jak podano w pracy Seymour, Nordwood (1993) w tej sytuacji program Deep Thought grając białymi zbił wieże, co pozwoliło czarnym przedostać się przez barierę pionów i łatwo wygrać.

Od początku istnienia komputerów tworzono programy grające w szachy. Jednym

(a)



(b)



Rys. 3.20: Dwie sytuacje kiedy znalezienie odpowiedniego ruchu wymaga analizy gry na dużą liczbę kroków w przód, podczas gdy nawet początkujący szachista łatwo znajduje rozwiązanie. (a) Seria szachów przez czarne przesuwa nieuniknione uzyskanie hetmana przez białe poza „horyzont“ metody szukania, (b) białym wystarcza wyłącznie dowolne poruszanie się królem, podczas gdy ulegnięcie „pokusie” zbicia czarnej wieży prowadzi białe do przegranej

z inicjatorów tej dziedziny był twórca teorii informacji Claude E. Shannon (patrz artykuł w Scientific American z roku 1950 (Shannon 1950)). W ciągu 50 lat poziom gry komputerów zmienił się od gry na bardzo niskim poziomie do gry mistrzowskiej (rys 3.21). W roku 1997 w komputer Deep Blue pokonał mistrza świata Kasparowa w stosunku 3:2 (patrz artykuły twórców koncepcji i oprogramowania tego komputera (Hsu i inni 1990, Campbell i inni 2002)).

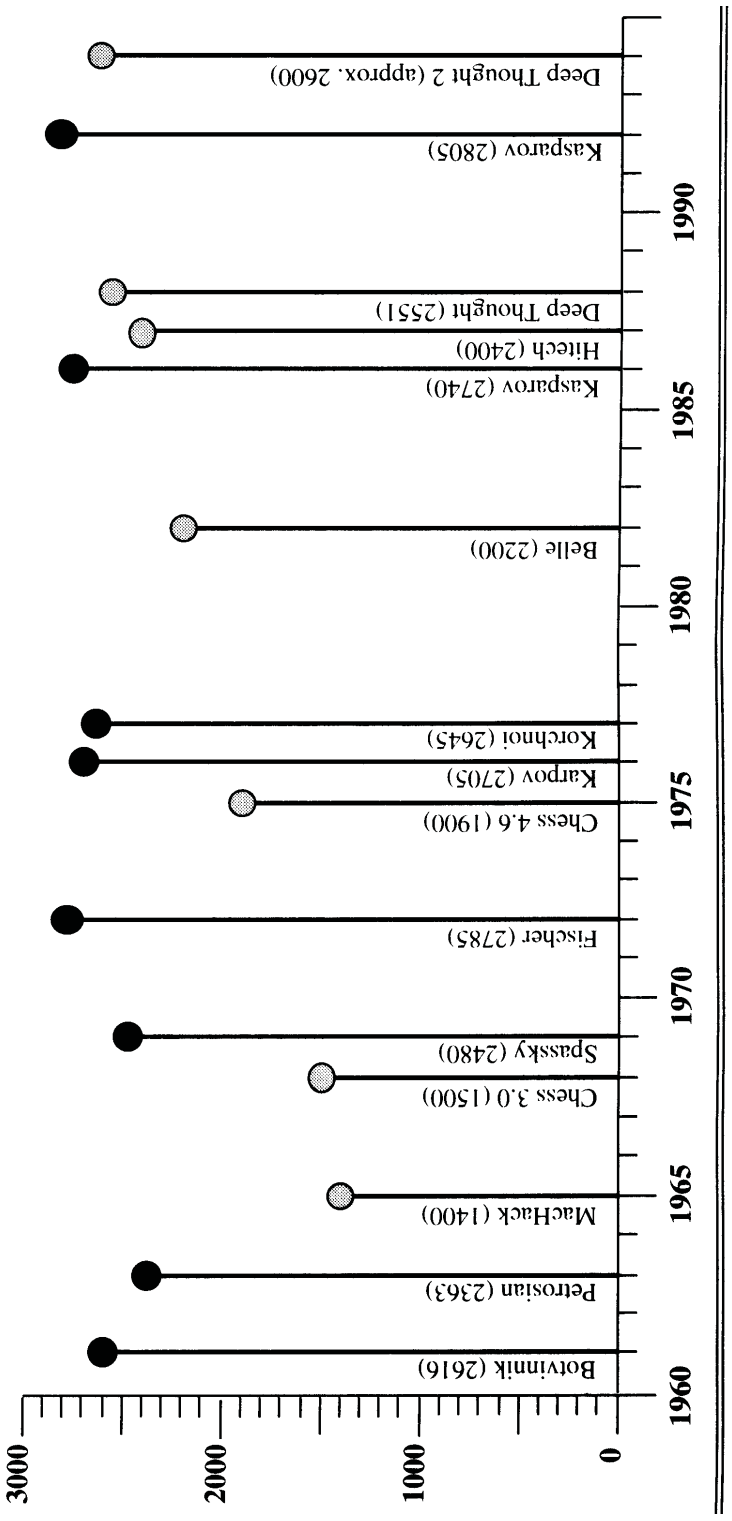
Komputer Deep Blue ma strukturę równoległą. System składa się z 30-węzłowego (30-procesorowego) komputera IBMRS/6000 SP i 480 jedno-układowych szachowych maszyn szukających. Każdy procesor komputera współpracuje z 16 maszynami szukającymi. System SP zawiera 28 węzłów z procesorem P2SC 120 MHz i 2 węzły z procesorem P2SC 135MHz. Węzły komunikują się ze sobą przez szybki przełącznik. Każdy węzeł wyposażony jest w RAM 1GB i dysk 4GB. Maszyny szukające zdolne są do przeglądania 2 do 2.5 miliona pozycji szachowych na sekundę. Cała struktura pozwalała na uzyskanie średnich szybkości przeglądania wynoszących $1,2 \times 10^8$ pozycji szachowych na sekundę.

Przy konstrukcji funkcji oceniającej zastosowano uczenie z nauczycielem (patrz rozdział 4). Polegało ono na automatycznym wyznaczaniu wag w funkcji oceniającej. Algorytm uczenia modyfikował wagi w taki sposób aby wyznaczany najlepszy ruch był zgodny z ruchem w przykładowych partiach szachowych granych przez wybitnych szachistów. Ogólnie, na wartość funkcji oceniającej mogło mieć wpływ około 8000 cech pozycji szachowych.

Innym sposobem uczenia była modyfikacja wag na podstawie różnicy pomiędzy wartością funkcji oceniającej dany stan a wartością oceny tego stanu uzyskaną metodą minimax po przeanalizowaniu grafu na dużą głębokość.

Inną wprowadzoną innowacją było zwiększanie głębokości szukania dla tych pozycji z których istnieje pojedynczy wyraźnie lepszy od innych ruch.

W październiku 2002 odbył się kolejny prestiżowy mecz szachowy pomiędzy czło-



Rys. 3.21: Ranking jakości gry w szachy ludzi i komputerów

wiekiem i komputerem. Poniżej przytoczony jest komentarz o tym meczu napisany przez polskiego arcymistrza szachowego zamieszczony w dzienniku „Rzeczpospolita" 23 października 2002r.

„Elektroniczni bracia. W Bahrajnie odbył się wielokrotnie przekładany mecz między zawodowym mistrzem świata Rosjaninem Władimirem Kramnikiem i najlepszym obecnie komputerowym programem szachowym „Deep Fritz", zainstalowanym na potężnym zespole multiprocesorowym. Ten pojedynek stanowił w pewnym sensie rewanż za mecz Garriego Kasparowa z superkomputerem „Deep Blue" sprzed pięciu lat, który ówczesny mistrz świata przegrał. Rewanż się nie udał – mimo ponadrocznego przygotowania Kramnik musiał się zadowolić remisem 4:4. Teoretycznie wiadomo, jak należy grać z komputerami. „Elektroniczni bracia" zdecydowanie górują nad ludźmi w tzw. taktyce szachowej, czyli rozpatrywaniu możliwych w danej pozycji wariantów i znajdowaniu ukrytych kombinacji. Natomiast strategia, czyli dokładna ocena pozycji i tworzenie długofalowych planów, jest ich piętą achillesową. Problem w tym, że bardzo trudno stworzyć pozycję, w której o zwycięstwie decyduje tylko i wyłącznie strategia, tym bardziej że nawet w trudnej sytuacji komputer się nie załamuje, lecz walczy do końca, wykorzystując wszystkie szanse lepiej od najwytrwalszego mistrza obrony.

A jednak w pierwszych trzech partiach mistrz świata umiejętnie stwarzał na szachownicy pozycje, w których komputer był bezradny (Kramnik prowadził wówczas 2,5:0,5). Wczesna wymiana hetmanów, wyraźnie zmniejszająca rolę taktyki, potem gromadzenie strategicznych plusów i komputer „dostrzegał" niebezpieczeństwo dopiero wtedy, gdy nawet najlepsza obrona nie mogła nic pomóc.

Od czwartej partii jednak wszystko się zmieniło. Autorzy programu wprowadzili zmianę zalecającą komputerowi unikanie wymiany hetmanów. Z kolei Kramnik jakby stracił koncentrację. W piątej partii w gorszej, ale remisowej pozycji popełnił fatalny błąd i musiał się poddać. W szóstej zaś zrealizował wygrywającą na pierwszy rzut oka kombinację, ale komputer znalazł ukryte fenomenalne wręcz obalenie (to właśnie maszyna umie najlepiej). Załamany Rosjanin uznał się za pokonanego w pozycji, w której jeszcze miał szanse na remis. Na tym mecz praktycznie się skończył. W pozostałych partiach człowiek nawet nie próbował wygrać, lecz konsekwentnie, z pełnym zresztą poparciem elektronicznego rywala, doprowadzał je do remisu. Mecz potwierdził, że siłą gry najlepsze komputery mimo wyżej wymienionych wad już nie ustępują czołowym szachistom świata. Jaki to będzie miało wpływ na dalszy rozwój szachów pokaże czas.

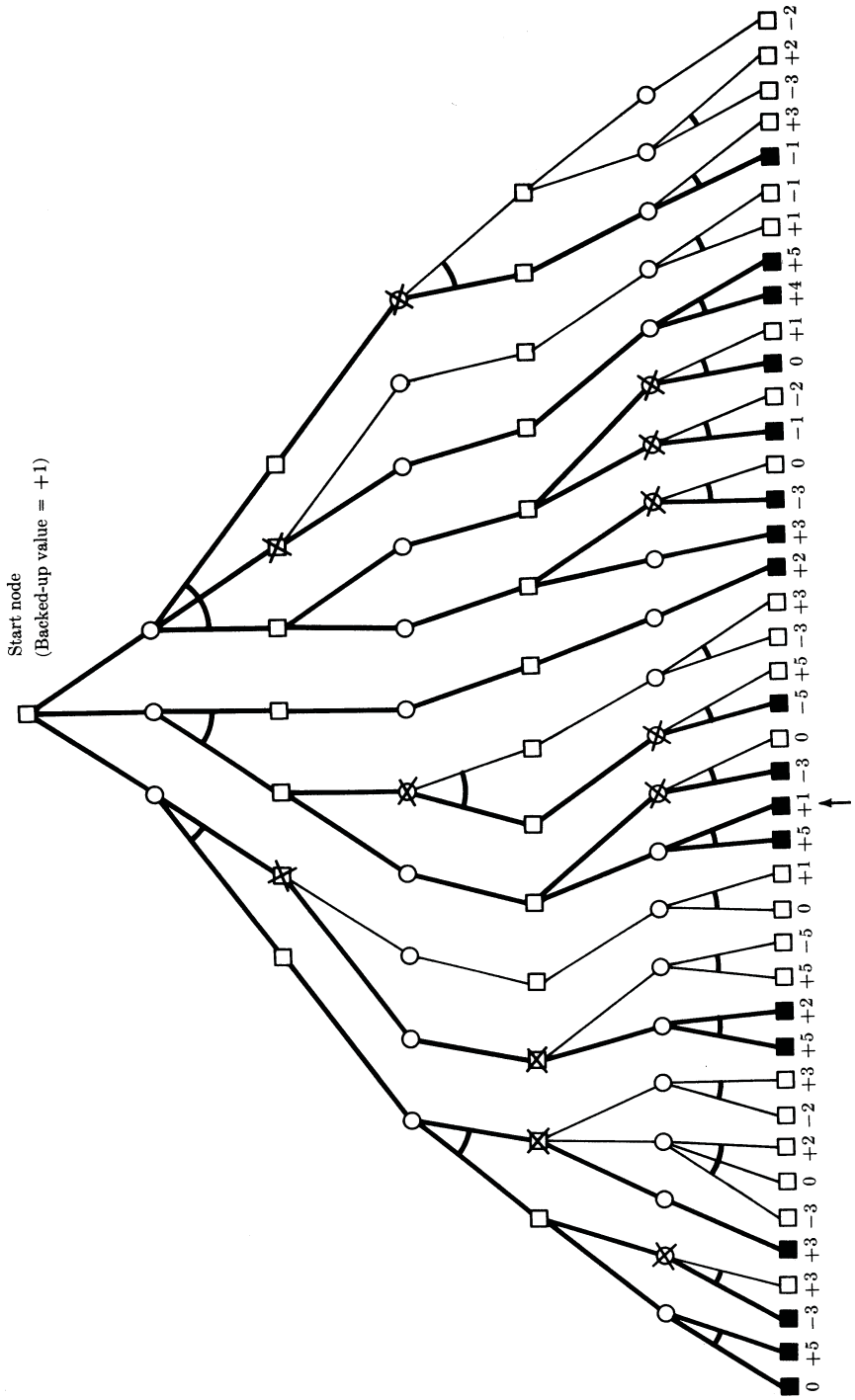
Michał Krasenkow Autor jest arcymistrzem szachowym."

3.11. Metoda $\alpha - \beta$

3.12. Uwagi

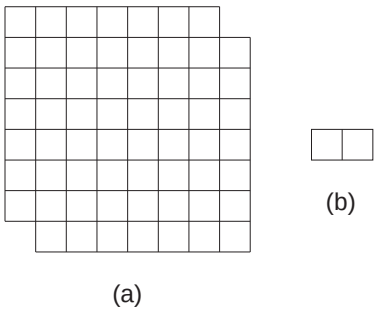
Stosowanie metod szukania jest uzasadnione, gdy nie jest znany algorytm rozwiązania zadania.

Przykład 3.5



Rys. 3.22: Ocena wartości węzła startowego grafu metodą $\alpha - \beta$. Na jego wartość wpływają wyłącznie wartości zaczernionych węzłów w ostatniej warstwie

Rozpatrzmy następujące zadanie: Sprawdzić czy możliwe jest pokrycie figury przedstawionej na rys. 3.23a 31 figurami o kształcie przedstawionym na rys. 3.23b. Aby to sprawdzić można zastosować jedną z metod szukania, ale jak sie okazuje, odpowiedź można otrzymać natychmiast na drodze prostego rozumowania logicznego:



Rys. 3.23: Czy figurę (a) można pokryć wieloma figurami (b)?

Figura przedstawiona na rys 3.23b ma kształt szachownicy, z której wycięto dwa pola. Jeżeli pokolorujemy jej pola na przemian kolorami białym i czarnym, to widać, że figura przedstawia szachownicę, z której wycięto dwa pola o tych samych kolorach i tym samym liczba pól białych i liczba pól czarnych są różne. Każde ustawienie na szachownicy figury z rys. 3.23b pokrywa jedno pole białe i jedno pole czarne. Ponieważ liczby pól białych i czarnych są różne więc niemożliwe jest ich pokrycie.

Poniżej przedstawiony jest inny interesujący przykład wskazujący na występowanie w przyrodzie zjawisk, których stan równowagi stanowi jednocześnie rozwiązanie pewnych zadań.

Przykład 3.6

Zamiast rozwiązywać metodą równomiernych kosztów zadanie znalezienia najkrótszej drogi pomiędzy dwoma miastami, wystarczy wykonać ze sznurka model, w którym węzły reprezentujące miasta połączone są sznurkami o długościach proporcjonalnych do długości drogi pomiędzy nimi. Wystarczy teraz rozciągnąć dwa węzły odpowiadające miastom, pomiędzy którymi poszukiwana jest droga, a uzyskany napięty sznurek reprezentował będzie najkrótszą drogę pomiędzy nimi.

W przeszłości postępowano podobnie stosując maszyny analogowe. Przez odpowiednie połączenie wzmacniaczy operacyjnych realizujących całkowanie i przekształtników nieliniowych otrzymywano układ elektryczny, w którym przebieg napięcia w czasie odpowiadał rozwiązaniu zadanego równania różniczkowego.

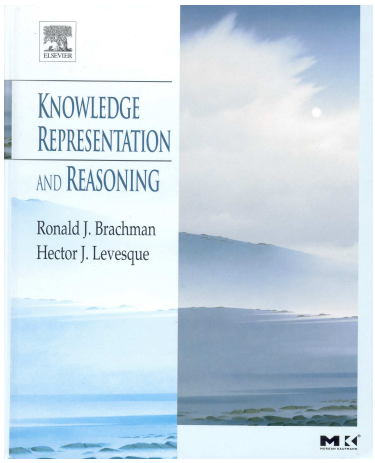
Rozdział 4

Reprezentacja wiedzy i wnioskowanie

Metody reprezentacji wiedzy i automatycznego wnioskowanie są ważnym działem informatyki intensywnie obecnie rozwijanym. Dział ten, łącznie ze zbliżonymi tematycznie metodami przetwarzania języka naturalnego można zaliczyć też do dziedziny sztucznej inteligencji. W rozdziale omówione zostaną podstawowe sposoby reprezentacji wiedzy: język logiki pierwszego rzędu i logika opisowaa.

4.1. Logika pierwszego rzędu

Logika pierwszrgo rzędu zwana też rachunkiem predykatów jest językiem formalnym, w którym można przedstawić opis pewnej rzeczywistości (w postaci faktów i reguł) i przeprowadzać w nim wnioskowanie. Rachunek predykatów jest rozszerzeniem algebry Boole'a poprzez wprowadzenie zmiennych i kwantyfikatorów. Stworzony został na przełomie XIX i XX wieku – badania nad nim rozpoczęli Ch. S. Peirce, G. Frege, B. Russel. Ważnym okresem w rozwoju rachunku predykatów były lata 30., w których badano zagadnienia pełności i rozstrzygalności systemów wnioskowania (Gödel, Herbrand, Skolem, Tarski). W 1965 A. Robinson podał algorytm dowodu oparty o rezolucję. Duży wkład w rozwój logiki matematycznej wnieśli też uczeni polscy w



Rys. 4.1: Okładka znanej monografii opisującej metody reprezentacji wiedzy i wnioskowania [8]

okresie przed drugą wojną światową. Np. klasyczna monografia Churcha (1956) cytuje prace następujących uczonych: Chwistek, Jaśkowski, Kuratowski, Leśniewski, Łukasiewicz, Mostowski, Rasiowa, Słupecki, Sobociński.

4.1.1. Elementy rachunku predykatów

Podstawowymi elementami rachunku predykatów sa:

predykaty które reprezentują własność elementu lub relację pomiędzy elementami. Argumentami predykatów mogą być zmienne (oznaczane tutaj słowami rozpoczynającymi się od dużych liter), stałe (oznaczane słowami rozpoczynające się od małej litery lub w cudzysłowie) oraz funkcje (oznaczane literami f, g, h). Na przykład predykat $ojciec(X, Y)$ może reprezentować relację ojcostwa. Podstawiając różne elementy za X i Y wartością predykatu jest prawda (T), gdy X jest ojcem Y lub fałsz (F), w przeciwnym przypadku.

operatory logiczne oddziałujace na predykaty. Najczęściej są to operatory: *nie* (*not*) ($\neg$), *i* (*and*) ($\wedge$), *lub* (*or*) ($\vee$), *jeżeli to* (*if*) ($\rightarrow$) (tablica 4.1).

a	$\neg a$
0	1
1	0

a	b	$a \vee b$	$a \wedge b$	$a \rightarrow b$
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	1

Tablica 4.1: Definicje podstawowych operatorów logicznych stosowanych w wyrażeniach predykatowych

kwantyfikatory które reprezentują zdania : *dla wszystkich* X (kwantyfikator uniwersalny $\forall X$) i *istnieje* X (kwantyfikator egzystencjalny $\exists X$).

Prawidłowe wyrażenia rachunku predykatów nazywane są *wyrażeniami poprawnie zbudowanymi* (*wpz*).

Przykład 4.1

Podamy tu kilka przykładów wpz.

Wyrażenie

$$mrówka(X) \wedge inteligentny(X)$$

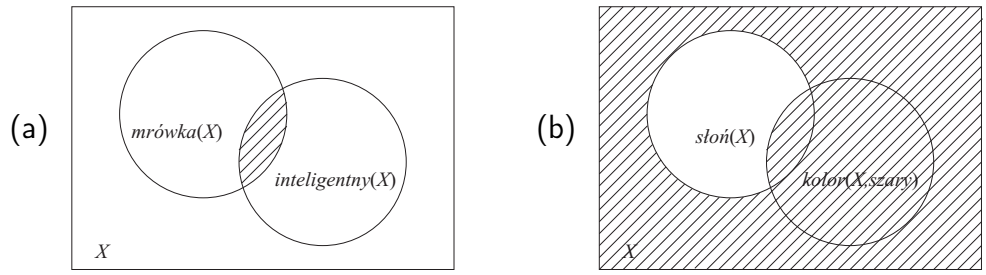
składa się z dwóch predykatów i nadal ma charakter predykatu, który może np. przyjmować wartość *prawda* (T) dla X będących inteligentnymi mrówkami [1] (rysunek 4.2a).

Wyrażenie

$$słoń(X) \rightarrow kolor(X, szary)$$

przyjmuje wartość T dla przedmiotów X nie będących słoniami lub będących koloru szarego co ilustruje diagram Venna na rys. 4.2b.

[1] Słowo „może" użyte tutaj i w dalszych przykładach oznacza, że wyrażenie predykatowe podanego znaczenia nabiera dopiero po odpowiedniej, jednej z wielu wielu możliwych, *interpretacji*, co wyjaśnione będzie później



Rys. 4.2: Diagramy Venna wyrażeń: (a) $mrówka(X) \wedge inteligentny(X)$ oraz (b) $słoń(X) \rightarrow kolor(X, szary)$

Poprawne jest wyrażenie

$$p(f(X), Y))$$

gdzie f nazywa się funktorem lub literą funkcyjną. ∎

Wyrażenie otrzymane przez kwantyfikację wpz po pewnej zmiennej jest także wpz. Jeżeli zmienna w wpz została skwantyfikowana, to nazywa się ją *zmienną związaną*, a w przeciwnym wypadku *zmienną swobodną*. Wpz w których nie ma zmiennych swobodnych nie są już predykatami, a reprezentują pewne zdania twierdzące. Takie wyrażenia nazywają się *zdaniami*. W dalszej części tego rozdziału będziemy zajmowali się tylko takimi wyrażeniami.

Przykład 4.2

Rozpatrzmy kilka wpz będących zdaniami.

Wyrażenie

$$pod(kot, stol)$$

może wyrażać fakt że „Kot znajduje się pod stołem".

Zdanie „Wszystkie słonie są szare" może być reprezentowane przez[2]

$$(\forall X)\,[słoń(X) \rightarrow kolor(X, szary)]$$

gdzie wyrażenie kwantyfikowane jest implikacją, a X jest zmienną kwantyfikowaną.

Mówimy, że wyrażenie zostało skwantyfikowane po X. Zakresem działania kwantyfikatora jest część następującego po nim wyrażenia wskazywana przez nawias.

Zdanie: „Istnieje osoba, która napisała ‚Pan Cogito' ", może być reprezentowane przez

$$(\exists X) napisal(X, „Pan\ Cogito")$$

Wyrażenie

$$\neg(\exists X)\,[mrówka(X) \wedge inteligentny(X)]$$

może reprezentować zdanie: „Nie ma inteligentnych mrówek".

[2]A nie $(\forall X)\,[słoń(X) \wedge kolor(X, szary)]$

$(\forall X)(\exists Y)kocha(X,Y)$ – może wyrażać zdanie (fakt): „Każdy kogoś kocha”

Wyrażenia:

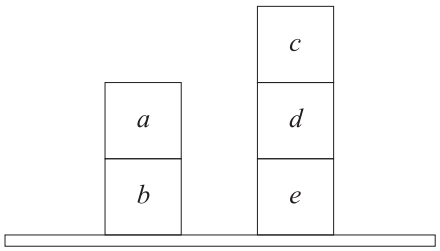
$$(\forall X \forall Y)dziecko(X,Y) \rightarrow potomek(X,Y)$$
$$(\forall X \forall Y \forall Z)dziecko(Z,Y) \wedge potomek(X,Z) \rightarrow potomek(X,Y)$$

przedstawiają zdanie (regułę): „Dziecko (jakiejś osoby) jest (jej) potomkiem i potomek dziecka (jakiejś osoby) jest potomkiem (tej osoby)” co jest rekurencyjną definicją bycia potomkiem.

Podobnie, przedstawiona na rysunku 4.3 scena ze „świata bloków” może być opisana następującymi wyrażeniami:

$$na(a,b)$$
$$na(c,d)$$
$$na(d,e)$$

$$(\forall X \forall Y)(na(X,Y) \rightarrow ponad(X,Y))$$
$$(\forall X \forall Y \forall Z)(na(X,Y) \wedge ponad(Y,Z) \rightarrow ponad(X,Z))$$
$$(\forall X)(\neg(\exists Y)na(Y,X) \rightarrow wolny(X))$$



Rys. 4.3: Świat bloków

W rachunku predykatów często wykorzystuje się poniższe formuły, które są zawsze prawdziwe

$$\neg(\exists X)P(X) \equiv (\forall X)[\neg P(X)]$$
$$\neg(\forall X)P(X) \equiv (\exists X)[\neg P(X)]$$

W rachunku predykatów stosuje się terminologie:

atom	– predykat
literał	– atom lub atom zanegowany
term	– argumenty atomu (predykatu)

4.1.2. Interpretacja

Wpz posiadają znaczenie tylko wtedy gdy wchodzącym w nią symbolom nadana zostanie jakakolwiek *interpretacja*.

Definicja 4.1 *Przez interpretację rozumiemy dowolny system składający się z niepustego zbioru D nazywającego się dziedziną interpretacji i zależności przyporządkującej każdemu predykatowi P^n, n-wymiarową relację w D, każdej literze funkcyjnej f^n, n-wymiarową funkcję w D i każdej stałej a_j element z D.*

Każde wpz bez zmiennych swobodnych przedstawia sobą pewne zdanie twierdzące prawdziwe lub fałszywe. Każde wpz ze zmiennymi swobodnymi przedstawia pewną relację (jest predykatem) w dziedzinie interpretacji, która może być spełniona lub nie w zależności od wyboru elementów z D.

Interpretacja definiuje semantykę języka predykatów.

Przykład 4.3

Rozpatrzmy poniżej podane wpz:

$$p(X, Y) \tag{4.1}$$

$$(\forall Y)p(X, Y) \tag{4.2}$$

$$(\exists X \forall Y)p(X, Y) \tag{4.3}$$

Uwaga: w (4.3) zwrócić uwagę na kolejność: istnieje, że dla każdego.

Przy interpretacji:

D – zbiór liczb całkowitych dodatnich
$p(X, Y)$ wyraża relację $X \leq Y$

poszczególne wpz przedstawiają:

$p(X, Y)$ – relację: $X \leq Y$,
$(\forall Y)p(X, Y)$ – relację: X jest mniejsze lub równe od każdej liczby całkowitej dodatniej (co jest spełnione dla $X = 1$),
$(\exists X \forall Y)p(X, Y)$ – zdanie (twierdzenie): istnieje liczba, która jest mniejsza od każdej liczby całkowitej dodatniej (istnieje najmniejsza liczba całkowita dodatnia). Twierdzenie to jest prawdziwe.

Zauważmy, że przy dziedzinie interpretacji D będącej zbiorem liczb całkowitych twierdzenie (4.3) jest nieprawdziwe. ∎

4.1.3. Wnioskowanie

Wnioskowaniem nazywamy proces stosujący *reguły wnioskowania* dla uzyskania wpz z zadanego zbioru wpz. Wpz otrzymane drogą wnioskowania nazywa się *wnioskiem*, a ciąg zastosowanych *reguł wnioskowania* tworzy dowód.

Wyrażenie

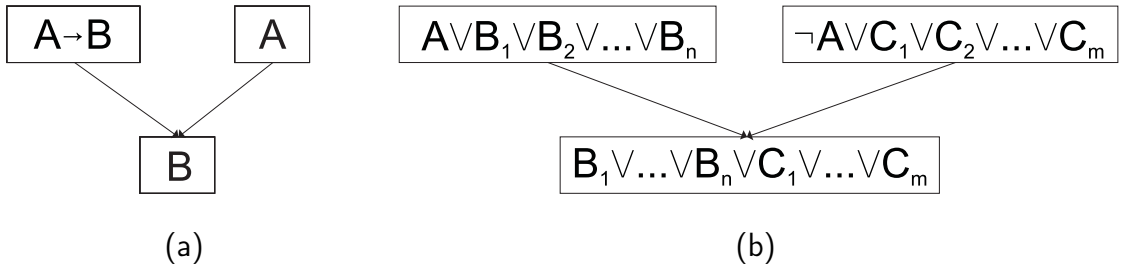
$$S \vdash W$$

oznacza, że wpz W jest wnioskiem ze zbioru wpz $S = \{S_1, S_2, \ldots, S_n\}$.

Reguły wnioskowania

Wnioskowanie jest na ogół procesem wielokrokowym i w każdym kroku stosowana jest odpowiednio dobrana *reguła wnioskowania*. Reguła wnioskowania składa się ze zbioru wyrażeń nazywanych *warunkami* i zbioru wyrażeń nazywanych *konkluzjami*. Gdy mamy zbiór wyrażeń odpowiadający warunkom reguły uznajemy, że wnioskiem są wyrażenia odpowiadające jej konkluzjom. W ten sposób stosując reguły wnioskowania, z wpz i zbiorów wpz otrzymujemy nowe wpz, aż do uzyskania wymaganego wyrażenia.

Jedną z ważnych i często stosowanych reguł wnioskowania jest *reguła odrywania (modus ponens)*. Mówi ona, że ze zdania (reguły) *jeżeli A to B*, i zdania (faktu) A wynika konkluzja (fakt) B (patrz rys. 4.4a).



Rys. 4.4: Reguły wnioskowania: (a) modus ponens, (b) rezolucja

Inną regułą wnioskowania jest *uniwersalna specjalizacja* (*reguła podstawiania*), która wytwarza wpz $W(a)$ z wpz $(\forall X)W(X)$.

Regułą wnioskowania, ne której oparte są metody automatycznego dowodzenia jest *reguła rezolucji*. Mówi ona, że ze zdania *A lub B* i zdania *nie A lub C* wynika zdanie *B lub C* (rys. 4.4b). Reguła rezolucji zawiera w sobie, jako szczególne przypadki, inne reguły wnioskowania. Usuwając np. w regule rezolucji wyrażenie B otrzymujemy regułę odrywania.

Klauzule

Regułę odrywania i rezolucję można stosować do pewnej klasy wpz zwanych *klauzulami*. Klauzula jest to wpz w postaci sumy logicznej literałów

$$L_1 \vee L_2 \vee \ldots \vee L_n$$

gdzie L_i, $i = 1, 2, \ldots, n$ są literałami. Na przykład:

$$P(X) \vee \neg P(Y) \vee Q(X, Y, Z)$$

Klauzule, które zawierają tylko jeden nie zanegowany atom nazywają się *klauzulami Horna*

$$A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n$$

gdzie A_i, $i = 1, 2, \ldots, n$ są atomami.

Zauważmy, że klauzule Horna można zapisać w postaci

$$A_n \wedge A_{n-1} \wedge \ldots \wedge A_2 \rightarrow A_1$$

Dowolne wpz może być przekształcone do postaci zbioru klauzul co będzie pokazane w dalszej części skryptu.

Strategie sterowania wnioskowaniem

Przy stosowaniu reguły odrywania możliwe są dwie podstawowe strategie sterowania wnioskowaniem: *w przód* (ang. forward chaining) i *w tył* (ang. backward chaining). Strategie te można stosować, gdy wyrażenia bazy wiedzy mają postać klauzul Horna. Gdy wyrażenia bazy wiedzy nie dadzą się sprowadzić do tej postaci, to dowodzenie może być przeprowadzone przy użyciu rezolucji, co jest omówione dalej.

Przykład 4.4

Przykłady przebiegu wnioskowania w tył i w przód przedstawiono na rysunkach 4.5 i 4.6. ■

Wnioskowanie w tył. Należy wykazać A.	
Baza wiedzy	Wnioskowanie
(1) $A \leftarrow B \wedge C$	1 Aby wykazać A należy wykazać B i C
(2) $A \leftarrow C \wedge D$	2 Aby wykazać B należy wykazać F i E
(3) $B \leftarrow F \wedge E$	3 Aby wykazać F należy wykazać R
(4) $C \leftarrow R \wedge S$	4 R jest faktem, więc R jest wykazane
(5) $F \leftarrow R$	5 R jest wykazane, więc F jest wykazane
(6) D	6 E jest faktem, więc E jest wykazane
(7) E	7 B jest wykazane
(8) R	8 Aby wykazać C należy wykazać R i S
(9) S	9 R jest faktem, więc R jest wykazane
	10 S jest faktem, więc S jest wykazane
	11 C jest wykazane
	12 A jest wykazane

Rys. 4.5: Przykład wnioskowania w tył (przykład 4.4)

4.1.4. Wynikanie logiczne

Dany jest zbiór wpz:

$$S_1, S_2, \ldots, S_n$$

Jeżeli dana interpretacja powoduje, że każdy wpz w tym zbiorze jest prawdziwy (ma wartość T), wtedy mówimy, że interpretacja *spełnia* zbiór wpz (jest jego *modelem*).

Wpz W *wynika logicznie* ze zbioru wpz (jest jego *konsekwencją*) S jeżeli każda interpretacja spełniająca S także spełnia W, co oznaczamy $S \models W$.

Istnieje ważny związek pomiędzy koncepcją wyrażenia wynikającego logicznie ze zbioru wpz i koncepcją wyrażenia wyprowadzonego ze zbioru wpz przy pomocy reguł wnioskowania:

Wnioskowanie w przód. Należy wykazać A.	
Baza wiedzy	Wnioskowanie
(1) $A \leftarrow B \wedge C$	(10) (8)(5) F
(2) $A \leftarrow C \wedge D$	(11) (3)(7)(10) B
(3) $B \leftarrow F \wedge E$	(12) (4)(8)(9) C
(4) $C \leftarrow R \wedge S$	(13) (1)(11)(12) A
(5) $F \leftarrow R$	
(6) D	
(7) E	
(8) R	
(9) S	

Rys. 4.6: Przykład wnioskowania w przód (przykład 4.4)

Twierdzenie 4.1 *Jeżeli S jest zbiorem wpz, zaś W jest wpz to: $S \models W$ wtedy i tylko wtedy, gdy $S \vdash W$*

4.1.5. Dowodzenie przy pomocy rezolucji

Dany jest zbiór wpz S. Należy wykazać, że wpz W jest wnioskiem ze zbioru S. Systemy oparte o rezolucję tworzą dowód przez wykazanie sprzeczności. W systemach tych neguje się W i dołącza $\neg W$ do zbioru S. Jeżeli W wynika logicznie z S, to zbiór S, $\neg W$ nie jest spełniony przy żadnej interpretacji.

Aby wykazać, że zbiór wpz nie jest spełniony przy żadnej interpretacji wystarczy pokazać, że jest sprzeczny, to znaczy że można z niego wyprowadzić zarówno pewne wpz P jak i wpz $\neg P$. Wyrażenia P i $\neg P$ generują klauzulę pusta *NIL* oznaczającą sprzeczność.

Aby wykazać sprzeczność zbioru wpz przy pomocy rezolucji, należy przekształcić go do postaci klauzul, a następnie stosować wielokrotnie regułe rezolucji aż do uzyskania klauzuli pustej *NIL*.

Przykład 4.5

Korzystając z rezolucji przedstawimy tu dowód w dziedzinie rachunku zdań.

Dany jest zbiór formuł rachunku zdań

1. $p \rightarrow \neg q$
2. $q \vee p \wedge \neg r$

Należy udowodnić, że:

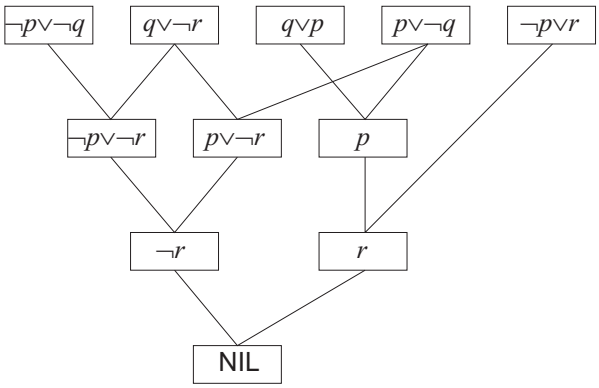
3. $\neg p \wedge q \vee p \wedge \neg r$

Wyrażenia 1, 2 oraz zanegowaną tezę 3 przekształcamy do postaci klauzul. Mamy np.: 2. $q \vee p \wedge \neg r \equiv (q \vee \neg r)(q \vee p)$ czemu odpowiadają dwie klauzule: $q \vee p \wedge \neg r$ i

$(q \vee \neg r)(q \vee p)$.

$$
\begin{array}{ll}
1. & \neg p \vee \neg q \\
2a. & q \vee \neg r \\
2b. & q \vee p \\
3a'. & p \vee \neg q \\
3b'. & \neg p \vee r
\end{array}
$$

Teraz stosując rezolucję do wybranych par klauzul staramy się otrzymać dwa wyrażenia przeciwne. Ich wyprowadzenie kończy dowód. Przebieg dowodu przedstawiono na rys. 4.7. ∎



Rys. 4.7: Przebieg dowodu do przykładu 4.5

Aby dowód przeprowadzać dla wyrażeń zawierających kwantyfikatory należy zastosować algorytm przekształcenia wyrażeń predykatowych do postaci klauzul a następnie uogólnioną regułę rezolucji w zastosowaniu do klauzul, co omówione jest poniżej.

Sprowadzenie wpz do postaci klauzul

Każde wpz można przekształcić do postaci klauzul. Kolejne kroki takiego przekształcania pokazane są na poniższym przykładzie (Nilsson 1980, str. 146).

Przykład 4.6

Przestawić w postaci klauzul wyrażenie:

$$(\forall X)\,[p(X) \rightarrow [(\forall Y)\,[p(Y) \rightarrow p(f(X,Y))] \wedge \neg(\forall Y)\,[q(X,Y) \rightarrow p(Y)]]]$$

1. Eliminacja symboli implikacji (korzystamy z wzoru $A \rightarrow B \equiv \neg A \vee B$):

$$(\forall X)\,[\neg p(X) \vee [(\forall Y)\,[\neg p(Y) \vee p(f(X,Y))] \wedge \neg(\forall Y)\,[\neg q(X,Y) \vee p(Y)]]]$$

2. Redukcja zakresu działania symbolu negacji:

Korzystamy tu z wzorów:

$$\begin{aligned}
\neg(A \wedge B) &\equiv \neg A \vee \neg B \\
\neg(A \vee B) &\equiv \neg A \wedge \neg B \\
\neg\neg A &\equiv A \\
\neg(\forall X)A(X) &\equiv (\exists X)\neg A(X) \\
\neg(\exists X)A(X) &\equiv (\forall X)\neg A(X)
\end{aligned}$$

$$(\forall X)\left[\neg p(X) \vee \left[(\forall Y)\left[\neg p(Y) \vee p(f(X,Y))\right] \wedge (\exists Y)\left[q(X,Y) \wedge \neg p(Y)\right]\right]\right]$$

3. Standaryzacja zmiennych:

   Każdy kwantyfikator ma swoją zmienną o innej nazwie;
   np. $(\forall X)[p(X) \rightarrow (\exists X)q(X)] \equiv (\forall X)[p(X) \rightarrow (\exists Y)q(Y)]$.

   $$(\forall X)\left[\neg p(X) \vee \left[(\forall Y)\left[\neg p(Y) \vee p(f(X,Y))\right] \wedge (\exists W)\left[q(X,W) \wedge \neg p(W)\right]\right]\right]$$

4. Eliminacja kwantyfikatorów szczegółowych:

   Korzystamy tu z wzorów:

   $$\begin{aligned}
   (\exists X)p(X) &\equiv p(a) \\
   (\exists X \forall Y)p(X,Y) &\equiv (\forall Y)p(a,Y) \\
   (\forall Y \exists X)p(X,Y) &\equiv p(g(Y),Y)
   \end{aligned}$$

   gdzie $g(\cdot)$ – funkcja Skolema. Na przykład

   $$[(\forall W)q(W)] \rightarrow (\forall X \forall Y \exists Z)[p(X,Y,Z) \rightarrow (\forall U)r(X,Y,U,Z)]$$

   jest równoważne

   $$[(\forall W)q(W)] \rightarrow (\forall X \forall Y)[p(X,Y,g(X,Y)) \rightarrow (\forall U)r(X,Y,U,g(X,Y))]$$

   W omawianym głównym przykładzie mamy:

   $$(\forall X)\left[\neg p(X) \vee \left[(\forall Y)\left[\neg p(Y) \vee p(f(X,Y))\right] \wedge \left[q(X,g(X)) \wedge \neg p(g(X))\right]\right]\right]$$

5. Przesunięcie kwantyfikatorów uniwersalnych na początek wyrażenia:

   $$(\forall X \forall Y)\left[\neg p(X) \vee \left[\neg p(Y) \vee p(f(X,Y))\right] \wedge \left[q(X,g(X)) \wedge \neg p(g(X))\right]\right]$$

6. Przedstawienie wyrażenia w normalnej postaci iloczynowej:

   Korzystamy tu z wzoru: $A \vee B \wedge C \equiv (A \vee B) \wedge (A \vee C)$.

   W naszym przykładzie postępując wg. schematu $a \vee (b \vee c) \wedge d \wedge e \equiv (a \vee b \vee c)(a \vee d \wedge e)$ mamy:

   $$(\forall X \forall Y)\left[\left[\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))\right] \wedge \left[\neg p(X) \vee q(X,g(X)) \wedge \neg p(g(X))\right]\right]$$

   i następnie postępując wg. schematu $(a \vee d \wedge e) \equiv (a \vee d) \wedge (a \vee e)$

   $$\begin{aligned}
   (\forall X \forall Y)\left[\left[\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))\right] \wedge \left[\neg p(X) \vee q(X,g(X))\right]\right. \\
   \left. \wedge \left[\neg p(X) \vee \neg p(g(X))\right]\right]
   \end{aligned}$$

7. Eliminacja kwantyfikatorów uniwersalnych:

   Wszystkie zmienne w wpz są kwantyfikowane uniwersalnie. Ponieważ kolejność kwantyfikatorów uniwersalnych nie ma znaczenia, to nie piszemy ich jawnie. Mamy

   $$[\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))] \wedge [\neg p(X) \vee q(X, g(X))] \wedge [\neg p(X) \vee \neg p(g(X))]$$

8. Eliminacja symboli koniunkcji:

   Wypisujemy wyrażenia rozdzielone symbolem koniunkcji – otrzymujemy wyrażenie początkowe doprowadzone do postaci klauzul.

   $$\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))$$
   $$\neg p(X) \vee q(X, g(X))$$
   $$\neg p(X) \vee \neg p(g(X))$$

■

Unifikacja

Unifikacja jest procesem sprawdzania czy dwa wyrażenia mogą stać się identyczne po zastosowaniu odpowiednich podstawień za swoje zmienne – jest to bardzo ważny proces pozwalający dokonywać rezolucji dwóch klauzul.

Przykład 4.7

Rozpatrzmy proces unifikacji dwóch literałów

$$P(a, X, f(g(Y))), \qquad P(Z, f(Z), f(U))$$

Przeglądając wyrażenia od strony lewej do prawej poszukując termów lub części termów, które są różne i znajdując podstawienia S zrównujące je, kolejno otrzymujemy

$$S = \{a/Z\}$$

$$P(a, X, f(g(Y))), \qquad P(a, f(a), f(U))$$
$$S = \{a/Z, f(a)/X\}$$
$$P(a, f(a), f(g(Y))), \qquad P(a, f(a), f(U))$$
$$S = \{a/Z, f(a)/X, g(Y)/U\}$$
$$P(a, f(a), f(g(Y))), \qquad P(a, f(a), f(g(Y)))$$

■

Przykład 4.8

$$P(f(X, g(a, Y)), g(a, Y)), \qquad P(f(X, Z), Z)$$
$$S = \{g(a, Y)/Z\}$$

$$P(f(X, g(a, Y)), g(a, Y)), \qquad P(f(X, g(a, Y)), g(a, Y))$$

Przykład 4.9

Przykład nie unifikowalności

$$P(X, f(X, f(X, X))), \qquad P(f(Y, Y), Y)$$

$$S = \{f(Y, Y)/X\}$$

$$P(f(Y, Y), f(f(Y, Y), f(f(Y, Y), f(Y, Y)))), \qquad P(f(Y, Y), Y)$$

W następnym podstawieniu pojawi się $S = \{\ldots,$ funkcja zmiennej $Y/Y\}$ co prowadzi do nieskończonej rekurencji i w konsekwencji powoduje, że wyrażenia są nie unifikowalne.

Rezolucja dwóch klauzul

Dane są dwie klauzule

$$L = \bigvee_i l_i, \qquad M = \bigvee_i m_i$$

gdzie l_i i m_i są literałami. Niech

$$l_j \subseteq \{l_i\}, \qquad m_k \subseteq \{m_i\}$$

Jeżeli istnieje unifikator S dla l_j i $\neg m_k$, to klauzule L i M mają rezolwentę

$$\left\{\bigvee_{i \neq j} l_i\right\}_S \vee \left\{\bigvee_{i \neq k} m_i\right\}_S$$

Przykład 4.10

Stosując podstawienie

$$S = \{X/Z, a/X\}$$

dwie klauzule

$$P(X, f(a)) \vee P(X, f(Y)) \vee Q(Y), \qquad \neg P(Z, f(Z)) \vee R(Z)$$

zostają przekształcone do postaci

$$P(a, f(a)) \vee P(a, f(Y)) \vee Q(Y), \qquad \neg P(a, f(a)) \vee R(a)$$

i ich rezolwenta wynosi

$$P(a, f(Y)) \vee Q(Y) \vee R(a),$$

Przykład dowodu metodą rezolucji

Przykład 4.11

Dany jest zbiór formuł rachunku predykatów (zbiór wpz)

1. $(\forall X)[pk(X) \rightarrow p(X)]$
2. $(\forall X)[d(X) \rightarrow \neg p(X)]$
3. $(\exists X)[d(X) \wedge i(X)]$

Należy udowodnić, że:

4. $(\exists X)[i(X) \wedge \neg pk(X)]$

Wyrażenia 1, 2, 3 wraz z zanegowaną tezą 4 po przekształceniu do postaci klauzul:

1. $\neg pk(X) \vee p(X)$
2. $\neg d(Y) \vee \neg p(Y)$

$3a$. $d(a)$

$3b$. $i(a)$

$4'$. $\neg i(Z) \vee pk(Z)$

Teraz stosując rezolucję do wybranych par klauzul otrzymujemy kolejno:

5. $pk(a)$ rezolwenta 3b i 4'
6. $p(a)$ rezolwenta 5 i 1
7. $\neg d(a)$ rezolwenta 6 i 2
8. *NIL* rezolwenta 7 i 3a.

Powyższe etapy dowodzenia zilustrowane są na rys 4.8.

Zauważmy, że np. nadając predykatom $pk(X)$, $p(X)$, $d(X)$, $i(X)$ interpretację: programujący komputery, programista, delfin i inteligentny, udowodniliśmy powyżej, że ze zdań:

programujący komputery jest programistą
delfiny nie są programistami
niektóre delfiny są inteligentne

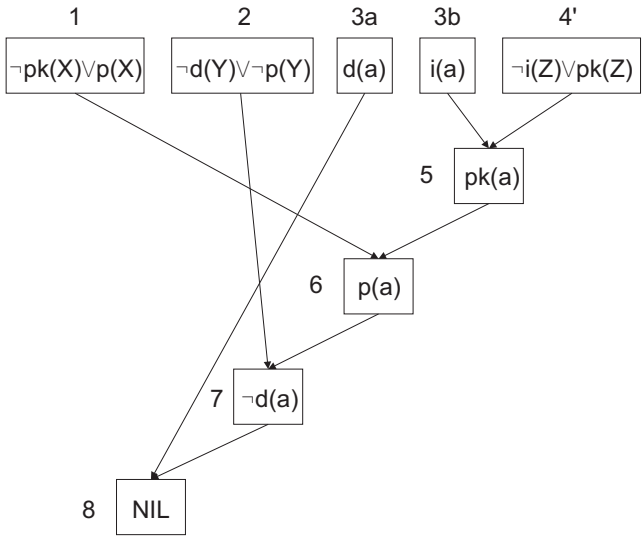
wynika zdanie:

niektórzy nie programujący komputerów są inteligentni.

■

4.1.6. Język PROLOG a wnioskowanie w rachunku predykatów

Na poniższym przykładzie pokażemy, że język PROLOG (patrz rozdział 7) jest systemem dowodzenia twierdzeń rachunku predykatów w oparciu o rezolucję.



Rys. 4.8: Przykład wnioskowania w oparciu o rezolucję

Przykład 4.12

Stosując regułę rezolucji należy pokazać, że wpz $(\exists Z)A(Z)$ wynika z podanego poniżej zbióru klauzul (2)–(7). Ponieważ $\neg(\exists Z)A(Z) \equiv (\forall Z)\neg A(Z)$ zadanie można sprowadzić do wykazania że zbiór klauzul (1)–(7) jest sprzeczny.

$$
\begin{array}{ll}
(1) & \neg A(Z) \\
(2) & A(X) \vee \neg P(X) \vee \neg Q(X,Y) \\
(3) & A(V) \vee \neg R(W) \vee \neg Q(W,V) \\
(4) & P(a) \\
(5) & Q(b,c) \\
(6) & R(a) \\
(7) & R(b)
\end{array}
$$

Stosując regułę rezolucji do wybranych par klauzul otrzymujemy:

$$
\begin{array}{ll}
(1)(2) & \neg P(X) \vee \neg Q(X,Y) \\
(1)(2)(4) & \neg Q(a,Y)
\end{array}
$$

Klauzula $\neg Q(a,Y)$ nie unifikuje się z $Q(b,c)$ i nie ma innych możliwych unifikacji, więc cofnięcie się do klauzuli (1)(2) i próba znalezienia innej unifikacji literału $\neg P(X)$. Ponieważ nie ma innych unifikacji $\neg P(X)$, więc cofnięcie się klauzuli (1) i próba wyznaczenia innej jej rezolwenty. Otrzymujemy

$$
\begin{array}{ll}
(1)(3) & \neg R(W) \vee \neg Q(W,Z) \\
(1)(3)(6) & \neg Q(a,Z)
\end{array}
$$

Klauzula $\neg Q(a,Z)$ nie unifikuje się z $Q(b,c)$ i nie ma innych możliwych unifikacji, więc cofnięcie się do klauzuli (1)(3) i wyznaczenie innej rezolwenty poprzez próby

innej unifikacji $\neg R(W)$. Tym razem otrzymujemy

$$\begin{array}{ll} (1)(3)(7) & \neg Q(b, Z) \\ (1)(3)(7)(5) & NIL, \text{ dla Z=c} \end{array}$$

W ten sposób wykazaliśmy, że wpz $(\exists Z)A(Z)$ wynika z klauzul (2)–(7) i zachodzi to dla $Z = c$.

Biorąc pod uwagę, że:

- Klauzule (1)–(7) są klauzulami Horna.
- Klauzule Horna można przedstawić w poniższej postaci

$$P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n \equiv P_1 \leftarrow P_2 \wedge \ldots \wedge P_n$$

- Przyjęto ustaloną strategię kolejności prób wyznaczania rezolwent polegającą na poszukiwaniu atomów unifikujących się z pierwszym literałem (w kolejności występowania w klauzuli) ostatnio otrzymanej klauzuli, a w przypadku niepowodzenia cofanie się do poprzedniej klauzuli i próba ponownego znalezienia unifikacji jej pierwszego literału.

widać, że klauzule (2)–(7) przedstawione w przykładzie są równoważne klauzulom programu w PROLOG'u oraz strategia wnioskowania jest identyczna z realizowaną w PROLOG'u. Wynika stąd, że wykonywanie programu w PROLOG'u jest równoważne wnioskowaniu w rachunku predykatów przy zastosowaniu reguły rezolucji i strategii wnioskowania w tył. ∎

4.1.7. Przykład zastosowania logiki pierwszego rzędu – świat smoka

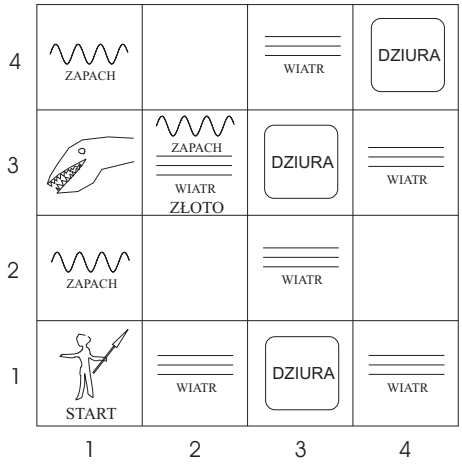
Często podawanym przykładem zastosowania metod logicznych jest wnioskowanie w tzw. *świecie smoka* (ang. wumpus world).

Opis świata smoka

System (jaskinia) składa sie z siatki kwadratów ograniczonych ścianami. W kwadracie może znajdować się agent (posłaniec) lub obiekty, którymi mogą być *smok*, *dziura* lub *złoto*. W stanie początkowym agent zajmuje kwadrat o współrzędnych (1,1). Celem agenta jest odnalezienie złota, powrót do kwadratu (1,1) i wyjście z systemu. Przykład systemu przedstawiono na rysunku 4.9.

Podczas działania systemu agent odbiera następujące sygnały, podejmuje następujące akcje i ma następujące cele:

- W kwadracie zawierającym smoka i w kwadratach przylegających bokami agent czuje zapach.
- W kwadratach przylegających bokami do dziury agent czuje powiew wiatru.
- W kwadracie zawierającym złoto agent zauważa blask.



Rys. 4.9: Przykład świata smoka

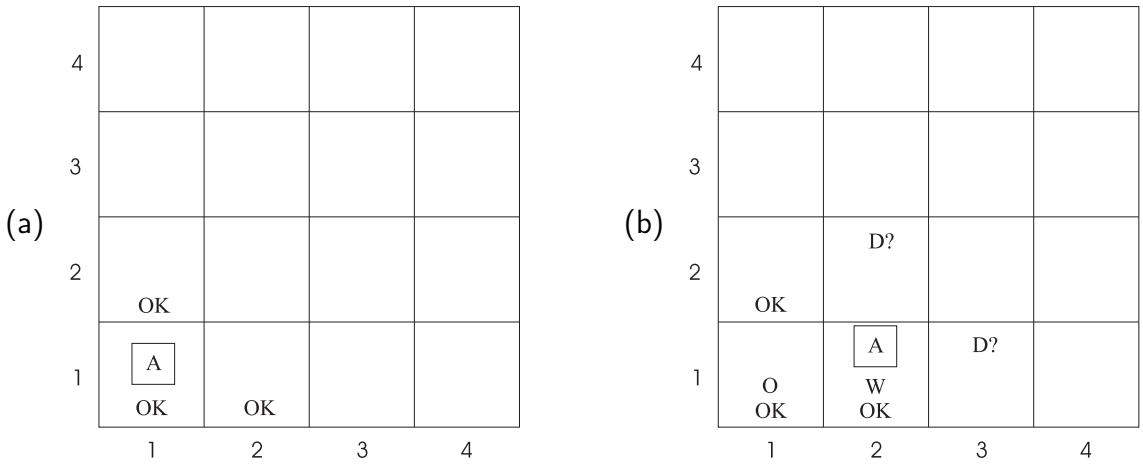
- Gdy agent usiłuje przejść przez ściane ograniczającą system, to czuje uderzenie.
- Gdy smok zostaje zabity, to wydaje przenikliwy dźwięk, który jest słyszalny w każdym kwadracie systemu.
- Agent odbiera docierające do niego sygnały w postaci list pięciu symboli. Na przykład, jeżeli jest zapach i wiatr ale nie ma blasku, uderzenia i przenikliwego dźwięku, agent otrzyma listę: [*Zapach*, *Wiatr*, *Nic*, *Nic*, *Nic*]. Agent nie postrzega swojego położenia.
- Agent może podjąć następujące akcje: przesunięcie się o jeden kwadrat do przodu, obrót w prawo o 90^0, obrót w lewo o 90^0, chwycenie obiektu znajdującego się w tym samym kwadracie, w którym znajduje się agent. Ponadto agent może wystrzelić strzałę, która leci wzdłuż lini prostej i albo trafi i zabije smoka, albo uderzy w ścianę. Agent ma tylko jedną strzałe, więc tylko jedna akcja strzału może mieć jakiś skutek. Wreszcie agent może podjąć akcje wyjścia z jaskini; co może być zrealizowane, gdy agent znajduje się w kwadracie $(1, 1)$.
- Agent ginie jeżeli wejdzie do kratki zawierającej dziurę lub żywego smoka.
- Celem agenta jest znalezienie złota i wyniesienie go z jaskini tak szybko jak to jest możliwe. W jednorazowej próbie, za wyniesienie złota z jaskini przyznane zostaje 1000 punktów, jedna podjęta akcja kosztuje 1 punkt zaś utrata życia przez agenta kosztuje 10000 punktów.

Na rysunkach 4.10 i 4.11 przedstawiono pozycję agenta w kolejnych krokach oraz otrzymywane przez niego sygnały i wyprowadzone wnioski.

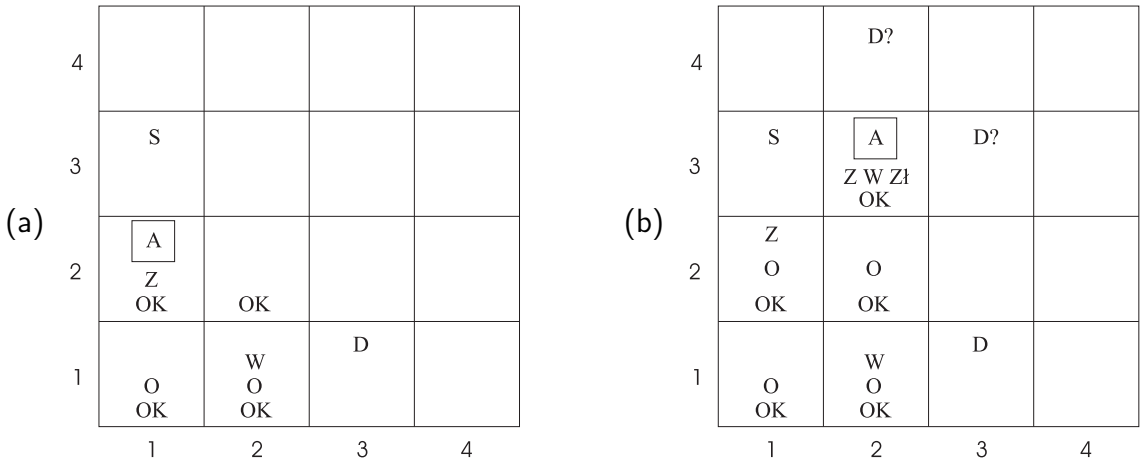
Świat smoka – wnioskowanie w rachunku zdań

Wiedza agenta o świecie smoka po trzech ruchach agenta, wyrażona w formalizmie rachunku zdań:

$$\begin{array}{rr} \neg Z_{1,1} & \neg W_{1,1} \\ \neg Z_{2,1} & W_{2,1} \\ Z_{1,2} & \neg W_{1,2} \end{array}$$



Rys. 4.10: Pierwszy krok agenta w świecie smoka. Zastosowano oznaczenia: A – agent, B – blask, D – dziura, O – kwadrat odwiedzony, OK – kwadrat bezpieczny, S – smok, W – wiatr, Z – Zapach: (a) stan początkowy i wnioski agenta po percepcji [*Nic, Nic, Nic, Nic, Nic*], (b) po pierwszym kroku i po percepcji [*Nic, Wiatr, Nic, Nic, Nic*]

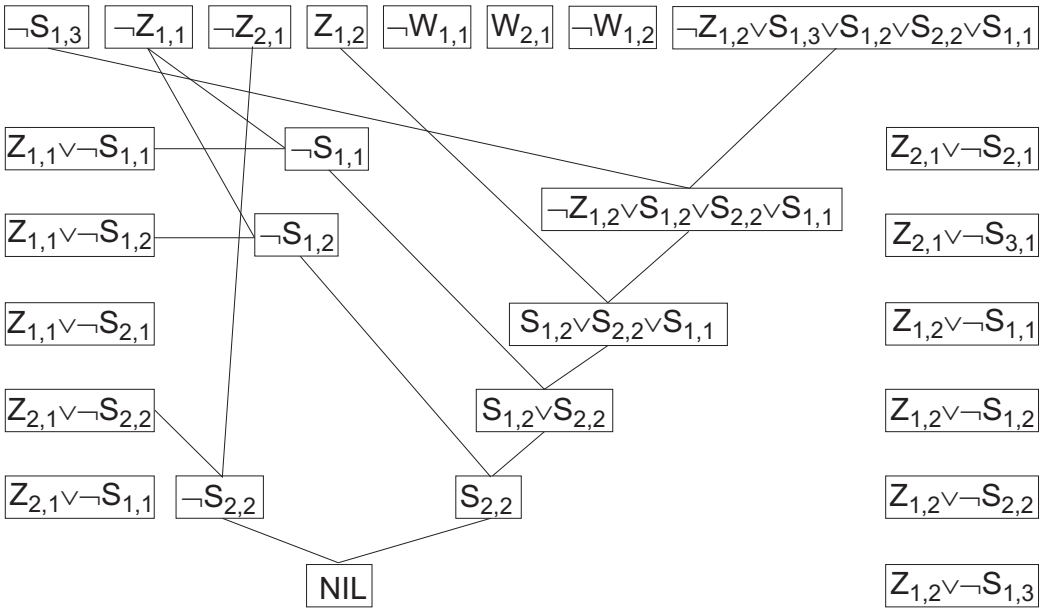


Rys. 4.11: Dalsze kroki agenta w świecie smoka: (a) stan po trzecim kroku i percepcji [*Zapach, Nic, Nic, Nic, Nic*], (b) po piątym kroku i percepcji [*Zapach, Wiatr, Blask, Nic, Nic*]

$$
\begin{array}{lllllllll}
\neg Z_{1,1} & \rightarrow & \neg S_{1,1} & \wedge & \neg S_{1,2} & \wedge & \neg S_{2,1} & & \\
\neg Z_{2,1} & \rightarrow & \neg S_{1,1} & \wedge & \neg S_{2,1} & \wedge & \neg S_{2,2} & \wedge & \neg S_{3,1} \\
\neg Z_{1,2} & \rightarrow & \neg S_{1,1} & \wedge & \neg S_{1,2} & \wedge & \neg S_{2,2} & \wedge & \neg S_{1,3}
\end{array}
$$

$$Z_{1,2} \rightarrow S_{1,3} \vee S_{1,2} \vee S_{2,2} \vee S_{1,1}$$

Na rysunku 4.12 przedstawiono przebieg dowodu, że smok znajduje się w kwadracie (3,1).



Rys. 4.12: Świat smoka. Rachunek zdań – przebieg dowodu, że smok znajduje się w kwadracie (3,1). Zanegowaną tezą jest $\neg S_{1,3}$

Świat smoka – wnioskowanie w rachunku predykatów

Opis formalny świata smoka można utworzyć na podstawie zdania w języku naturalnym:
„Zapach występuje w kwadracie wtedy i tylko wtedy, gdy przynajmniej w jednym sąsiednim kwadracie znajduje się smok“ Na podstawie tego zdania mamy

$$(\forall X, Y)[ws(X, Y) \wedge z(X, Y)] \equiv (\exists X_s, Y_s)[ws(X_s, Y_s) \wedge ks(X, Y, X_s, Y_s) \wedge s(X_s, Y_s)]$$

gdzie predykaty ws, z, ks i s określają odpowiednio znajdowanie sie w systemie, zapach, kwadrat sąsiedni i obecność smoka.

Nie sprawdzając znajdowania sie współrzędnych wewnątrz systemu, oraz zapisując sąsiedztwo w jawny sposób i nie piszac jawnie kwantyfikatorów można na podstawie powyższego wyrażenia napisać dwie równoważne reguły:

$$\neg z(X, Y) \rightarrow \neg s(X, Y) \wedge \neg s(X - 1, Y) \wedge \neg s(X + 1, Y) \wedge \neg s(X, Y - 1) \wedge \neg s(X, Y + 1)$$

$$z(X, Y) \rightarrow s(X, Y) \vee s(X - 1, Y) \vee s(X + 1, Y) \vee s(X, Y - 1) \vee s(X, Y + 1)$$

Bezpośrednie zapisanie powyższych wyrażeń w PROLOGu nie jest możliwe ponieważ nie są one klauzulami Horna (nie dają się do nich sprowadzić). Aby uzyskać program w PROLOGu należy obok predykatów z i s oznaczających zapach i smoka wprowadzić predykaty nz i ns oznaczające niewystępowanie zapachu i smoka. Ponadto należy pamiętać o występującym w PROLOGu założeniu o *zamkniętości świata* czyli generowaniem przez PROLOG odpowiedzi $FALSE$ w przypadku niemożności wykazania czegoś. Np. przy zapytaniu $s(4, 4)$ odpowiedź $FALSE$ może oznaczać brak smoka w kwadracie (4,4) ale także brak danych do wykazania występowania smoka w tym kwadracie. Czyli aby uzyskać wiarygodną odpowiedź należy zapytać dwukrotnie: o istnienie smoka i o jego nieistnienie.

Struktury agentów

Rozróżnia się trzy podstawowe struktury agentów:

- agent refleksowy (bezinercyjny) (ang. reflex) – działa bezpośrednio, wyłącznie na podstawie percepcji,
- z modelem świata (ang. model-based) – ma wbudowany model świata i przed podjęciem decyzji przeprowadza wnioskowanie korzystając z tego modelu.
- ukierunkowany na osiągnięcie celu (ang. goal-based) – formułuje cele i usiłuje je osiągnąć; zwykle wykorzystuje także model świata.

Działania refleksowe mają szereg niedogodności. W przypadku świata smoka agent nie może korzystać z wcześniej zaobserwowanych faktów, ponieważ ich nie pamięta. Może wpaść w nieskończone pętle, np. ponownie dochodząc do miejsca, w ktorym już był.

Wewnętrzny model świata może być przechowywany poprzez pamiętanie wszystkich dotychczasowych percepcji, lub co jest równoważne, a znacznie bardziej efektywne, poprzez pamiętanie stanu. Przykładem obu sposobów przechowywnia modelu jest zadanie poszukiwania klucza przez człowieka: może on go znaleźć na podstawie pamiętania stanu otoczenia lub poprzez przeglądanie zapisów pamięci czynności dokonywanych wcześniej.

Zmienność świata w czasie (chociażby poprzez zmienność położenia agenta) powoduje konieczność aktualizacji modelu świata, a także pamiętania niektórych poprzednich jego parametrów aby móc wnioskować o trendach.

4.1.8. Własności logik

Monotoniczność

Logikę nazywamy monotoniczną, jeżeli wnioski wynikające z bazy wiedzy B_1 wynikają także z bazy $B_1 \cup B_2$, czyli

$$\text{jeżeli } B_1 \models \alpha, \text{ to } (B_1 \cup B_2) \models \alpha$$

Logiki wyższego rzędu

Logiki wyższego rzędu pozwalają dodatkowo na kwantyfikacje po relacjach (predykatach) jak i funkcjach.

Np. w logice wyższego rzędu można napisać wyrażenie

$$(\forall XY)(X = Y) \equiv ((\forall p)p(X) \equiv p(Y))$$

oznaczające, że wszystkie obiekty są równe wtedy i tylko wtedy, gdy wszystkie ich własności są równe, lub wyrażenie

$$(\forall f, g)(f = g) \equiv ((\forall X)f(X) = g(X))$$

oznaczające, że dwie funkcje są równe wtedy i tylko wtedy, gdy mają takie same wartości dla wszystkich argumentów.

Zagadnienie pełności

Mówimy, że procedura wnioskowania jest *pełna* wtedy i tylko wtedy gdy dowolne wyrażenie wynikające logicznie z dowolnej bazy danych moę być także z niej wyprowadzone przy użyciu tej procedury. Algorytm dowodu wykorzystujący regułę modus ponens jest pełny w przypadku wyrażeń zapisanych w postaci klauzul Horna (to znaczy w postaci $\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee q$ co jest równoważne wyrażeniu $p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow q$).

Przykład 4.13

Stosując wyłącznie regułę modus ponens wykazać $s(a)$ na podstawie następujących wyrażeń:

$$(\forall X)p(X) \rightarrow q(X)$$
$$(\forall X)\neg p(X) \rightarrow r(X)$$
$$(\forall X)q(X) \rightarrow s(X)$$
$$(\forall X)r(X) \rightarrow s(X)$$

Jak łatwo się przekonać, stosując wyłącznie regułę modus ponens dowodu przeprowadzić się nie daje (drugie wyrażenie nie jest klauzulą Horna), a z drugiej strony stosując regułę rezolucji dowód uzyskuje się bardzo łatwo. Czyli procedura dowodu oparta wyłącznie o regułę wnioskowania modus ponens nie jest pełna, gdy wyrażenia nie są w postaci klauzul Horna. ∎

Zagadnienie istnienia pełnej procedury dowodu jest bardzo ważne. Jeżeli pełna procedura dowodzenia twierdzeń matematycznych by istniała i była znana to: (a) wszystkie hipotezy mogłyby być wykazane mechanicznie, (b) cała matematyka mogłaby być wygenerowana z zespołu podstawowych aksjomatów. Pytania o pełność doprowadziły do powstania największych prac matematycznych XX wieku.

Kurt Gödel w 1930 roku podał twierdzenie o pełności w logice pierwszego rzędu: każde wyrażenie wynikające ze zbioru wpz BW może być z niego wyprowadzone przez pewną procedurę R [3], czyli

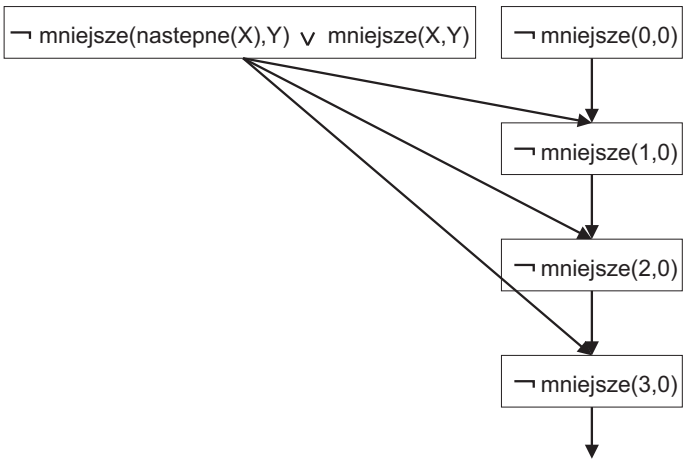
$$\text{jeżeli } BW \models \alpha \text{ to } BW \vdash_R \alpha$$

Nie jest to trywialne ponieważ występowanie kwantyfikatorów i zagnieżdżonych funkcji prowadzi do baz o nieskończonych rozmiarach.

Gödel nie podał algorytmu R, stwierdził tylko, że istnieje. J.A. Robinson (1965) podał algorytm dowodu oparty o rezolucję i wykazał jego pełność.

Logika pierwszego rzędu jest *częściowo rozstrzygalna*, tzn. że jeżeli jakieś wyrażenie wynika z danej bazy wiedzy, to w skończonym czasie można to wykazać, natomiast jeżeli nie wynika, to procedura dowodzenia może nigdy się nie zakończyć. Z częściowej rozstrzygalności logiki pierwszego rzędu wynika, że wykazanie sprzeczności bazy wiedzy jest też częściowo rozstrzygalne.

[3] Poprzez niewielkie rozszerzenie języka pierwszego rzędu o schemat indukcji Gödel sformułował i udowodnił słynne twierdzenie o niepełności: istnieją zdania arytmetyczne prawdziwe, których nie da się wyprowadzić z przyjętych dla arytmetyki aksjomatów. Twierdzenie Gödla wraz z odkryciem fizyki kwantowej, zasadą nieoznaczoności Heisenberga, problemem stopu w maszynie Turinga oraz odkryciem procesów chaotycznych spowodowało odejście od deterministycznego i przewidywalnego świata Newtona dominującego w fizyce XVIII i XIX wieku.



Rys. 4.13: Przykład niekończącego się wnioskowania

4.2. Obiekt atrybut wartość

Oprócz języka logiki pierwszego rzedu, stosowane są reprezentacje o mniejszej sile ekspresji ale bardziej efektywne obliczeniowo. Najstarszym sposobem reprezentacji są tróki obiekt-atrybut-wartość (ang. Object Attribute Value triples (O A V)).

Na przykład reguła systemu MYCIN: „If the site of the culture is blood, and the morphology of the organism is rod, and the Gram stain of the organism is Gram-neg, and the patient is a compromised host, then there is suggestive evidence (0.6) that the identity of the organism is Pseudomonas aeuroginosas" składa się z trójek (O-A-V) przedstawionych na rys. 4.14

	Obiekt	Atrybut	Wartość
If	Culture	Site	Blood
	Organism	Morphology	Rod
	Organism	Gram stain	Gram-neg
	Patient	Compromised host	True
Then	Organism	Identity	Pseudomonas aeuroginosas

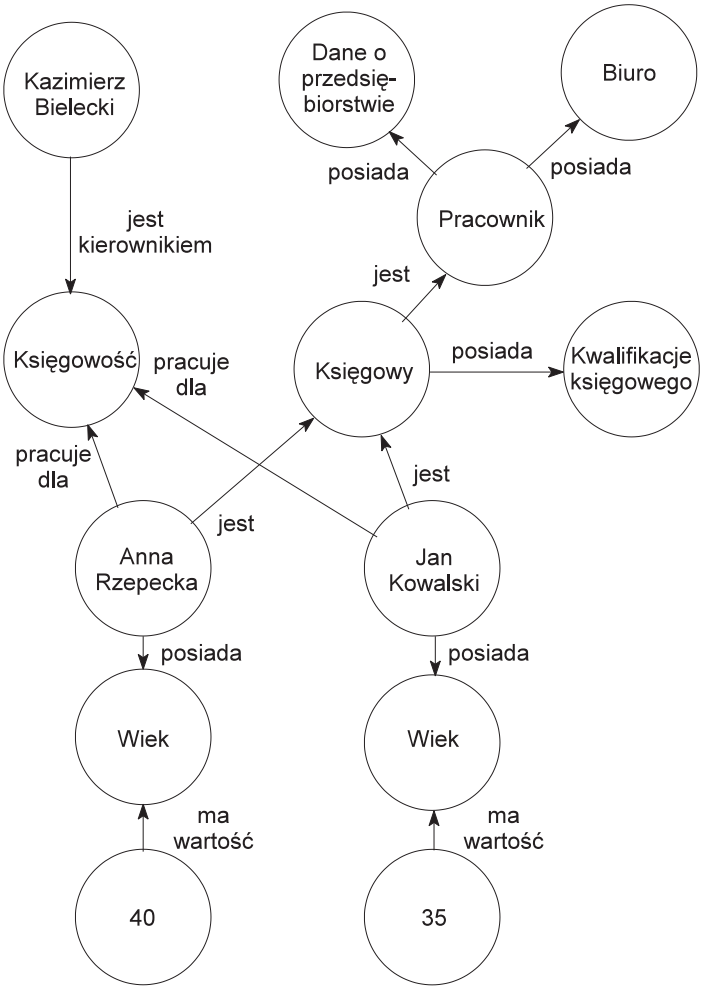
Rys. 4.14: Trójki O-A-V wybranej reguły systemu MYCIN

4.3. Sieci semantyczne

Sieci semantyczne są bardziej elastycznym sposobem reprezentacji wiedzy niż trójki (O A V). Składają się z węzłów i krawędzi z etykietami.

Węzły mogą oznaczać następujące kategorie (patrz przykład sieci semantycznej przedstawiony na rys. 4.15):

1. Obiekty. W następującym przykładzie Anna Rzepecka, Jan Kowalski i Kazimierz Bielecki są obiektami fizycznymi. Obiekty abstrakcyjne także są umieszczane w węzłach. 40 jest obiektem abstrakcyjnym.



Rys. 4.15: Przykład sieci semantycznej

2. Pojęcia ogólne takie jak „pracownik" lub „księgowy".
3. Atrybuty. Mogą one być dwóch typów: wymagające podania wartości i nie potrzebujące wartości. „Wiek" wymaga podania wartości, a „kwalifikacje księgowego" są atrybutem „księgowego" i nie wymaga podania wartości.

Krawędzie są jednokierunkowymi połączeniami pomiędzy węzłami. Można je podzielić na następujące klasy:

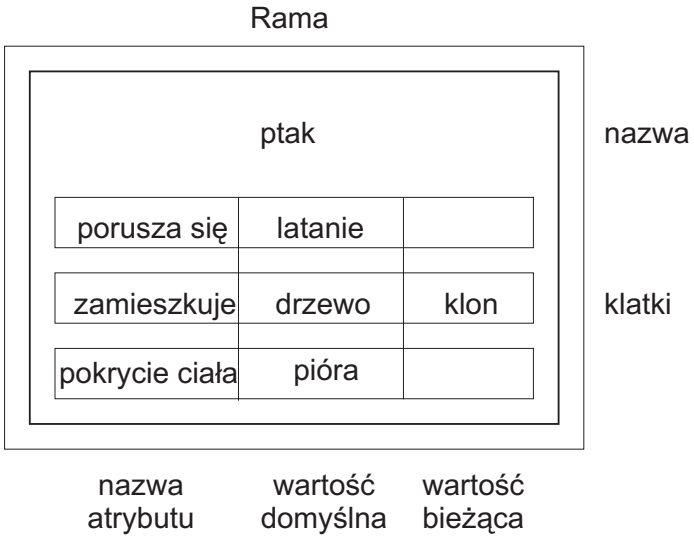
1. Połączenie „posiada" przyporządkowuje węzły z atrybutem do węzłów z pojęciami ogólnymi lub z obiektami. Połączenia te stwierdzają, że pojęcie ogólne posiada atrybut, np. że pracownik posiada biuro lub księgowy posiada kwalifikacje księgowego lub że obiekt ma atrybut, np. Anna Rzepecka posiada wiek 40.
2. Połączenie „jest" reprezentuje relację pomiędzy węzłami reprezentującymi obiekty i węzłami reprezentującymi pojęcia ogólne lub pomiędzy różnymi węzłami reprezentującymi pojęcia ogólne.
3. Inne połączenia. W przykładzie, połączenie pomiędzy Anną Rzepecką i działem księgowości reprezentuje związek pomiędzy dwoma obiektami. Tym związkiem

jest „pracuje dla". Częste zastosowanie ma połączenie „jest częścią" (np. cylinder jest częścią silnika).

4.4. Ramy

Ramy są wygodnym sposobem reprezentacji wiedzy. Stosują podstawowe zasady reprezentacji (O-A-V) ale oferują bardziej efektywną reprezentację. Uzasadnieniem użycia ram było stwierdzenie, że ekspert człowiek przechowuje swoją wiedzę w postaci różnych powiązanych koncepcji (ram).

Rama jest prostą strukturą. Ma nazwę, poprzez którą następuje do niej odwołanie i zbiór klatek (ang. slots) – patrz rys. 4.16.



Rys. 4.16: Przykład ramy

Wartości domyślne są zwykle związane z ogólnymi własnościami klasy lub pojęcia – pozostają niezmienne podczas trwania ramy. Wartości domyślne są używane tylko wtedy, gdy nie ma wartości bieżących dla danej klatki. Wartości bieżące dotyczą zwykle specyficznych obiektów – mogą być zmieniane w czasie działania systemu. Gdy wartość bieżąca istnieje, to jej wartość jest wartością klatki. Rama może być uważana za tablicę dynamiczną mającą trzy kolumny i dowolną liczbę wierszy.

Ramy mogą być łączone w struktury obrazujące ich wzajemne relacje. Podstawową strukturą jest połączenie typu rama rodzicielska – rama potomna. Ramę potomną można uważać za pewną specjalizację ramy rodzicielskiej i odwrotnie rama rodzicielska może być uważana za pewną generalizację ramy potomnej. Zbiór takich połączeń pomiędzy ramami tworzy ich hierarchię, w której bardziej ogólne koncepcje pojawiają się na górze zaś bardziej specyficzne na dole struktury.

Prostą i jedyną korzyścią łączenia ram w hierarchie, jest to, że informacja może być rozłożona niekoniecznie będąc duplikowaną

Rama potomna może dziedziczyć wartości (zarówno domyślne jak i bieżące) od ram rodzicielskich, które z kolei mogą je dziedziczyć od swoich ram rodzicielskich, itd. W ten sposób informacja przepływa z góry na dół struktury.

4.5. Logika opisowa

Trudności w prowadzeniu wnioskowania przy reprezentacji wiedzy metodami sieci semantycznych i ram doprowadziły do powstania metody nazywanej logiką opisową. Logika opisowa zachowuje wygodę tworzenia opisu właściwą ramom przy jednocześnie dużej efektywności wnioskowania.

Rozdział 5

Wnioskowanie w warunkach niepewności

Przypuśćmy, że projektujemy system ekspertowy przeprowadzający analizę działania sklepów. Moglibyśmy napisać regułę:

$$(\forall X)\mathit{symptom}(X, \mathit{małe\_zyski}) \rightarrow \mathit{diagnoza}(X, \mathit{mały\_kapitał})$$

Ale reguła ta jest błędna ponieważ nie zawsze małe zyski są wynikiem zbyt małego zainwestowanego kapitału. Czasami małe zyski są wynikiem złej lokalizacji sklepu lub złego doboru towarów. Możemy więc napisać

$$\begin{aligned}(\forall X)\mathit{symptom}(X, \mathit{małe\_zyski}) \quad \rightarrow \quad & \mathit{diagnoza}(X, \mathit{zły\_dobór\_towarów}) \vee \\ & \mathit{diagnoza}(X, \mathit{mały\_kapitał}) \vee \\ & \mathit{diagnoza}(X, \mathit{zła\_lokalizacja})\end{aligned}$$

Ale może być jeszcze wiele innych przyczyn małych zysków jak np. niewłaściwy personel, czy niewłaściwe godziny otwarcia; praktycznie przyczyn może być bardzo dużo. Odwrócenie kierunku implikacji:

$$(\forall X)\mathit{diagnoza}(X, \mathit{mały\_kapitał}) \rightarrow \mathit{symptom}(X, \mathit{małe\_zyski})$$

też jest błędne ponieważ za mały kapitał niekoniecznie może prowadzić do małych zysków. Jedynym rozwiązaniem jest wypisanie wszystkich możliwych diagnoz prowadzących do symptomu małych zysków.

W większości zadań o dużym znaczeniu praktycznym pojawiają się podobne trudności. W szczególności w problemach diagnostyki medycznej i technicznej opisanie rzeczywistości w języku rachunku predykatów jest często bardzo trudne lub niemożliwe do wykonania.

Aby rozwiązywać zadania wnioskowania w sytuacjach niedeterministycznych początkowo postępowano nieformalnie np. przyporządkowując wyrażeniom predykatowym liczby wskazujące wiarygodność wyrażeń, a następnie w procesie wnioskowania takie liczby nadawano kolejnym wnioskom jako iloczyn wiarygodności wyrażeń, z których wnioski te wynikały (system ekspertowy MYCIN). Później w latach 60. zaproponowano wnioskowanie rozmyte, a od połowy lat 90. nastąpił powrót do metod probabilistycznych (sieci bayesowskie).

5.1. Wnioskowanie rozmyte

Logika rozmyta jest rozszerzeniem logiki bulowskiej włączającym pojęcie prawdy częściowej – znajdującej się pomiędzy prawdą a fałszem. Zaproponowana została w latach 60. przez Lotfi Zadeha z UC/Berkeley jako narzędzie do modelowania niokreśloności występujących w języku naturalnym.

5.1.1. Zbiory rozmyte

W klasycznej teorii zbiorów, podzbiór U zbioru X może być zdefiniowany jako odwzorowanie elementów zbioru X w elementy zbioru $\{0, 1\}$,

$$U : X \to \{0, 1\}$$

Odwzorowanie to może być reprezentowane jako zbiór uporządkowanych par. Pierwszym elementem uporządkowanej pary jest element zbioru X, a elementem drugim jest element należący do zbioru $\{0, 1\}$, gdzie 0 reprezentuje nienależenie do podzbioru U, a 1 reprezentuje należenie. Prawdziwość lub fałszywość zdania

$$x \in U$$

można określić przez odszukanie pary uporządkowanej, której pierwszym elementem jest x. Zdanie jest prawdziwe jeżeli drugim elementem pary jest 1 lub jest fałszywe jeżeli drugim elementem jest 0.

Podobnie rozmyty podzbiór F zbioru X może być zdefiniowany jako odwzorowanie elementów zbioru X w elementy zbioru $[0, 1]$,

$$F : X \to [0, 1]$$

reprezentowane przez zbiór uporządkowanych par, gdzie każda para zawiera pierwszy element należący do X, a drugi element należący do przedziału $[0, 1]$ przy czym każdemu elementowi X odpowiada jedna para uporządkowana. To definiuje odwzorowanie pomiędzy elementami zbioru X i wartościami z przedziału $[0, 1]$. Wartość 0 reprezentuje całkowite nie należenie do zbioru, wartość 1 – całkowite należenie, a wartości pośrednie – reprezentują częściową przynależność. Zbiór X jest nazywany *obszarem rozważań* lub *przestrzenią* (ang. universe of discourse). Odwzorowanie nazywa się *funkcją przynależności* (ang. membership function).

Reasumując, niech X oznacza pewien stały zbiór. Zbiór rozmyty F można określić w X jako zbiór uporządkowanych par:

$$F = \{(x, \mu_F(x)) \mid x \in X\}$$

gdzie $\mu_F(x)$ jest funkcją przynależności zbioru F.

$\mu_F(x) = 1$ oznacza pełną przynależność x do F,

$\mu_F(x) = 0$ oznacza brak przynależności x do F,

wartości pośrednie $\mu_F(x)$ oznaczają częściową przynależność x do F.

Stopień prawdziwości stwierdzenia

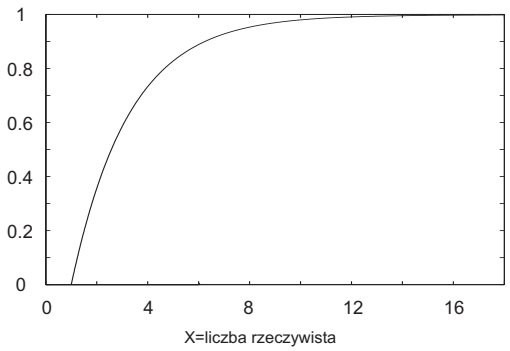
$$x \in F$$

można określić znajdując uporządkowaną parę, której pierwszym elementem jest x. Stopniem prawdziwości stwierdzenia jest drugi element uporządkowanej pary. W praktyce, określenia funkcja przynależności i podzbiór rozmyty używane są wymiennie.

Wiele określeń występujących w języku naturalnym wygodnie jest reprezentować w postaci zbiorów rozmytych. Ilustrują to poniższe przykłady.

Przykład 5.1

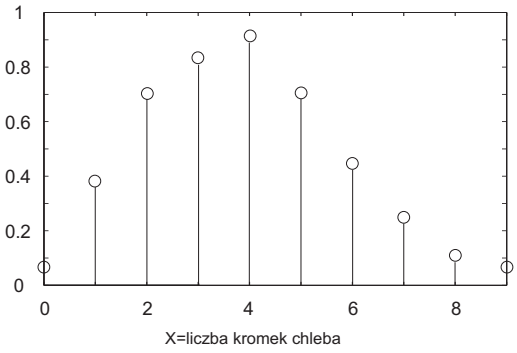
Zbiór „liczb rzeczywistych dużo większych od 1" może być opisany przez funkcję przynależności przedstawioną na rysunku 5.1. ■



Rys. 5.1: Funkcja przynależności zbioru rozmytego: „liczb rzeczywistych dużo większych od 1"

Przykład 5.2

Zbiór „liczby kromek chleba zjadanych w czasie typowego śniadania" może być zdefiniowany funkcją przynależności przedstawioną na rysunku 5.2. ■

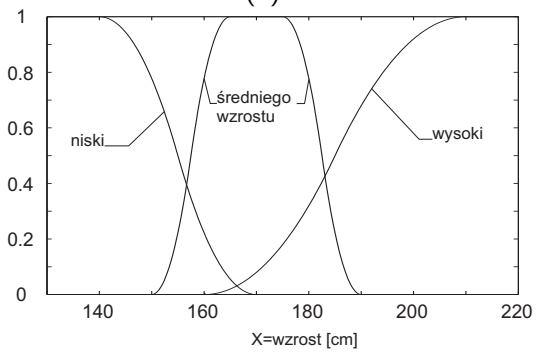


Rys. 5.2: Funkcja przynależności zbioru rozmytego: „liczba kromek chleba zjadanych w czasie typowego śniadania"

Przykład 5.3

Ludzie i ich wzrost. Niech przestrzeń X będzie zbiorem liczb rzeczywistych w przedziale $[130, 220]$ reprezentującym wzrost osób wyrażony w cm. Zdefiniujmy trzy zbiory rozmyte: *niski, średniego wzrostu, wysoki*, które będą odpowiadały odpowiednim pojęciom w języku naturalnym. Na rysunku 5.3 przedstawiono przykładowe funkcje przynależności definiujące te zbiory rozmyte oraz wartości funkcji przynależności zbioru *wysoki* dla kilku przykładowych osób.

(a)



(b)

osoba	wzrost [cm]	$\mu_{\text{wysoki}}(x)$
Jan	145	0.00
Tomasz	185	0.50
Andrzej	195	0.82
Mariusz	210	1.00

Rys. 5.3: Funkcje przynależności trzech zbiorów rozmytych dotyczących wzrostu (a) oraz wartości funkcji przynależności zbioru *wysoki* dla kilku przykladowo wybranych osób (b)

W terminologii zbiorów rozmytych przestrzeń nazywa się *zmienną lingwistyczną*, a zdefiniowane na niej podzbiory nazywa się *wartościami lingwistycznymi*. W omawianym przykładzie zmienną lingwistyczną jest „wzrost", a zbiorem możliwych wartości tej zmiennej jest zbiór

$$T = \{\textit{niski, średniego wzrostu, wysoki, }\}$$

Funkcje przynależności używane w zastosowaniach zwykle mają kształt trójkąta, trapezu, są funkcjami Gaussa, asymetrycznymi funkcjami Gaussa, funkcjami sigmoidalnymi, funkcjami *typu s*, funkcjami *typu z*, funkcjami *typu* π lub funkcjami dzwonowymi (rysunek 5.4).

Funkcja Gaussa opisana jest wzorem

$$g(x, \sigma, c) = \exp\left(\frac{-(x-c)^2}{2\sigma^2}\right)$$

Asymetryczna funkcja Gaussa składa się z dwóch funkcji Gaussa o oddzielnie zadanych parametrach σ i c. Funkcja sigmoidalna dana jest wzorem

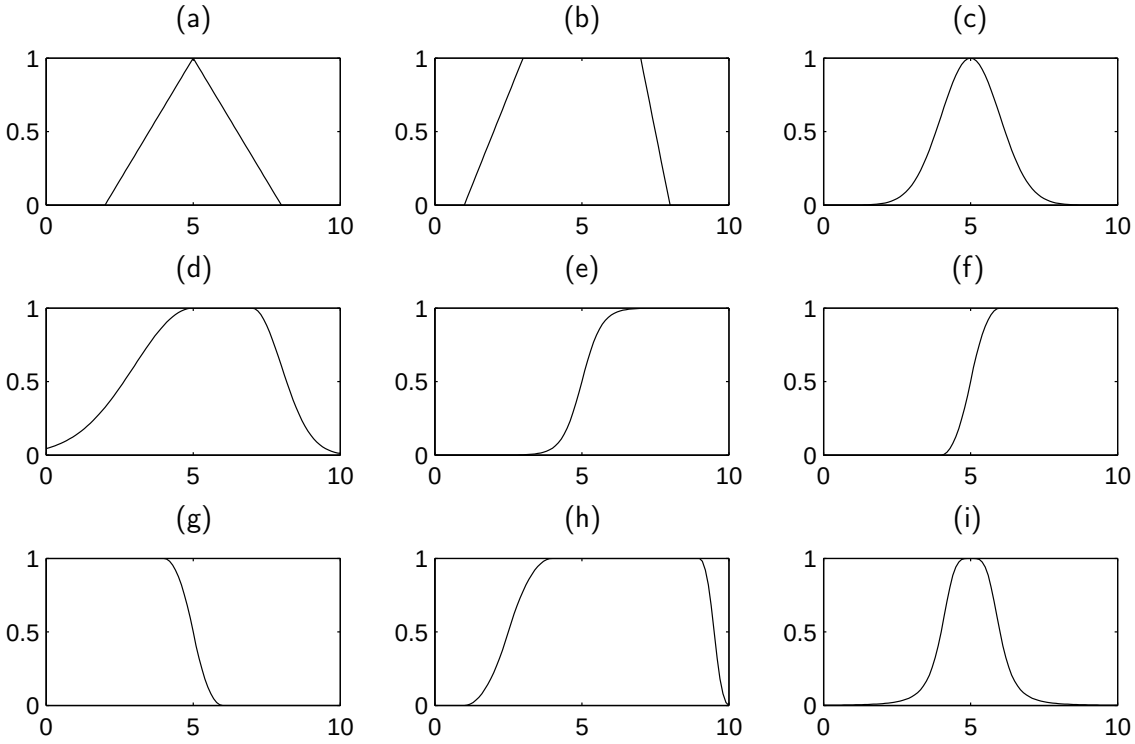
$$si(x, a, c) = \frac{1}{1 + \exp(-a(x-c))}$$

Jeden z wariantów funkcji typu s (ang. s–shaped, s-curve) jest opisany wzorem

$$s(x, a, b) = \begin{cases} 0 & , x < a \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-a}{b-a}\pi) & , a \leq x \leq b \\ 1 & , x > b \end{cases}$$

Funkcja typu z jest lustrzanym odbiciem funkcji typu s. Funkcja typy π jest złożeniem funkcji s i z. Uogólniona funkcja dzwonowa dana jest wzorem

$$f(x, a, b, c) = \frac{1}{1 + \left| \frac{x - c}{a} \right|^{2b}}$$



Rys. 5.4: Typowe funkcje przynależności: (a) trójkąt, (b) trapez, (c) funkcja Gaussa, (d) asymetryczna funkcja Gaussa, (e) funkcja sigmoidalna, (f) funkcja typu s, (g) funkcja typu z, (h) funkcja typu π, (i) uogólniona funkcja dzwonowa

Funkcje przynależności mogą być wieloargumentowe. Np. przynależność do podzbioru rozmytego *wysoki na swój wiek* jest zależna od dwóch argumentów: wzrostu i wieku. Funkcja przynależności takiego zbioru może mieć kształt jak na rys. 5.5. Innym przykładem może być funkcja przynależności do zbioru zdefiniowanego jako „liczba x dużo większa od liczby y".

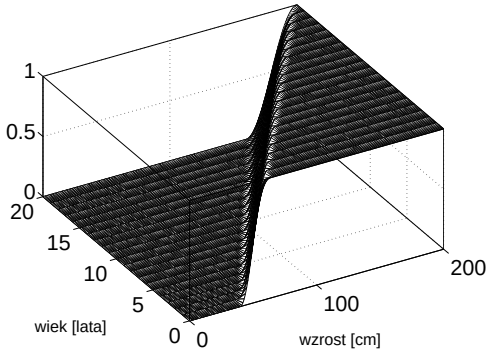
5.1.2. Operacje na zbiorach rozmytych

Operacje logiczne w dziedzinie zbiorów rozmytych zdefiniowane są następująco:

Definicja 5.1 *Zawieranie zbiorów*

Zbiór rozmyty A zawiera się w zbiorze rozmytym B wtedy i tylko wtedy, gdy

$$\forall x \in X : \quad \mu_A(x) \leq \mu_B(x)$$



Rys. 5.5: Funkcja przynależności zbioru rozmytego: *wysoki na swój wiek*

Definicja 5.2 *Równość zbiorów*

Zbiór rozmyty A jest równy zbiorowi rozmytemu B wtedy i tylko wtedy, gdy

$$\forall x \in X: \quad \mu_A(x) = \mu_B(x)$$

Definicja 5.3 *Iloczyn zbiorów*

Iloczynem zbiorów rozmytych A i B jest zbiór rozmyty $A \cap B$ o funkcji przynależności

$$\forall x \in X: \quad \mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

Definicja 5.4 *Suma zbiorów*

Sumą zbiorów rozmytych A i B jest zbiór rozmyty $A \cup B$ o funkcji przynależności

$$\forall x \in X: \quad \mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

Definicja 5.5 *Negacja (dopełnienie) zbioru*

Negacją lub dopełnieniem zbioru rozmytego A jest zbiór rozmyty $\overline{A}$ o funkcji przynależności

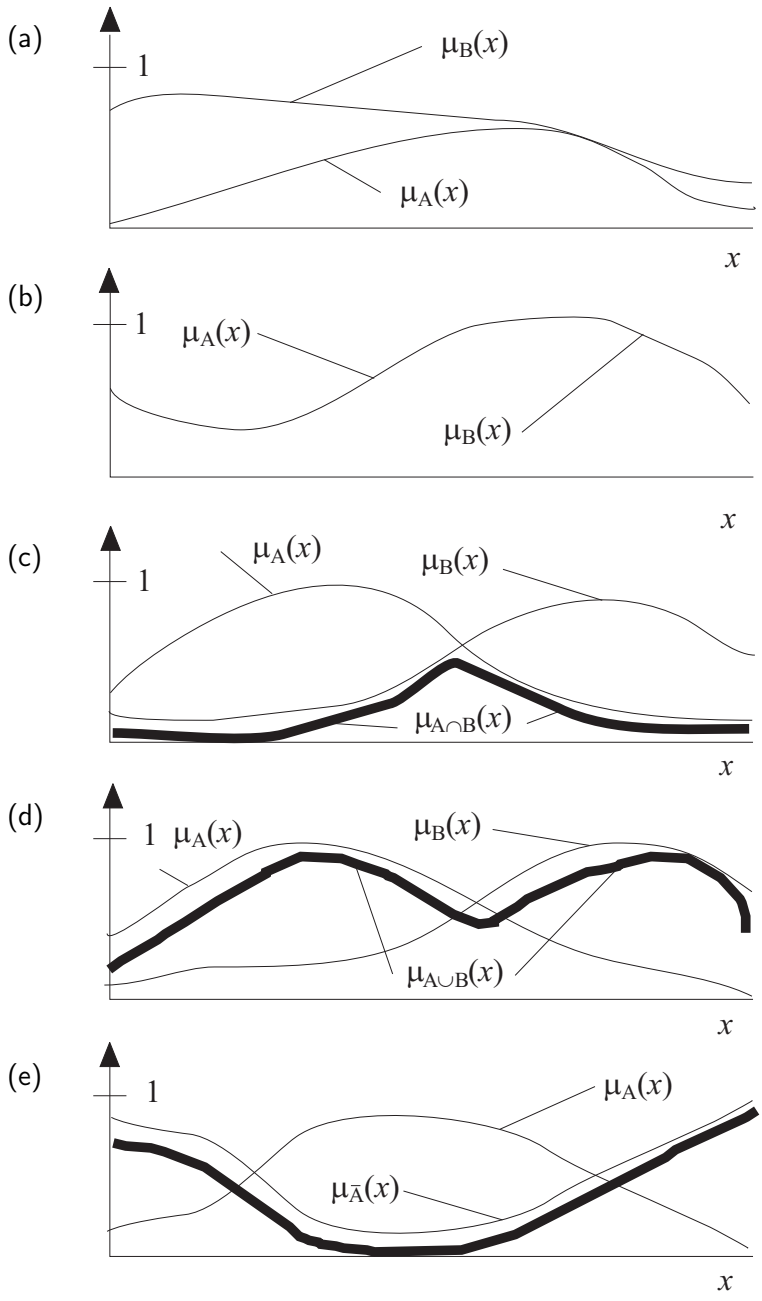
$$\forall x \in X: \quad \mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

Na rysunku 5.6 zilustrowano powyższe definicje.

Czasami tworzone są inne definicje operacji AND i OR ale definicja operacji NOT pozostaje bez zmian. Zauważmy, że gdy funkcje przynależności przyjmują tylko wartości 0 i 1 powyższe operacje pokrywają się ze znanymi operacjami na zbiorach nierozmytych. Jest to potwierdzeniem, że zbiory i logika rozmyta stanowią uogólnienie teorii zbiorów i logiki bulowskiej.

Operacje iloczynu i sumy mogą być zastosowane także do dwóch zbiorów rozmytych zdefiniowanych w róznych przestrzeniach. Wtedy z dwóch zbiorów rozmytych otrzymujemy jeden zbiór dwuargumentowy. Jeżeli przestrzeniami są X i Y wtedy zbiór dwuargumentowy jest określony na iloczynie kartezjańskim $X \times Y$.

Definicja 5.6 *Iloczyn (produkt) kartezjański zbiorów*



Rys. 5.6: Operacje na zbiorach rozmytych: (a) Zbiór rozmyty A zawiera się w zbiorze rozmytym B, (b) Zbiór A jest równy zbiorowi B, (c) iloczyn zbiorów A i B, (d) suma zbiorów A i B, (e) negacja (dopełnienie zbioru) A

Iloczyn (produkt) kartezjański dwóch zbiorów rozmytych A i B zdefiniowanych w przestrzeniach X i Y jest zbiorem rozmytym w przestrzeni iloczynu kartezjańskiego $X \times Y$ i posiada funkcję przynależności

$$\mu_{A\times B}(x, y) = \min(\mu_A(x), \mu_B(y))$$

Definicja 5.7 *Suma (koprodukt) kartezjański zbiorów*

Suma (koprodukt) kartezjański dwóch zbiorów rozmytych A i B zdefiniowanych w przestrzeniach X i Y jest zbiorem rozmytym w przestrzeni iloczynu kartezjańskiego

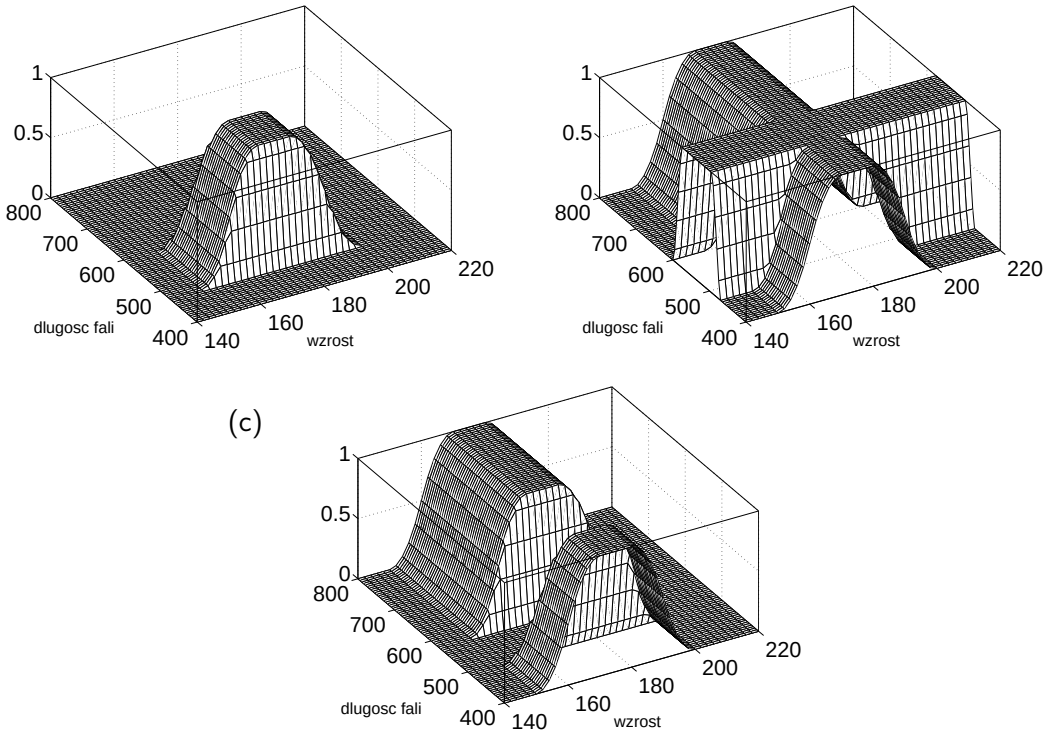
$X \times Y$ i posiada funkcję przynależności

$$\mu_{A+B}(x, y) = \max(\mu_A(x), \mu_B(y))$$

Przykład 5.4

Przyjmijmy definicję podzioru *średniego wzrostu* jak w przykładzie 5.3 i ponadto przyjmijmy, że mamy podzbiór rozmyty *zielonooki* określony trapezową funkcją przynależności, gdzie przestrzenią podzbioru *średniego wzrostu* jest wzrost w cm, a przestrzenią podzbioru *zielonooki* jest długość fali światła w nm (kolor zielony – 530nm).

Na tej podstawie dla przykładowych podzbiorów: *średniego wzrostu i zielonooki*, *średniego wzrostu lub zielonooki* i *średniego wzrostu i nie zielonooki* można wyznaczyć dwuargumentowe funkcje przynależności stosując odpowiednio operacje iloczynu, sumy, i iloczynu zbioru i zbioru zanegowanego. Otrzymane rezultaty przedstawione są na rys. 5.7



Rys. 5.7: Funkcje przynależności podzbiorów: (a) *średniego wzrostu i zielonooki*, (b) *średniego wzrostu lub zielonooki* i (c) *średniego wzrostu i nie zielonooki*

Definiuje się także poniższe operacje algebraiczne:

Definicja 5.8 *Koncentracja zbioru*

Koncentracją zbioru rozmytego A jest zbiór $CON(A)$ o funkcji przynależności

$$\forall x \in X : \quad \mu_{CON(A)}(x) = (\mu_A(x))^2$$

Definicja 5.9 *Rozcieńczenie zbioru*

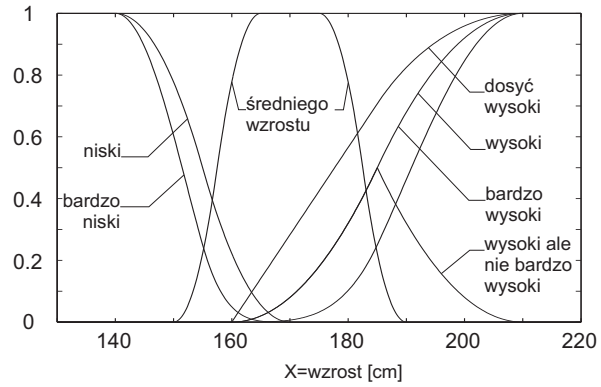
Rozcieńczeniem zbioru rozmytego A jest zbiór $DIL(A)$ o funkcji przynależności

$$\forall x \in X : \quad \mu_{CON(A)}(x) = (\mu_A(x))^{0,5}$$

Powyższe dwie operacje często używa się do tworzenia funkcji przynależności zbiorów których definicje słowne zawierają przedrostki *bardzo* lub *dosyć, dość*. Np. jeżeli zbiór rozmyty określony słowem „gorący" ma funkcje przynależności $\mu(\cdot)$ to zbiory „bardzo gorący" i „dość gorący" mogą mieć funkcje przynależności $\mu(\cdot)^2$ i $\mu(\cdot))^{0,5}$. Ilustruje to poniższy przykład

Przykład 5.5

Ludzie i ich wzrost. Niech przestrzeń X będzie zbiorem liczb rzeczywistych w przedziale $[130, 220]$ reprezentującym wzrost osób wyrażony w cm. Zdefiniujmy kilka podzbiorów rozmytych: *bardzo niski, niski, średniego wzrostu, wysoki ale nie bardzo wysoki, dosyć wysoki,wysoki, bardzo wysoki*, które będą odpowiadały odpowiednim pojęciom w języku naturalnym. Korzystając z funkcji przynależności zbiorów „niski", „średniego wzrostu", „wysoki" z przykładu 5.3 oraz operacji negacji, a także CON i DIL uzyskujemy funkcje przynależności przedstawione rysunku 5.8. ■



Rys. 5.8: Funkcje przynależności kilku zbiorów rozmytych dotyczących wzrostu

5.1.3. Relacje rozmyte

Relacje w klasycznej teorii zbiorów określa się za pomocą zbioru elementów, par elementów, trójek elementów, itd. Zbiory te określaja odpowiednio, relacje unarne (jednoczłonowe), binarne i ogólnie n-arne. Podobnie, zbiór rozmyty reprezentuje relację rozmytą. Funkcja przynależności relacji rozmytej określa stopień zachodzenia relacji.

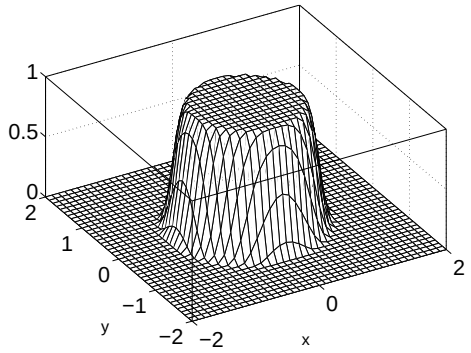
Przykład 5.6

Nierówność

$$x^2 + y^2 \leq R^2$$

określa relację pomiędzy liczbami x i y spełnianą przez punkty o współrzędnych x, y znajdujące się w kole o środku w początku układu współrzędnych i promieniu R.

Analogicznie zbiór rozmyty o funkcji przynależności przedstawionej na rys. 5.9 reprezentuje relację rozmytą znajdowania się punktu wewnątrz koła jednostkowego. ■



Rys. 5.9: Funkcja przynależności relacji rozmytej: *punkt o współrzędnych x, y znajduje się wewnątrz koła jednostkowego*

Złożenie relacji

Złożenie relacji klasycznych (nie rozmytych):

Niech A, B i C będą trzema zbiorami. Dana jest relacja $\mathcal{R}$ od A do B i relacja $\mathcal{S}$ z B do C, wtedy złożeniem $\mathcal{R} \circ \mathcal{S}$ relacji $\mathcal{R}$ i $\mathcal{S}$ jest relacja od A do C zdefiniowana jako:

$$a(\mathcal{R} \circ \mathcal{S})c \Leftrightarrow \text{istnieje pewne } b \in B \text{ takie, że } a\mathcal{R}b \text{ i } b\mathcal{S}c$$

Na przykład, jeżeli $\mathcal{R}$ jest relacją „bycia matką" i $\mathcal{S}$ jest relacją „żonaty z", wtedy $\mathcal{R} \circ \mathcal{S}$ jest relacją „bycia teściową".

Definicja 5.10 *Złożenie relacji rozmytych*

Niech A, B i C będą trzema zbiorami rozmytymi. Dana jest relacja $\mathcal{R}$ od A do B i relacja $\mathcal{S}$ z B do C, wtedy złożenie max-min $\mathcal{R} \circ \mathcal{S}$ relacji $\mathcal{R}$ i $\mathcal{S}$ jest relacją rozmytą od A do C zdefiniowaną natępująco:

$$\mu_{\mathcal{R}\circ\mathcal{S}}(a, c) = \max_b \min[\mu_{\mathcal{R}}(a, b), \mu_{\mathcal{S}}(b, c)], \qquad a \in A, b \in B, c \in C$$

Przykład 5.7

$$\begin{aligned} \mathcal{R} &= \text{„}x \text{ jest w relacji z } y\text{"} - \text{relacja rozmyta zdefiniowana na } X \times Y \\ \mathcal{S} &= \text{„}y \text{ jest w relacji z } z\text{"} - \text{relacja rozmyta zdefiniowana na } Y \times Z \end{aligned}$$

gdzie przestrzeniami są $X = \{1, 2, 3, 4\}$, $Y = \{a, b, c\}$, $Z = \{\alpha, \beta, \gamma, \delta, \epsilon\}$

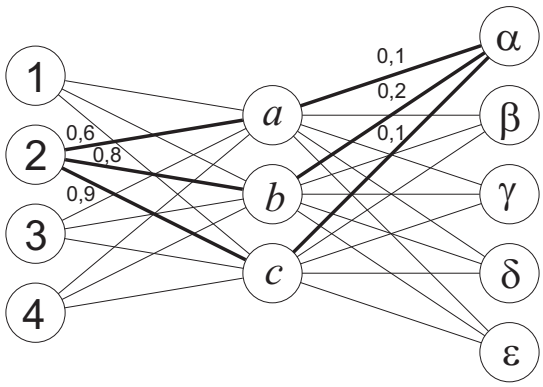
Niech relację $\mathcal{R}$ i $\mathcal{S}$ są zdefiniowane przez następujące dwuwymiarowe funkcje przynależności:

$$\mathcal{R} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \left[\begin{array}{ccc} 0.4 & 0.6 & 0.8 \\ 0.6 & 0.8 & 0.9 \\ 0.7 & 0.7 & 0.7 \\ 0.8 & 0.4 & 0.1 \end{array}\right] \end{array}$$

$$\mathcal{S} = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccccc} \alpha & \beta & \gamma & \delta & \epsilon \end{array} \\ \left[\begin{array}{ccccc} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \\ 0.2 & 0.3 & 0.5 & 0.3 & 0.2 \\ 0.1 & 0.2 & 0.9 & 0.6 & 0.3 \end{array}\right] \end{array}$$

Wtedy wyznaczenie funkcji przynależności złożenia tych relacji przebiega jak przedstawiono poniżej, gdzie wyznaczono wartość funkcji przynależności tej relacji dla wartości $x = 2$ i $z = \alpha$ (patrz też rys. 5.10).

$$\begin{aligned} \mu_{\mathcal{R}\circ\mathcal{S}}(2,\alpha) &= \max[\min(0.6,0.1), \min(0.8,0.2), \min(0.9,0.1)] \\ &= \max(0.1, 0.2, 0.1) \\ &= 0.2 \end{aligned}$$

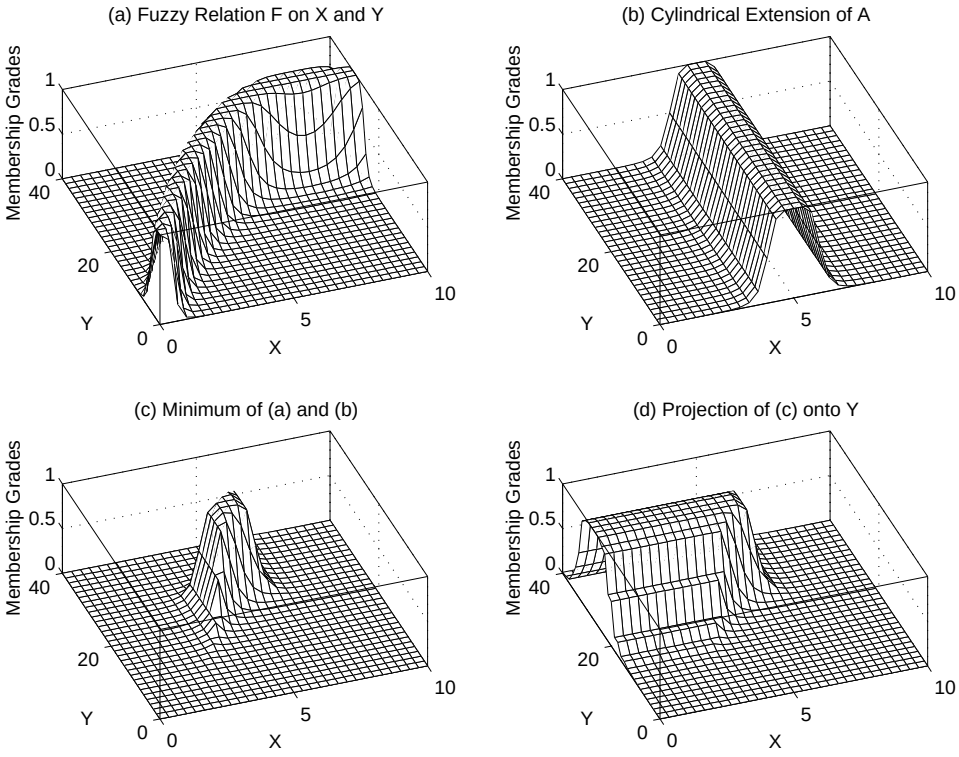


Rys. 5.10: Przykład złożenia relacji rozmytych. Linie pogrubione ilustrują wyznaczenie wartości relacji pomiędzy 2 i α

Definicja 5.11 *Złożenie zbioru rozmytego i relacji rozmytej*

Dany jest zbiór rozmyty $A \subseteq X$ oraz relacja rozmyta $\mathcal{F} \subseteq X \times Y$, wtedy złożenie $B = A \circ \mathcal{F}$ zbioru A i relacji $\mathcal{F}$ jest zbiorem rozmytym $B \subseteq Y$ zdefiniowanym natępująco (patrz rys. 5.11):

$$\mu_B(y) = \max_x \min[\mu_A(x), \mu_{\mathcal{F}}(x,y)],$$



Rys. 5.11: Ilustracja złożenia zbioru rozmytego i relacji rozmytej (Jang, 99)

5.1.4. Wnioskowanie rozmyte

Wnioskowanie nierozmyte przy zastosowaniu reguły odrywania (modus ponens) przebiega według schematu:

przesłanka 1 (fakt):	x jest A
przesłanka 2 (reguła):	jeżeli x jest A to y jest B
konkluzja	y jest B

Np. ze zdań: „jeżeli jedzenie jest dobre to napiwek jest wysoki" i „jedzenie jest dobre" wynika zdanie „napiwek jest wysoki".

We wnioskowaniu rozmytym obie przesłanki i konkluzja są zbiorami rozmytymi i przebiega ono według schematu:

przesłanka 1 (fakt):	x jest A'
przesłanka 2 (reguła):	jeżeli x jest A to y jest B
konkluzja	y jest B'

Np. ze zdań „jeżeli jedzenie jest dobre to napiwek jest wysoki" (przesłanka 2) i „jedzenie jest takie sobie" (przesłanka 1) wynika zdanie „napiwek jest nienajwyższy" (konkluzja).

Formalnie, we wnioskowaniu rozmytym przyjmuje się, że wynikowy zbiór rozmyty $B' \subseteq Y$ wynika z operacji złożenia zbioru rozmytego $A' \subseteq X$ i relacji rozmytej $A \to B \subseteq X \times Y$ (definicja 5.11). Tak więc

$$\mu_{B'}(y) = \max_x \min[\mu_{A'}(x), \mu_{A\to B}(x, y)] \tag{5.1}$$

lub inaczej

$$B' = A' \circ (A \to B)$$

Relacja rozmyta $A \to B$ może być definiowana na wiele sposobów. Najczęściej korzysta się z następującej definicji zaproponowanej przez Mamdaniego, w której implikacje utożsamia się z iloczynem kartezjańskim zbiorów rozmytych (definicja 5.6)

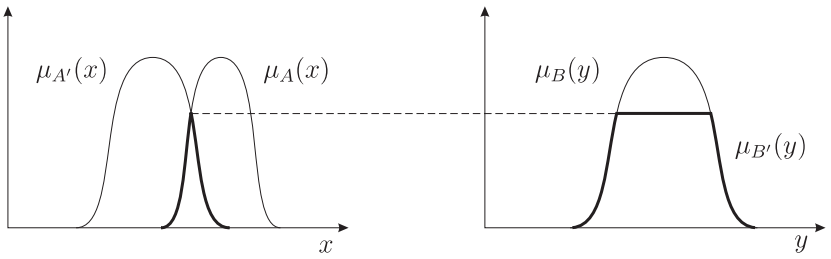
$$A \to B = A \times B$$

czyli

$$\mu_{A\to B} = \mu_{A\times B}(x, y) = \min(\mu_A(x), \mu_B(y)) \tag{5.2}$$

Podstawiając (5.2) do (5.1) otrzymujemy (patrz też rys. 5.12)

$$\begin{aligned} \mu_{B'}(y) &= \max_x \min[\mu_{A'}(x), \min(\mu_A(x), \mu_B(y))] \\ &= \min[\max_x \min[\mu_{A'}(x), (\mu_A(x)], \mu_B(y)] \end{aligned} \tag{5.3}$$



Rys. 5.12: Interpretacja graficzna wnioskowania rozmytego (patrz wzór (5.3))

Wykorzystując uzyskaną zależność oraz rozszerzając dotychczasowe rozważania na wiele reguł, większą od jednego liczbę sygnałów wejściowych i uwzględniając fakt, że sygnały wejściowe przyjmują wartość dokładną (nie rozmytą) proces wnioskowania rozmytego sprowadza się do poniżej opisanego postępowania.

Kompletny system wnioskowania rozmytego zawiera pewną liczbę reguł rozmytych o postaci:

$$\begin{array}{llllll} \text{jeżeli } x_1 \text{ jest } A_{11} & \text{i} & x_2 \text{ jest } A_{12} & \text{i} \ldots \text{i} & x_n \text{ jest } A_{1n}, & \text{to } y \text{ jest } B_1 \\ \text{jeżeli } x_1 \text{ jest } A_{21} & \text{i} & x_2 \text{ jest } A_{22} & \text{i} \ldots \text{i} & x_n \text{ jest } A_{2n}, & \text{to } y \text{ jest } B_2 \\ & & \vdots & & & \\ \text{jeżeli } x_1 \text{ jest } A_{m1} & \text{i} & x_2 \text{ jest } A_{m2} & \text{i} \ldots \text{i} & x_n \text{ jest } A_{mn}, & \text{to } y \text{ jest } B_m \end{array}$$

gdzie A_{ij}, B_j, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, są zbiorami rozmytymi o znanych funkcjach przynależności, x_i, $i = 1, 2, \ldots, n$ są sygnałami wejściowymi, a y jest sygnałem wyjściowym.

Typowy proces wnioskowania rozmytego polega na odwzorowaniu liczbowych sygnałów wejściowych (nie rozmytych) na liczbowe sygnały wyjściowe (nie rozmyte). Proces wnioskowania zachodzi w pięciu etapach nazywanych *rozmywaniem* (*fuzyfikacją*), zastosowaniem operacji rozmytych, zastosowaniem implikacji (omówiona wcześniej definicja Mamdaniego lub inne), kompozycją (metoda *MAX* lub *sumacyjna*) oraz *precyzowaniem* (*defuzyfikacją*).

Proces wnioskowania rozmytego omówimy na poniższym przykładzie.

PRZYKŁAD 5.8

Przyjmijmy, że dane są następujące reguły:

reguła 1: jeżeli wysokość jest duża, to siła jest mała
reguła 2: jeżeli wysokość jest mała i prędkość jest mała, to siła jest mała
reguła 3: jeżeli wysokość jest mała i prędkość jest duża, to siła jest duża

oraz funkcje przynależności zbiorów rozmytych występujących w regułach:

$$wm(x) = \begin{cases} 1 - x/160 & \text{gdy } x \leq 160 \\ 0 & \text{gdy } 160 < x \end{cases}, \quad wd(x) = \begin{cases} 0 & \text{gdy } x \leq 80 \\ x/80 - 1 & \text{gdy } 80 < x \leq 160 \\ 1 & \text{gdy } 160 < x \end{cases}$$

$$pm(x) = 1 - x/20, \qquad pd(x) = x/20$$

$$sm(x) = \begin{cases} 1 - x/50 & \text{gdy } x \leq 50 \\ 0 & \text{gdy } 50 < x \end{cases}, \quad sd(x) = \begin{cases} 0 & \text{gdy } x \leq 50 \\ x/50 - 1 & \text{gdy } 50 < x \end{cases}$$

gdzie oznaczenia i zakresy zmiennych zestawione są poniżej

nazwa zbioru	funkcja przynależności	zakres zmiennej
„wysokość mała”	wm	$[0, 200\text{m}]$
„wysokość duża”	wd	$[0, 200\text{m}]$
„prędkość mała”	pm	$[0, 20\text{m/s}]$
„prędkość duża”	pd	$[0, 20\text{m/s}]$
„siła mała”	sm	$[0, 100\text{kN}]$
„siła duża”	sd	$[0, 100\text{kN}]$

Omawiany system mogłby np. dotyczyć sterowania lądującym pojemnikiem w polu grawitacyjnym. Zadaniem jest wyznaczanie wartości siły na podstawie zadanych wartości wysokości i prędkości. W przykładzie obliczymy siłę dla wartości wysokości i prędkości odpowiednio 100m i 12m/s.

Rozmywanie

W etapie rozmywania (fuzyfikacji) liczbom wejściowym przyporządkowane zostają wartości funkcji przynależności. W omawianym przykładzie sprowadza się to do wyznaczenia następujących wartości:

reguła 1: $wd(100) = 0,25$
reguła 2: $wm(100) = 0.375$, $pm(12) = 0,4$
reguła 3: $wm(100) = 0.375$, $pd(12) = 0,6$

Operacje rozmyte

Na etapie operacji rozmytych przeprowadzane są operacje logiczne na przesłankach występujących w regułach. W omawianym przykładzie odpowiada to wyznaczaniu wartości rozmytych iloczynów logicznych przesłanek. Wartości tych iloczynów dla wszystkich reguł wynoszą:

reguła 1: $\min[wd(100) = 0,25] = 0,25$
reguła 2: $\min[wm(100) = 0,375, pm(12) = 0,4] = 0,375$
reguła 3: $\min[wm(100) = 0.375, pd(12) = 0,6] = 0,375$

Implikacje

Na etapie obliczania implikacji wyznaczona zostaje funkcja przynależności wniosku (następnika) w każdej z reguł.

Stosując definicje Mamdaniego (patrz wzór 5.3 i rys. 5.12) funkcja przynależności wniosku dla każdej reguły zostaje odcięta na wysokości wynikającej z operacji rozmytych.

W omawianym przykładzie oznaczając przez s_1 funkcję *sm* obciętą na poziomie 0,25, przez s_2 funkcję *sm* obciętą na poziomie 0,375 oraz przez s_3 funkcję *sd* obciętą na poziomie 0,375 otrzymujemy:

$$\text{reguła 1:}\quad s_1(x) = \begin{cases} 0,25 & \text{gdy } x \leq 37,5 \\ 1 - x/50 & \text{gdy } 37,5 < x \leq 50 \\ 0 & \text{gdy } 50 < x \end{cases}$$

$$\text{reguła 2:}\quad s_2(x) = \begin{cases} 0,375 & \text{gdy } x \leq 31,25 \\ 1 - x/50 & \text{gdy } 31,25 < x \leq 50 \\ 0 & \text{gdy } 50 < x \end{cases}$$

$$\text{reguła 3:}\quad s_3(x) = \begin{cases} 0 & \text{gdy } x \leq 50 \\ x/50 - 1 & \text{gdy } 50 < x \leq 68,75 \\ 0,375 & \text{gdy } 68,75 < x \end{cases}$$

Kompozycja

Na etapie kompozycji wyznaczona zostaje funkcja przynależności wniosku będąca złożeniem funkcji przynależności wniosków w poszczególnych regułach. Stosuje się dwie metody kompozycji.

W metodzie kompozycji MAX funkcja przynależności wniosku przyjmuje wartości, które są maksimami spośród wartości funkcji przynależności otrzymanych dla wszystkich reguł.

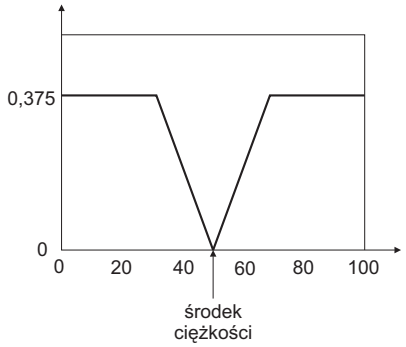
W omawianym przykładzie przyjmując, że implikacje wyznaczono metodą Mamdaniego, kompozycja MAX wyznacza funkcję przynależności wyniku (oznaczoną jako $s(x)$ (rys. 5.13):

$$s(x) = \begin{cases} 0,375 & \text{gdy } x \leq 31,25 \\ 1 - x/50 & \text{gdy } 31,25 < x \leq 50 \\ x/50 - 1 & \text{gdy } 50 < x \leq 68,75 \\ 0,375 & \text{gdy } 68,75 < x \end{cases}$$

W metodzie kompozycji sumacyjnej funkcja przynależności wniosku jest sumą funkcji przynależności otrzymanych dla wszystkich reguł. Może ona przyjmować wartości przekraczające 1 i wtedy, gdy to jest konieczne można ją normalizować.

Precyzowanie

Proces otrzymywania jednej liczbowej wartości odpowiedzi na podstawie otrzymanej funkcji przynależności nazywa sie *prcyzowaniem* (defuzyfikacją). Proponowanych jest wiele algorytmów precyzowania spośród których najczęściej stosowane jest obliczenie środka ciężkości i wyznaczenie maksimum.



Rys. 5.13: Funkcja przynależności wyjściowego zbioru rozmytego $s(x)$ dla wartości wejściowych wysokości i prędkości odpowiednio 100m i 12m/s oraz wartość wynikowa wynosząca 50

W pierwszej metodzie rozwiązaniem jest środek ciężkości funkcji przynależności, który wyznaczany jest z wzoru

$$s_c = \frac{\int_a^b xs(x)dx}{\int_a^b s(x)dx}$$

gdzie $s(\cdot)$ jest funkcją przynależności, zaś przedział $[a, b]$ jest jej dziedziną. W omawianym przykładzie środek ciężkości otrzymanej funkcji przynależności jest równy $s_c = 50$ i ta wartość jest odpowiedzią systemu na sygnały wejściowe 100 i 12 co pokazano na rys. 5.13.

W drugiej metodzie rozwiązaniem jest wartość zmiennej niezależnej, dla której funkcja przynależności przyjmuje wartość maksymalną. ■

Przykład 5.9

Na rys. 5.14 przedstawiono inny przykład wnioskowania rozmytego. Zwróćmy uwagę na występowanie w regułach funkcji OR. ■

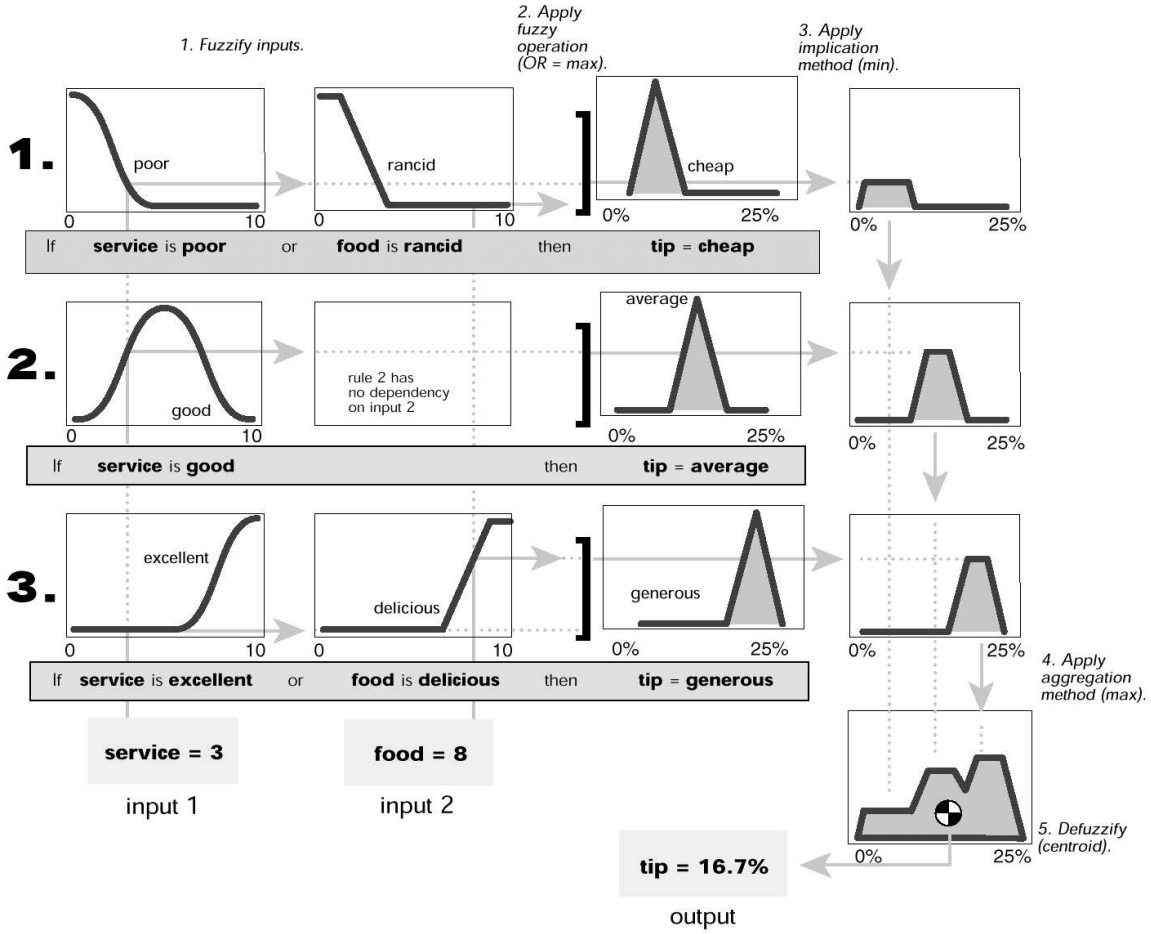
Przykład 5.10

Na rys. 5.15 przedstawiono przykład wnioskowania rozmytego metodą Sugeno. ■

5.2. Sieci bayesowskie

W połowie lat 90. do wnioskowania w sytuacjach niedeterministycznych zaczęto stosować metody rachunku prawdopodobieństwa, a w szczególności *sieci bayesowskie* lub *sieci przekonań* (ang. *bayesian networks* lub *bayesian belief networks*) (Russel, Norvig 1995 str. 436, Heckerman, Wellman 1995, Li 1998). Ich stosowanie wynika z następujących spostrzeżeń:

- Wiedza posiadana przez ludzi i obserwowane fakty stanowiące podstawę wnioskowania są niepewne.



Rys. 5.14: Inny przykład wnioskowania rozmytego – ustalanie wielkości napiwku (Matlab Fuzzy Logic Toolbox)

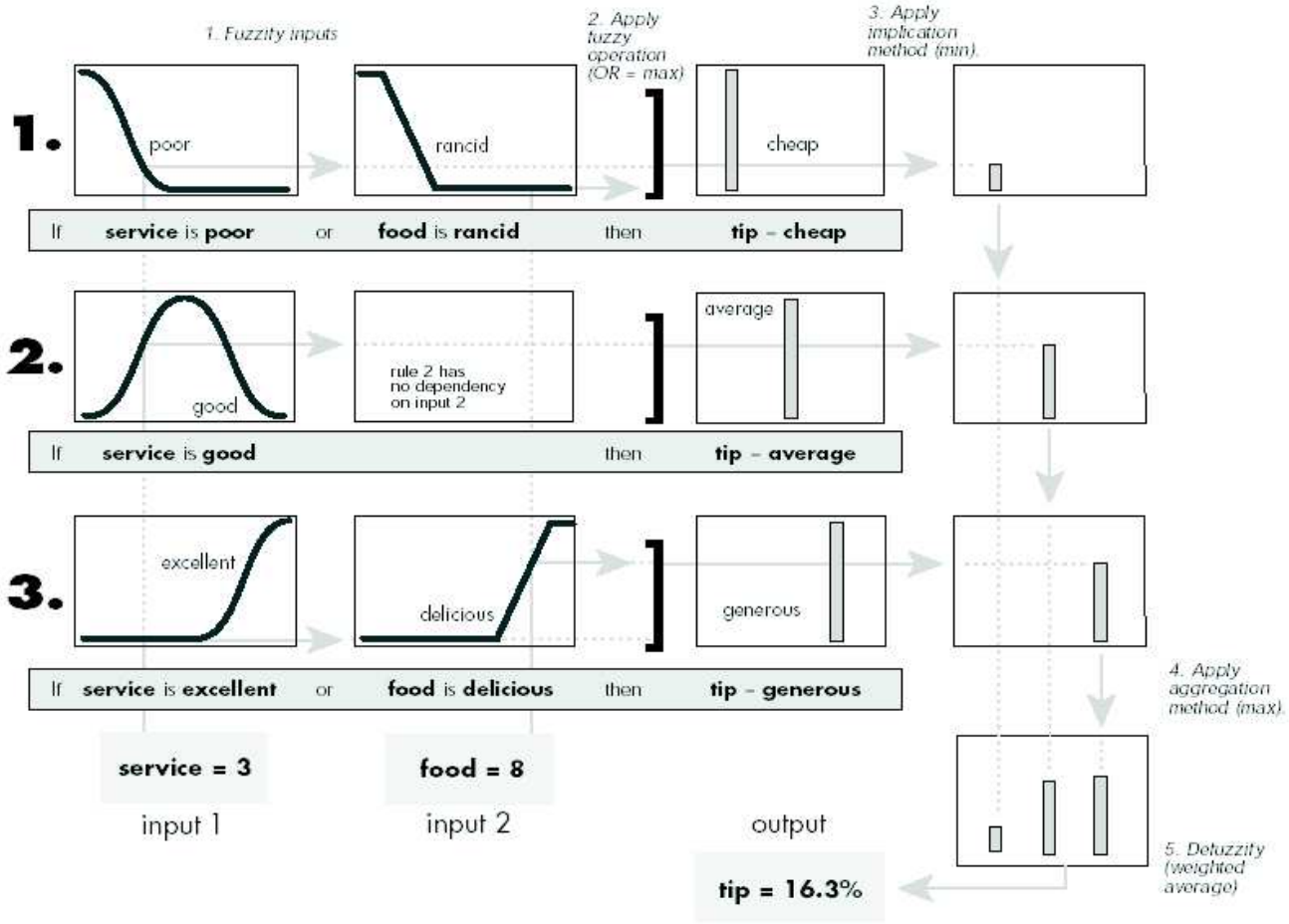
- Wiedza zapisana w systemie ekspertowym będąca modelem wiedzy posiadanej przez człowieka ma najczęściej charakter probabilistyczny, a nie deterministyczny.

W początkowym okresie rozwoju systemów ekspertowych stosowanie podejścia probabilistycznego napotykało na poważne trudności wynikające z wielkiej liczby wymaganych obliczeń i stąd opracowano metody przybliżone obliczające na każdym etapie wnioskowania liczby mówiące o wiarygodności wyprowadzonych wniosków jako pewne funkcje wiarygodności przesłanek. Systemy rozmyte oferowały całkowite odejście od metod probabilistycznych. Przełom w stosowaniu metod probabilistycznych stanowiło wprowadzenie graficznego języka modelowania dla reprezentacji zależności niepewnych. Reprezentacje takie były znane od lat dwudziestych ale zostały niedawno ponownie odkryte i zastosowane.

Sieć bayesowska jest grafem reprezentującym zbiór zależnych od siebie zmiennych losowych i wskazujacym strukturę ich zależności.

Definicja 5.12 *Sieć bayesowska jest acyklicznym grafem skierowanym, w którym:*

- *Węzły reprezentują zmienne losowe (wnioski w systemie). Zmienne losowe mogą*



Rys. 5.15: Przykład wnioskowania rozmytego metodą Sugeno – ustalanie wielkości napiwku (Matlab Fuzzy Logic Toolbox)

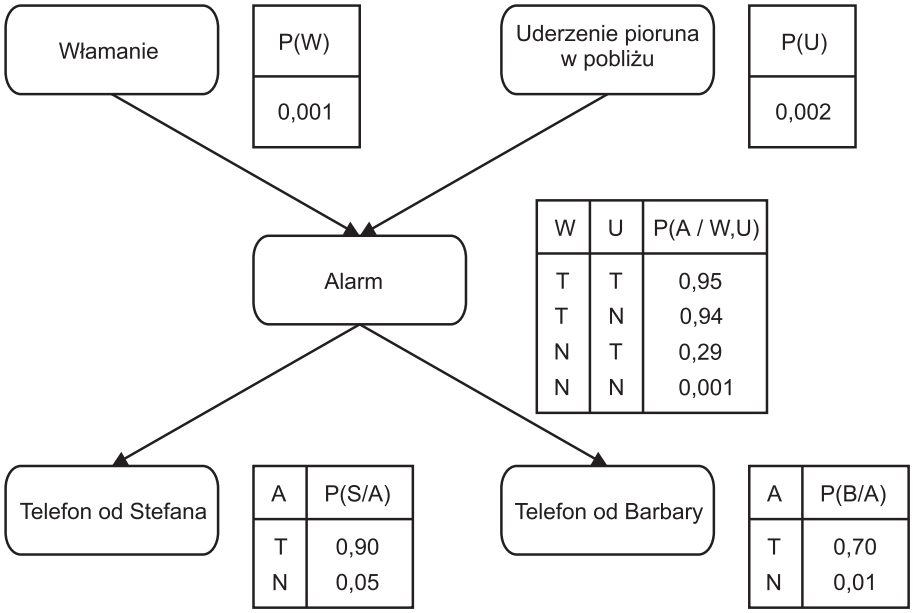
być dyskretne (przyjmujące wartości ze zbiorów skończonych lub przeliczalnych) lub ciągłe.

- *Ukierunkowane krawędzie reprezentują zależności przyczynowe pomiędzy zmiennymi losowymi.*
- *Każdy węzeł potomny A skojarzony jest z prawdopodobieństwem warunkowym $p(A/\mathbf{B})$, gdzie $\mathbf{B}$ jest zbiorem wszystkich jego węzłów rodzicielskich. Każdy korzeń grafu C skojarzony jest z prawdopodobieństwem bezwarunkowym $p(C)$.*

Przykład 5.11

Rozpatrzmy następującą sytuację. W mieszkaniu został zainstalowany system alarmowy. Jest prawie niezawodny przy wykrywaniu włamania, ale czasami w czasie burzy bliskie uderzenie pioruna także uruchamia alarm. Właściciel alarmu ma dwóch zaprzyjaźnionych sąsiadów Stefana i Barbarę, którzy zobowiązali się telefonować do niego do pracy gdyby usłyszeli alarm. Stefan prawie zawsze telefonuje gdy usłyszy alarm, ale czasami bierze za alarm dzwięki klaksonu dochodzące z ulicy i też telefonuje. Barbara z kolei często słucha głośnej muzyki i czasami nie słyszy alarmu. Ta

prosta sytuacja może być przedstawiona w postaci sieci bayesowskiej pokazanej na rys. 5.16, gdzie w tablicach obok węzłów podano prawdopodobieństwa warunkowe i bezwarunkowe realizacji zmiennych losowych reprezentowanych przez węzły.



Rys. 5.16: Przykładowa sieć bayesowska. Litery W, U, A, S i B oznaczają odpowiednio zmienne losowe „Włamanie", „Uderzenie pioruna w pobliżu", „Alarm", „Telefon od Stefana" i „Telefon od Barbary" (na podstawie: Russel S., Norvig P., Artificial Intelligence, Prentice-Hall, London 2003)

Zadanie wnioskowania w systemach probabilistycznych polega na wyznaczaniu rozkładu prawdopodobieństwa zbioru hipotez $\mathbf{H} = H_1, H_2, \ldots, H_m$ przy zadanych wartościach przesłanek $\mathbf{e} = e_1, e_2, \ldots, e_k$. W rachunku prawdopodobieństwa znajomość łącznego rozkładu prawdopodobieństwa zmiennych losowych umożliwia wyznaczenie dowolnego prawdopodobieństwa warunkowego tych zmiennych. Czyli wnioskowanie sprowadza się do obliczenia prawdopodobieństwa warunkowego $p(\mathbf{H}|\mathbf{e})$ według wzoru

$$p(\mathbf{H}|\mathbf{e}) = \frac{p(\mathbf{e}, \mathbf{H})}{p(\mathbf{e})}$$

lub według reguły Bayesa

$$p(\mathbf{H}|\mathbf{e}) = \frac{p(\mathbf{e}|\mathbf{H})p(\mathbf{H})}{p(\mathbf{e})}$$

5.2.1. Niezależność warunkowa

Obliczenie łącznego rozkładu prawdopodobieństwa dokonuje się na podstawie zadanych prawdopodobieństw warunkowych według reguły łańcuchowej

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|x_1, \ldots, x_{i-1})$$

Zadanie wyznaczenia rozkładu $p(x_1, x_2, \ldots, x_n)$ znacznie się upraszcza jeżeli w systemie występuje *niezależność warunkową* zmiennych losowych zdefiniowana następująco: Jeżeli zachodzi:

$$p(x_i|x_1, \ldots, x_{i-1}) = p(x_i|\mathbf{R_i})$$

gdzie $\mathbf{R}_i \subseteq \{x_1, \ldots, x_{i-1}\}$, to zmienna losowa x_i jest warunkowo niezależna od zmiennych losowych zawartych w zbiorze $\{x_1, \ldots, x_{i-1}\} \setminus \mathbf{R_i}$. Przy występowaniu niezależności warunkowej rozkład zmiennej losowej x_i zależy od zbioru $\mathbf{R}_i$ nazywanego rodzicielskim, a nie zależy od zbioru $\{x_1, \ldots, x_{i-1}\} \setminus \mathbf{R_i}$.

Struktura sieci bayesowskiej bezpośrednio wskazuje zależności pomiędzy zmiennymi losowymi w systemie. Stosuje się konwencje, że węzły rodzicielskie danego węzła tworzą zbiór rodzicielski $\mathbf{R}_i$, czyli warunkowy rozkład prawdopodobieństwa danego węzła zależy od wartości węzłów rodzicielskich, a nie zależy od wartości węzłów je poprzedzających. Węzły rodzicielskie „zasłaniają" poprzedzające węzły.

Ze względu na występowanie w sieci bayesowskiej niezależności warunkowych możliwe jest na podstawie opisujących sieć prawdopodobieństw warunkowych wyznaczenie prawdopodobieństwa łącznego i następnie dowolnych innych prawdopodobieństw.

Pokazują to dwa poniższe przykłady.

Przykład 5.12

Dana jest prosta sieć (łańcuch Markowa) o strukturze $x \to y \to z$ i prawdopodobieństwach danych w tablicy 5.1. Należy wyznaczyć rozkład prawdopodobieństwa $p(x|z)$.

$\neg x$	$0,73$
x	$0,27$

$p(x)$

	$\neg y$	y
$\neg x$	$0,20$	$0,80$
x	$0,70$	$0,30$

$p(y|x)$

	$\neg z$	z
$\neg y$	$0,32$	$0,68$
y	$0,75$	$0,25$

$p(z|y)$

Tablica 5.1: Rozkłady prawdopodobieństwa zmiennych losowych w przykładowej sieci bayesowskiej o strukturze $x \to y \to z$

Z wzoru na prawdopodobieństwo warunkowe mamy

$$p(x|z) = \frac{p(x,z)}{p(z)} = \frac{\sum_y p(x,y,z)}{\sum_{xy} p(x,yz)}$$

Na podstawie reguły łańcuchowej otrzymujemy wzór na rozkład łączny

$$p(x,y,z) = p(z|x,y)p(y|x)p(x)$$

Ze względu na to, że zmienna losowa z jest warunkowo niezależna od zmiennej x mamy $p(z|x,y) = p(z|y)$, i stąd

$$p(x,y,z) = p(z|y)p(y|x)p(x)$$

oraz ostatecznie

$$p(x|z) = \frac{p(x)\sum_y p(z|y)p(y|x)}{\sum_x \left[p(x)\sum_y p(z|y)p(y|x)\right]}$$

Podstawiając do powyższego wzoru wartości prawdopodobieństw z tablicy 5.1 otrzymujemy rozkład $p(x|z)$ przedstawiony w tablicy 5.2. ■

	$\neg z$	z
$\neg x$	$0,3536$	$0,1840$
x	$0,6464$	$0,8160$

$p(x|z)$

Tablica 5.2: Obliczony rozkład prawdopodobieństwa $p(x|z)$ w przykładowej sieci bayesowskiej o strukturze $x \rightarrow y \rightarrow z$

Przykład 5.13

Korzystając z niezależności warunkowej można łatwo dla sieci bayesowskiej z rys. 5.16 wyznaczyć prawdopodobieństwo łączne. Korzystając z reguły łańcuchowej i niezależności warunkowych otrzymujemy

$$\begin{aligned} p(A,B,S,U,W) &= p(B|S,A,U,W)p(S,A,U,W) \\ &= p(B|A)p(S|A,U,W)p(A,U,W) \\ &= p(B|A)p(S|A)p(A|U,W)p(U,W) \\ &= p(B|A)p(S|A)p(A|U,W)p(U)p(W) \end{aligned}$$

Z otrzymanego wzoru i rozkładów warunkowych mamy np.

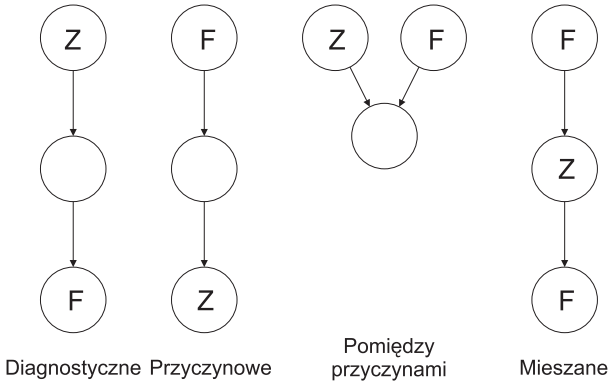
$$p(A,\neg B,\neg S,U,W) = 0,3 \cdot 0,1 \cdot 0,95 \cdot 0,002 = 5,7 \cdot 10^{-8}$$

■

5.2.2. Rodzaje wnioskowania w sieciach bayesowskich

Można wyróżnić cztery następujące typy wnioskowania w sieciach bayesowskich, (rysunek 5.17):

- Wnioskowanie diagnostyczne: Od objawów do przyczyn.
- Wnioskowanie przyczynowe: Od przyczyn do objawów.
- Wnioskowanie pomiędzy przyczynami (wyjaśnienie osłabiające (ang. explaining away)): Pomiędzy kilkoma przyczynami tego samego objawu (jakie jest prawdopodobieństwo wystąpienia przyczyny, gdy zaszły objawy i inne przyczyny). Często występowanie objawu i jednej przyczyny zmniejsza prawdopodobieństwo przyczyny drugiej.
- Wnioskowanie mieszane: Połączenie dwóch lub więcej powyższych typów.



Rys. 5.17: Cztery typy wnioskowania w sieciach bayesowskich. Wszystkie one polegają na wyznaczaniu prawdopodobieństwa $p(Z|F)$, gdzie Z jest zmienną, której prawdopodobieństwo chcemy otrzymać (zmienną reprezentującą zapytanie), a F jest zmienną reprezentującą zaistniały fakt

5.2.3. Algorytmy wnioskowania w siecach bayesowskich

W punkcie 5.2.1. pokazano, że struktura sieci wraz z zadanymi prawdopodobieństwami warunkowymi stanowią opis równoważny prawdopodobieństwu łącznemu. Prowadzi to do algorytmu wyznaczania dowolnego prawdopodobieństwa polegającego na wyznaczeniu rozkładu łącznego, a następnie na jego podstawie dowolnego rozkładu poszukiwanego.

Ilość danych potrzebna do opisu sieci wraz z zadanymi prawdopodobieństwami warunkowymi jest znacznie mniejsza od ilości danych potrzebnych do opisu prawdopodobieństwa łącznego. Z tego powodu algorytm wnioskowania polegający na wyznaczeniu najpierw rozkładu łącznego, a następnie rozkładów poszukiwanych, które też mogą zawierać niewielką ilość danych jest nieefektywny, gdyż rozpoczynając się od niewielkiej liczby danych, generuje ich dużą liczbę aby w wyniku znowu otrzymać niewielką liczbę danych.

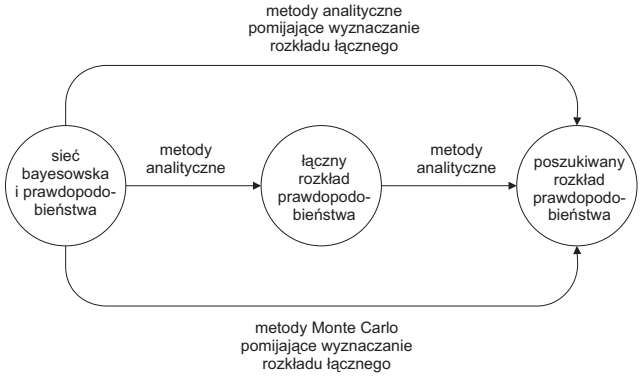
Znaleziono algorytmy pozwalające bezpośrednio wyznaczyć poszukiwany dowolny rozkład prawdopodobieństwa bez konieczności przechodzenia przez rozkład łączny (patrz rys. 5.18). Algorytmy te ze względu na ich złożony opis nie będą tu omawiane.

Oprócz algorytmów dokładnych często stosuje się metody symulacji stochastycznej (metody Monte Carlo).

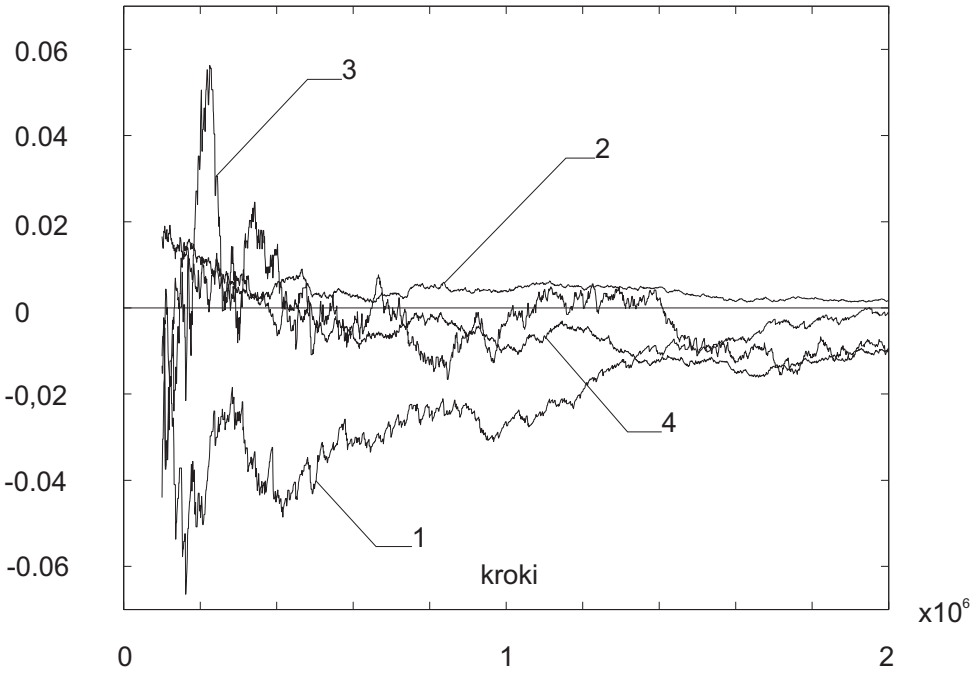
W metodach symulacji stochastycznej najpierw losowo generuje się wartości korzeni drzewa zgodnie z zadanymi prawdopodobieństwami, a następnie generuje się wartości pozostałych węzłów zgodnie z zadanymi prawdopodobieństwami warunkowymi. Na podstawie otrzymanych wartości oblicza się poszukiwane częstości, które dla dużej liczby eksperymentów zmierzają do odpowiednich prawdopodobieństw.

Przykład 5.14

Dla sieci z rys. 5.16 przeprowadzono symulację stochastyczną. Wykresy błędów otrzymywanych prawdopodobieństw w funkcji liczby kroków symulacji przedstawiono na rys. 5.19, a otrzymane końcowe wyniki oraz wyniki dokładnych obliczeń przedstawiono w tablicy 5.3. ■



Rys. 5.18: Trzy rodzaje algorytmów stosowanych do wyznaczania poszukiwanych rozkładów prawdopodobieństwa w sieciach bayesowskich



Rys. 5.19: Symulacja stochastyczna prawdopodobieństw dla sieci z rys. 5.16. Wykresy błędów $\frac{\hat{p}-p}{p}$ w funkcji liczby kroków symulacji, gdzie $\hat{p}$ oznacza estymowane prawdopodobieństwo, a p prawdopodobieństwo wyznaczone dokładnie: (1) wykres błędu dla prawdopodobieństwa $p(S|W)$, (2) $p(S|\neg W)$, (3) $p(B|W)$, (4)$p(B|\neg W)$

	$p(S\|W)$	$p(S\|\neg W)$	$p(B\|W)$	$p(B\|\neg W)$
dokładne	0,8490	0.0513	0,6586	0,0110
estymowane	0.8481	0.0514	0.6524	0.0110

Tablica 5.3: Obliczone i estymowane prawdopodobieństwa warunkowe dla sieci z rys. 5.16. Liczba kroków symulacji wynosiła $2 \cdot 10^6$

Rozdział 6

Metody uczenia maszyn

Uczenie się komputerów jest jednym z ważnych sposobów automatycznego pozyskiwania wiedzy lub uzyskiwania algorytmów (optymalnych lub suboptymalnych). W czasie uczenia systemowi uczącemu się przedstawiane są zadania do rozwiązania (na wejscie podawane są obrazy) i ewentualnie odpowiedzi, a system stopniowo ulepsza swoje działanie generując coraz lepsze rozwiązania (coraz lepiej klasyfikuje obrazy). Uczenie prowadzone na podstawie obserwacji nazywa się uczeniem *indukcyjnym*.

Celem działania systemu uczącego się jest znalezienie maszyny realizujacej pewne odwzorowanie $\mathbf{x} \to d$. Np. $\mathbf{x}$ może być wektorem pikseli fotografii przedstawiających drzewo, a $d \in \{-1, 1\}$ jest sygnałem mówiącym czy fotografia przedstawia drzewo liściaste lub iglaste.

Zadanie uczenia sprowadza się do znalezienia funkcji $y = f(\mathbf{x}, \mathbf{w})$ realizującej możliwie dokładnie to odwzorowanie, gdzie $\mathbf{w}$ jest wektorem parametrów. Zazwyczaj postać funkcji jest wybrana wcześniej i w trakcie uczenia wyznaczane są wartości parametrów $\mathbf{w}$. Przyjęcie konkretnych wartości tych parametrów ostatecznie określa funkcję f.

Aby określić jakość odwzorowania $y = f(\mathbf{x}, \mathbf{w})$ przyjmuje się zazwyczaj, że wektory $\mathbf{x}$ i sygnały d są realizacjami zmiennych losowych o łacznej gęstości prawdopodobieństwa $p(\mathbf{x}, d)$. W najbardziej ogólnym modelu d jest losowe przy ustalonym $\mathbf{x}$. Np. przy przesyłaniu sygnałów w obecności szumów d reprezentuję nadaną wiadomość, a $\mathbf{x}$ zaszumioną wiadomość odebraną, na podstawie której należy odtworzyć wiadomość nadaną. Przy ustalonym $\mathbf{x}$ możemy mieć różne d. Jednak w większości zastosowań adekwatny jest model prostszy, w którym d jest deterministyczną funkcją $\mathbf{x}$.

Przyjmując jako kryterium błędu systemu, dla pojedynczego sygnału wejściowego $\mathbf{x}$, kwadrat różnicy pomiędzy wartością oczekiwaną d, a wyjściem systemu y, całkowity błąd systemu (zwany też ryzykiem średnim) można wyrazić wzorem

$$E(\mathbf{w}) = \mathcal{E}(y-d)^2 = \int (y-d)^2 p(\mathbf{x}, d) d\mathbf{x} dd = \int \left[f(\mathbf{x}, \mathbf{w}) - d\right]^2 p(\mathbf{x}, d) d\mathbf{x} dd \quad (6.1)$$

gdzie $\mathcal{E}$ oznacza operację uśredniania.

Znając gestość $p(\mathbf{x}, d)$ można wyznaczyć wartość parametrów $\mathbf{w}$ minimalizujących bład (6.1). Najczęściej w zastosowaniach praktycznych rozkład ten jednak nie jest znany i wyznaczanie parametrów $\mathbf{w}$ odbywa się na drodze uczenia.

6.1. Metody iteracyjne szukania ekstremum jako algorytmy uczenia

W czasie tzw. uczenia z nauczycielem korzysta się ze zbioru par $\mathbf{x}_i, d_i, i = 1, 2, \ldots n$, zwanego zbiorem uczącym. Na podstawie tych par można obliczyć empiryczny błąd systemu (ryzyko empiryczne, błąd uczenia) jako

$$E_{emp}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - d_i)^2 = \sum_{i=1}^{n}[f(\mathbf{x}_i, \mathbf{w}) - d_i]^2 \tag{6.2}$$

Błąd ten zależy od konkretnego zbioru uczącego i wartości parametrów $\mathbf{w}$, nie zależy natomiast od rozkładu prawdopodobieństwa $p(\mathbf{x}, d)$. Ponieważ postać funkcji f jest zadana, funkcja $E_{emp}(\mathbf{w})$ jest znana w postaci jawnej. Powstałą sytuację ilustruje poniższy prosty przykład.

Przykład 6.1

Tablica (rys. 6.1a) zawiera zbiór uczący. Maszyna realizująca odwzorowanie $\mathbf{x} \rightarrow d$ ma postać: $y = f(x, \mathbf{w}) = w_1x^2 + w_2x$.

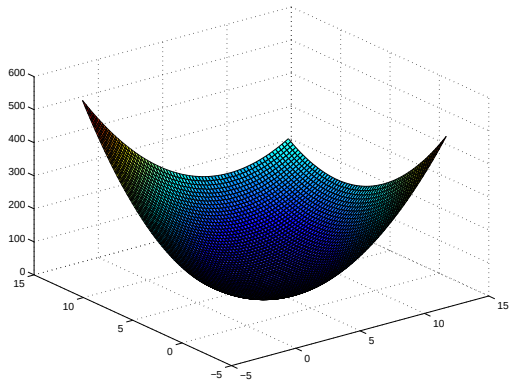
Stąd po prostych podstawieniach do wzoru (6.2) ryzyko empiryczne wyraża się wzorem (patrz też rys. 6.1b)

$$E_{emp}(\mathbf{w}) = \frac{1}{4}(2,62w_1^2 + 2,31w_2^2 - 2,31w_1w_2 - 16,91w_1 - 9,53w_2 + 82,55)$$

(a)

x	d
−1,21	2,3
0,10	3,4
0,40	5,1
0,82	6,3

(b)



Rys. 6.1: Zbiór uczący (a) oraz wykres powierzchni ryzyka empirycznego (b) do przykładu 6.4

Zadanie uczenia indukcyjnego sprowadza się do znalezienia wektora parametrów $\mathbf{w}$ minimalizujących ryzyko empiryczne dane wzorem (6.2), mając nadzieje że funkcja $f(\mathbf{x}, \mathbf{w})$ określona poprzez wektor $\mathbf{w}$ zapewni też minimum ryzyka średniego danego wzorem (6.1). Zależności pomiędzy uzyskiwanym ryzykiem empirycznym, a ryzykiem średnim i zdolność nauczonego systemu do generalizacji omówione beda w dalszej części rozdziału.

Metody analityczne poszukiwania minimum funkcji (6.2) poprzez przyrównywanie pochodnych do zera zazwyczaj nie mają znaczenia praktycznego ze względu

na nieliniowośc funkcji i występowanie licznych minimów lokalnych. W pierwszym przypadku nie można uzyskać rozwiązań powstałych równań w postaci jawnej, zaś w drugim uzyskuje się wielką liczbę rozwiązań. Z tego powodu do poszukiwania minimum stosuje się metody numeryczne. Zadania poszukiwania minimum funkcji wielu zmiennych są trudne i wymagaja dużych mocy obliczeniowych i wciąż poszukuje się efektywnych algorytmów minimalizacji. Podstawowe, stosowane, algorytmy omówione są w następnym punkcie. Są to algorytmy gradientowe, szukania przypadkowego i symulowanego wyżarzania, algorytmy ewolucyjne i algorytmy roju. Oprócz nich istnieją inne algorytmy związane z uczeniem specyficznych struktur, bedą one omówione w dalszych częściach rozdziału.

6.2. Rodzaje uczenia i struktury uczące sie

W poprzednim punkcie omówiliśmy zasadę uczenia się gdy znane są pary: sygnał wejściowy $\mathbf{x}_i$ i poprawna odpowiedź d_i. Uczenie takie nazywa się uczeniem z nauczycielem (z nadzorem) (ang. supervised). Oprócz tego wyróżnia się uczenie *z krytykiem* oraz *bez nadzoru*.

Przy uczeniu z krytykiem nie dysponujemy parami uczącymi natomiast dla każdej pary: sygnału wejsciowego $\mathbf{x}_i$ i odpowiedzi $y_i = f(\mathbf{x}_i, \mathbf{w})$, otrzymujemy ocenę odpowiedzi. Ocena pochodzi od środowiska w którym działa system uczący się i na ogół nie jest znana jej analityczna zależność od sygnałów $\mathbf{x}$ i y. Uczenie z krytykiem ma miejsce w tzw. *wieloetapowych procesach decyzyjnych* gdzie po ciągu decyzji otrzymujemy ocenę tego ciągu bez wskazania jaki powinien być.

W czasie uczenia bez nadzoru dysponujemy tylko sygnałami wejściowymi $\mathbf{x}$, nie znamy zwiazanych z nimi poprawnych odpowiedzi, ani nie znamy błędu podejmując jakąś decyzję. Uczenie to nazywa się czasami samoorganizacją i zazwyczaj polega na znajdywaniu centrów skupisk (klastrów) sygnałów wejściowych.

Funkcje odwzorowujące $y = f(\mathbf{x}, \mathbf{w})$ mogą być dowolnymi funkcjami zawierającymi parametry $\mathbf{w}$. Naogół jednak, funkcje te tworzone są jako pewne struktury składane ze standardowych prostych funkcji. Typowymi strukturami są sieci neuronowe, systemy rozmyte, drzewa decyzyjne lub sieci bayesowskie.

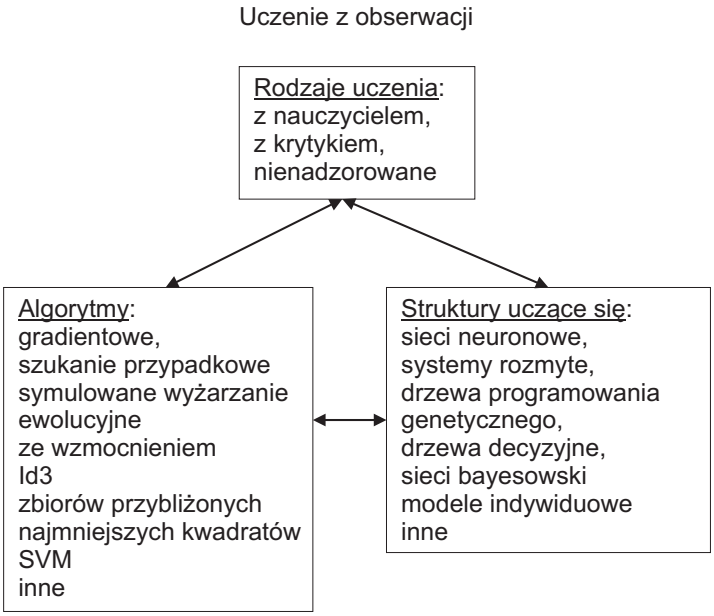
Można zauważyć, że w trakcie uczenia, w systemie uczącym się pojawia się wiedza. Może to być wiedza niejawna, uzyskiwana w postaci parametrów jakiegoś systemu przetwarzającego informację, np. wagi sieci neuronowej. System z takimi parametrami przejawia oczekiwane, optymalne działanie, można więc mówić o uczeniu się działania. Drugą postacią uzyskiwanej wiedzy jest wiedza w postaci jawnej – wyrażona np. w postaci zbioru faktów i reguł lub drzewa decyzyjnego.

Wyodrębnioną grupą problemów rozwiązywanych metodami uczenia są zadania wydobywania (odkrywania) wiedzy z baz danych. Na podstawie przeglądania danych, w procesie uczenia tworzy się wiedzę regułową o relacjach zachodzących pomiędzy danymi.

Jeszcze innym podziałem metod uczenia jest podział na metody iteracyjne wymagające wielokrotnego podawania danych uczących, w czasie ktorego system stopniowo poprawia swoje działanie i metody jednorazowe, w których na podstawie danych uczących wyznaczane są parametry systemu w jednorazowym procesie obliczeniowym.

Jedną z podstawowych pożądanych własności systemu uczącego się jest *generalizacja*, czyli zdolnośc nauczonego systemu do poprawnego działania dla sygnałów wejściowych różnych od stosowanych w procesie uczenia.

Reasumując problemy uczenia z obserwacji można podzielić na rodzaje uczenia, algorytmy uczenia oraz struktury uczące się, co ilustruje rys. 6.2. Elementy tego podziału zostaną omówione bardziej szczegółowo w dalszej części tego rozdziału.



Rys. 6.2: Uczenie indukcyjne – rodzaje, stosowane algorytmy i struktury uczące się

6.3. Podstawowe metody iteracyjne szukania ekstremum

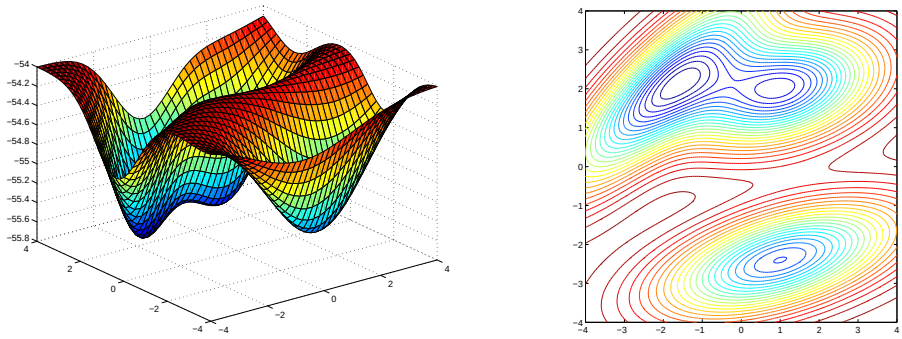
Jak pokazano powyżej, większość problemów indukcyjnego uczenia się maszyn można sprowadzić do zadania określenia przepisu funkcji f poprzez poszukiwanie minimum funkcji ryzyka empirycznego $E(\mathbf{w})_{emp}$ danej wzorem (6.2). Przy wyznaczaniu minimum dostępne są wartości funkcji $E(\mathbf{w})_{emp}$ dla zadanego punktu $\mathbf{w}$, a w przypadku stosowania uczenia z nauczycielem znana jest postać analityczna funkcji $E(\mathbf{w})_{emp}$.

Metody iteracyjne szukania ekstremum funkcji polegają na wybraniu wartości początkowej (w niektórych algorytmach wielu wartości początkowych) argumentu funkcji, a następnie na modyfikacjach argumentu prowadzących do wyznaczenia ekstremum funkcji.

Porównując metody iteracyjne z metodami szukania omówionymi w rozdziale 2 można zauważyć, że metody szukania dokładają w każdym kroku fragment rozwiązania (np. w zadaniu poszukiwania najkrótszej drogi), podczas gdy w metodach iteracyjnych szukania ekstremum, na początku tworzone jest kompletne rozwiązanie, które następnie jest iteracyjnie poprawiane.

Problem poszukiwania minimum funkcji dwóch zmiennych ilustruje rys. 6.3. Efektem poszukiwania minimum funkcji $E(\mathbf{w})$ jest punkt (wektor) $\mathbf{w}$. Zwróćmy

uwagę, że punkt może reprezentować (specyfikować) dowolne obiekty. Może np. reprezentować obrazy graficzne, sygnały czasowe (np. akustyczne), czy algorytmy lub programy komputerowe.



Rys. 6.3: Przykładowa funkcja dwóch zmiennych: wykres powierzchni i wykres warstwicowy. Należy wyznaczyć wartości argumentów dla których funkcja przyjmuje wartość minimalną (globalnie)

Poniżej omówione zostaną najczęciej stosowane, podstawowe, metody numerycznego poszukiwania ekstremum funkcji: gradientowe, szukania przypadkowego, symulowanego wyżarzania, ewolucyjne i roju. Algorytmy omawiane będą dla przypadku poszukiwania minimum funkcji.

6.3.1. Metody gradientowe

Omówiona tu będzie podstawowa metoda gradientowa nazywająca się metodą najszybszego spadku[1]. Działa ona w następujących krokach:

Poszukiwanie minimum funkcji $E(\mathbf{w})$ przebiega w nastepujacych etapach:

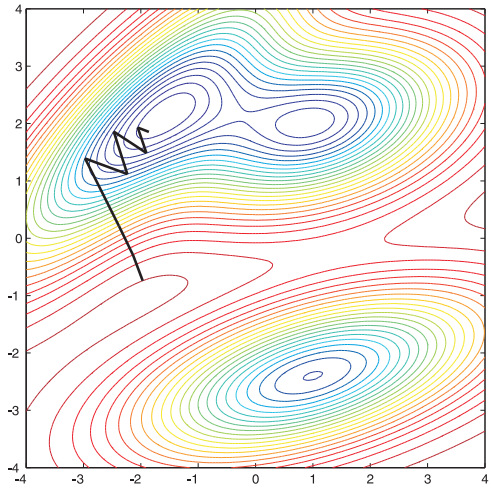
1. Losowy wybór punktu startowego $\mathbf{w}$.
2. Obliczenie gradientu funkcji $E(\mathbf{w})$ w punkcie $\mathbf{w}$

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{bmatrix}$$

3. Wyznaczenie nowego punktu według wzoru

$$\mathbf{w}' = \mathbf{w} - \eta \nabla E(\mathbf{w})$$

[1]Metody gradientowe poszukiwania ekstremum omówione są w tym opracowaniu pobieżnie. Wyczerpujące omówienie tych metod znaleźć można w monografiach i podręcznikach dotyczących algorytmów optymalizacyjnych i badań operacyjnych



Rys. 6.4: Przykładowe kolejne punkty wyznaczane w metodzie najszybszego spadku

gdzie η, jest arbitralnie wybraną stałą zwaną współczynnikiem korekcji.

Wielkość współczynnika korekcji wpływa na długość kroków algorytmu.

4. Sprawdzenie warunku zatrzymania (numer kroku lub wartość funkcji $E(\mathbf{w})$). Jeżeli warunek nie jest spełniony to skok do kroku 2.
5. Zakończenie algorytmu.

Przykładowe kroki algorytmu przedstawione są na rys. 6.4.

Do metod gradientowych należy też metoda propagacji wstecznej błędu uczenia sieci neuronowych i metoda doboru funkcji przynależności systemu wnioskowania rozmytego.

Gdy nie jest znany przepis funkcji, a funkcja dana jest przez wartości, to można zastosować przybliżenie

$$\nabla E(\mathbf{w}) \approx \frac{1}{h} \begin{bmatrix} E(\mathbf{w} + h\mathbf{e}_1) - E(\mathbf{w}) \\ E(\mathbf{w} + h\mathbf{e}_2) - E(\mathbf{w}) \\ \vdots \\ E(\mathbf{w} + h\mathbf{e}_n) - E(\mathbf{w}) \end{bmatrix}$$

gdzie h jest arbitralnie wybraną niewielką liczbą, a $\mathbf{e}_i$, $i = 1, 2, \ldots, n$ oznacza wektor jednostkowy w kierunku i. Metody gradientowe mogą zatrzymywać się w ekstremach lokalnych.

Aproksymacja stochastyczna

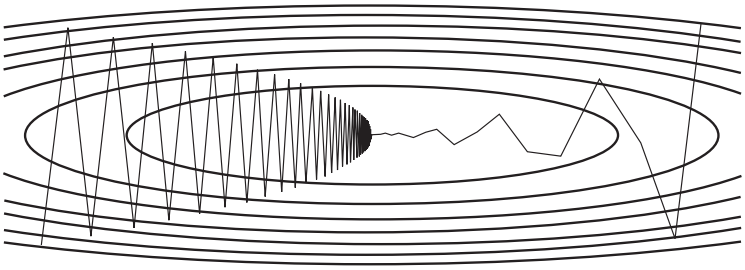
Aproksymacja stochastyczna jest odmianą metody gradientowej pozwalającą znajdować ekstremum w przypadku, gdy pomiary wartości funkcji lub obliczony gradient obarczone są przypadkowymi błędami. Aproksymacja stochastyczna jest metodą gradientową ze współczynnikiem korekcji zmienianym według wzoru

$$\eta_k = \frac{\eta_0}{k}$$

gdzie k jest numerem kroku. Malejący w ten sposób współczynnik korekcji zapewnia wytłumienie błędów, a jednocześnie umożliwia przesunięcie się punktu $\mathbf{w}$ od dowolnej wartości początkowej do ekstremum. Przyjęcie np. współczynnika $\eta_k = \eta_0/k^2$ tłumiło by błędy ale zbyt szybkie jego malenie spowodowałoby zatrzymywanie się punktu $\mathbf{w}$ poza ekstremum.

Przyspieszanie zbieżności

- Zmienianie parametru η:
  jeżeli błąd uczenia monotonicznie maleje zwiększ współczynnik η o małą wartość
  jeżeli błąd uczenia wzrósł pomnóż η przez liczbę mniejszą od 1.
- Metoda momentu
$$\Delta w(k) = -\eta \nabla E(k) + \alpha \Delta w(k-1)$$
  gdzie $0 < \alpha < 1$, patrz rys. 6.5
- Metoda Levenberga-Marquarda



Rys. 6.5: Przykład zastosowania metody najszybszego spadku i metody momentu

6.3.2. Szukanie przypadkowe

Minimalizacja funkcji $E(\mathbf{w})$ przebiega w następujących etapach:

1. Losowy wybór punktu startowego $\mathbf{w}$, przyjęcie licznika iteracji $k = 1$.
2. Wyznaczenie wartości funkcji $E(\mathbf{w})$.
3. Wyznaczenie punktu $\mathbf{w}' = \mathbf{w} + \Delta\mathbf{w}$, gdzie $\Delta\mathbf{w}$ jest realizacją zmiennej losowej o rozkładzie normalnym
$$g(\Delta\mathbf{w}, \sigma) = (2\pi\sigma)^{-n/2} \exp\left[-|\Delta\mathbf{w}|^2/(2\sigma)\right]$$
   zaś σ jest wariancją (stałą).
4. Wyznaczenie wartości funkcji $E(\mathbf{w}')$ i $\Delta E = E(\mathbf{w}') - E(\mathbf{w})$.
5. Gdy $\Delta E < 0$, to podstawienie $\mathbf{w} \leftarrow \mathbf{w}'$
6. $k \leftarrow k+1$. Jeżeli k osiągnęło przyjętą wartość lub $E(\mathbf{w})$ jest mniejsze od założonej wartości, to zakończenie obliczeń; w przeciwnym przypadku skok do 3.

6.3.3. Symulowane wyżarzanie

Symulowane wyżarzanie (ang. simulated annealing) jest odmianą metody szukania przypadkowego. Nazwa metody pochodzi od analogii jej procesu szukania ekstremum do procesu chłodzenia metali, w czasie którego, na skutek kontrolowanego, powolnego obniżania temperatury energia wewnętrzna układu osiąga minimum globalne.

Minimalizacja funkcji $E(\mathbf{w})$ przebiega w następujących etapach:

1. Losowy wybór punktu startowego $\mathbf{w}$; przyjęcie współczynnika $T = T_{max}$ nazywanego temperaturą; przyjęcie licznika iteracji $k = 1$.
2. Wyznaczenie wartości funkcji $E(\mathbf{w})$.
3. Wyznaczenie punktu $\mathbf{w}' = \mathbf{w} + \Delta\mathbf{w}$, gdzie $\Delta\mathbf{w}$ jest realizacją zmiennej losowej o rozkładzie normalnym
$$g(\Delta\mathbf{w}, T) = (2\pi T)^{-n/2} \exp\left[-|\Delta\mathbf{w}|^2/(2T)\right]$$
4. Wyznaczenie wartości funkcji $E(\mathbf{w}')$.
5. Podstawienie $\mathbf{w} \leftarrow \mathbf{w}'$ z prawdopodobieństwem danym rozkładem Boltzmanna[2]
$$h(\Delta E, T) = \frac{1}{1 + \exp(\Delta E/(cT))}$$
gdzie $\Delta E = E(\mathbf{w}') - E(\mathbf{w})$, c jest stałą.
6. Zmniejszenie temperatury T; zwykle $T \leftarrow \eta T$, gdzie η jest stałą z przedziału (0,1).
7. $k \leftarrow k + 1$. Jeżeli k osiągnęło przyjętą wartość, to zakończenie obliczeń; w przeciwnym przypadku skok do 3.

Przykładowe kroki algorytmów szukania przypadkowego i symulowanego wyżarzania przedstawione są na rys. 6.6.

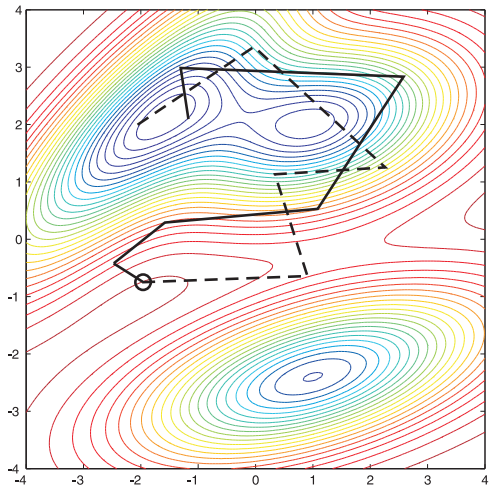
Przykład 6.2

Zastosowano metodę symulowanego wyżarzania do rozwiązania zadania o komiwojażerze. Generacja nowego stanu drogi polegała na zmianie istniejącej drogi poprzez trzy losowo wybierane operacje przedstawione na rys. 6.7. ■

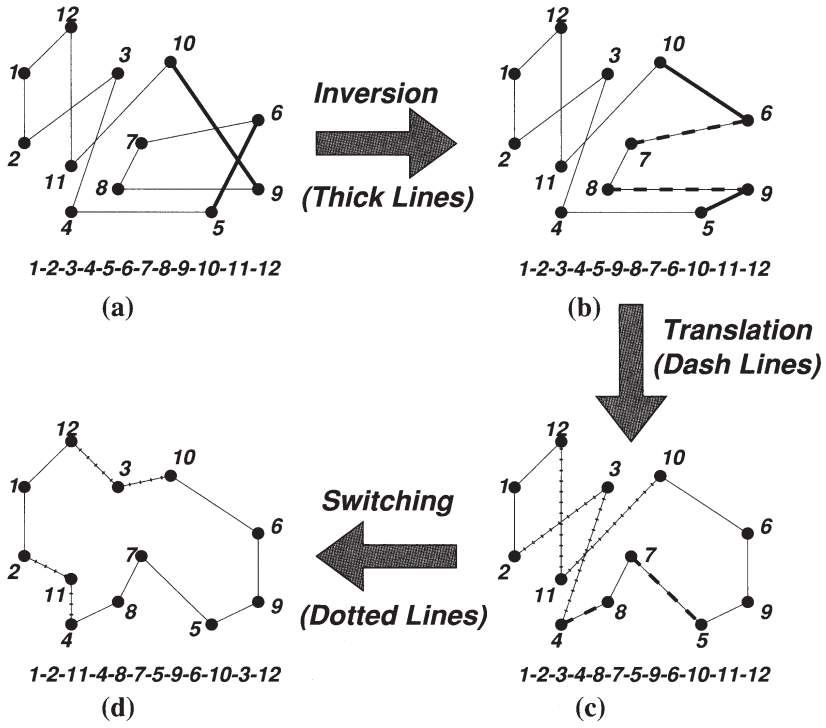
6.3.4. Algorytmy genetyczne (metody ewolucyjne)

Algorytmy ewolucyjne są ważnymi metodami poszukiwania minimum funkcji. Zasada działania algorytmu genetycznego polega na początkowym wyborze pewnej liczby punktów początkowych, a następnie na przeprowadzeniu wielu cykli uczenia. W czasie cyklu oceniana jest jakość punktów, wybierane są punkty najlepsze (selekcja) i tworzone są z nich pary, na podstawie których tworzone są punkty nowe

[2] w większości wariantów algorytmu jeżeli $\Delta E < 0$, to przyjmuje się $h = 1$, tzn., że nowy punkt jest napewno akceptowany.



Rys. 6.6: Przykładowe kolejne punkty wyznaczane w metodach szukania przypadkowego (linia ciągła) i symulowanego wyżarzania (linia przerywana). Zauważmy, że w symulowanym wyżarzaniu kolejne wartości funkcji mogą wzrastać lub maleć, podczas gdy w metodzie szukania przypadkowego mogą tylko maleć



Rys. 6.7: Trzy operacje zmieniające stan drogi w zadaniu o komiwojażerze

(krzyżowanie) oraz dokonywane są losowe zmiany punktów (mutacja). Dzięki tym operacjom w kolejnych cyklach otrzymuje się punkty coraz bliższe minimum, dotyczy to zarówno całej populacji jak i punktu w danej chwili najlepszego. Po przeprowadzeniu pewnej liczby cykli proces uczenia się kończy i z populacji punktów wybiera się punkt najlepszy.

Algorytmy genetyczne są próbą naśladowania procesów biologicznej ewolucji, której działanie w taki sposób przypuszczalnie zachodzi.

Klasyczny algorytm genetyczny

Istnieje wielka ilość odmian i wariantów algorytmów genetycznych. Poniżej podane zostaną zasady działania tzw. *klasycznego algorytmu genetycznego*, a później podane zostaną niektóre możliwe modyfikacje.

W klasycznym algorytmie genetycznym punkt rozwiązujący zadanie reprezentowany jest przez łańcuch binarny, nazywany czasami *chromosomem* o określonej i stałej długości.

Działanie algorytmu genetycznego przebiega w następujących etapach:

1. wybór początkowej populacji łańcuchów,
2. ocena jakości łańcuchów,
3. sprawdzenie czy jakość całej populacji łańcuchów lub jakość łańcucha najlepszego osiągnęła wymaganą wartość. Jeżeli tak to zakończenie algorytmu, jeżeli nie to przejście do etapu następnego.
4. selekcja łańcuchów,
5. krzyżowanie łańcuchów,
6. mutacja

Poszczególne etapy realizowane są następująco:

1. Wybór początkowej populacji łańcuchów można realizować na wiele sposobów ale bardzo często dokonywane jest to losowo, co zapewnia w miarę równomierny rozkład początkowej populacji, a w konsekwencji ułatwia znajdywanie globalnego maksimum jakości (zapobiega zatrzymaniu się w lokalnych maksimach).
2. Ocena jakości może być dokładna, jak np. w przypadku poszukiwania ekstremum funkcji, lub – co ma miejsce bardzo często – przybliżona lub obarczona błędem. Przybliżenie to może wynikać z faktu, że jakość łańcucha może być zależna od parametru przy którym przeprowadzano ocenę jakości, np. od stanu początkowego w przypadku sterowania (patrz omówiony dalej przykład sterowania odwróconym wahadłem). Aby się od wpływu parametru uwolnić należy przeprowadzić obliczenia jakości dla wielu wartości parametrów, a otrzymane wyniki uśrednić.
4. Selekcja polega na wyborze z populacji łańcuchów najlepszych i pozostawienie ich w populacji. Łańcuchy nie wybrane zostają usunięte z systemu. W ten sposób w nowo utworzonej populacji niektóre łańcuchy występują wielokrotnie. Najczęściej wyboru łańcuchów dokonuje się w sposób losowy.

   W metodzie tzw. selekcji ruletkowej przeprowadza się n losowań łańcuchów populacji, gdzie n jest liczebnością populacji, przy czym prawdopodobieństwo wyboru łańcucha w jednym losowaniu jest proporcjonalne do jego jakości. Np. jeżeli jakość czterech łańcuchów wynosiła 2, 3, 5, 10, to prawdopodobieństwa ich wylosowania w jednym kroku wynoszą odpowiednio 2/20, 3/20, 5/20 i 10/20.

W metodzie tzw. selekcji turniejowej losuje się n par łańcuchów i z każdej pary wybiera się łańcuch lepszy. Metoda ta ma duże znaczenie, gdy ocena jakości łańcucha jest kosztownym procesem wieloetapowym (np. jak w omawianym w p. 2 przypadku zależności jakości od wartości parametru), a ostateczna ocena jest sumą ocen w poszczególnych etapach. W takim przypadku możemy obliczać na bieżąco jakości pary łańcuchów i gdy różnica jakości przekroczy pewien ustalony próg przerywać proces obliczeń i wybierać łańcuch do tej pory lepszy. Może to znacznie zmniejszyć ilość obliczeń, przy dopuszczalnie małym prawdopodobieństwie wybrania łańcucha gorszego.

5. Krzyżowanie łańcuchów realizowane jest w następujących etapach: (a) losowy wybór $n_k \leq n$ łańcuchów, (b) połączenie wybranych łańcuchów w losowo dobrane pary, (c) dla każdej pary losowanie numeru bitu w łańcuchu, b_k, (d) wytworzenie z każdej pary dwóch łańcuchów potomnych, przy czym pierwszy łańcuch potomny otrzymuje pierwsze b_k bitów od pierwszego łańcucha rodzicielskiego, a pozostałe bity od drugiego, zaś drugi łańcuch potomny otrzymuje pierwsze b_k bitów od drugiego łańcucha rodzicielskiego, a pozostałe bity od pierwszego. Np. z pary łańcuchów rodzicielskich 101101011111 i 010101111101 dla $b_k = 5$ otrzymuje się łańcuchy potomne 101101111101 i 010101011111.
6. Mutacja polega na losowych zmianach bitów w łańcuchach. W czasie mutacji losowana jest zamiana poszczególnych bitów wszystkich łańcuchów z prawdopodobieństwem p_m.

Stosowane w praktyce algorytmy genetyczne często znacznie różnią się od przedstawionego powyżej klasycznego algorytmu genetycznego. Podstawową modyfikacją jest przyjęcie reprezentacji punktów innych niż łańcuchy binarne – np. reprezentacja za pomocą liczb rzeczywistych. Pociąga to za sobą zmianę sposobu krzyżowania punktów. Inną modyfikacją jest kumulacja oceny jakości punktu. Ma to miejsce wtedy, gdy nie potrafimy dokładnie ocenić jakości punktu. Np. nie możemy ocenić dokładnie jakości predyktora przebiegu czasowego na podstawie jednego lub kilku predykcji. Przy kumulacji oceny jakości możemy posługiwać się np. wzorem

$$c_i = \frac{1}{2}(c_{i-1} + f_i)$$

gdzie f_i oceną punktu w kroku i, a c_i jest oceną skumulowaną. Przyjęcie powyższego wzoru zapewnia malejący wpływ poprzednich ocen jakości. Stosując ten wzór[3] mamy

$$c_i = \frac{1}{2}f_i + \frac{1}{4}f_{i-1} + \ldots + \frac{1}{2^{k+1}}f_{i-k} + \ldots$$

Algorytmy genetyczne mają zdolność wychodzenia z lokalnych ekstremów, a tym samym zdolność do znajdywania ekstremów globalnych. Ponadto algorytmy genetyczne działające przez cały czas mają zdolność do nadążania za zmieniającym się środowiskiem. Zasady działania algorytmu genetycznego też mogą zmieniać się według algorytmu genetycznego!

[3]Zastosowanie analogicznego wzoru, w którym współczynnik 1/2 zastąpiony zostałby zmiennym współczynnikiem $1/i$, prowadziłoby do skumulowanej oceny będącej średnią arytmetyczną wszystkich cząstkowych ocen.

Własności algorytmów genetycznych

Wyprowadzimy tu kilka ogólnych własności zachowania się klasycznego algorytmu genetycznego.

Schemat jest to łańcuch o długości k zbudowany z elementów $\{0, 1, *\}$, gdzie $*$ reprezentuje pozycje nieokreślone. Np. $0{*}{*}1{*}1$. Schemat S reprezentuje łańcuchy składające się z elementów 0, 1, gdy ich długość wynosi k i są z nim identyczne na pozycjach określonych.

Przykład 6.3

Schemat $1{*}0{*}$ reprezentuje łańcuchy 1000, 1001, 1100 i 1101. Schemat 01001 reprezentuje tylko jeden łańcuch, a schemat $*****$ reprezentuje wszystkie łańcuchy o długości 5.

Łańcuch 010 reprezentowany jest przez schematy: 010, $*10$, $0{*}0$, $01{*}$, $**0$, $*1{*}$, $0{**}$, $***$. ■

Dowolny schemat S zawierający n symboli $*$ reprezentuje 2^n łańcuchy.

Dowolny łańcuch o długości k reprezentowany jest przez 2^k schematów.

Definicja 6.1 *Licznością schematu S jest liczba $L(S)$ równa liczbie pozycji określonych w schemacie. Np. $S = (*1*0*11)$, $L(S) = 4$.*

Definicja 6.2 *Długością schematu S jest liczba $D(S)$ równa odległości pomiędzy pierwszą i ostatnią określoną pozycją w tym schemacie. Np. $S = (*1*0{**})$, $D(S) = 4 - 2 = 2$.*

Przyjmijmy oznaczenia:

ch	–	łańcuch
P	–	populacja łańcuchów
k	–	długość łańcucha
$l(P)$	–	liczba łańcuchów w populacji P
$F(ch_i)$	–	kryterium jakości łańcucha ch_i
t	–	numer cyklu działania algorytmu genetycznego
$I(P, S, t)$	–	liczba łańcuchów w populacji P reprezentowana przez schemat S w cyklu t
p_m	–	prawdopodobieństwo mutacji

Definicja 6.3 *Przystosowaniem schematu S w cyklu t nazywamy liczbę*

$$F(S,t) = \frac{\sum\limits_{ch_i \in S} F(ch_i)}{I(P,S,t)}$$

Definicja 6.4 *Przystosowaniem populacji P w cyklu t nazywamy liczbę*

$$F(P,t) = \frac{\sum\limits_{i=1}^{l(P)} F(ch_i)}{l(P)}$$

Wyznaczymy teraz liczbę łańcuchów reprezentowanych przez schemat S w cyklu $t+1$ zależną od tej liczby w cyklu t, czyli

$$I(P,S,t+1) = f(I(P,S,t))$$

Etap selekcji

Prawdopodobieństwo wylosowania w trakcie selekcji łańcucha reprezentowanego przez S wynosi

$$p = \frac{\sum\limits_{ch_i \in S} F(ch_i)}{\sum\limits_{i=1}^{l(P)} F(ch_i)} = \frac{F(S,t)I(P,S,t)}{F(P,t)l(P)}$$

Sposób losowania łańcuchów na etapie selekcji jest schematem Bernoulliego (z prawdopodobieństwem p wylosowania łańcucha reprezentowanego przez S (sukces) i prawdopodobieństwem $1-p$ wylosowania łańcucha nie reprezentowanego przez S). Ponieważ wartość średnia liczby sukcesów w schemacie Bernoulliego wynosi $p \times$ *liczba prób* mamy

$$\mathcal{E}I(P,S,t+1) = I(P,S,t)\frac{F(S,t)}{F(P,t)}$$

Etap krzyżowania

W czasie krzyżowania pary łańcuchów, z których jeden należy, a drugi nie należy do schematu S, prawdopodobieństwo zamiany łańcucha należącego do schematu S na nie należący do niego wynosi (im większa jest długość schematu, $D(S)$, tym większa jest szansa, że dany łańcuch przestanie być przez niego reprezentowany)

$$p_{zk}(S) = \frac{D(S)}{k-1}$$

Etap mutacji

Prawdopodobieństwo, że podczas mutacji łańcuch nie przestanie należeć do schematu S wynosi

$$p_{pm} = (1-p_m)^{L(s)} \approx 1 - L(S)p_m$$

przy założeniu $p_m \ll 1$.

Uwzględniając te prawdopodobieństwa otrzymujemy ostatecznie

$$\mathcal{E}I(P,S,t+1) = I(P,S,t)\frac{F(S,t)}{F(P,t)}\left[1 - \frac{D(S)}{k-1}\right][1 - L(S)p_m]$$

Przykład zastosowania algorytmu genetycznego – dylemat więźnia

Za pomocą algorytmu genetycznego poszukiwano najlepszej strategii w dwuosobowej grze nazywanej *dylematem więźnia*. Posiada ona tablicę wypłat przedstawioną na rys. 6.8. Tablicę można interpretować jako wyniki dwóch możliwych działań wieźniów. Mogą oni zdradzać (tzn. współpracować z władzami) lub współpracować ze sobą.

	Zdrada	Współpraca
Zdrada	1	5
Współpraca	0	3

Rys. 6.8: Tablica wypłat w dwuosobowej grze *dylemat więźnia*

Przyjęto, że posunięcie gracza będzie opierało się o poprzednie 3 gry (sekwencja 6 decyzji). Strategia gracza reprezentowana może być więc przez łańcuch 64 bitowy przedstawiający posunięcia gracza będące reakcją na wszytkie kombinacje decyzji mogące zaistnieć w poprzednich 3 grach. Dodatkowo dla wyznaczania posunięcia na początku gry, gdy jeszcze nie ma poprzednich gier, potrzebny jest 6 bitowy łańcuch zawierający hipotetyczny zapis 3 ostatnich gier. Ostatecznie przyjęto łańcuch 70 bitowy reprezentujący kompletną strategię gracza.

Prowadzono uczenie algorytmem genetycznym, przy czym jakość strategii określano na podstawie wyników gier algorytmów między sobą. Otrzymano bardzo rozsądne strategie. np. (kolejność ja, on):

$$\begin{array}{cccc} (WW) & (WW) & (WW) & \rightarrow \; W \\ (WW) & (WW) & (WZ) & \rightarrow \; Z \\ (WZ) & (ZW) & (WW) & \rightarrow \; W \\ (ZZ) & (ZZ) & (ZZ) & \rightarrow \; Z \end{array}$$

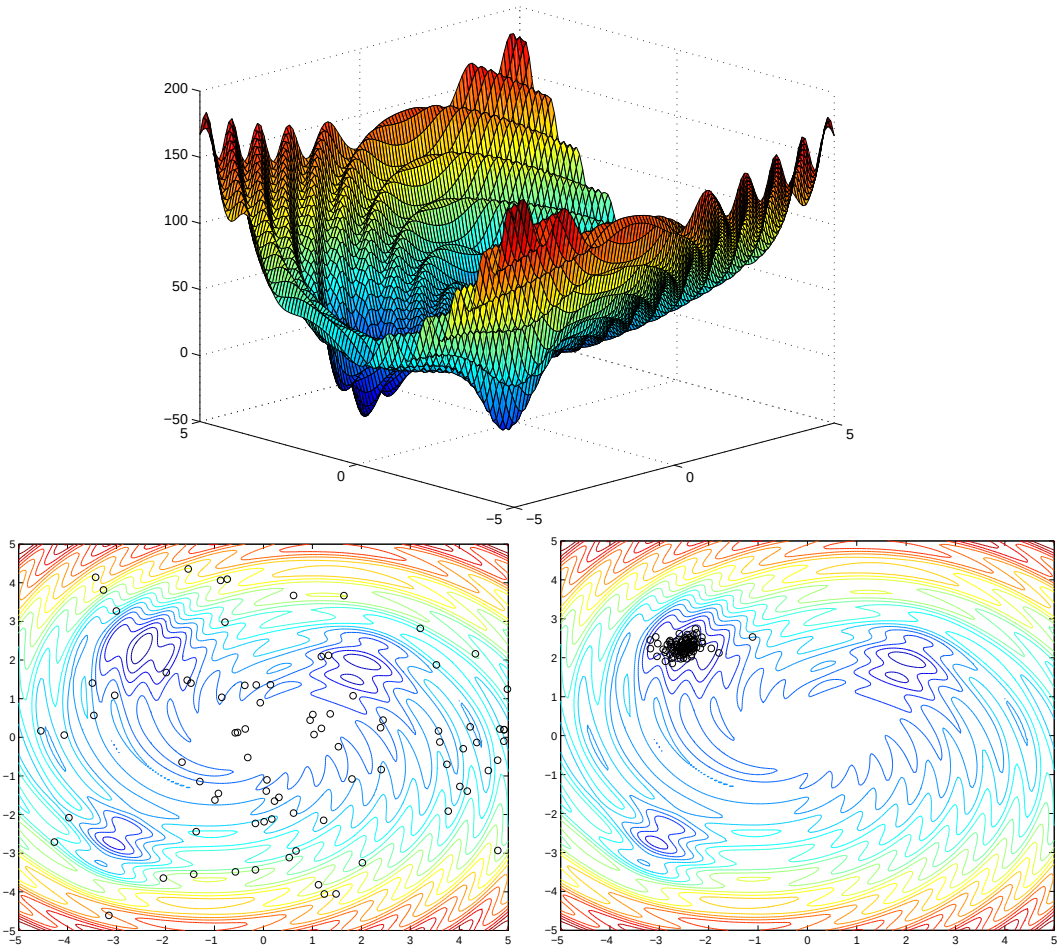
Przykład zastosowania algorytmu genetycznego – predykcja

Jednym z typowych zastosowań algorytmów genetycznych jest predykcja czyli przewidywanie przyszłych stanów jakiegoś procesu, na podstawie określonej liczby stanów dotychczasowych i ewentualnie innych parametrów mających wpływ na stan procesu.

W tym zastosowaniu algorytmem predykcji może być wielomian, którego zmienne reprezentują wartości przeszłych stanów. Populacja algorytmów jest zbiorem takich wielomianów o różniących się wartościach współczynników. W trakcie trwania algorytmu genetycznego dobierane są wartości współczynników wielomianów. Jako kryterium jakości algorytmów przyjmuje się różnice pomiędzy wartością przewidywaną, a wartością rzeczywiście zachodzącą. Uczenie można przeprowadzać *off line* na danych zgromadzonych wcześniej.

Przykład zastosowania algorytmu genetycznego – wyznaczenie minimum funkcji

Na rys. 6.9 przedstawiono początkową i końcową populacje punktów przy poszukiwaniu minimum funkcji.



Rys. 6.9: Przykład zastosowania algorytmu ewolucyjnego do poszukiwania minimum funkcji 2 argumentów: (a) wykres funkcji, (b) populacja początkowa, (c) populacja po 77 epokach działania algorytmu

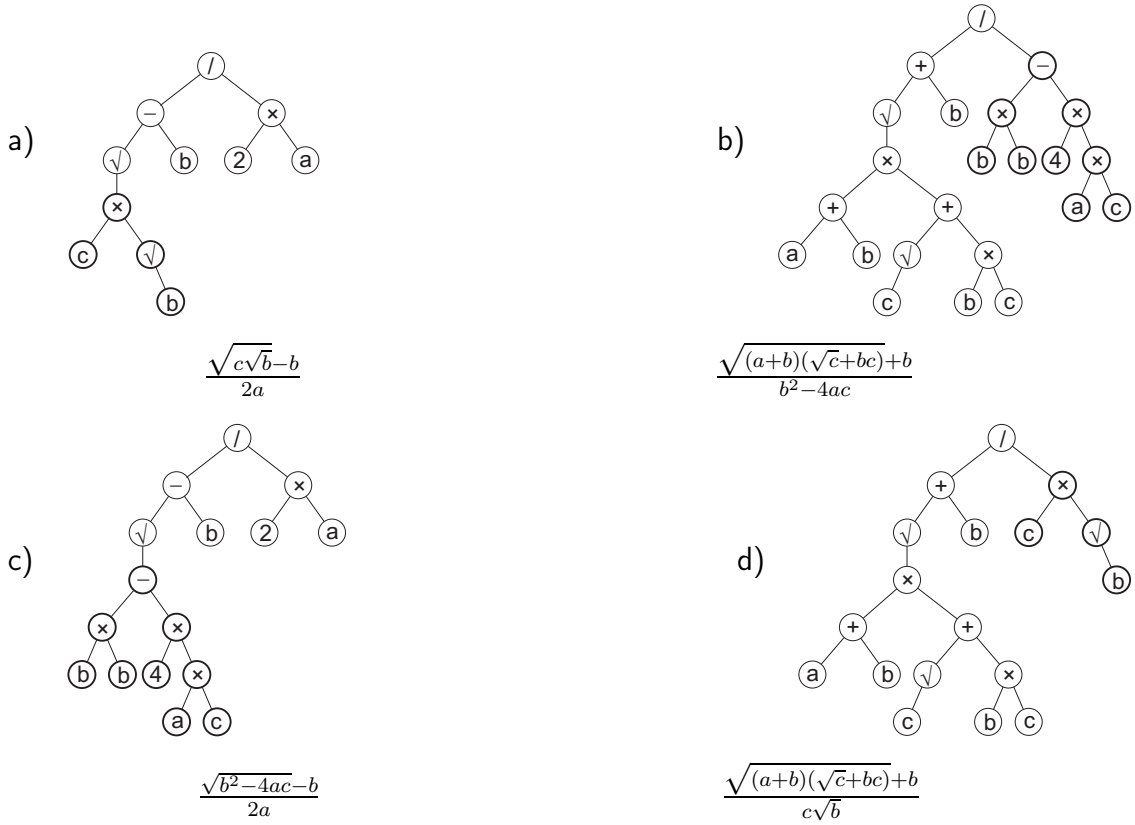
6.3.5. Programowanie genetyczne

Programowanie genetyczne jest odmianą algorytmu genetycznego, w którym ewolucji podlegają programy. Przykłady operacji krzyżowania i mutacji przedstawiono na rysunkach 6.10, 6.11, 6.12.

6.3.6. Algorytmy roju

Omówione tu będą algorytmy poszukiwania ekstremum funkcji nazywane algorytmami roju (ang. PSO – particle swarm optimization). W algorytmach tych podobnie jak w algorytmach ewolucyjnych ekstremum funkcji jest poszukiwane przez populację punktów.

Metoda roju została zaproponowana przez Kennedy'ego i Eberhardta w roku 1995 (patrz rys. 6.13). Tak jak algorytmy ewolucyjne naśladują podstawowe operacje ewolucji biologicznej, tak metody roju naśladują zachowanie się jednostek w społecznościach. Jednostka wybiera algorytm swojego postępowania jako wypadkową



Rys. 6.10: Przykładowa operacja krzyżowania w programowaniu genetycznym, gdy programy krzyżujące się są różne: (a) i (b) programy przed krzyżowaniem, (c) i (d) programy po krzyżowaniu. Liniami pogrubionymi narysowano fragmenty drzew podlegających zamianie

swojego dotychczas najlepszego postępowania, postępowania jednostki w populacji odnoszącej największe sukcesy oraz pewnej swojej dynamiki zmian.

W metodzie roju populacja punktów (cząstek) przemieszcza się w przestrzeni. W każdym kroku iteracyjnym cząstka przemieszcza się w kierunku określonym przez trzy wektory: wektor wskazujący najlepsze dotychczasowe położenie cząstki, tzn. takie w którym wartość funkcji była najmniejsza, wektor wskazujący najlepsze dotychczasowe położenie cząstek sąsiednich i chwilowy wektor swojej prędkości. W zależności od definicji sąsiedztwa cząstek wyróżnia się dwa podstawowe warianty metod: metoda *najlepszy globalnie* oraz metoda *najlepszy lokalnie*.

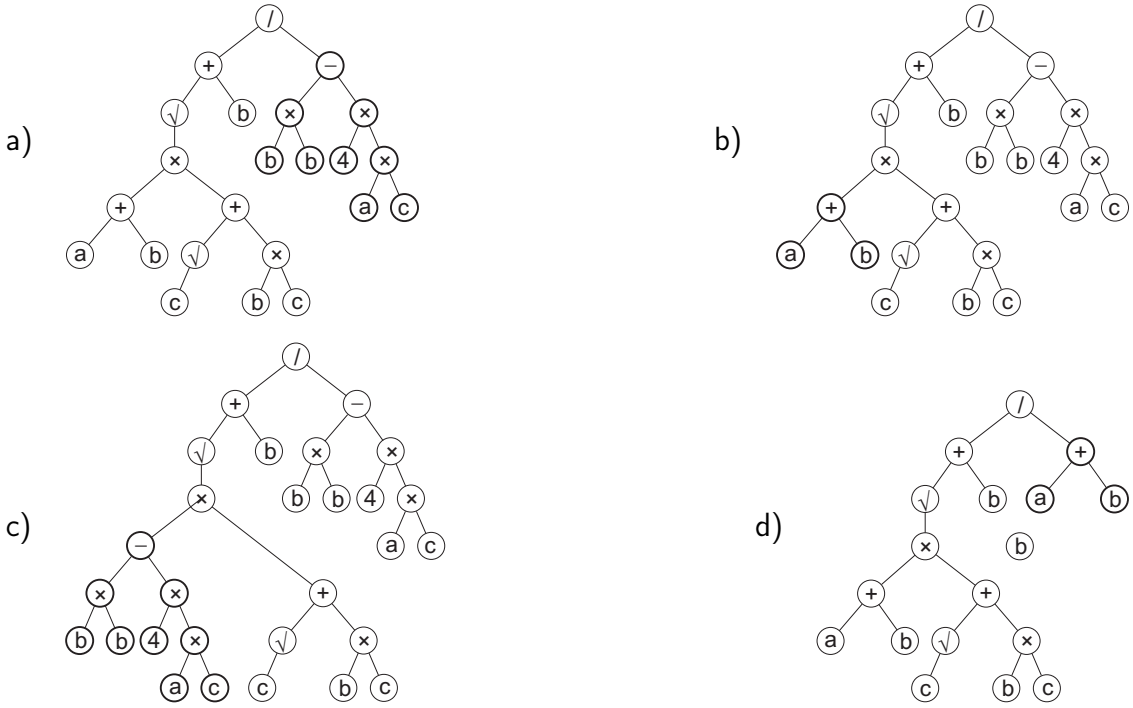
Wariant „najlepszy globalnie"

W wariancie *najlepszy globalnie* (ang. global best) położenie cząstki w kroku iteracyjnym $t+1$ zmienia się według wzoru

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \mathbf{v}_i(t+1)$$

gdzie $\mathbf{w}_i(t)$ oznacza pozycję cząstki i w kroku t, a $\mathbf{v}_i(t+1)$ oznacza prędkość cząstki obliczaną według wzoru

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{w}_i(t)] + c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{w}_i(t)] \qquad (6.3)$$



Rys. 6.11: Przykładowa operacja krzyżowania w programowaniu genetycznym, gdy programy krzyżujące się są identyczne: (a) i (b) programy przed krzyżowaniem, (c) i (d) programy po krzyżowaniu. Liniami pogrubionymi narysowano fragmenty drzew podlegających zamianie

gdzie c_1 i c_2 są dodatnimi stałymi, $\mathbf{r}_1(t)$ and $\mathbf{r}_2(t)$ są macierzami diagonalnymi o losowych elementach wybranych z rozkładu równomiernego w przedziale [0,1], $\mathbf{y}_i(t)$ jest najlepszą dotychczasową pozycją cząstki (najlepszą osobistą (ang. personal best)) spośród wszystkich pozycji osiąganych od pierwszego kroku, a $\hat{\mathbf{y}}(t)$ jest najlepszą pozycją spośród najlepszych osobistych pozycji wszystkich cząstek (najlepszą globalną (ang. global best)).

W literaturze dotyczącej algorytmów roju powyższe składowe nazywane są następująco:

- poprzednia prędkość cząstkii, $\mathbf{v}_i$ – składową momentu lub inercji,
- $c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{x}_i(t)]$ – składową kognitywną,
- $c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{x}_i(t)]$ – składową socjalną.

Na rys. 6.14 zilustrowano wyznaczanie nowej pozycji cząstki.

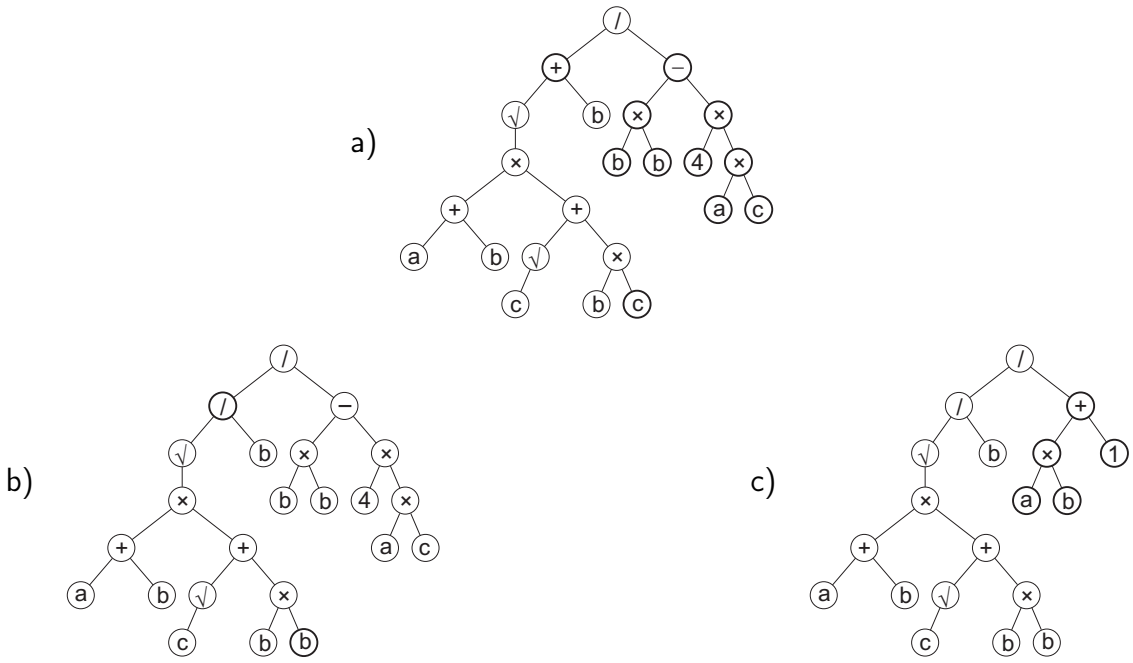
Najlepsza pozycja osobista jest wyznaczana z wzoru

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{jeżeli } E(\mathbf{w}_i(t+1)) \geq E(\mathbf{y}_i(t)) \\ \mathbf{w}_i(t+1) & \text{w przeciwnym razie} \end{cases}$$

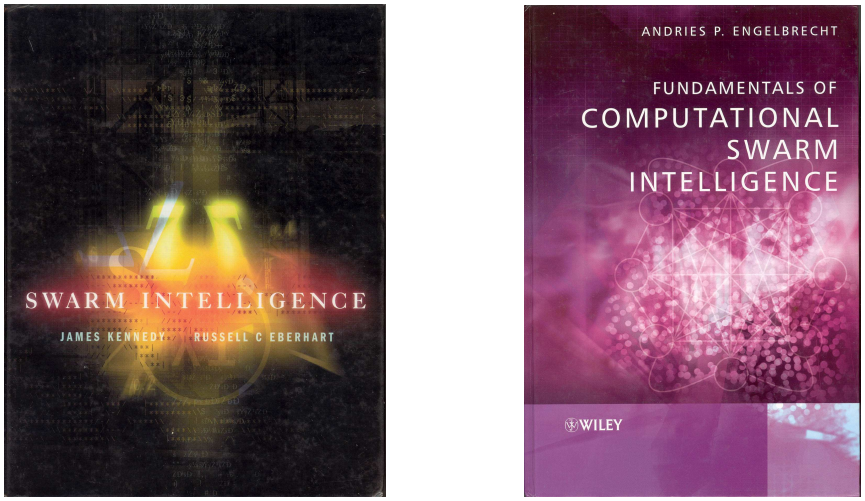
gdzie E jest minimalizowaną funkcją.

Najlepsza pozycja globalna obliczana jest z wzoru

$$\hat{\mathbf{y}}(t) = \arg \min_{\mathbf{y}_i, i=1,\ldots n_s} \{E(\mathbf{y}_i(t))\}$$



Rys. 6.12: Przykładowe operacje mutacji w programowaniu genetycznym: (a) program początkowy, (b) program po zastąpieniu wybranego symbolu terminalnego przez inny symbol terminalny i wybranej operacji przez inną operację, (c) program po zastąpieniu jednej z gałęzi drzewa przez inną gałąź

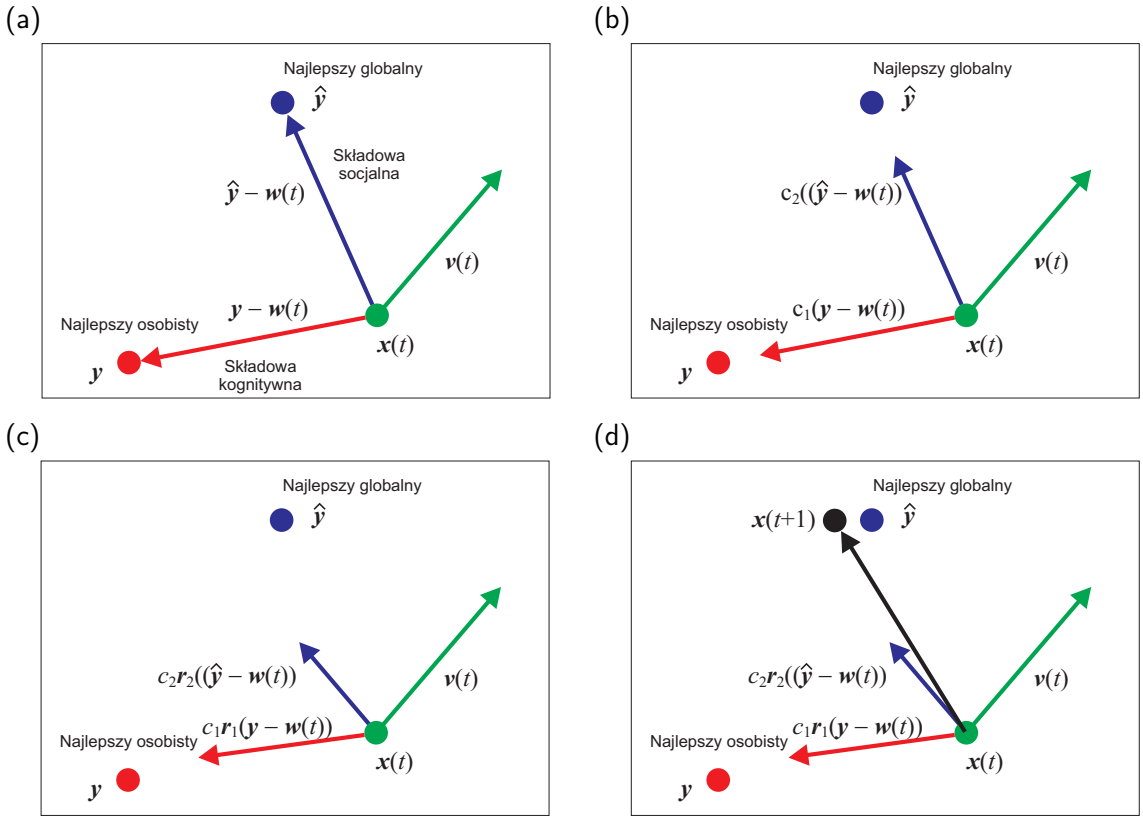


Rys. 6.13: Okładki książki autorów algorytmu roju i obszernej monografii dotyczącej algorytmów roju i algorytmów mrówkowych

gdzie n_s jest całkowitą liczbą cząstek.

W niektórych wariantach metody pozycja najlepsza globalnie wyznaczana jest spośród najlepszych pozycji w aktualnym roju.

$$\hat{\mathbf{y}}(t) = \arg \min_{\mathbf{w}_i, i=1,\ldots n_s} \{E(\mathbf{w}_i(t))\}$$



Rys. 6.14: Etapy wyznaczania nowego położenia czastki: (a) dotychczasowe położenie cząstki, $\mathbf{x}(t)$, oraz wektory wskazujące: aktualną prędkość cząstki, $\mathbf{v}(t)$, najlepszą dotychczasową pozycję cząstki (najlepszy osobisty), $\mathbf{y}-\mathbf{w}(t)$, oraz najlepszą dotychczasową pozycję spośród wszystkich cząstek (najlepszy globalny), $\hat{\mathbf{y}}(t)-\mathbf{w}(t)$, (b) wektory po przemnożeniu przez stałe, c_i, (c) wektory po przemnożeniu przez liczby losowe oraz (d) wypadkowy wektor przesuniecia, $\mathbf{x}(t+1)$, będący sumą trzech wektorów

Wariant „najlepszy lokalnie"

W wariancie *najlepszy lokalnie* (ang. local best) każdej cząstce przypisuje się pewne inne cząstki, które nazywane są sąsiednimi. Teraz, zamiast pozycji najlepszej globalnie $\hat{\mathbf{y}}(t)$, bierze się pozycję najlepszą lokalnie $\hat{\mathbf{y}}_i(t)$, tzn. najlepszą pozycję osobistą spośród cząstek sąsiednich. Czyli mamy

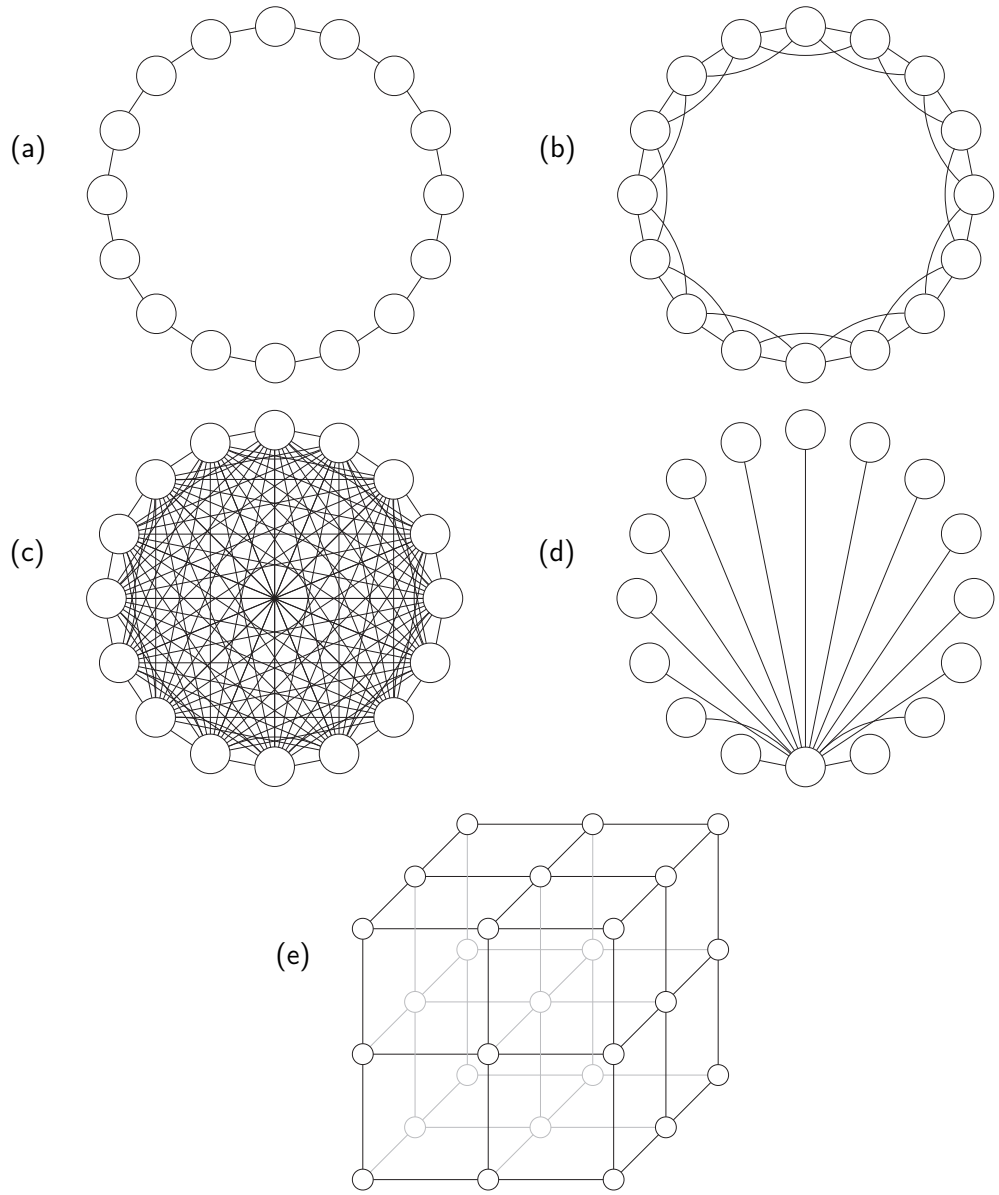
$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{w}_i(t)] + c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}_i(t) - \mathbf{w}_i(t)]$$

zaś pozycja najlepsza lokalnie jest obliczana ze wzoru

$$\hat{\mathbf{y}}_i(t) = \arg \min_{\hat{\mathbf{y}}_j, j \in \mathcal{N}_i} \{f(\hat{\mathbf{y}}_j(t))\}$$

gdzie $\mathcal{N}_i$ jest zbiorem cząstek sąsiednich względem cząstki i (wraz z cząstką i).

Na rysunkach 6.15 przedstawiono typowe sąsiedztwa cząstek. Nie ma ogólnie najlepszej topologii sąsiedztwa dla wszystkich problemów. Na ogół ze wzgledu na większą liczbę połączeń duże sąsiedztwa zapewniają szybszą zbieżność do ekstremum, natomiast cząstki są bardziej skupione niż w przypadku sąsiedztw małych.



Rys. 6.15: Typowe sąsiedztwa cząstek: (a),(b),(c) struktury pierścieniowe, (d) struktura kołowa i (e) tzw. struktura von Neumana

Odwrotnie, sąsiedztwa małe zapewniają większy rozrzut położeń cząstek, tym samym są mniej podatne na ugrzęźnięcia w lokalnych ekstremach

Sąsiedztwa cząstek nie są tworzone według odległości geometrycznych, mogą być tworzone np. według numerów czastek lub przypadkowo. Raz ustalone sąsiedztwa pozostają niezmienione w całym czasie działania algorytmu, niezależnie od aktualnego położenia cząstek.

Wartości początkowe

Położenia początkowe cząstek wybiera się najczęściej z rozkładu równomiernego, $(U(0,1))$, w przestrzeni poszukiwań.

Prędkości początkowe

$$\mathbf{v}_i = 0$$

Początkowe najlepsze pozycje osobiste

$$\mathbf{y}_i(0) = \mathbf{w}_i(0)$$

Ograniczenie prędkości

Podczas działania algorytmu PSO prędkości cząstek szybko wzrastają do bardzo dużych wartości, szczególnie cząstek odległych od najlepszych globalnych lub osobistych pozycji. W krokach iteracyjnych poprawki pozycji są coraz większe i cząstki wychodzą poza obszar poszukiwań. Aby temu zapobiec ogranicza się prędkości cząstek.

Jednym ze sposobów ograniczenia jest przyjęcie zasady:

$$v_{ij}^{'}(t+1) = \begin{cases} v_{ij}(t+1) & \text{if } v_{ij}(t+1) < V_{max,j} \\ V_{max,j} & \text{w przeciwnym razie} \end{cases}$$

gdzie $V_{max,j}$ oznacza maksymalną dopuszczalną prędkość w kierunku j. Zazwyczaj przyjmuje się

$$V_{max,j} = \delta(w_{max,j} - w_{min,j})$$

gdzie $w_{max,j}$ and $w_{min,j}$ są granicami obszaru poszukiwań w kierunku j, a $\delta \in (0, 1]$ jest stałą dobieraną w zależności od problemu.

Maksymalna prędkość może zostać zmniejszona, gdy w ciągu τ kolejnych kroków nie poprawiła się pozycja najlepsza globalnie

$$V_{max,j}(t+1) = \begin{cases} \beta V_{max,j}(t) & \text{if } f(\hat{\mathbf{y}}(t)) \geq f(\hat{\mathbf{y}}(t-t^{'})) \\ & \forall t^{'} = 1, \ldots, \tau \\ V_{max,j}(t) & \text{w przeciwnym razie} \end{cases}$$

gdzie $\beta \in (0, 1]$ jest stałą lub zmienianym parametrem.

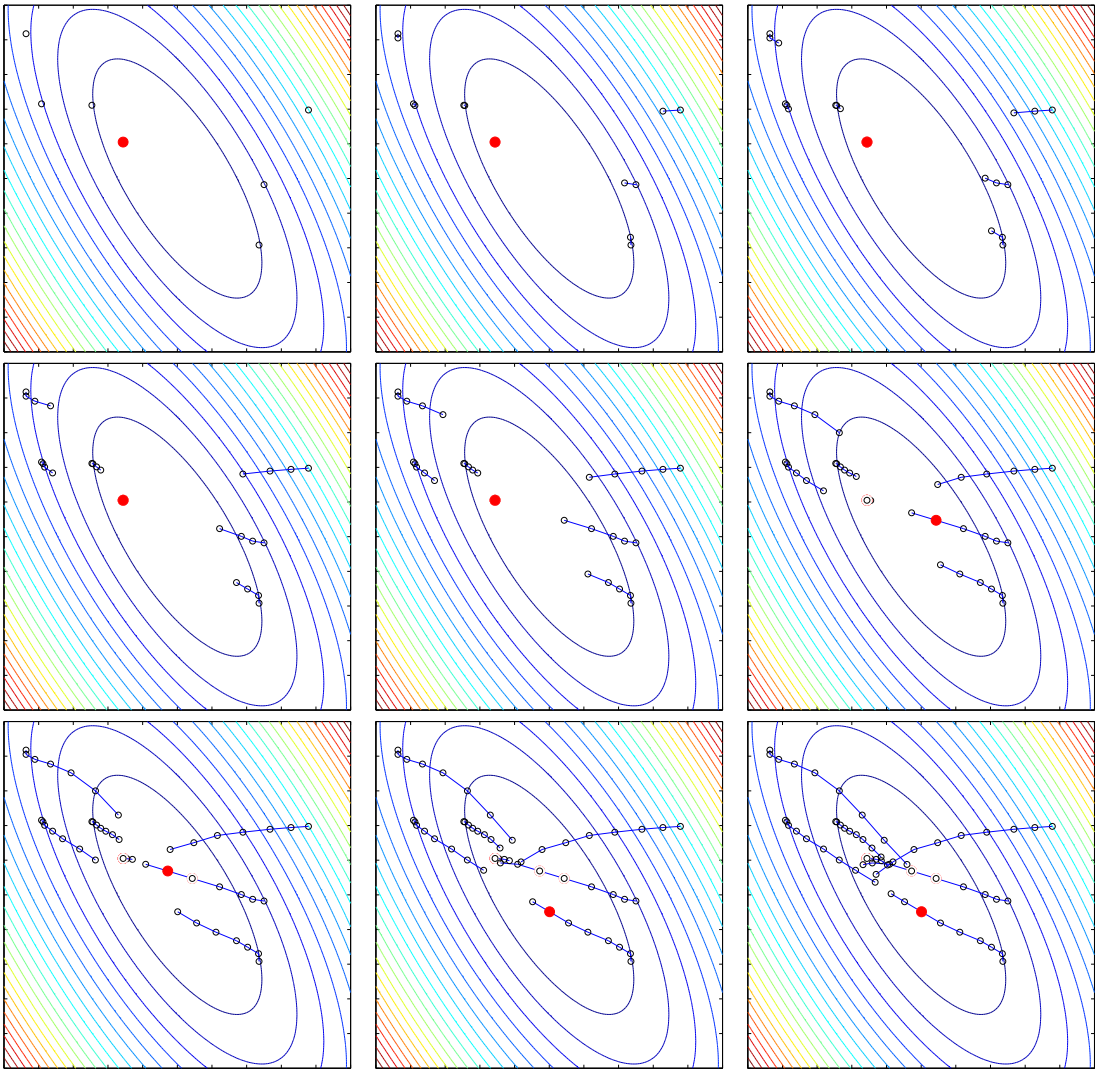
W innym postępowaniu prędkość może być tłumiona w każdym kroku zgodnie ze zmodyfikowanym wzorem (6.3)

$$\mathbf{v}_i(t+1) = \alpha \mathbf{v}_i(t) + c_1 \mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{w}_i(t)] + c_2 \mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{w}_i(t)]$$

gdzie α jest stałym parametrem.

6.4. Sztuczne sieci neuronowe

W procesie uczenia wykorzystuje się kilka typowych struktur. Jedną z nich są *sztuczne sieci neuronowe*. Ich uniwersalność spowodowała ich szerokie zastosowania jako systemów uczacych się.



Rys. 6.16: Ilustracja działania algorytmu PSO. Trajektorie ruch cząstek poszukujacych minimum funkcji. Pierwsze 8 kroków algorytmu. Czarne kółko oznacza punkt globalnie najlepszy

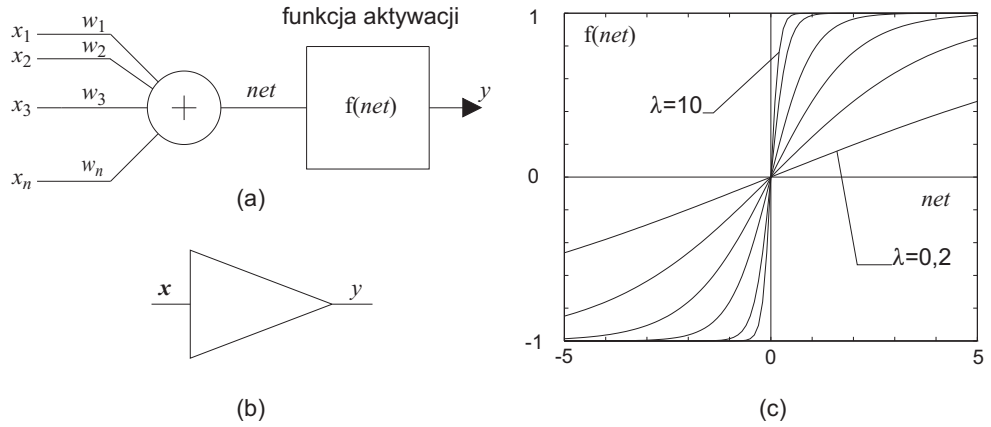
6.4.1. Struktury sieci neuronowych

Sieci neuronowe składają się z dużej liczby wzajemnie połączonych układów zwanych *sztucznymi neuronami*, które realizują proste przekształcenia sygnałów wejściowych w wyjściowe.

Sztuczny neuron jest układem statycznym o schemacie blokowym i symbolu przedstawionym na rys. 6.17a,b. Sygnał wyjściowy neuronu dany jest zależnością:

$$y = f(net), \qquad net = \mathbf{w}^t\mathbf{x} = \left(\sum_{i=1}^{n} w_i x_i\right) \tag{6.4}$$

gdzie sygnał net będący ważoną sumą sygnałów wejściowych nazywany jest *pobudzeniem łącznym*, funkcja $f(net)$ oddziałująca na pobudzenie łączne nazywa się *funkcją*



Rys. 6.17: Sztuczny neuron: (a) schemat blokowy, (b) symbol i (c) bipolarna funkcja aktywacji dla różnych wartości współczynnika λ

aktywacji, $\mathbf{w}$ jest *wektorem wag* zdefiniowanym jako

$$\mathbf{w} \stackrel{\text{def}}{=} [\, w_1 \;\; w_2 \;\; \cdots \;\; w_n \,]^t$$

$\mathbf{x}$ zaś jest wektorem wejściowym:

$$\mathbf{x} \stackrel{\text{def}}{=} [\, x_1 \;\; x_2 \;\; \cdots \;\; x_n \,]^t$$

Typowymi funkcjami aktywacji są

$$f(net) = \frac{2}{1 + \exp(-\lambda net)} - 1, \quad \lambda > 0 \tag{6.5}$$

oraz

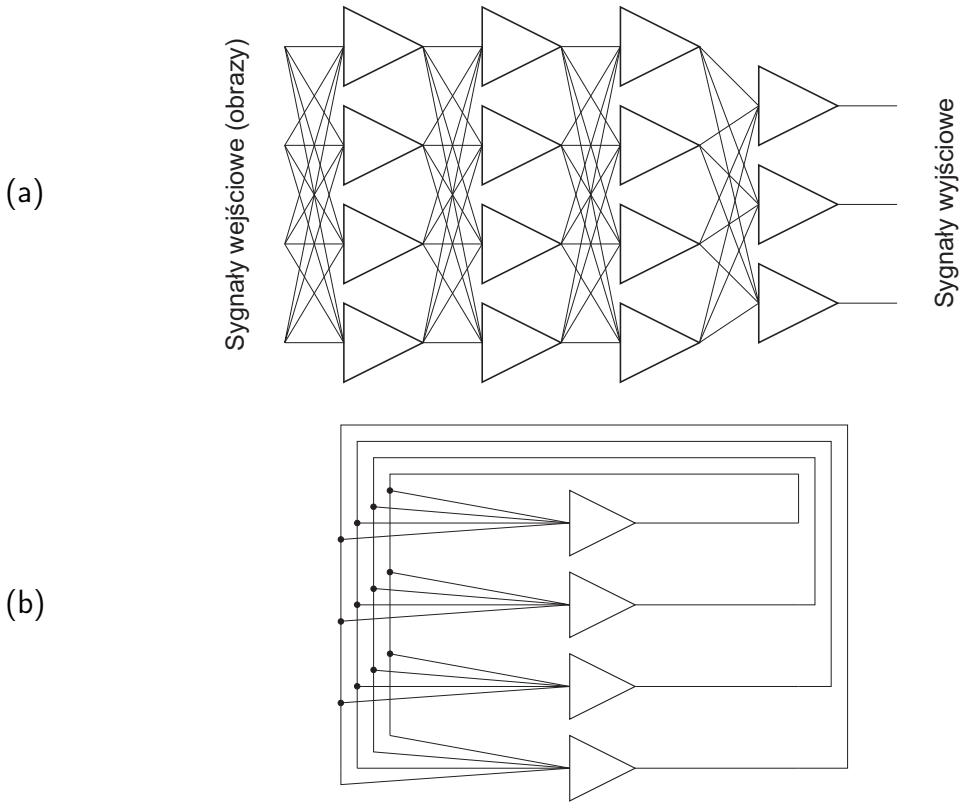
$$f(net) = \mathrm{sgn}(net) = \begin{cases} +1, & \text{gdy } net \geq 0, \\ -1, & \text{gdy } net < 0. \end{cases} \tag{6.6}$$

zwane odpowiednio ciągłą i dyskretną funkcją *bipolarną*. Ciągła funkcja aktywacji dla różnych wartości parametru λ pokazana jest na rys. 6.17c. Zauważmy, że gdy $\lambda \to \infty$, wówczas $f(net)$ dąży do funkcji $\mathrm{sgn}(net)$ zdefiniowanej wzorem (6.6). Często też stosuje się funkcję aktywacji *unipolarną*, która przyjmuje wartości z przedziału $(0, +1)$.

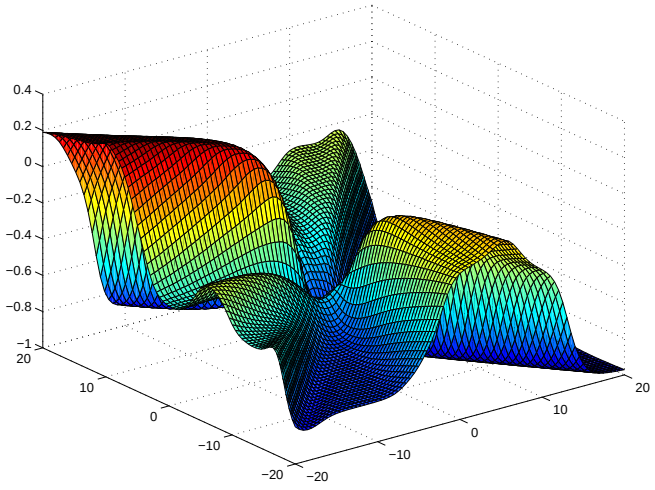
Neurony łączone są w większe struktury, spośród których można wyróżnić dwie podstawowe – sieci warstwowe bez sprzężeń zwrotnych oraz sieci ze sprzężeniami zwrotnymi (rys. 6.18).

Sieci warstwowe bez sprzężeń zwrotnych są układami statycznymi realizującymi złożone nieliniowe odwzorowanie, którego kształt zależy od przyjętej funkcji aktywacji, stałej λ i wektorów wag $\mathbf{w}$ wszystkich neuronów sieci. Sygnały wyjściowe tych układów zależą wyłącznie od aktualnych sygnałów wejściowych i reagują na zmianę sygnałów wejściowych z opóźnieniem wynikającym z opóźnień wnoszonych przez czas reakcji poszczególnych neuronów. Przykładowa funkcja realizowana przez sieć warstwową przedstawiona jest na rys. 6.19.

Sieci ze sprzężeniami zwrotnymi są nieliniowymi układami dynamicznymi. Wprowadzone w zadany stan początkowy, dzięki sprzężeniom zwrotnym zmieniają swój stan, aż do osiągnięcia stanu równowagi.



Rys. 6.18: Podstawowe struktury sieci neuronowych: (a) sieć warstwowa bez sprzężeń zwrotnych i (b) sieć ze sprzężeniami zwrotnymi



Rys. 6.19: Przykładowa funkcja realizowana przez dwuwejściową warstwową sieć neuronową

Potencjalny zakres zastosowań sieci warstwowych jest nadzwyczaj szeroki. Mogą one być stosowane we wszelkich zadaniach klasyfikacji, rozpoznawania, diagnostyki, monitorowania i sterowania. Podobnie, zakres zastosowania sieci ze sprzężeniem zwrotnym obejmuje zadania, gdzie potrzebny jest określony złożony, dynamiczny układ nieliniowy. Specyficznymi zastosowaniami sieci ze sprzężeniami zwrotnymi są

asocjacyjne pamięci rekurencyjne, które po pobudzeniu zmierzają do jednego ze swoich stanów równowagi odpowiadającemu jednej z zapamiętanych w nich wartości. Inną grupą zastosowań sieci ze sprzężeniami zwrotnymi są rozwiązania zadań optymalizacji (np. zadanie o komiwojażerze), gdzie sieć o odpowiednio dobranych wagach zmierza do stanu równowagi, którym jest poszukiwane rozwiązanie.

Sieci neuronowe mogą być realizowane w postaci specjalizowanych układów elektronicznych lub symulowane przy pomocy uniwersalnych komputerów.

Przygotowanie sieci do rozwiązywania określonych zadań odbywa się na drodze jej *uczenia*, w którym wyznaczone zostają wagi neuronów sieci powodujące jej oczekiwane działanie. W przypadku uczenia z nauczycielem stosować można dowolną z metod uczenia omówionych w tym rozdziale, przy czym najczęściej stosuje się metodę gradientową najszybszego spadku.

Wagi sieci ze sprzężeniami zwrotnymi zwykle oblicza się na podstawie odpowiednich wzorów, a nie są one zmieniane w procesie iteracyjnym. Dzieje się tak w przypadku pamięci asocjacyjnych czy sieci rozwiązujących zadania optymalizacyjne.

Nadanie sieci odpowiednich własności poprzez naukę, a także równoległy sposób jej działania stanowi nowy sposób podejścia do problemu programowania i działania urządzeń przetwarzających informację.

Podsumowując, sztuczne sieci neuronowe:

- zbudowane są z dużej liczby wzajemnie połączonych prostych jednostek – neuronów,
- wykonują obliczenia równolegle, jednocześnie w całej swej strukturze,
- przygotowanie ich do działania odbywa się poprzez uczenie, a nie poprzez programowanie,
- mają zastosowania w zadaniach, które potrafią rozwiązywać ludzie, a bardzo trudno do ich rozwiązania zaprogramować komputery.

Korzyści z zastosowania sieci neuronowych:

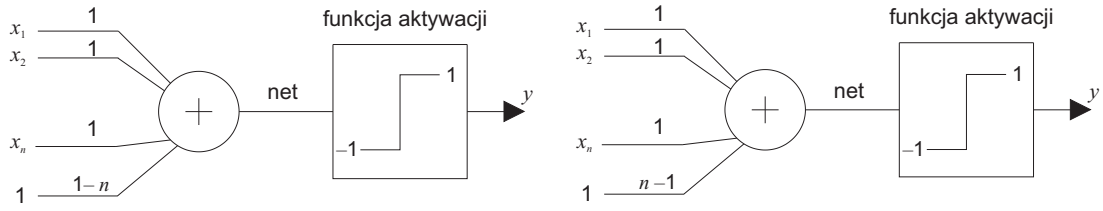
- zastąpienie programowania przez uczenie,
- duża szybkość działania (w przypadku realizacji sprzętowych),

Wady zastosowania sieci neuronowych:

- czasami bardzo długa nauka bez osiągnięcia dostatecznie małego błędu lub zła generalizacja,
- nieznajomość algorytmu działania nauczonej sieci.

6.4.2. Sieci warstwowe jako bramki logiczne

Pojedyncze neurony dyskretne mogą realizować dowolne iloczyny lub sumy logiczne argumentów wejściowych co ilustruje rys. 6.20. Gdy argument x_i, $i = 1, 2, \ldots, n$ wchodzący w skład sumy lub iloczynu ma być zanegowany, wtedy należy zamienić odpowiadającą mu wagę w_i z 1 na -1.



Iloczyn logiczny

$$y = x_1 \wedge x_2 \wedge \ldots \wedge x_n$$

Suma logiczna

$$y = x_1 \vee x_2 \vee \ldots \vee x_n$$

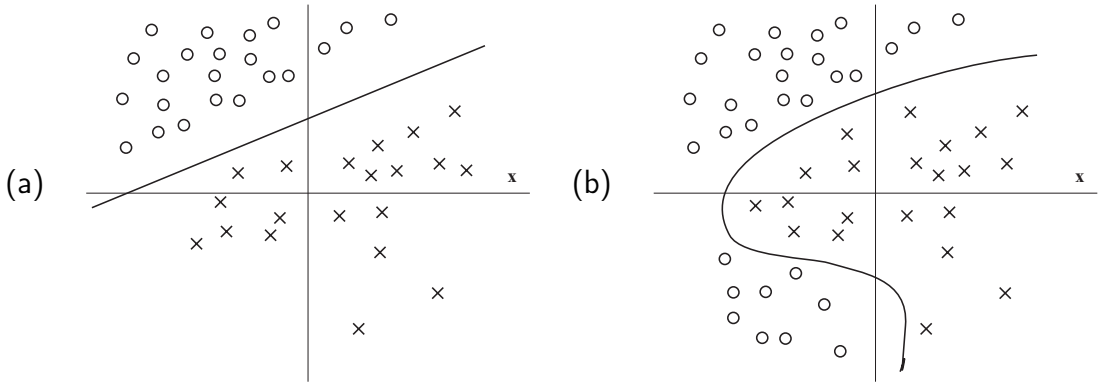
Rys. 6.20: Realizacja funkcji iloczynu i sumy logicznej przez bipolarne neurony dyskretne. Przyjęto następujące przyporządkowanie sygnałów wartościom logicznym: 0 logiczne – sygnał -1, 1 logiczne – sygnał $+1$

Ponieważ każdą funkcję logiczną można przedstawić w postaci sumy iloczynów (lub iloczynu sum), to można ją zrealizować za pomocą dwuwarswtwowj sieci neuronowej obliczającej w pierwszej warstwie sumy (iloczyny), a w drugiej iloczyny (sumy).

6.4.3. Sieci warstwowe jako klasyfikatory

Wygodnym sposobem przedstawiania zagadnień klasyfikacji jest interpretacja geometryczna. Stosując ją, każdemu obrazowi wejściowemu reprezentowanemu przez wektor $\mathbf{x} = [x_1, x_2, \ldots, x_n]^t$ odpowiada punkt w przestrzeni n wymiarowej nazywanej *przestrzenią obrazów*. Obszary zajmowane przez obrazy należące do poszczególnych klas nazywają się *obszarami decyzyjnymi*.

Ważną własnością jest *liniowa separowalność* obszarów decyzyjnych. Obszary te są liniowo separowalne, jeżeli dają się parami rozdzielić przez hiperpłaszczyzny. Na rysunku 6.21 przedstawiono przykłady separowalności obszarów decyzyjnych.



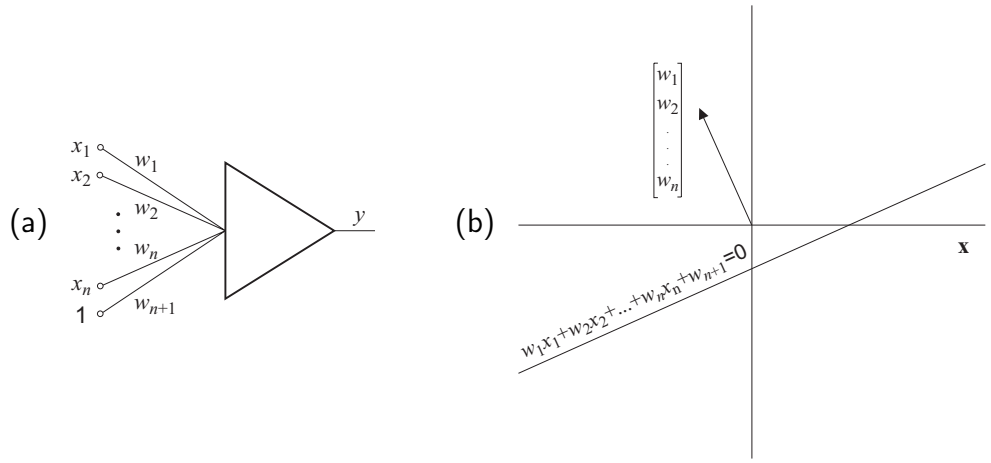
Rys. 6.21: Przykłady obszarów decyzyjnych w przypadku dwóch klas: (a) obszary liniowo separowalne i (b) nie separowalne liniowo

Jeżeli na neuron o wagach $\mathbf{w} = [w_1, w_2, \ldots, w_n, w_{n+1}]$ podany zostanie obraz wejściowy $\mathbf{x}$,to na wyjściu neuronu pojawia się sygnał, który może mieć wartość

mniejszą, równą lub większą od zera. Miejscem geometrycznym zerowych odpowiedzi neuronu są punkty $\mathbf{x}$ dane równaniem

$$w_1x_1 + w_2x_2 + \ldots w_nx_n + w_{n+1} = 0$$

Równanie to jest równaniem hiperpłaszczyzny w przestrzeni obrazów. Czyli interpretacją geometryczną neuronu w przestrzeni obrazów jest hiperpłaszczyzna dzieląca przestrzeń obrazów na dwie części; część, w której odpowiedzi neuronu będą ujemne i część, w której będą dodatnie (rys. 6.22).



Rys. 6.22: Pojedynczy neuron: (a) sygnały wejściowe i wyjściowe oraz (b) interpretacja geometryczna. Hiperpłaszczyzna $w_1x_1 + w_2x_2 + \ldots w_nx_n + w_{n+1} = 0$ reprezentuje zbiór sygnalów wejsciowych $\mathbf{x}$, dla których odpowiedź neuronu ma wartość zero. Hiperplaszczyzna dzieli przestrzeń $\mathbf{x}$ na dwie części: na część dla których odpowiedzi neuronu są dodatnie i część gdzie są ujemne. Wektor wag $[\,w_1 \;\; w_2 \;\; \cdots \;\; w_n\,]^t$ jest prostopadły do hiperpłaszczyzny, a jego zwrot wskazuje tę część przestrzeni $\mathbf{x}$, dla której odpowiedzi neuronu są dodatnie

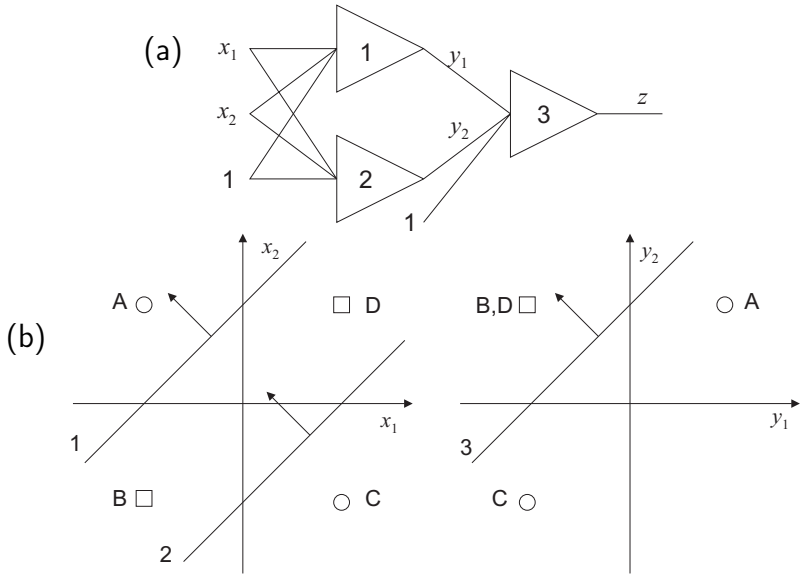
Stąd widać, że pojedynczy neuron może przeprowadzać klasyfikację obrazów należących do dwóch klas separowalnych liniowo. W przypadku liczby klas większej od dwóch separowalność liniowa jest warunkiem koniecznym przeprowadzenia klasyfikacji przez sieć jednowarstwową.

Gdy klasy nie są separowalne liniowo, to klasyfikację można przeprowadzić przy pomocy sieci o dwu lub więcej warstwach.

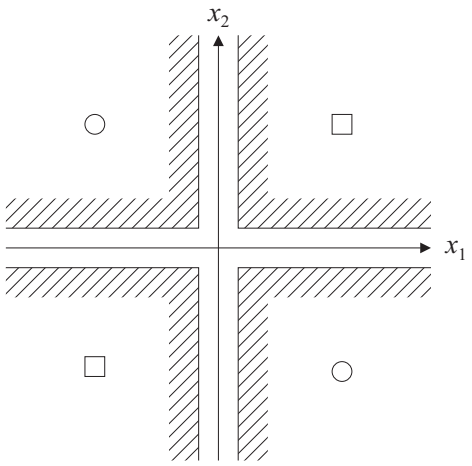
Przykład 6.4

Na rysunku przedstawiono 6.23 sieć dwuwarstwową klasyfikującą obrazy w przypadku gdy klasy nie są separowalne liniowo (zadanie XOR). Jak widać z rysunku obrazy należące do dwóch klas początkowo nie separowalne liniowo po przejściu przez pierwszą warstwę stają się liniowo separowalne.

Na rysunku 6.24 przedstawiono zadanie, gdy klasy reprezentowane są przez zbiory nieskończone. Wtedy do klasyfikacji można użyć jednej warstwy zamieniającej zbiory w pojedyncze punkty, a poźniej wystarczają dwie warstwy, czyli łącznie trzy warstwy. Aby pierwsza warstwa zamieniła zbiory w punkty, hiperpłaszczyzny



Rys. 6.23: Klasyfikacja w przypadku klas nie separowalnych liniowo (zadanie XOR): (a) dwuwarstwowa sieć neuronowa, (b) obrazy w przestrzeni obrazów (przestrzeń x_1x_2) i obrazy w *przestrzeni odwzorowań* (przestrzeń y_1y_2)



Rys. 6.24: Przykład zadania klasyfikacji, gdy klasy reprezentowane są przez zbiory nieskończone. Wtedy do klasyfikacji trzeba użyć jednej warstwy zamieniającej zbiory w pojedyncze punkty (w tym przypadku dwa neurony reprezentowane przez proste $x_1 = 0$ i $x_2 = 0$), a później wystarczają dwie warstwy, czyli łącznie trzy warstwy.

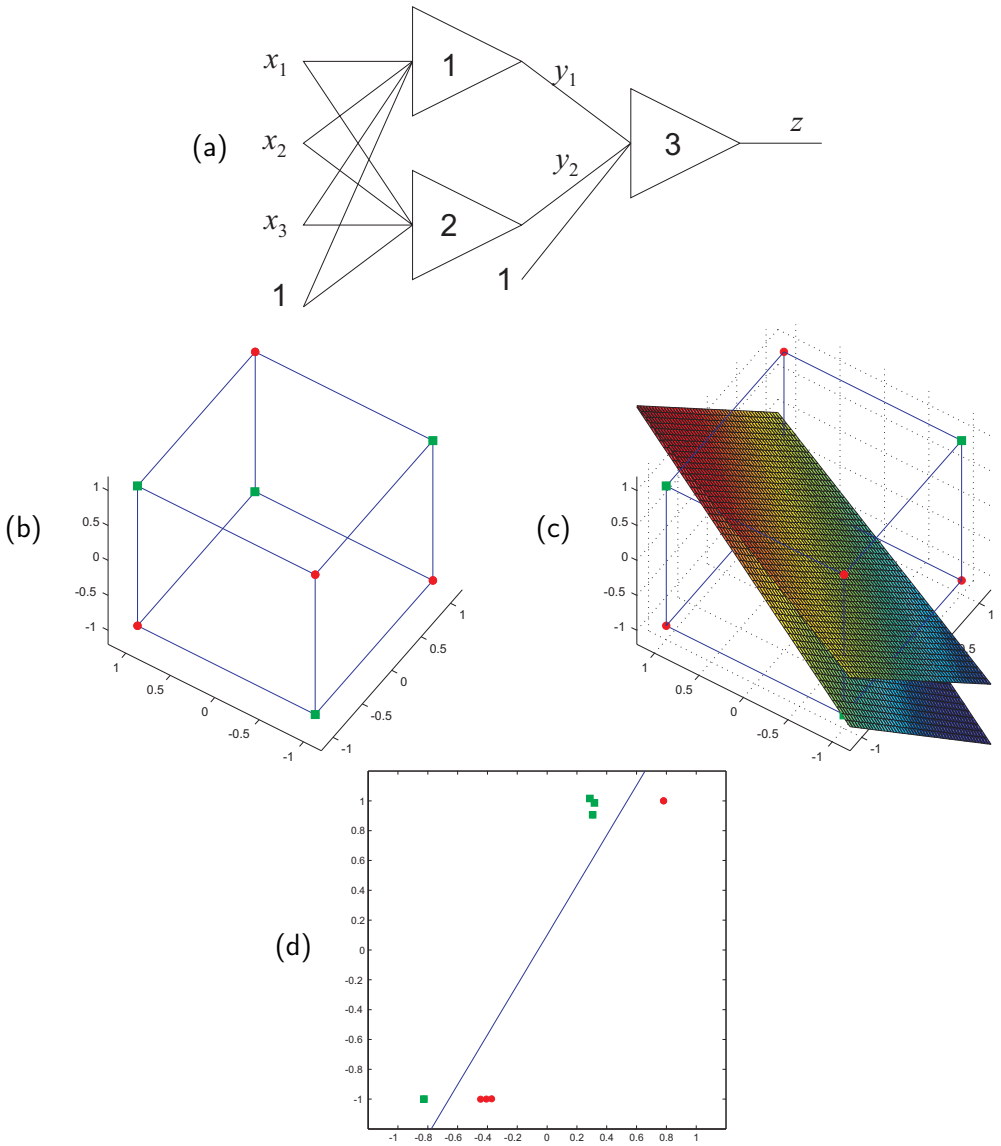
reprezentujące neurony pierwszej warstwy powinny tak podzielić przestrzeń obrazów aby w powstałych podobszarach występowały obrazy należące do pojedynczych klas. ■

Przykład 6.5

Pokazano tu klasyfikacje obrazów nie separowalnych liniowo (trójwymiarowe zadanie XOR) przez dwuwarstwową sieć neuronową o dwóch neuronach w warstwie wejściowej (rys. 6.25). Wyznaczone w procesie uczenia wagi neuronów w perwszej i drugiej

warstwie są następujące:

$$W_1 = \begin{bmatrix} -0,3725 & 0,3667 & -0,3725 & -0,0628 \\ -8,5935 & 10,2626 & -8,0737 & -4,9021 \end{bmatrix}, [W_2 = 28,7597 - 17,19161,7220]$$



Rys. 6.25: Klasyfikacja trojwymiarowej funkcji XOR: (a) sieć neuronowa, (b) obrazy w przestrzeni trójwymiarowej, (c) płaszczyzny reprezentujące neurony pierwszej warstwy, (d) obrazy po przekształceniu przez warstwę pierwszą i linia prosta reprezentujaca neuron warstwy drugiej

Zwrócmy uwagę, że dwa neurony wejściowe nie są w stanie podzielić przestrzeni w taki sposób aby w powstałych podobszarach występowały obrazy należące wyłącznie do identycznych klas, a mimo to poprawna klasyfikacja następuje – dwa neurony warstwy wejściowej przenoszą obrazy do przestrzeni odwzorowań, w której są one już liniowo separowalne. Zachowanie to można łatwo wytłumaczyć jeśli zauważymy,

że wielkości wag neuronów warstwy wejściowej znacznie między sobą się różnią. ■

Powstaje ogólne pytanie: ile warstw jest koniecznych do klasyfikacji obrazów należących do klas nie separowalnych liniowo? Można teoretycznie wykazać (patrz. np Bishop 1996), że dwie warstwy wystarczają do klasyfikacji klas dowolnie rozmieszczonych w przestrzeni. Wcześniej w tym skrypcie wykazano to dla przypadku obrazów binarnych należących do dwóch klas przez pokazanie, że dowolną funkcję logiczną można zrealizować przy pomocy sieci dwuwarstwowej.

W konkretnych zadaniach, najczęściej, korzystnie jest użyć więcej niż dwie warstwy.

6.4.4. Sieci warstwowe jako aproksymatory funkcji

Innym obok klasyfikacji często spotykanym zastosowaniem jest użycie sieci warstwowych do aproksymacji funkcji ciągłych.

Zadanie aproksymacji polega na przybliżaniu zadanej ciągłej, wieloargumentowej funkcji $h(\mathbf{x})$ funkcją $H(\mathbf{w}, \mathbf{x})$ realizowaną przez warstwową sieć neuronową, gdzie $\mathbf{x} = [\, x_1 \;\; x_2 \;\; \cdots \;\; x_n \,]^t$ jest wektorem wejściowym, natomiast $\mathbf{w} = [\, w_1 \;\; w_2 \;\; \cdots \;\; w_m \,]^t$ jest wektorem parametrów (wag). Poniżej pokażemy, że przy doborze odpowiedniej liczby neuronów sieć trójwarstwowa może aproksymować z dowolną dokładnością dowolną funkcję $h(\mathbf{x})$.

Zauważmy, że sieć warstwowa złożona z neuronów unipolarnych może aproksymować (realizować) następującą funkcję (nazywaną bramką)

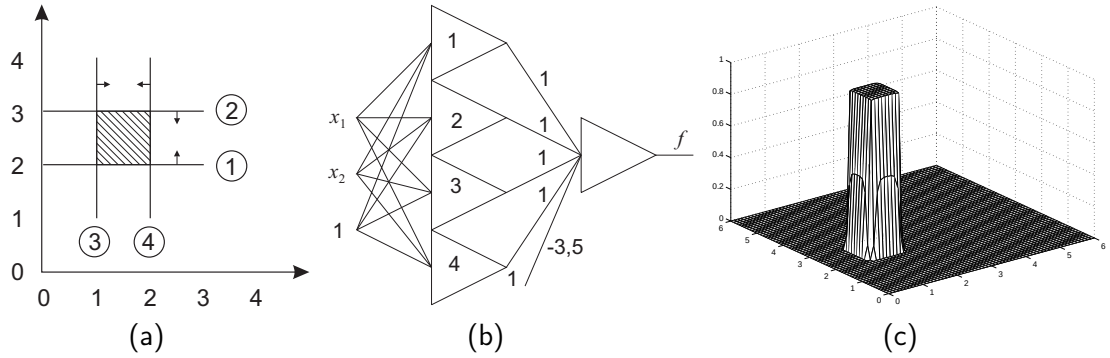
$$f(x_1, x_2, \ldots x_n) = \begin{cases} 1 & \text{gdy } x_{0i} - \dfrac{\Delta x}{2} < x_i < x_{0i} + \dfrac{\Delta x}{2}, i = 1, 2, \ldots, n \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{6.7}$$

Funkcja ta przyjmuje wartość 1 dla punktu $\mathbf{x}$ znajdującego się wewnątrz hiperkostki o środku w punkcie $\mathbf{x}_0 = [\, x_{01} \;\; x_{02} \;\; \cdots \;\; x_{0n} \,]^t$ i krawędzi Δx, a wartość 0 poza nią. Funkcję tę może aproksymować dwuwarstwowa sieć neuronowa mająca $2n$ neuronów w warstwie wejściowej i jeden neuron w warstwie wyjściowej. Każdy neuron w warstwie wejściowej reprezentuje jedną z hiperpłaszczyzn tworzących hiperkostkę, a wartości sygnałów wyjściowych tych neuronów (~ 0 lub ~ 1) mówią po której stronie hiperpłaszczyzn znajduje się punkt x, zaś neuron wyjściowy decyduje czy punkt x znajduje się wewnątrz hiperkostki.

Przykład 6.6

Wyznaczymy tu sieć aproksymującą funkcję (6.7) dla przypadku dwuwymiarowego: $n = 2$, $x_0 = [3{,}5\ 4{,}5]^t$, $\Delta x = 1$.

Na rysunku 6.26a przedstawiono kwadrat wewnątrz którego funkcja przyjmuje wartość 1, przy czym strzałki wskazują półpłaszczyzny dla których odpowiedź neuronu ma być >0,5. Na podstawie równań prostych ograniczających ten kwadrat można wyznaczyć wagi czterech neuronów warstwy wejściowej sieci. Np. równanie prostej oznaczonej numerem 1 ma postać $x_2 - 2 = 0$ i stąd wagi odpowiadającego jej neuronu wynoszą: $w_{11} = 0$, $w_{12} = 1$, $w_{13} = -2$. Wyznaczone w ten sposób wagi



Rys. 6.26: Przykład konstrukcji sieci neuronowej realizującej funkcję (6.7) (bramkę): (a) kwadrat wewnątrz którego funkcja ma przyjmować wartość 1. Proste oznaczone numerami 1 do 4 reprezentują neurony tworzonej sieci, przy czym strzałki wskazują półpłaszczyzny dla których odpowiedzi neuronów są $> 0,5$, (b) sieć neuronowa realizująca funkcję (6.7) oraz (c) odpowiedź tej sieci na sygnały wejściowe z zakresu $(0, 6)$

neuronów w pierwszej warstwie zostały następnie pomnożone przez stały współczynnik $>> 1$ (przykładowo 20), aby w uzyskanej sieci przejście pomiędzy wartościami wyjścia 0 i 1 było odpowiednio strome. Ostatecznie otrzymano

$$\begin{array}{lll} w_{11} = 0, & w_{12} = 20, & w_{13} = -40, \\ w_{21} = 0, & w_{22} = -20, & w_{23} = 60, \\ w_{31} = 20, & w_{32} = 0, & w_{33} = -20, \\ w_{41} = -20, & w_{42} = 0, & w_{43} = 40 \end{array}$$

Biorąc te wagi oraz przyjmując wagi neuronu wyjściowego: $w_{5i} = 1$, $i = 1, 2, 3, 4$, $w_{55} = -3,5$ otrzymujemy sieć przedstawioną na rysunku 6.26b, zaś odpowiedź tej sieci na sygnały wejściowe z zakresu $(0, 6)$ przedstawiono na rys 6.26c. ■

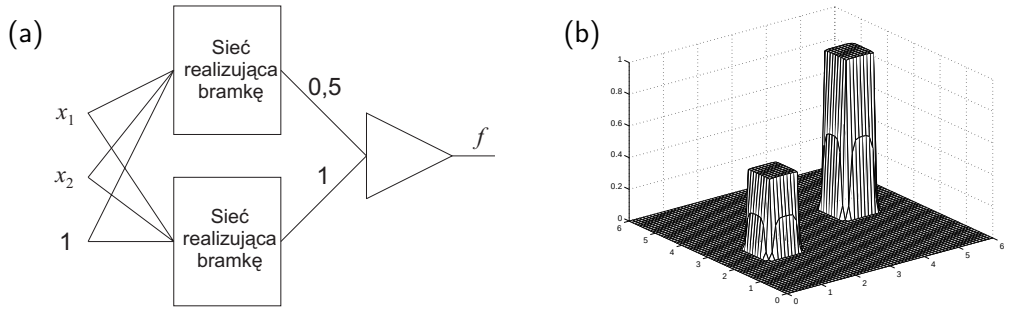
Podając wyjścia sieci realizujących bramki na odpowiednio ważone wejścia neuronu liniowego (sumującego) otrzymujemy trójwarstwową sieć mogącą aproksymować dowolną funkcję, przy czym dokładność aproksymacji zależy od szerokości bramki Δx.

Przykład 6.7

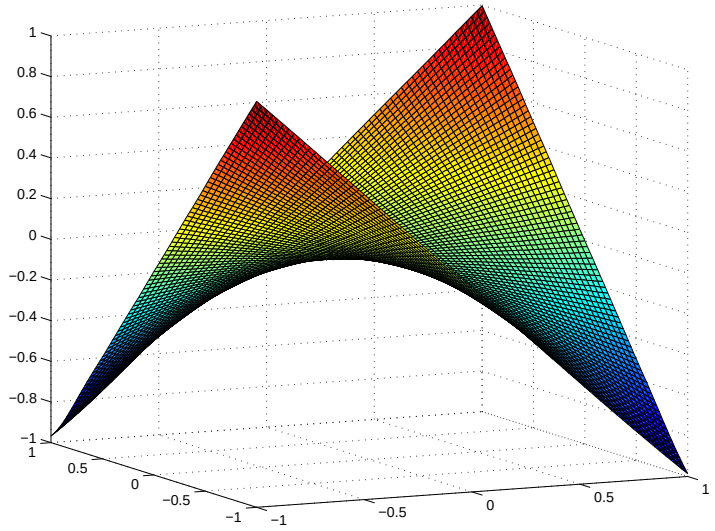
Na rysunku 6.27a przedstawiono przykładową sieć zbudowaną z dwóch sieci realizujących bramki i jednego neuronu liniowego, zaś na rysunku 6.27b funkcję realizowaną przez całą sieć. ■

Pokazano powyżej, że trójwarstwowa sieć neuronowa może aproksymować dowolną funkcję. Jak wspomniano wcześniej do takiej aproksymacji wystarczy już sieć dwuwarstwowa.

W praktycznych realizacjach nie aproksymuje się funkcji za pomocą sumowania bramek, tylko przyjmuję się sieć warstwową o określonej liczbie warstw i liczbach neuronów w warstwach i bezpośrednio poprzez naukę uzyskuje się pożądane przybliżenie. Na rys. 6.28 przedstawiono przykład aproksymacji funkcji $y = x_1 x_2$ przez sieć dwuwarstwową mającą 16 neuronów wejściowych.



Rys. 6.27: Przykładowa sieć neuronowa sumująca dwie bramki (a) oraz funkcja przez nią realizowana (b)



Rys. 6.28: Aproksymacja funkcji $y = x_1 x_2$ przez dwuwarstwową sieć neuronową o 16 neuronach wejściowych. Aproksymację przeprowadzono korzystając z 1000 par uczących

6.4.5. Metoda propagacji wstecznej błędu

Sieci warstwowe poprzez odpowiedni dobór wag są w stanie aproksymować dowolne funkcje, czyli są w stanie po nauczeniu rozwiązywać dowolne zadania. Spośród metod uczenia sieci warstwowych najczęściej stosowana jest gradientowa metoda największego spadku, która w przypadku uczenia sieci neuronowych nazywana jest *metodą propagacji wstecznej błędu*, (ang. error back propagation method), którą poniżej krótko omówimy dla przypadku sieci dwuwarstwowej.

Przyjmijmy oznaczenia jak na schemacie sieci przedstawionym na rys. 6.29. Przy oznaczeniach jak na rysunku 6.29, sygnał wyjściowy sieci dwuwarstwowej wynosi

$$\mathbf{z} = \mathbf{\Gamma}\left[\mathbf{W}\mathbf{y}\right] = \mathbf{\Gamma}\left[\mathbf{W}\mathbf{\Gamma}\left[\mathbf{V}\mathbf{x}\right]\right] \tag{6.8}$$

gdzie:

$$\mathbf{x} = [\, x_1 \;\; x_2 \;\; \cdots \;\; x_I \,]^t, \qquad \mathbf{y} = [\, y_1 \;\; y_2 \;\; \cdots \;\; y_J \,]^t, \qquad \mathbf{z} = [\, z_1 \;\; z_2 \;\; \cdots \;\; z_K \,]^t$$

$$\mathbf{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1I} \\ v_{21} & v_{22} & \cdots & v_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ v_{J1} & v_{J2} & \cdots & v_{JI} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^t \\ \mathbf{v}_2^t \\ \vdots \\ \mathbf{v}_J^t \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1J} \\ w_{21} & w_{22} & \cdots & w_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KJ} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^t \\ \mathbf{w}_2^t \\ \vdots \\ \mathbf{w}_K^t \end{bmatrix}$$

zaś $\mathbf{\Gamma}$ jest nieliniowym operatorem o postaci

$$\mathbf{\Gamma}[\cdot] = \begin{bmatrix} f(\cdot) \\ f(\cdot) \\ \vdots \\ f(\cdot) \end{bmatrix}$$

gdzie funkcje $f(\cdot)$ są funkcjami aktywacji dane wzorami:

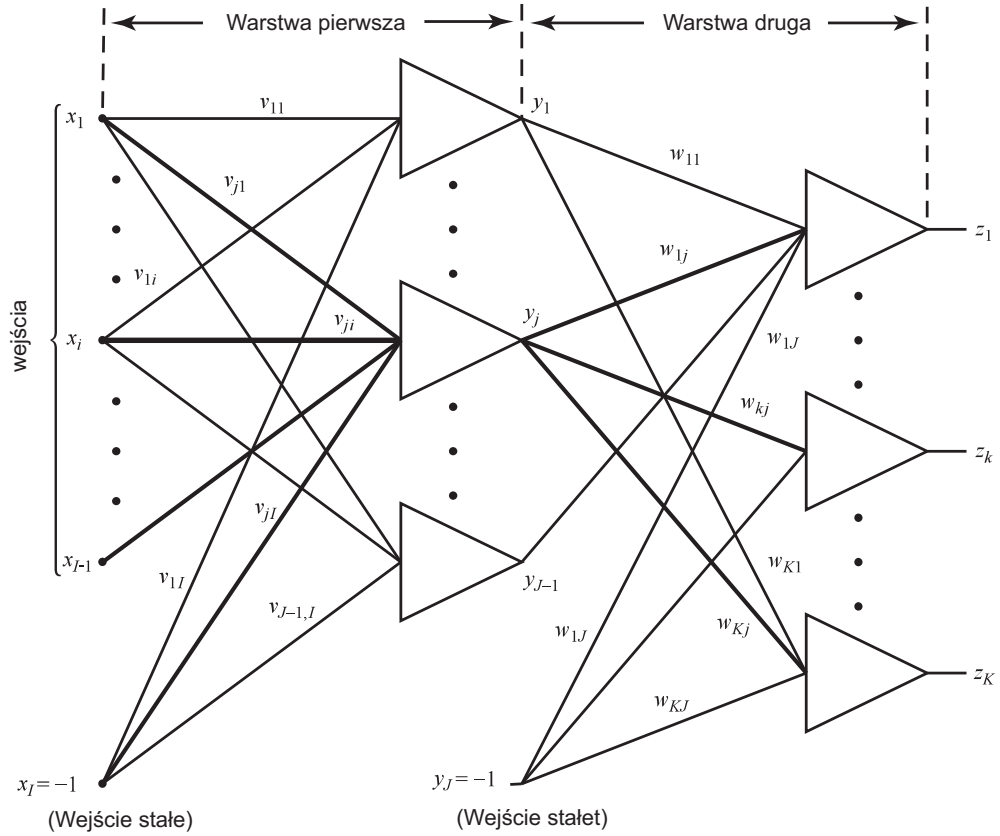
$$f(net) = \frac{2}{1 + \exp(-\lambda net)} - 1 \qquad \text{(funkcja bipolarna)} \tag{6.9}$$

lub

$$f(net) = \frac{1}{1 + \exp(-\lambda net)} \qquad \text{(funkcja unipolarna)} \tag{6.10}$$

Oznaczając pożądany sygnał wyjściowy jako

$$\mathbf{d} = [\, d_1 \;\; d_2 \;\; \cdots \;\; d_K \,]^t$$



Rys. 6.29: Schemat sieci dwuwarstwowej

błąd sieci dla jednego wektora wejściowego $\mathbf{x}_l$, $l = 1, 2, \ldots, p$, wynosi

$$E_l(\mathbf{V}, \mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (d_{lk} - z_{lk})^2 = \tfrac{1}{2} \|\mathbf{d}_l - \mathbf{z}_l(\mathbf{V}, \mathbf{W})\|^2, \tag{6.11}$$

gdzie $\mathbf{d}_l$ i $\mathbf{z}_l$ oznaczają odpowiednio pożądane i aktualne sygnały wyjściowe, gdy na wejście podano obraz $\mathbf{x}_l$, zaś błąd łączny dla wszystkich wektorów uczących wynosi

$$E(\mathbf{V}, \mathbf{W}) = \sum_{l=1}^{p} E_l(\mathbf{V}, \mathbf{W}) \tag{6.12}$$

Uczenie polega na zastosowaniu metody najszybszego spadku do znalezienia wag $\mathbf{V}$, $\mathbf{W}$ minimalizujących funkcję $E(\mathbf{V}, \mathbf{W})$ daną wzorem (6.12). Ponieważ funkcja (6.12) jest sumą składników, obliczenie jej gradientu sprowadza się do obliczenia i zsumowania gradientów funkcji $E_l(\mathbf{V}, \mathbf{W})$ danej wzorem (6.11). Mamy więc

$$\Delta \mathbf{V}, \mathbf{W} = -\eta \nabla E(\mathbf{V}, \mathbf{W}) = -\eta \sum_{l=1}^{p} \nabla E_l(\mathbf{V}, \mathbf{W}) \tag{6.13}$$

Możliwe i realizowane są dwa rodzaje postępowania. W jednym z nich, będącym zwykłą metodą gradientową, korekcja wektorów wag następuje po obliczeniu gradientu funkcji (6.12) (po podaniu wszystkich wektorów uczących). W drugim, korekcja wag następuje po każdorazowym podaniu wektora uczącego i obliczeniu gradientu funkcji (6.11).

Przeprowadzając korekcje wag po każdorazowym podaniu wektora uczącego w kierunku malejącego gradientu otrzymujemy ogólne wzory na korekcję wag w warstwie pierwszej i drugiej

$$\begin{aligned} \Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}}, \qquad k = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, J \\ \Delta v_{ji} &= -\eta \frac{\partial E}{\partial v_{ji}}, \qquad j = 1, 2, \ldots, J-1, \quad i = 1, 2, \ldots, I \end{aligned} \tag{6.14}$$

gdzie η jest stałym współczynnikiem, błąd E jest zdefiniowany wzorem (6.11) zaś indeks pominięto dla uproszczenia zapisu.

Ponieważ błąd E zależy od wartości wagi w_{kj} tylko poprzez funkcję $net_k = \mathbf{w}_k^t \mathbf{y}$, oraz od wartości wagi v_{ji} tylko poprzez funkcję $net_j = \mathbf{v}_j^t \mathbf{x}$, to stosując wzór na pochodne cząstkowe funkcji złożonej możemy napisać

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} y_j \tag{6.15}$$

oraz

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial v_{ji}} = \frac{\partial E}{\partial net_j} x_i \tag{6.16}$$

Ujemna pochodna błędu po wielkości net_k (net_j) nazywa się *sygnałem błędu delta* generowanym przez k-ty (j-ty) neuron i jest zdefiniowana następująco

$$\delta_{zk} \stackrel{\text{def}}{=} -\frac{\partial E}{\partial net_k}, \qquad \delta_{yj} \stackrel{\text{def}}{=} -\frac{\partial E}{\partial net_j}$$

Obliczymy teraz wartość δ_{zk}. Na podstawie (6.11) mamy

$$\delta_{zk} = -\frac{\partial E}{\partial net_k} = -\frac{1}{2}\frac{\partial}{\partial net_k}\left(d_k - z_k\right)^2 = \left(d_k - z_k\right)\frac{\partial z_k}{\partial net_k}$$

i ostatecznie

$$\delta_{zk} = \left(d_k - z_k\right) f'(net_k) \tag{6.17}$$

Podstawiając powyższe wyrażenie do równań (6.15) i (6.14) otrzymujemy

$$w'_{kj} = w_{kj} + \eta(d_k - z_k)f'(net_k)y_j \tag{6.18}$$

Obliczymy teraz wartość sygnału błędu δ_{yj}. Pobudzenie j-tego neuronu, net_j, wpływa teraz na wszystkie składowe błędu (6.11) poprzez wyjście y_j, podczas gdy poprzednio pobudzenie net_k wpływało tylko na jedną zależną od niego składową. Biorąc to pod uwagę, sygnał błędu δ_{yj} w węźle j wynosi

$$\delta_{yj} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j} = -\frac{\partial E}{\partial y_j}f'(net_j) \tag{6.19}$$

a następnie mamy

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{K}\left(d_k - z_k\right)^2\right] = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{K}\left\{d_k - f\left[net_k(\mathbf{y})\right]\right\}^2\right] =$$

$$= -\sum_{k=1}^{K}\left[\left(d_k - z_k\right)\frac{\partial f\left[net_k\left(\mathbf{y}\right)\right]}{\partial y_j}\right] = -\sum_{k=1}^{K}\left[\left(d_k - z_k\right)\frac{\partial f}{\partial net_k}\frac{\partial net_k}{\partial y_j}\right] =$$

$$= -\sum_{k=1}^{K}\left[\left(d_k - z_k\right)f'\left(net_k\right)w_{kj}\right] = -\sum_{k=1}^{K}\delta_{zk}w_{kj} \tag{6.20}$$

gdzie δ_{zk} dane jest przez wzór (6.17). Podstawiając (6.20) do (6.19) mamy

$$\delta_{yj} = f'_j(net_j)\sum_{k=1}^{K}\delta_{zk}w_{kj}, \qquad j = 1, 2, \ldots, J-1 \tag{6.21}$$

Podstawiając (6.21) do (6.16) i (6.14) otrzymujemy ostatecznie wyrażenie opisujące zasadę korekcji wag w warstwie pierwszej

$$v'_{ji} = v_{ji} + \eta f'_j(net_j)x_i\sum_{k=1}^{K}\delta_{zk}w_{kj} \tag{6.22}$$

Jak z niego wynika, korekcja wag dochodzących do neuronu j-tego w warstwie pierwszwj jest proporcjonalna do ważonej sumy wszystkich wartości δ w warstwie następnej. Wagi w_{kj}, $k = 1, 2, \ldots, K$, w warstwie wyjściowej, które wpływają na błąd δ_{yj} oraz wagi v_{ji}, dla $i = 1, 2, \ldots, I$, na które ma wpływ błąd δ_{yj} zostały wyróżnione na rysunku pogrubionymi liniami.

Dla obliczenia $f'(net)$ rozpatrzmy dwie typowe funkcje aktywacji $f(net)$ dane wzorami (6.9) i (6.10) Dla funkcji bipolarnej mamy

$$f'(net) = \frac{2\exp(-net)}{\left[1 + \exp(-net)\right]^2} = \frac{1}{2}\left(1 - f(net)^2\right) \tag{6.23}$$

Dla funkcji unipolarnej mamy

$$f'(net) = \frac{\exp(-net)}{[1+\exp(-net)]^2} = f(net)(1-f(net)) \tag{6.24}$$

Przedstawiona metoda nauki sieci wielowarstwowej nazywa się *metodą propagacji wstecznej*. Jest ona uniwersalnym algorytmem nauki sieci wielowarstwowych i jej odkrycie stanowiło początek nowego etapu rozwoju badań nad sieciami neuronowymi. Nazwa metody wynika z kolejności obliczania sygnałów błędu δ, które przebiega w odwrotnym kierunku niż przechodzenie sygnałów przez sieć, to znaczy od warstwy wyjściowej poprzez warstwy ukryte w kierunku wejść.

Podsumowanie metody propagacji wstecznej błędu

Dane jest p par uczących

$$\{\mathbf{x}_1, \mathbf{d}_1, \mathbf{x}_2, \mathbf{d}_2, \ldots, \mathbf{x}_p, \mathbf{d}_p\}$$

gdzie $\mathbf{x}_i$ ma rozmiar $(I \times 1)$, $\mathbf{d}_i$ ma rozmiar $(K \times 1)$, a $x_{iI} = -1$, dla $i = 1, 2, \ldots, p$. Wektor $\mathbf{y}_i$ o rozmiarach $(J \times 1)$ jest wektorem wyjściowym warstwy ukrytej, a $y_{iJ} = -1$, dla $i = 1, 2, \ldots, p$. Wektor $\mathbf{z}$ o rozmiarach $(K \times 1)$ jest wyjściem warstwy wyjściowej. Parametr q oznacza numer cyklu nauki a l oznacza numer kroku wewnątrz cyklu nauki.

Krok 1 Wybór $\eta > 0$, $E_{max} > 0$.

Krok 2 Wybór elementów macierzy wag $\mathbf{W}$ i $\mathbf{V}$ jako niewielkich liczb losowych. Macierz $\mathbf{W}$ ma rozmiar $(K \times J)$, macierz $\mathbf{V}$ ma rozmiar $(J \times I)$.

Krok 3 Ustawienie wartości początkowych liczników oraz zerowanie błędu

$$q \leftarrow 1, \quad l \leftarrow 1, \quad E \leftarrow 0$$

Krok 4 Podanie obrazu na wejście (obrazy mogą być podawane w kolejności losowej) i obliczenie sygnału wyjściowego

$$\mathbf{x} \leftarrow \mathbf{x}_l, \quad \mathbf{d} \leftarrow \mathbf{d}_l$$

$$y_j \leftarrow f\left(\mathbf{v}_j^t \mathbf{x}\right), \qquad j = 1, 2, \ldots, J-1$$

gdzie $\mathbf{v}_j$ jest j-tym wierszem $\mathbf{V}$

$$z_k \leftarrow f\left(\mathbf{w}_k^t \mathbf{y}\right), \qquad k = 1, 2, \ldots, K$$

gdzie $\mathbf{w}_k$ jest k-tym wierszem $\mathbf{W}$, zaś funkcja aktywacji $f(\cdot)$ jest dana wzorem (6.9) – neuron bipolarny lub wzorem (6.10) – neuron unipolarny.

Krok 5 Uaktualnienie błędu:

$$E \leftarrow E + \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2$$

Krok 6 Obliczenie wektorów sygnałów błędów $\boldsymbol{\delta}_z$ i $\boldsymbol{\delta}_y$ obydwu warstw. Wektor $\boldsymbol{\delta}_z$ ma rozmiar $(K \times 1)$, a wektor $\boldsymbol{\delta}_y$ ma rozmiar $(J \times 1)$. Dla bipolarnej funkcji aktywacji (6.9)

$$\delta_{zk} = \frac{1}{2}(d_k - z_k)(1 - z_k^2), \qquad k = 1, 2, \ldots, K$$

$$\delta_{yj} = \frac{1}{2}(1 - y_j^2) \sum_{k=1}^{K} \delta_{zk} w_{kj}$$
$$j = 1, 2, \ldots, J-1$$

Dla unipolarnej funkcji aktywacji (6.10)

$$\delta_{zk} = (d_k - z_k) z_k (1 - z_k), \quad k = 1, 2, \ldots, K$$

$$\delta_{yj} = y_j (1 - y_j) \sum_{k=1}^{K} \delta_{zk} w_{kj}, \quad j = 1, 2, \ldots, J-1$$

Krok 7 Uaktualnienie wag warstwy wyjściowej

$$w_{kj} \leftarrow w_{kj} + \eta \delta_{zk} y_j$$

$$k = 1, 2, \ldots, K, \quad j = 1, 2, \ldots, J$$

Krok 8 Uaktualnienie wag warstwy ukrytej

$$v_{ji} \leftarrow v_{ji} + \eta \delta_{yj} x_i$$

$$j = 1, 2, \ldots, J-1, \quad i = 1, 2, \ldots, I$$

Krok 9 Jeżeli $l < p$ to $p \leftarrow l + 1$ i przejście do kroku 4.

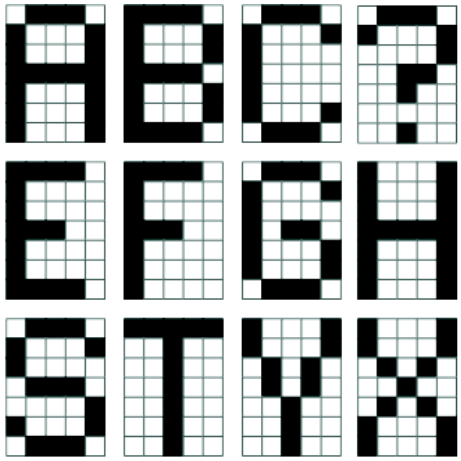
Krok 10 Cykl nauki został zakończony. Jeżeli teraz $E < E_{max}$, to zakończenie nauki, w przeciwnym razie rozpoczęcie nowego cyklu przez przejście do kroku 3.

Omówiono wariant dokonujacy korekty po każdym podaniu nowego obrazu. W przypadku stosowania wariantu dokonywania korekty po podaniu wszystkich obrazów uczących, należy po podaniu każdego obrazu obliczyć wektor korekcji wag ale nie dokonywać korekcji, a po podaniu wszystkich obrazów, zsumować wektory korekcji i wynikowy wektor korekcji dodać do wektora wag.

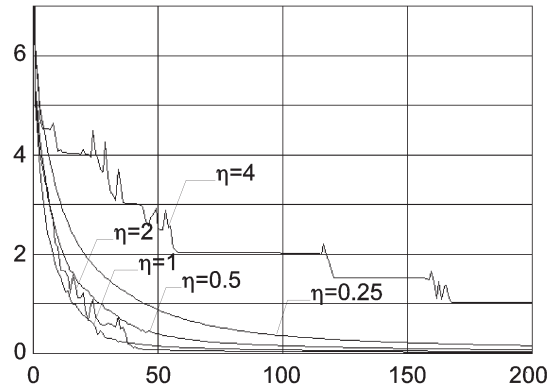
6.4.6. Przykład zastosowania metody propagacji wstecznej

Omówimy tu przykład uczenia metodą propagacji wstecznej jednowarstwowej sieci klasyfikującej dwanaście znaków przedstawionych na rys. 6.30a. Znaki te, z których każdy reprezentuje inną klasę, zajmują wierzchołki 35-wymiarowej kostki i są oczywiście liniowo separowalne, a tym samym nauczona sieć jednowarstwowa powinna je poprawnie klasyfikować. Wyniki klasyfikacji odczytywano przyporządkowując sygnałom na wyjściach neuronów liczbę 1, gdy przekraczają poziom 0.5, a liczbę 0 w

(a)



(b)



Rys. 6.30: Uczenie sieci neuronowej metodą propagacji wstecznej: (a) klasyfikowane znaki, (b) przebiegi błędu w procesie uczenia dla kilku różnych współczynników η

przeciwnym razie, co jest równoważne zastąpieniu układzie neuronów ciągłych neuronami dyskretnymi. Badana sieć zawierała 12 unipolarnych neuronów o ciągłych funkcjach aktywacji z parametrami $\lambda = 1$.

Przeprowadzono 200 cykli uczenia sieci dla kilku wartości współczynnika η, po których uzyskano następujące błędy łączne:

$$\begin{aligned} \eta &= 4, & E &= 1,0040 \\ \eta &= 2, & E &= 0,0138 \\ \eta &= 1, & E &= 0,0294 \\ \eta &= 0,5, & E &= 0,0651 \\ \eta &= 0,25, & E &= 0,1469. \end{aligned}$$

Przebieg uczenia pokazano na rys. 6.30b. Z rysunku widać wyraźnie wpływ współczynnika η na szybkość uczenia. Zbyt duża jego wartość ($\eta = 4$) powoduje duże oscylacje i przeregulowania i w efekcie duży błąd końcowy. Pośrednie wartości ($\eta = 2$ lub 1) powodują, pomimo niewielkich oscylacji, szybkie osiąganie małych wartości błędu. Wreszcie zbyt małe wartości ($\eta = 0,5$ lub 0,25) powodują systematyczne i bez oscylacji, ale powolne malenie błędu, który po ustalonej liczbie kroków pozostaje jednak stosunkowo duży. Widać więc, że w omawianym przykładzie optymalnymi wartościami współczynnika η było 2 lub 1. Kontynuując proces uczenia poza przyjęte 200 cykli, uzyskiwano małe wartości błędu także dla wolniej zbieżnych procesów

uczenia.

Przy przyjętym sposobie odczytu klasy i definicji błędu łącznego, będącego sumą błędu po wszystkich obrazach wejściowych, widać, że klasyfikacja wszystkich obrazów będzie bezbłędna, gdy wartość błędu łącznego będzie mniejsza od 0,125. Wartość ta zapewnia, że różnica pomiędzy sygnałem oczekiwanym a rzeczywistym na wyjściu dowolnego neuronu i dla każdego obrazu nigdy nie przekroczy wartości progowej 0,5, a tym samym odczyt klasy będzie bezbłędny. Na podstawie przytoczonych wyżej wartości błędów widać, że bezbłędne klasyfikacje zapewniły procesy uczenia dla współczynników η =0,5, 1 i 2.

Dla ilustracji charakteru reakcji sieci poniżej przedstawiono sygnały wyjściowe neuronów nauczonej sieci (dla $\eta = 2$) odpowiadającej na przykładowy obraz wejściowy (znak A), gdzie w nawiasach przy liczbach podano znaki reprezentowane przez poszczególne neurony:

$$\begin{array}{cccccccc} (A) & 0,9799, & (B) & 0,0039, & (C) & 0,0113, & (?) & 0,0103 \\ (E) & 0,0001, & (F) & 0,0100, & (G) & 0,0020, & (H) & 0,0123 \\ (S) & 0,0002, & (T) & 0,0014, & (Y) & 0,0001, & (F) & 0,0086 \end{array}$$

6.4.7. Sieci neuronowe jako pamięci asocjacyjne

Podstawowe pojęcia

Pamięc asocjacyjna jest układem odwzorowującym wektory $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, $\ldots, \mathbf{x}^{(p)}$ należące do R^n w wektory $\mathbf{y}^{(1)}$, $\mathbf{y}^{(2)}$, $\ldots, \mathbf{y}^{(p)}$ należące do R^m. Wektory $\mathbf{x}^{(i)}$ i $\mathbf{y}^{(i)}$ nazywane są *wzorcowymi* lub *prototypowymi*.

Umieszczenie w pamięci wektorów wzorcowych nazywa się *zapisem* i pociąga za sobą odpowiednie dobranie wartości wag. Odwzorowanie wykonane przez pamięć na wektorze wejściowym $\mathbf{x}$ nazywa się *odczytem*.

Jeśli sieć przechowuje p asocjacji o postaci

$$\mathbf{x}^{(i)} \rightarrow \mathbf{y}^{(i)}, \qquad \mathbf{y}^{(i)} \neq \mathbf{x}^{(i)}, \qquad i = 1, 2, \ldots, p, \tag{6.25}$$

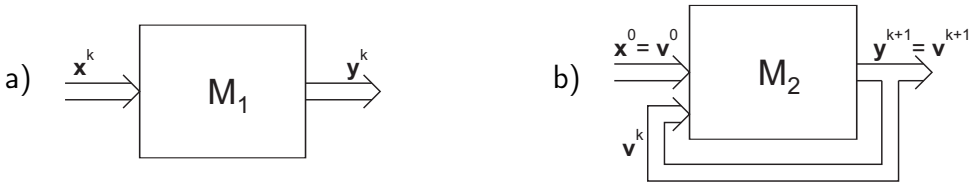
to pamięć nazywa się *heteroasocjacyjną*. Podczas odczytu odwzorowuje ona wektory ze zbioru $\left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)}\right\}$ na wektory wyjściowe ze zbioru $\left\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(p)}\right\}$. Jeżeli natomiast asocjacje są postaci

$$\mathbf{x}^{(i)} \rightarrow \mathbf{x}^{(i)}, \qquad i = 1, 2, \ldots, p, \tag{6.26}$$

to pamięć nazywa się *autoasocjacyjną*. Podczas odczytu odwzorowuje wektory ze zbioru $\left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)}\right\}$ w siebie.

Inną ważną cechą pamięci jest charakter jej odpowiedzi na zakłócone wzorcowe wektory wejściowe $\mathbf{x}$. W sieciach *progowych* mamy $\mathbf{x} + \boldsymbol{\Delta} \rightarrow \mathbf{y}$, gdzie $\boldsymbol{\Delta}$ jest niewielkim błędem. W sieciach *interpolacyjnych* $\mathbf{x} + \boldsymbol{\Delta} \rightarrow \mathbf{y} + \boldsymbol{\eta}$, gdzie $|\boldsymbol{\eta}| \rightarrow 0$, gdy $|\boldsymbol{\Delta}| \rightarrow 0$.

Pamięci asocjacyjne realizowane są jako statyczne i dynamiczne co przedstawia rys. 6.31, gdzie pokazano ich schematy blokowe. Pamięci statyczne są sieciami bez sprzężeń zwrotnych i ich odczyt następuje po jednorazowym przejściu sygnału od wejścia do wyjścia. Pamięci dynamiczne (rekurencyjne) są sieciami ze sprzężeniami



Rys. 6.31: Struktury pamięci asocjacyjnych: (a) pamięć statyczna, (b) pamięc dynamiczna

zwrotnymi i ich odczyt następuje po osiągnięciu przez sieć stanu stabilnego po wielokrotnym przejściu sygnałów sprzężenia zwrotnego przez warstwy neuronów.

Pamięć statyczna realizuje odwzorowanie

$$\mathbf{y}^k = \mathrm{M}_1\left[\mathbf{x}^k\right], \tag{6.27}$$

gdzie k oznacza chwilę czasu, a operator M_1 jest układem funkcji nieliniowych. Należy zauważyć, że sygnał wyjściowy układu w danej chwili zależy tylko od aktualnego sygnału wejściowego i nie zależy od sygnału wejściowego w przeszłości.

Pamięci dynamiczne są układami, w których sygnały wyjściowe w danej chwili są zależne od sygnałów wejściowych w przeszłości. Korzystając z koncepcji stanu, działanie pamięci dynamicznej można opisać równaniem

$$\begin{aligned} \mathbf{v}^{k+1} &= \mathrm{M}_2\left[\mathbf{v}^k\right], \quad \mathbf{v}^0 = \mathbf{x}^0 \\ \mathbf{y}^{k+1} &= \mathbf{v}^{k+1} \end{aligned} \tag{6.28}$$

gdzie $\mathbf{v}^k$ oznacza stan w chwili k, a M_2 jest układem funkcji nieliniowych. Operator M_2 odziaływuje w bieżącej chwili k na wektor stanu $\mathbf{v}^k$, tworząc nowy stan $\mathbf{v}^{k+1}$. Stan początkowy $\mathbf{v}^0$ jest zadany poprzez $\mathbf{x}^0$, a odczytywaną wartością jest sygnał $\mathbf{y}^{k+1}$ tożsamy ze stanem $\mathbf{v}^{k+1}$

W następnych punktach zajmiemy się pamieciami dynamicznymi.

Architektura pamięci dynamicznej (rekurencyjnej)

Na rys. 6.32 przedstawiono schemat pamięci rekurencyjnej. Użyto neurony bipolarne, dyskretne (stany i wyjścia są reprezentowane przez sygnały -1 i $+1$.

Algorytm zapisu

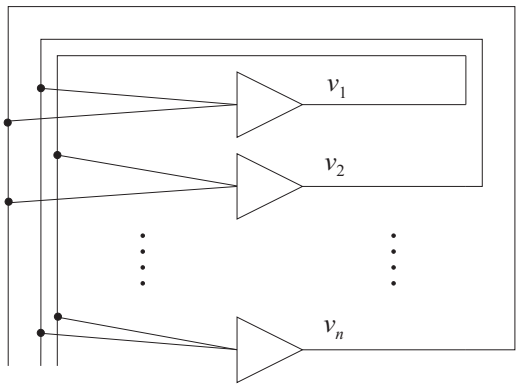
Przy zapisie do pamięci wektorów wzorcowych $\mathbf{s}^{(m)}$, $m = 1, 2, \ldots, p$, o składowych $+1, -1$, wagi wyznaczane są według następującego algorytmu Hebba:

$$\mathbf{W} = \sum_{m=1}^{p} \mathbf{s}^{(m)}\mathbf{s}^{(m)t} - p\mathbf{I}, \tag{6.29}$$

gdzie $\mathbf{I}$ jest macierzą jednostkową, lub

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^{p} s_i^{(m)} s_j^{(m)}, \tag{6.30}$$

gdzie δ_{ij} oznacza funkcję delta Kroneckera.



Rys. 6.32: Schemat pamięci dynamicznej (rekurencyjnej). Przedstawione na rysunku neurony są dyskretne i bipolarne

Proces odczytu

Zmianę wartości i-tej składowej stanu autoasocjacyjnej pamięci Hopfielda można wyrazić wzorem

$$v_i^{'} = \operatorname{sgn}\left(\sum_{j=1}^{n} w_{ij} v_j\right) \tag{6.31}$$

Na tej podstawie pamięć dynamiczna odczytywana jest według następującego algorytmu zwanego odczytem asynchronicznym:

1. Ustawienie stanu początkowego $\mathbf{v}^{(0)} = \mathbf{x}$.
2. Ustawienie liczb $1, 2, \ldots, n$ w losowej kolejności $\alpha_1, \alpha_2, \ldots, \alpha_n$.
3. Dla i od 1 do n wykonanie

$$v'_{\alpha_i} = \operatorname{sgn}\left(\sum_{j=1}^{n} w_{\alpha_i j} v_j\right).$$

4. Jeżeli $v'_{\alpha_i} = v_{\alpha_i}$, $i = 1, 2, \ldots, n$ (czyli w danym cyklu nie było zmian), to zakończenie procesu odczytu i uznanie sygnałów $v'_1, v'_2, \ldots, v'_n$ za sygnały wyjściowe; w przeciwnym razie przejście do 2.

Bardzo użyteczną wielkością charakterystyczną stanu pamięci autoasocjacyjnej Hopfielda jest funkcja

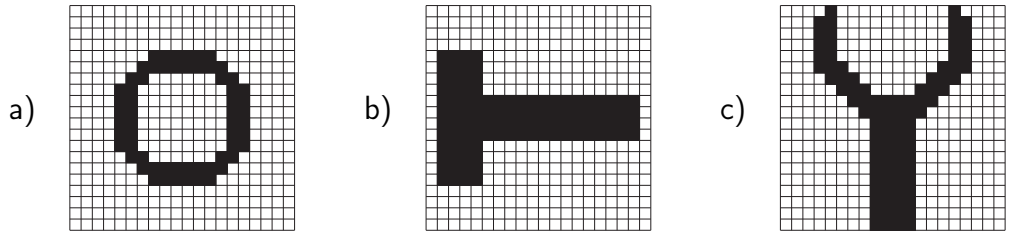
$$E(\mathbf{v}) = -\tfrac{1}{2}\mathbf{v}^t \mathbf{W} \mathbf{v}. \tag{6.32}$$

zwana *funkcją energii*. Asynchroniczne uaktualnianie stanu podczas odczytu powoduje (co można łatwo wykazać), że wartość funkcji energii maleje przy każdej zmianie stanu, a minimalne przyrosty energii sieci w kolejnych zmianach stanu są ustalone (są krotnością liczby 1 dla macierzy wag obliczonych według wzoru (6.29)). Można także pokazać, że jeżeli ze stanem, w którym sieć się znajduje sąsiadują stany o mniejszej wartości funkcji energii, to sieć napewno przejdzie do jednego z tych stanów o mniejszej energii. Stąd, ponieważ istnieje dolna granica wartości funkcji

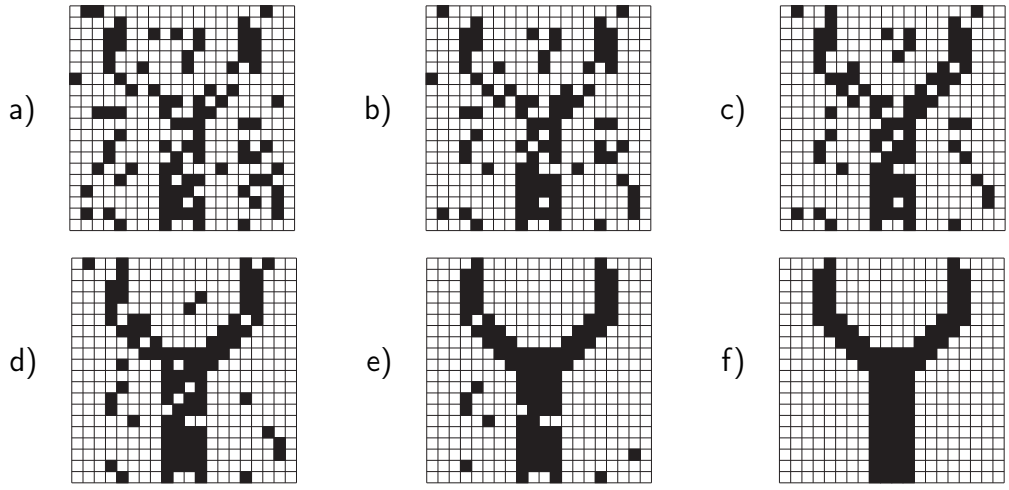
energii, to sieć musi osiągnąc jakiś stan stabilny, który znajduje się w minimum energii lokalnym lub globalnym.

Obliczając ze wzoru (6.32) energię układu dla wektora komplementarnego $-\mathbf{v}$, łatwo widać, że $E(-\mathbf{v}) = E(\mathbf{v})$. Stąd wynika, że wielkość minimum funkcji $E(-\mathbf{v})$ jest taka sama jak funkcji $E(\mathbf{v})$. Prowadzi to do ważnego wniosku, że układ może zmierzać zarówno do punktu $\mathbf{v}$, jak i $-\mathbf{v}$. Do którego z nich dojdzie, zależy od odległości punktu reprezentującego stan początkowy do punktów $\mathbf{v}$ i $-\mathbf{v}$.

Pamięć autoasocjacyjna Hopfielda jest często nazywana w literaturze dekoderem korygującym przekłamania. Odzwierciedla to fakt, że pamięć na podstawie podanego na wejście wektora różniącego sie nieznacznie od wektora zapamiętanego odtwarza na wyjściu wektor zapamiętany, czyli jest układem progowym. Ilustrują to rysunki 6.33 i 6.34 gdzie przedstawiono przykładowe obrazy wzorcowe i przebieg procesu odczytu przy podaniu na wejście pamięci zniekształconego wzorca.



Rys. 6.33: Przykładowe obrazy wzorcowe zapisane do pamięci rekurencyjnej



Rys. 6.34: Przykładowe etapy odczytu pamięci rekurencyjnej, do której zapisano obrazy przedstawione na rys. 6.33

Ortogonalność wzorców

Przyjmijmy, że w pamięci zapisano obrazy $\mathbf{s}^{(m)}$, $m = 1, 2, \ldots, p$, a na wejście podany jest obraz $\mathbf{s}^{(0)}$. Pobudzenie łączne i-tego neuronu dla reguły aktualizacji danej wzorem (6.31) ma wartość

$$net_i^{(0)} = \sum_{j=1}^{n} w_{ij} s_j^{(0)}. \tag{6.33}$$

Ciekawa sytuacja powstaje, gdy w pamięci zapisane są wektory ortogonalne $\mathbf{s}^{(m)}$. Na podstawie (6.29) i (6.33) możemy wyznaczyć wektor pobudzeń łącznych, gdy stanem sieci jest wektor $\mathbf{s}^{(m')}$:

$$\mathbf{net}^{(m')} = \left(\sum_{m=1}^{p} \mathbf{s}^{(m)} \mathbf{s}^{(m)t} - p\mathbf{I} \right) \mathbf{s}^{(m')}.$$

Przyjmując, że wektor $\mathbf{s}^{(m')}$ jest jednym z wektorów wzorcowych, oraz na podstawie warunku ortogonalności, który dla wektorów o składowych ± 1 przybiera postać $\mathbf{s}^{(i)t}\mathbf{s}^{(j)} = 0$ dla $i \neq j$ i $\mathbf{s}^{(i)t}\mathbf{s}^{(j)} = n$ dla $i = j$, otrzymujemy natychmiast

$$\mathbf{net}^{(m')} = (n - p)\mathbf{s}^{(m')}.$$

Przyjmując, że spełniona jest nierówność $n > p$, widać, że stan $\mathbf{s}^{(m')}$ jest stanem stabilnym.

Pojemność pamięci rekurencyjnej

Jednym z najważniejszych parametrów pamięci autoasocjacyjnej jest jej pojemność, czyli liczba obrazów, które może efektywnie zapamiętać. Przeprowadzimy analizę pojemności pamięci przy założeniu, że wszystkie składowe wektorów wzorcowych $s_i^{(m)}$, $i = 1, 2, \ldots, n$, $m = 1, 2, \ldots, p$, są niezależnymi dyskretnymi zmiennymi losowymi przyjmującymi wartości ± 1 z prawdopodobieństwem $P = 1/2$.

Podstawiając do wzoru (6.33) wzór (6.30) otrzymujemy

$$net_i^{(0)} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{m=1}^{p} s_i^{(m)} s_j^{(m)} s_j^{(0)} = \sum_{m=1}^{p} s_i^{(m)} \sum_{\substack{j=1 \\ j \neq i}}^{n} s_j^{(m)} s_j^{(0)}. \tag{6.34}$$

W wyrażeniu (6.34) na wartość pobudzenia łącznego i-tego neuronu, nie uwzględniając zerowych wag na głównej przekątnej, kładąc $\mathbf{s}^{(0)} = \mathbf{s}^{(m')}$ oraz wyłączając przed sumy składnik $m = m'$, otrzymujemy

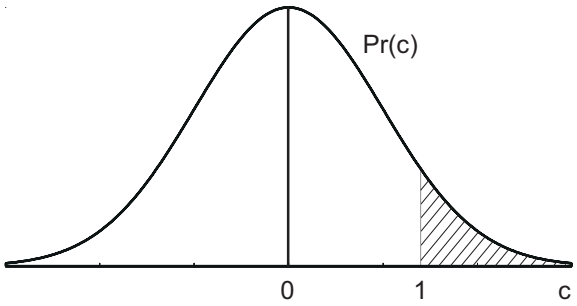
$$net_i^{(m')} = n s_i^{(m)} + \sum_{j=1}^{n} \sum_{\substack{m=1 \\ m \neq m'}}^{p} s_i^{(m)} s_j^{(m)} s_j^{(m')}. \tag{6.35}$$

Wektor stanu pamięci $\mathbf{v}$ jest stabilny, gdy $\operatorname{sgn}(net_i) = \operatorname{sgn}\left(\sum_{j=1}^{n} w_{ij} v_j\right) = v_i$, $i = 1, 2, \ldots, n$. Stąd na podstawie (6.35) widać, że składowa $s_i^{(m')}$ jest stabilna, gdy znak sumy w (6.35) jest taki sam jak znak $s_i^{(m')}$ lub gdy znaki te są przeciwne, ale wartość sumy jest mniejsza od n.

Wprowadźmy parametr

$$c_i^{(m')} = -\frac{1}{n} s_i^{(m')} \sum_{j=1}^{n} \sum_{\substack{m=1 \\ m \neq m'}}^{p} s_i^{(m)} s_j^{(m)} s_j^{(m')}$$

nazywany *przesłuchem* i będący iloczynem składowej $s_i^{(m')}$ i sumy z wyrażenia (6.35). Łatwo zauważyć, że gdy $c_i^{(m')} < 1$, to składowa $s_i^{(m')}$ jest stabilna. Wyznaczymy



Rys. 6.35: Rozkład prawdopodobieństwa przesłuchu c. Zakreskowane pole przedstawia prawdopodobieństwo niestabilności jednej składowej

teraz rozkład prawdopodobieństwa $c_i^{(m')}$, a następnie na jego podstawie różne wielkości określające pojemność sieci. Parametr $c_i^{(m')}$ jest iloczynem zmiennej losowej $s_i/$ i sumy $n(p-1) \approx np$ zmiennych losowych przyjmujących wartości ± 1 z prawdopodobieństwem $P = 1/2$. Z rachunku prawdopodobieństwa wiadomo, że suma dwuwartościowych zmiennych losowych ma rozkład dwumianowy, który w naszym przypadku przybiera postać

$$\Pr(c) = 2^{np} \left(\begin{array}{c} np \\ (c+p)n/2 \end{array} \right), \qquad c = -p, \frac{2}{n} - p, \frac{4}{n} - p, \ldots, p.$$

Dla dużych wartości np rozkład ten zbliża się do rozkładu normalnego o wariancji $\sigma^2 = p/$ (patrz np. Feller (1968), str. 162)

$$\Pr(c) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{p/}} \exp\left(-\frac{c^2}{2p/} \right). \tag{6.36}$$

Rozkład ten przedstawiono na rys. 6.35. Na podstawie rozkładu (6.36) można znaleźć prawdopodobieństwo niestabilności pojedynczej składowej, równe prawdopdobieństwu tego, że $c > 1$:

$$\Pr(c > 1) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{p/}} \int_1^\infty \exp\left[-\frac{c^2}{2p/} \right] dc = \frac{1}{2} \left[1 - \Phi\left(\sqrt{\frac{n}{p}} \right) \right], \tag{6.37}$$

gdzie funkcja

$$\Phi(x) \stackrel{\text{def}}{=} \frac{2}{\sqrt{2\pi}} \int_0^x \exp\left(-t^2/2 \right) dt$$

nazywa się całką prawdopodobieństwa i jej wartości są podawane w tablicach.

Korzystając ze wzoru (6.37) można na podstawie danego prawdopodobieństwa niestabilności obliczyć stosunek $p/$. Przykładowo dla $\Pr(c > 1) = 0,001$ otrzymujemy $p/ = 0,105$, a dla $\Pr(c > 1) = 0,1$ mamy $p/ = 0,61$.

Obliczone prawdopodobieństwo niestabilności jednej składowej wskazuje tylko niestabilność początkową, natomiast nie daje informacji, czy początkowa niestabilność nie pociągnie za sobą w kolejnych krokach przełączenia innych składowych. Na przykład początkowa niestabilność 1% składowych i przyjmowanie przez nie

w rezultacie przeciwnych wartości mogłaby być do przyjęcia, natomiast oczywiście nie do przyjęcia byłaby początkowa niestabilność powodująca w efekcie kilkudziesięcioprocentowy błąd. Z bardziej złożonych analiz wynika, że krytyczną wartością jest $p/ = 0,138$. Dla większych wartości początkowa niewielka liczba niestabilnych składowych w kolejnych krokach lawinowo zmienia inne składowe dając ostatecznie zupełnie błędny odczyt.

W innej definicji pojemności żąda się, aby z prawdopodobieństwem P_1 bliskim jedności wszystkie wzorce były stabilne, czyli $[1 - Pr(c > 1)]^{np} > P_1$. Po zastosowaniu słusznego dla dużych np przybliżenia $[1 - \Pr(c > 1)]^{np} \approx -\Pr(c > 1)np + 1$ warunek przybiera postać

$$\Pr(c > 1) < \frac{1 - P_1}{np}. \tag{6.38}$$

Dla małych $p/$ można zastosować rozwinięcie asymptotyczne funkcji błędu:

$$1 - \operatorname{erf}(x) \to \frac{1}{\sqrt{\pi}x} \exp\left(-x^2\right) \qquad \text{dla} \qquad x \to \infty.$$

Stosując to rozwinięcie oraz logarytmując obie strony nierówności (6.38) otrzymujemy

$$\ln\left[\Pr(c > 1)\right] = -\ln 2 - \frac{1}{2}\ln \pi - \frac{1}{2}\ln\left(\frac{n}{2p}\right) - \frac{n}{2p} < \ln(1 - P_1) - \ln(np)$$

lub dla dużych n

$$\frac{n}{2p} > \ln(np).$$

Przybliżonym rozwiązaniem tej nierówności, zbliżającym się ze wzrostem n do rozwiązania dokładnego, jest

$$p < \frac{n}{4 \ln n}. \tag{6.39}$$

Gdy liczba zapamiętanych obrazów p spełnia warunek (6.39), wtedy wszystkie zapamiętane obrazy są stabilne z prawdopodobieństwem bliskim jedności.

6.5. Systemy neuronowo–rozmyte

Systemy neuronowo–rozmyte (ang. ANFIS od Adaptive Neuro–Fuzzy Inference System) są to systemy wnioskowania rozmytego, w których wartości parametrów funkcji przynależności dobierane są w procesie uczenia z nauczycielem. Nazwa tych systemów bierze się z podobieństwa struktury obliczeń systemu rozmytego do struktury warstwowej sieci neuronowej i z algorytmu uczenia, którego jeden z etapów jest identyczny z metodą propagacji wstecznej.

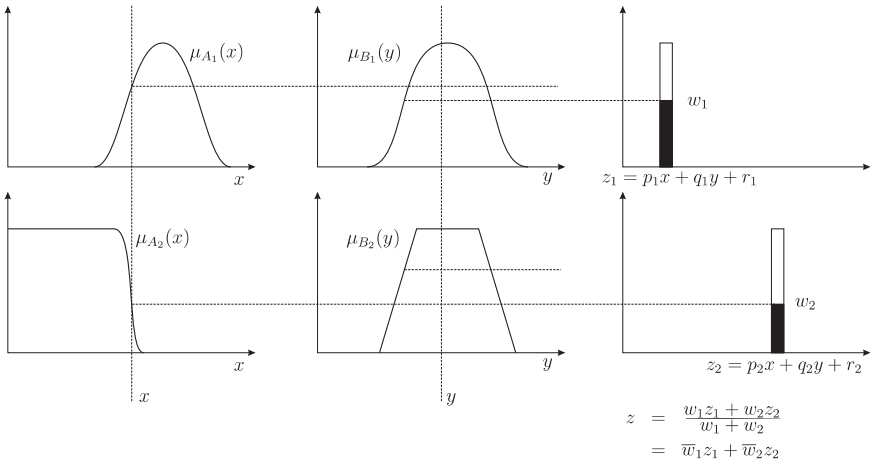
6.5.1. Struktura systemu ANFIS

Typową strukturą uczącego sie systemu rozmytygo (ANFIS) jest system Sugeno. Rozpatrzmy przykładowy system Sugeno o dwóch wejsciach i następujacych regułach:

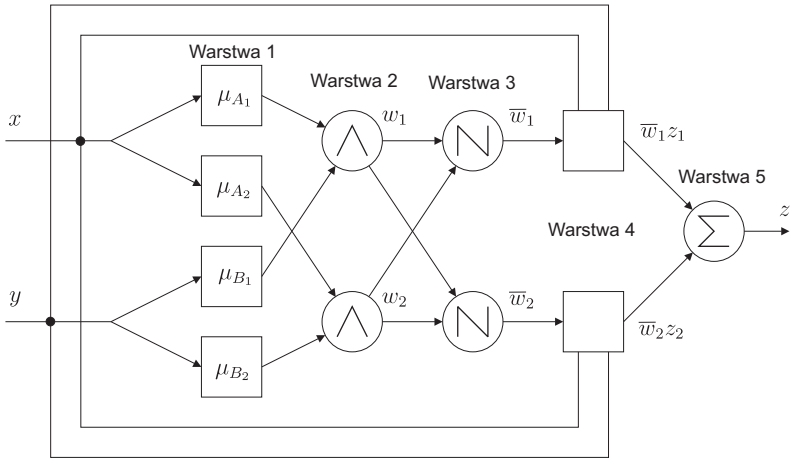
$$\begin{array}{l} \text{Jeżeli } x \text{ jest } A_1 \text{ i } y \text{ jest } B_1 \text{ to } z_1 = p_1 x + q_1 y + r_1 \\ \text{Jeżeli } x \text{ jest } A_2 \text{ i } y \text{ jest } B_2 \text{ to } z_2 = p_2 x + q_2 y + r_2 \end{array} \tag{6.40}$$

gdzie x i y są wejściami, A_i, B_i, $i = 1, 2$ są zbiorami rozmytymi, z_1, z_2 są położeniami funkcji przynależnosci wyjść, p_i, q_i, r_i, $i = 1, 2$ są parametrami.

Oznaczając przez $\mu_{A_i}(x)$, $\mu_{B_i}(y)$, $i = 1, 2$ funkcje przynależności zbiorów A_i, B_i, na rysunku 6.36 przedstawiono operację wymagane do obliczenia wartości wyjścia systemu,z. Operacje te mogą być wykonywane przez warstwową strukture obliczeniową przedstawioną na rys. 6.37. Warstwy te realizują następujące obliczenia:



Rys. 6.36: Ilustracja obliczeń realizowanych przez przykładowy system Sugeno



Rys. 6.37: Struktura obliczeń systemu Sugeno z rysunku 6.36

Warstwa 1 (adaptacyjna[4]) Warstwa oblicza wartości funkcji przynależności μ dla wejść x lub y

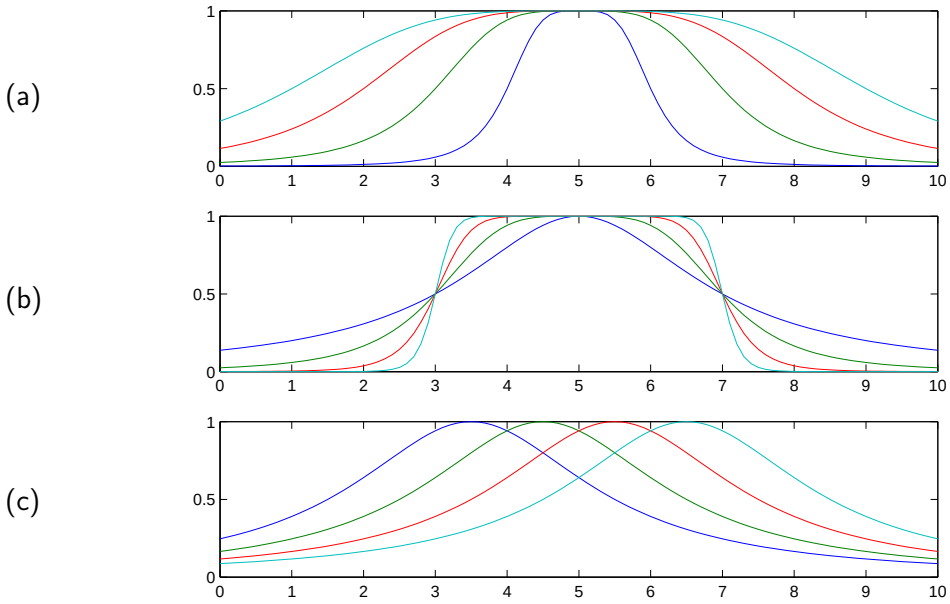
$$O_{1,i} = \begin{cases} \mu_{A_i}(x) & \text{dla} \quad i = 1, 2 \\ \mu_{B_{i-2}}(y) & \text{dla} \quad i = 3, 4 \end{cases} \tag{6.41}$$

[4]Określenie adaptacyjna oznacza tutaj, że wartości parametrów występujących w tej warstwie bedą podlegały zmianom w czasie uczenia.

Typową funkcją przynależności jest uogólniona funkcja dzwonowa wyrażona wzorem

$$\mu(x) = \frac{1}{1 + |\frac{x-c}{a}|^{2b}}$$

gdzie a, b, c są parametrami. Parametry te bedziemy nazywali *parametrami przesłanek*. Kształt funkcji dzwonowej dla różnych wartości tych parametrów przedstawiony jest na rysunku 6.38.



Rys. 6.38: Wykresy uogólnionej funkcji dzwonowej dla różnych wartości parametrów a, b, c: (a) $a = 2, 3, 4,\ b = 2,\ c = 5$, (b) $a = 2,\ b = 1, 2, 4, 8,\ c = 5$, (c) $a = 1,\ b = 1,\ c = 3,5,\ 4,5,\ 5,5,\ 6,5$

Warstwa 2 (ustalona) Węzły tej warstwy obliczają iloczyny sygnałów na ich wejściach. Na ich wyjściach otrzymujemy siły zadziałania poszczególnych reguł.

$$O_{2,i} = w_i = \mu_{A_i}(x)\mu_{B_i}(y), \qquad i = 1, 2 \tag{6.42}$$

Warstwa 3 (ustalona) Węzły tej warstwy obliczają znormalizowane siły zadziałania poszczególnych reguł według wzoru

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \qquad i = 1, 2 \tag{6.43}$$

Warstwa 4 (adaptacyjna) Węzły tej warstwy obliczają następujące iloczyny

$$O_{4,i} = \overline{w}_i z_i = \overline{w}_i(p_i x + q_i y + r_i), \qquad i = 1, 2 \tag{6.44}$$

Parametry p, q, r nazywane są *parametrami konsekwencji.*

Warstwa 5 (ustalona) Pojedynczy węzeł tej warstwy oblicza sumę sygnałów z węzłów warstwy 4

$$z = O_{5,1} = \sum_{i=1}^{2} \overline{w}_i z_i = \frac{\sum_i w_i z_i}{\sum_i w_i} \tag{6.45}$$

6.5.2. Hybrydowy algorytm uczenia

Na podstawie wzorów (6.41) do (6.45) mamy

$$\begin{aligned} z &= \frac{w_1}{w_1+w_2}z_1 + \frac{w_2}{w_1+w_2}z_2 \\ &= \overline{w}_1(p_1x+q_1y+r_1) + \overline{w}_2(p_2x+q_2y+r_2) \\ &= (\overline{w}_1x)p_1 + (\overline{w}_1y)q_1 + (\overline{w}_1)r_1 + \\ &+ (\overline{w}_2x)p_2 + (\overline{w}_2y)q_2 + (\overline{w}_2)r_2 \end{aligned}$$

Z powyższego wzoru widać, że wyjście systemu Sugeno z jest zależne liniowo od parametrów konsekwencji p, q, r. Ponieważ $\overline{w}_1$ i $\overline{w}_1$ zależą od funkcji przymależności μ, a te z kolei zależą nieliniowo od parametrów przesłanek (np. parametrów a, b, c w przypadku uogólnionej funkcji dzwonowej), to tym samym wyjście z jest nieliniowo zależne od parametrów przesłanek.

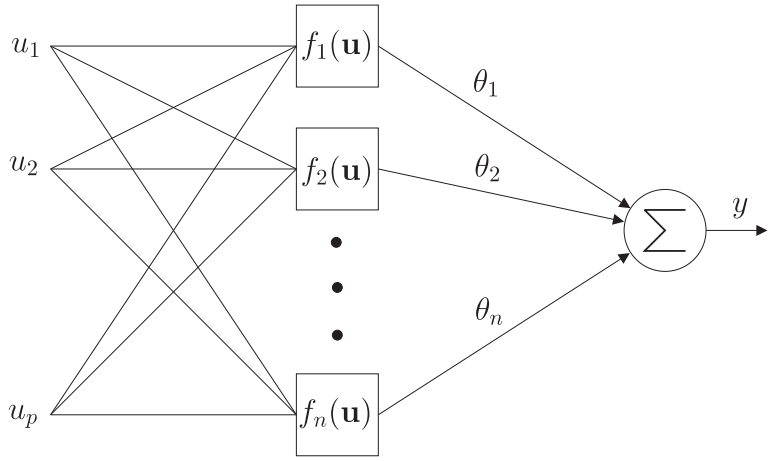
Liniowa zależność wyjścia od jednych parametrów i nieliniowa od innych, sugeruje zastosowanie róznych algorytmów do poszczególnych grup parametrów. Metodę najmniejszych kwadratów dla parametrów wpływających liniowo i metodę gradientową dla parametrów wpływających nieliniowo.

Metoda najmniejszych kwadratów

Mamy model liniowy opisany równaniem (patrz też rys. 6.39)

$$y = \theta_1 f_1(\mathbf{u}) + \theta_2 f_2(\mathbf{u}) + \ldots + \theta_n f_n(\mathbf{u}) \tag{6.46}$$

gdzie y jest wyjściem, $\mathbf{u} = [u_1, \ldots, u_p]$ jest wektorem wejściowym, f_1, ..., f_n sa znanymi funkcjami, a $\theta_1, \theta_2, \ldots, \theta_n$ parametrami.



Rys. 6.39: Model liniowy – ilustracja do równania (6.46)

Metoda najmniejszych kwadratów rozwiązuje następujące zadanie. Mamy zbiór par sygnałów wejściowych i sygnału wyjściowego (par sygnałów uczących) $\{(\mathbf{u}_i; y_i), i = 1, \ldots, m\}$. Należy tak dobrać parametry modelu liniowego $\theta_1, \theta_2, \ldots, \theta_n$ aby podając na jego wejście sygnały $\{\mathbf{u}_i, i = 1, \ldots, m\}$, otrzymać na wyjściu sygnały możliwie bliskie $\{y_i, i = 1, \ldots, m\}$ (w sensie kryterium kwadratowego) .

Podstawiając do równania (6.46) kolejne sygnały wejściowe $\{\mathbf{u}_i, i = 1, \ldots, m\}$ otrzymujemy układ równań liniowych, zwanych równaniami regresji

$$\begin{array}{rcl} f_1(\mathbf{u}_1)\theta_1 + f_2(\mathbf{u}_1)\theta_2 + \ldots + f_n(\mathbf{u}_1)\theta_n & = & y_1 \\ f_1(\mathbf{u}_2)\theta_1 + f_2(\mathbf{u}_2)\theta_2 + \ldots + f_n(\mathbf{u}_2)\theta_n & = & y_2 \\ \vdots & \vdots & \vdots \\ f_1(\mathbf{u}_m)\theta_1 + f_2(\mathbf{u}_m)\theta_2 + \ldots + f_n(\mathbf{u}_m)\theta_n & = & y_m \end{array} \tag{6.47}$$

Stosując oznaczenia

$$\mathbf{A} = \begin{bmatrix} f_1(\mathbf{u}_1) \ldots f_n(\mathbf{u}_1) \\ \vdots \\ f_1(\mathbf{u}_m) \ldots f_n(\mathbf{u}_m) \end{bmatrix}, \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

równanie (6.47) możemy zapisać w postaci

$$\mathbf{A}\boldsymbol{\theta} = \mathbf{y} \tag{6.48}$$

Jeżeli $m = n$ (czyli liczba równań jest równa liczbie niewiadomych), to otrzymujemy rozwiązanie dokładne

$$\boldsymbol{\theta} = \mathbf{A}^{-1}\mathbf{y}$$

Jeżeli (co jest typową sytuacją) $m > n$, to układ równań nie ma rozwiązania[5]. Wtedy możemy poszukiwać rozwiązania przybliżonego. W tym celu zastąpimy równanie (6.48) równaniem

$$\mathbf{A}\boldsymbol{\theta} + \mathbf{e} = \mathbf{y} \tag{6.49}$$

gdzie wektor $\mathbf{e}$ reprezentuje róznicę wartości lewych i prawych stron równania.

Korzystając z równania (6.49) poszukiwanie rozwiązania przybliżonego sprowadza się do znalezienia rozwiązania $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ minimalizującego sumę kwadratów składowych wektora $\mathbf{e}$, czyli minimum wyrażenia

$$E(\boldsymbol{\theta}) = \mathbf{e}^T\mathbf{e} = (\mathbf{y} - \mathbf{A}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{A}\boldsymbol{\theta}) = \sum_{i=1}^{m}(y_i - \mathbf{a}_i^T\boldsymbol{\theta})^2$$

Po prostych, nie przytaczanych tu przekształceniach (patrz np. Jang 1997) oraz spełnieniu pewnych ogólnych założeń, otrzymujemy rozwiazanie

$$\hat{\boldsymbol{\theta}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \tag{6.50}$$

Z powyższego wynika, że jeżeli układ uczący się możemy przedstawić równaniem (6.46), gdzie parametry $\theta_1, \theta_2, \ldots, \theta_n$ dobierane sa w procesie uczenia, to możemy je wyznaczyć w jednym kroku obliczeniowym, danym równaniem (6.50).

[5] za wyjątkiem pewnych szczególnych doborów danych wejściowych

Algorytm uczenia systemu rozmytego

W uczącym się systemie rozmytym mamy dwa rodzaje parametrów:

zbiór parametrów przesłanek (nieliniowo wpływających na wynik) (a, b, c),
zbiór parametrów konsekwencji (wpływających liniowo na wynik) (p, q, r)

Każdy cykl nauki podzielony jest na dwa etepy. W etapie pierwszym parametry konsekwencji są wyznaczane meodą najmniejszych kwadratów, a w etapie drugim parametry przesłanek są wyznaczane metodą gradientową (metodą propagacji wstecznej).

Kroki algorytmu

1 Parametry przesłanek są ustalone i obliczenia w systemie sa wykonywane przez warstwy od 1 do 4, po kolei dla wszystkich danych wejściowych i na tej podstawie tworzone sa równania regeresji.

2 Parametry konsekwencji obliczane sa metoda najmniejszych kwadratów.

3 Przy ustalonych wartościach wszystkich parametrów obliczana jest odpowiedź sieci dla wszystkich danych wejściowych.

4 Obliczny jest błąd jako suma kwadratów różnic pomiędzy wyjściami systemu i oczekiwanymi odpowiedziami dla wszystkich danych wejściowych.

5 Wykonywany jest jeden cykl metody propagacji wstecznej wyznaczający nowe wartości parametrów przesłanek.

6 Przejście do kroku 1.

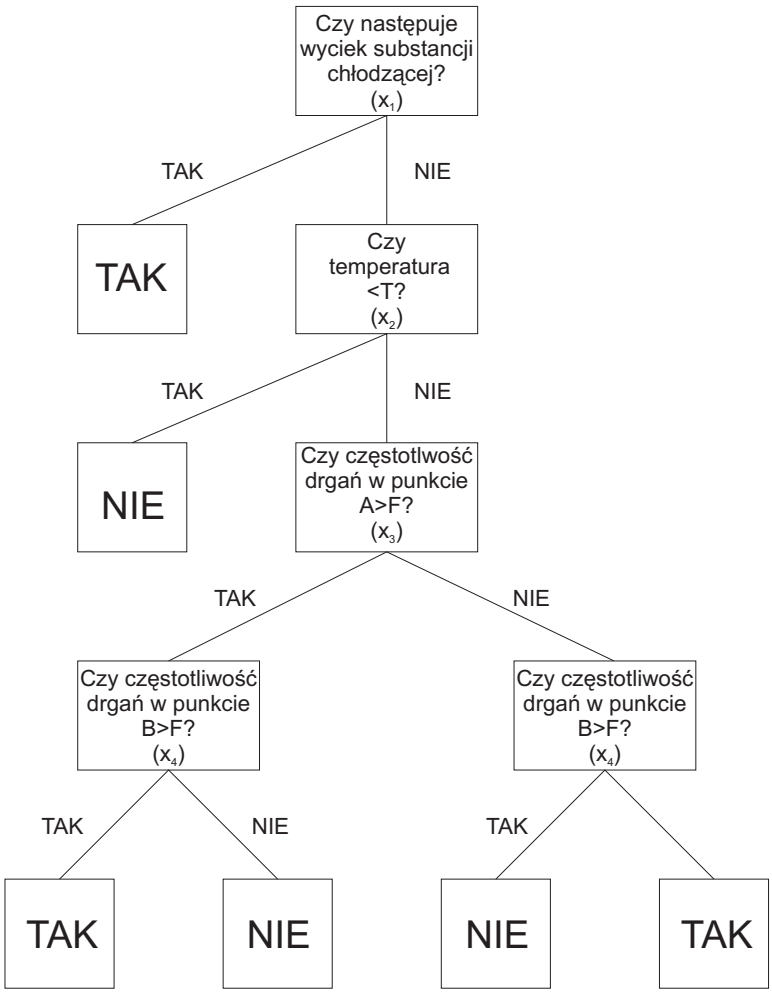
6.6. Uczenie drzew decyzyjnych

Uczenie drzew decyzyjnych należy do metod uczenia, w których uzyskiwana wiedza wyrażona jest w postaci jawnej – a ponadto typowe metody uczenia drzew decyzyjnych są jednorazowe (nie są iteracyjne).

Drzewo decyzyjne jest systemem, który na podstawie podanych danych generuje konkluzje będącą decyzją typu *tak/nie*. Celem uczenia jest uzyskanie drzewa decyzyjnego na podstawie zadanej pewnej liczby przykładów zawierających wartości atrybutów danych i odpowiadające im decyzje. Drzewa decyzyjne generujące odpowiedzi *tak/nie* są specyficznym zapisem funkcji logicznych. W poniższym przykładzie zostaną przedstawione podstawowe zasady uczenia drzew decyzyjnych.

Przykład 6.8

Na rys. 6.40 przedstawiono przykład drzewa decyzyjnego generującego sygnał alarmu w pewnym urządzeniu w zależności od wartości czterech parametrów: występowania wycieku substancji chłodzącej (x_1), temperatury $< T$ (x_2), częstotliwości drgań w punkcie pomiarowym $A > F$ (x_3) oraz częstotliwości drgań w punkcie pomiarowym $B > F$ (x_4).



Rys. 6.40: Przykład drzewa decyzyjnego

Zadaniem uczenia jest utworzenie drzewa decyzyjnego z rys. 6.40 na podstawie znajomości 6 zestawów pomiarowych (obrazów) przedstawionych w tablicy 6.1.

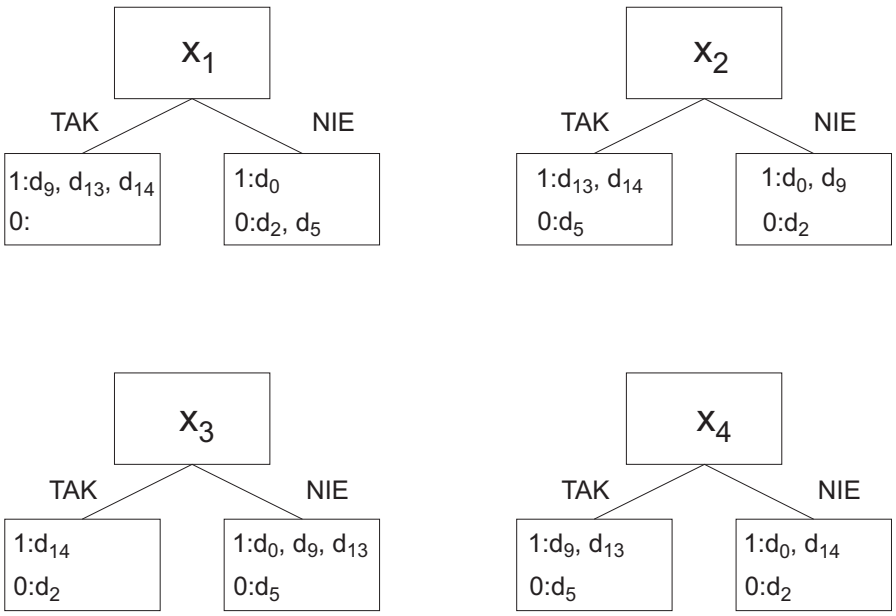
numer obrazu	wyciek x_1	temp. $< T$ x_2	częst. $A > F$ x_3	częst. $B > F$ x_4	ALARM
d_0	0	0	0	0	1
d_2	0	0	1	0	0
d_5	0	1	0	1	0
d_9	1	0	0	1	1
d_{13}	1	1	0	1	1
d_{14}	1	1	1	0	1

Tablica 6.1: Sześć zestawów wartości parametrów i decyzji stanowiący zbiór uczący (obrazy uczące) drzewa decyzyjnego

Podstawową zasadą uczenia indukcyjnego jest tzw. zasada *brzytwy Ockhama* mówiąca, że: „bytów nie należy mnożyć więcej niż potrzeba" co tutaj można wyrazić jako: najbardziej prawdopodobną hipotezą jest hipoteza najprostsza, która jest

zgodna z wszystkimi obserwacjami. W myśl tej zasady poszukujemy zmiennej, która ma największy wpływ na podejmowaną decyzję (np. jej określona wartość pozwala na jednoznaczną decyzję); zmienna ta staje się korzeniem budowanego drzewa. Następnie tworzymy fragment drzewa decyzyjnego oparty o tą zmienną i dalej kolejno wyznaczamy następne zmienne o największych wpływach na decyzje i tworzymy dalsze fragmenty drzewa. Przy wyznaczaniu wpływu kolejnych zmiennych bierzemy pod uwagę tylko te obrazy uczące, które spełniają warunki aby dostać się do określonego węzła drzewa.

W omawianym przykładzie, na podstawie rysunku 6.41 widać, że naistotniejszą zmienną jest zmienna x_1, której pozytywna wartość zawsze powoduje alarm. Po jej wybraniu otrzymujemy pierwszy węzeł drzewa decyzyjnego (rysunek 6.42). Gdy zmienna x_1 przyjmuje wartość 1, to decyzją jest alarm; gdy zmienna x_1 przyjmuje wartość 0, to należy na nowo określić zmienną (spośród pozostałych) o największym wpływie na decyzję biorąc pod uwagę dane uczące d_0, d_2 i d_5. Postępując wg. tej zasady otrzymujemy całe drzewo przedstawione na rysunku 6.42. Drzewo to w niewielkim stopniu różni się od drzewa oryginalnego pokazanego na rys. 6.40, co ilustruje tablica 6.2. ■

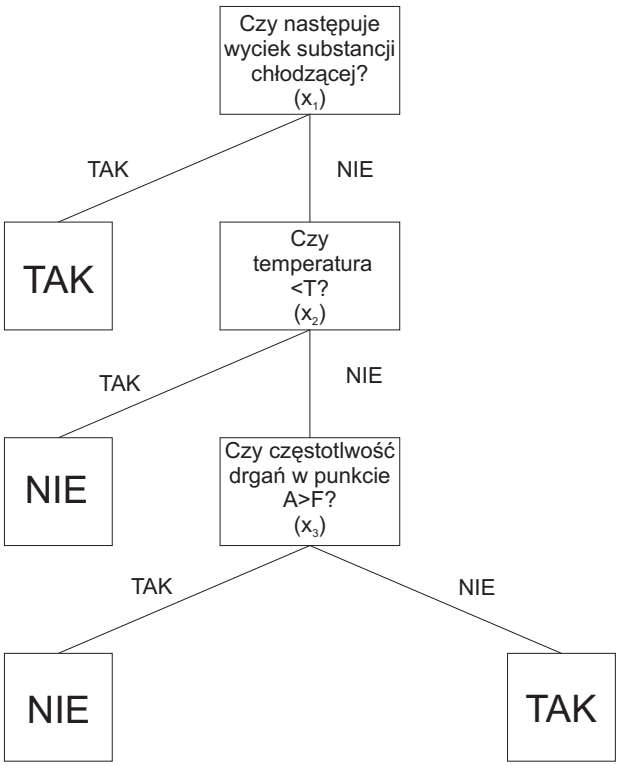


Rys. 6.41: Wyznaczanie argumentu o największym wpływie – okazuje się nim zmienna x_1

6.6.1. Metoda wyboru atrybutów

Najczęściej we wczesnych etapach tworzenia drzewa decyzyjnego nie ma zmiennych, których określona wartość pozwala na jednoznaczną decyzję. Wtedy jako miarę wpływu zmiennej na podejmowaną decyzję przyjmuje się entropię [6] rozkładu prawdopodobieństwa decyzji wynikających z przyjęcia tej zmiennej jako węzła drzewa decyzyjnego.

[6]Jest to zarazem miara ilości informacji



Rys. 6.42: Drzewo decyzyjne otrzymane w procesie uczenia na podstawie 6 danych uczących

x_1	x_2	x_3	x_4	f	$\hat{f}$
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

Tablica 6.2: Funkcja logiczna f reprezentująca zadane drzewo decyzyjne z rysunku 6.40 oraz funkcja logiczna $\hat{f}$ reprezentująca drzewo decyzyjne otrzymane w procesie uczenia przedstawione na rys. 6.42

Entropia rozkładu prawdopodobieństwa p_i, $i = 1, 2, \ldots, n$ wynosi

$$E = -\sum_{i=1}^{n} p_i \log_2 p_i$$

W przypadku nieznajomości prawdopodobieństw p_i można je zastępować liczebnościami zdarzeń.

Przyjmijmy oznaczenia: $i = 1, 2, \ldots, I$ – numer atrybutu, $j = 1, 2, \ldots, J_i$ – indeks wartości przyjmowanych przez i-ty atrybut, $k = 0, 1$ – możliwe decyzje, L – liczba elementów zbioru uczącego, L_{ijk} – liczba decyzji o wartości k, gdy wybrano wartość o indeksie j atrybutu o numerze i (patrz tablica 6.3).

x_1	x_2	x_3	x_4	x_5	f
0	0	1	0	2	0
2	0	1	1	1	0
0	1	0	0	1	1
2	1	2	1	0	0
1	0	0	0	1	1
1	0	0	1	2	1
1	0	0	1	1	1
1	1	0	0	2	0
1	1	0	1	2	1
1	1	2	1	1	1
2	1	0	0	0	1
2	1	1	1	2	1

Tablica 6.3: Przykładowy zbiór uczący. Stosując przyjęte oznaczenia mamy: $I = 5$, $J_1 = J_3 = J_5 = 3$, $J_2 = J_4 = 2$, $L = 12$, oraz np. $L_{110} = 1$, $L_{111} = 1$, $L_{120} = 1$, $L_{121} = 5$, $L_{130} = 2$, oraz $L_{131} = 2$.

Zastępując prawdopodobieństwa liczebnościami, entropia rozkładu prawdopodobieństwa decyzji, gdy wybrano wartość j atrybutu i wynosi

$$E(D_{ij}) = -\frac{L_{ij0}}{L_{ij0} + L_{ij1}} \log_2 \frac{L_{ij0}}{L_{ij0} + L_{ij1}} - \frac{L_{ij1}}{L_{ij0} + L_{ij1}} \log_2 \frac{L_{ij1}}{L_{ij0} + L_{ij1}} \tag{6.51}$$

gdzie D_{ij} oznacza poddrzewo decyzyjne wychodzące z atrybutu i wzdłuż krawędzi j (po wyborze wartości j atrybutu i).

Podobnie, entropia rozkładu prawdopodobieństwa decyzji gdy wybrano atrybut i jest ważoną sumą entropii dla poszczególnych wartości atrybutów:

$$E(D_i) = \sum_{j=1}^{J_i} \frac{L_{ij0} + L_{ij1}}{L} E(D_{ij}) \tag{6.52}$$

gdzie D_i jest poddrzewem decyzyjnym o korzeniu będącym atrybutem i.

Zasada wyboru atrybutu: wybierany jest atrybut, dla którego entropia dana wzorem (6.52) przyjmuje wartość minimalną. Gdy pewna wartość wybranego atrybutu jednoznacznie określa wartość funkcji, to wychodząca z tego atrybutu krawędź drzewa opisana tą wartością kończy się liściem drzewa decyzyjnego

Przykład 6.9

W poniższym prostym przykładzie pokazano sposób wyboru jednego spośród dwóch atrybutów

...	x_i	...	f
	0		0
	0		0
	0		1
	0		1
	1		0
	1		0
	1		0
	1		0
	1		0
	1		1

$$\begin{aligned}\text{entropia} &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} \\ &= 1.00\end{aligned}$$

$$\begin{aligned}\text{entropia} &= -\frac{5}{6}\log_2\frac{5}{6} - \frac{1}{6}\log_2\frac{1}{6} \\ &= 0.65\end{aligned}$$

$$\text{ważona entropia} = \frac{4}{10}\cdot 1.00 + \frac{6}{10}\cdot 0.65 = 0.79$$

...	x_j	...	f
	0		0
	0		1
	0		1
	0		1
	1		0
	1		0
	1		0
	1		1
	1		1
	1		0

$$\begin{aligned}\text{entropia} &= -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \\ &= 0.81\end{aligned}$$

$$\begin{aligned}\text{entropia} &= -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6} \\ &= 0.92\end{aligned}$$

$$\text{ważona entropia} = \frac{4}{10}\cdot 0.81 + \frac{6}{10}\cdot 0.92 = 0.88$$

---

x_i powinno zostać wybrane jako korzeń drzewa

■

Przykład 6.10

W tablicy 6.3 podany jest zestaw danych uczących. Wyznaczyć korzeń i i pozostałe węzły drzewa decyzyjnego wybierając atrybuty o najmniejszej entropii.

Na podstawie wzoru (6.51) mamy:

$$\begin{aligned}E(D_{11}) &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} &= 1.0000 \\ E(D_{12}) &= -\frac{1}{6}\log_2\frac{1}{6} - \frac{5}{6}\log_2\frac{6}{6} &= 0.6500 \\ E(D_{13}) &= -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} &= 1.0000\end{aligned}$$

i następnie

$$\begin{array}{lll} E(D_{21}) = 0.9710, & E(D_{22}) = 0.8631, & \\ E(D_{31}) = 0.5917, & E(D_{32}) = 0.9183, & E(D_{33}) = 1.0000 \\ E(D_{41}) = 0.9710, & E(D_{42}) = 0.8631, & \\ E(D_{51}) = 1.0000, & E(D_{52}) = 0.7219, & E(D_{53}) = 0.9710 \end{array}$$

Podstawiając otrzymane wyniki do wzoru (6.52) mamy:

$$E(D_1) = \frac{2}{12}1 + \frac{6}{12}0.65 + \frac{4}{12}1 = 0.8250$$

i następnie $E(D_2) = 0.9080$, $E(D_3) = 0.7414$, $E(D_4) = 0.9080$, $E(D_5) = 0.8720$. Stąd widać, że należy wybrać atrybut x_3, ponieważ poddrzewo decyzyjne z korzeniem x_3 posiada najmniejszą entropie.

Przyjmując, że korzeniem drzewa jest atrybut x_3, w drugim etapie należy dla każdej krawędzi wychodzącej z węzła x_3, tzn dla każdej wartości atrybutu x_3 wyznaczyć nowy atrybut spośród pozostałych atrybutów.

Drugi etap dla $x_3 = 0$:

$$\begin{aligned} E(D_{11}) &= -\frac{1}{1}\log_2\frac{1}{1} - \frac{0}{1}\log_2\frac{0}{1} = 0.0000 \\ E(D_{12}) &= -\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5} = 0.7219 \\ E(D_{13}) &= -\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1} = 0.0000 \end{aligned}$$

i następnie

$$\begin{array}{lll} E(D_{21}) = 0.0000, & E(D_{22}) = 0.8113, & \\ E(D_{41}) = 0.8113, & E(D_{42}) = 0.0000, & \\ E(D_{51}) = 0.0000, & E(D_{52}) = 0.0000, & E(D_{53}) = 0.9183 \end{array}$$

Podstawiając otrzymane wyniki do wzoru (6.52) mamy:

$$E(D_1) = \frac{1}{7}0 + \frac{5}{7}0.7219 + \frac{1}{7}0 = 0.5156$$

i następnie $E(D_2) = 0.4636$, $E(D_4) = 0.4636$, $E(D_5) = 0.3936$. Stąd wynika, że należy wybrać atrybut x_5.

Drugi etap dla $x_3 = 1$:

$$\begin{array}{lll} E(D_{11}) = 0.0000, & & E(D_{13}) = 1.0000 \\ E(D_{21}) = 0.0000, & E(D_{22}) = 0.0000, & \\ E(D_{41}) = 0.0000, & E(D_{42}) = 1.0000, & \\ & E(D_{52}) = 0.0000, & E(D_{53}) = 1.0000 \end{array}$$

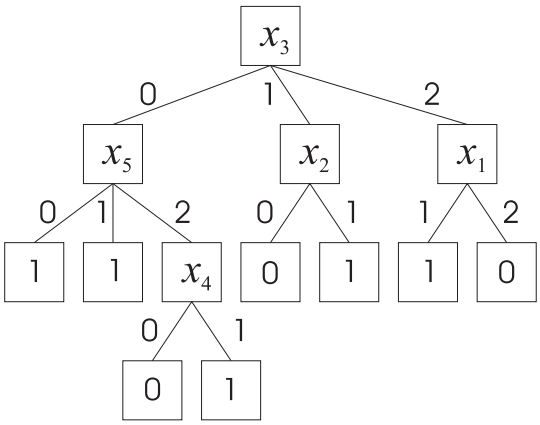
i następnie $E(D_1) = 0.6666$, $E(D_2) = 0.0000$, $E(D_4) = 0.6666$, $E(D_5) = 0.6666$. Stąd wynika, że w tym przypadku należy wybrać atrybut x_2.

Drugi etap dla $x_3 = 2$:

$$\begin{array}{rl} & E(D_{12}) = 0.0000, \quad E(D_{13}) = 0.0000 \\ & E(D_{22}) = 1.0000, \\ & E(D_{42}) = 1.0000, \\ E(D_{51}) = 0.0000, & E(D_{52}) = 0.0000, \end{array}$$

i następnie $E(D_1) = 0.0000$, $E(D_2) = 1.0000$, $E(D_4) = 1.0000$, $E(D_5) = 0.0000$. Czyli w tym przypadku należy wybrać atrybut x_1 lub x_5.

Kontynuując obliczenia otrzymujemy drzewo decyzyjne przedstawione na rysunku 6.43. ■



Rys. 6.43: Drzewo decyzyjne uzyskane przy wyborze atrybutów o minimalnej entropii

6.7. Generalizacja

Zdolność systemu do poprawnych odpowiedzi na sygnały wejściowe nie występujące w zbiorze uczącym nazywa się *generalizacją*. Podstawowym zadaniem uczenia jest utworzenie systemu który będzie dobrze generalizował.

Ocena zdolności do generalizacji systemu uczącego się jest jednym z podstawowych problemów teorii uczenia.

W praktycznych zastosowaniach uczenia powstają pytania: jaką zdolność do generalizacji będzie miał nauczony system, jaką da się uzyskać jakość generalizacji przy dostępnych danych uczących, dane są rózne zestawy parametrów nauczonych systemów, który będzie generalizował lepiej?

Błąd generalizacji jest tożsamy z ryzykiem średnim danym wzorem (6.1)

$$E(\mathbf{w}) = \mathcal{E}(y-d)^2 = \int (y-d)^2 p(\mathbf{x},d) d\mathbf{x} dd = \int \left[f(\mathbf{x},\mathbf{w}) - d\right]^2 p(\mathbf{x},d) d\mathbf{x} dd$$

który nie może być wykorzystany do obliczeń, ponieważ nie jest znana gęstość prawdopodobieństwa $p(\mathbf{x}, d)$. Natomiast znane jest ryzyko empiryczne dane wzorem (6.2)

$$E_{emp}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - d_i)^2 = \sum_{i=1}^{n}\left[f(\mathbf{x}_i,\mathbf{w}) - d_i\right]^2$$

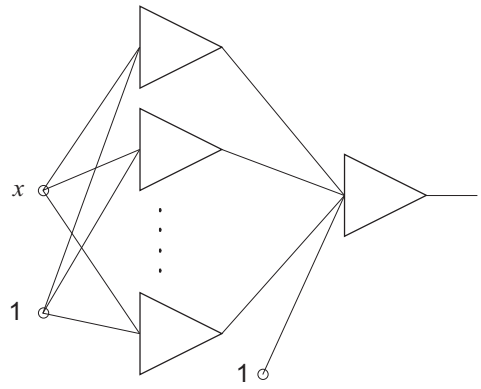
Powstaje pytanie: znamy ryzyko empiryczne, jakiego możemy się spodziewać błędu generalizacji (ryzyka średniego)? W dalszej części tego punktu odpowiemy na to pytanie.

6.7.1. Przykłady

Przedstawimy tu kilka przykładów przybliżających problemy generalizacji.

Przykład 6.11

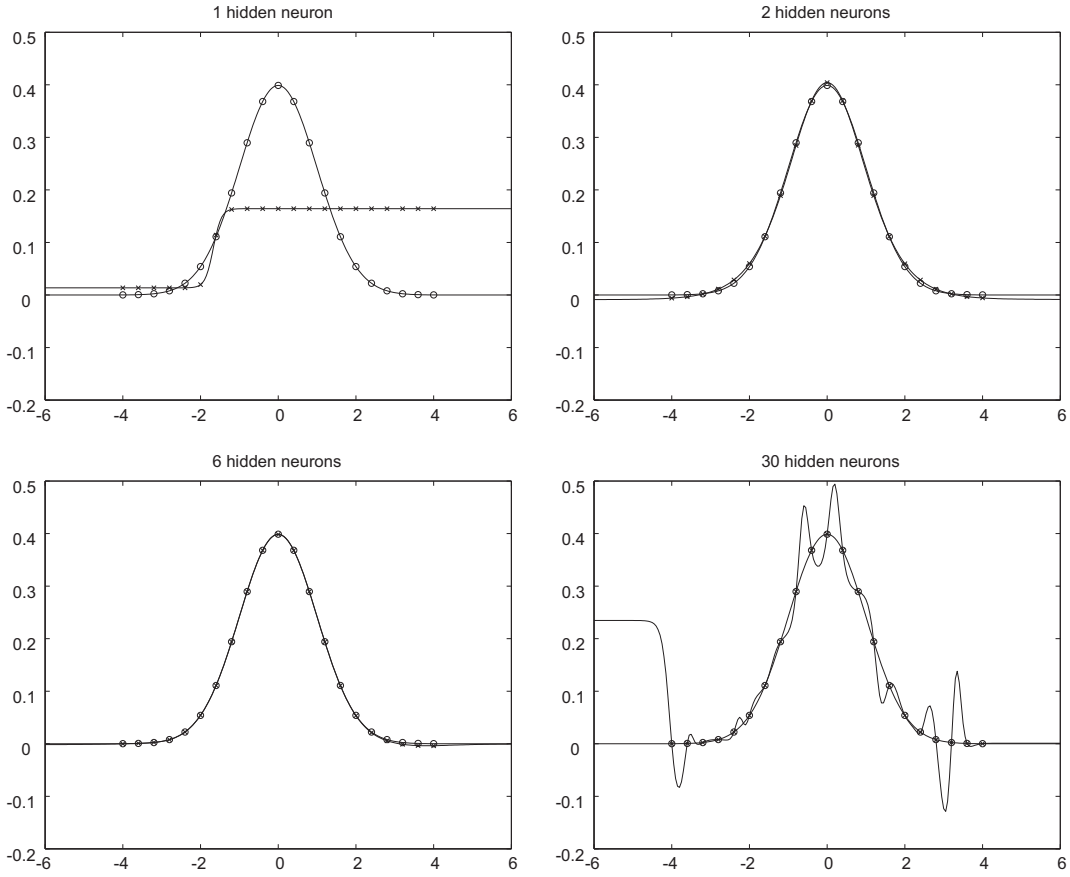
Przeprowadzono aproksymację funkcji Gaussa przy pomocy dwuwarstwowej sieci neuronowej przedstawionej na rys, 6.44. Na rysunku 6.45 przedstawiono wyniki aproksymacji realizowane przez sieci neuronowe o różnych liczbach neuronów. Zbiorem uczącym w każdym przypadku było 21 par argumentów i wartości funkcji. Na rysunkach kółka oznaczają pary uczące, krzyżyki odpowiedzi generowane przez sieci neuronowe dla sygnałów wejściowych ze zbioru uczącego, linie ciągłe przedstawiają funkcję Gaussa i funkcje realizowane przez sieci neuronowe.



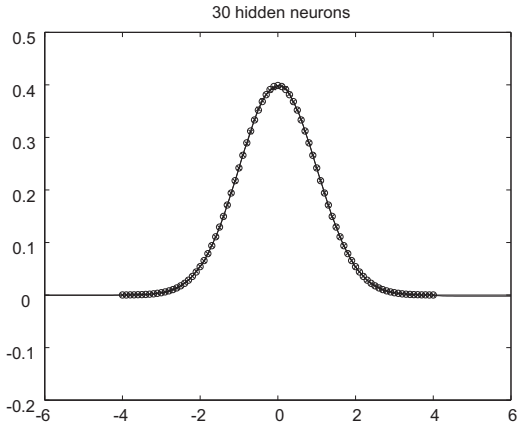
Rys. 6.44: Sieć neuronowa wykorzystana do aproksymacji funkcji Gaussa

W omawianym przykładzie błędem empirycznym jest średnia z sumy kwadratów różnic pomiędzy rzędnymi kółek i krzyżyków, zaś błędem generalizacji średni kwadrat różnicy pomiędzy liniami ciągłymi. Jak widać z rysunków zbyt mała liczba neuronów (1 lub 2) powoduje zarówno duży błąd empiryczny jak i duży błąd generalizacji. Z kolei duża liczba neuronów (30) powoduje bardzo mały błąd empiryczny natomiast bardzo duży błąd generalizacji. Optymalną w tym przypadku jest sieć o 6 neuronach. Przy jej zastosowaniu uzyskujemy mały błąd empiryczny i mały (najmniejszy z rozpatrywanych) błąd generalizacji. Można z tego wyciągnąć wniosek, że istnieje optymalna liczba neuronów (złożoność systemu uczącego się) zapewniająca najlepszą generalizację.

Przeprowadzono też aproksymację stosując sieć o 30 neuronach w warstwie wejściowej ale korzystając tym razem z 81 par uczących. Wyniki aproksymacji przedstawiono na rys. 6.46. Jak widać zwiększona liczba par uczących spowodowała, że tym razem sieć o 30 neuronach zapewnia bardzo dobrą generalizację, czyli można się spodziewać, że zwiększenie liczebności zbioru uczącego zmienia (zwiększa) optymalną złożonośc sieci. ■



Rys. 6.45: Aproksymacja funkcji Gaussa przez sieci neuronowe o różnych liczbach neuronów wejściowych dla zbioru uczącego zawierającego 21 par

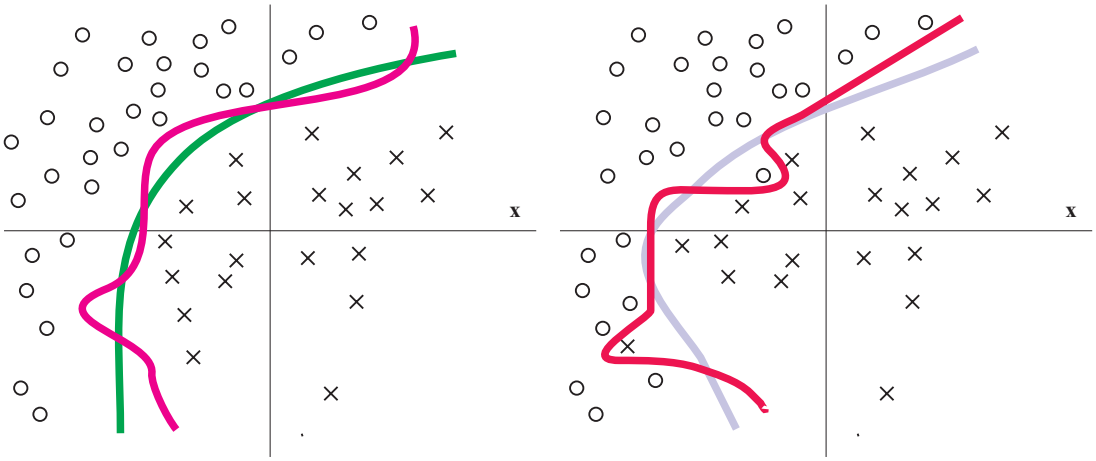


Rys. 6.46: Aproksymacja funkcji Gaussa przez sieć neuronową o 30 neuronach w arstwie wejściowej i zbioru uczącego zawierającego 81 par

Przykład 6.12

Na rysunku 6.47 przedstawiono przykładowe rozmieszczenia punktów uczących w dwuwymiarowej przestrzeni należących do dwóch klas i linii separujących klasy. Powstają pytania, które linie separujace zapewnią lepszą generalizację? Na rysunku 6.47a intuicyjnie lepsza wydaje się linia gładsza. Na rysunku 6.47b sytuacja jest

bardziej niejednoznaczna bowiem linia gładsza powoduje dwa błędy klasyfikacji. Bezbłędną klasyfikacje zapewnia linia bardziej skomplikowana. Które linie zapewnią lepszą generalizację? ■



Rys. 6.47: Przykładowe rozmieszczenia punktów uczących w dwuwymiarowej przestrzeni należących do dwóch klas i linii separujących klasy. Powstają pytania, które linie separujace zapewnią lepszą generalizację?

Przykład 6.13

Na rysunku 6.48 przedstawiono wyniki eksperymentu klasyfikacji, w którym obraz wejściowy był wektorem binarnym o 12 składowych. a wynikiem klasyfikacji była funkcja boolowska. Klasyfikator, którym była sieć neuronowa, a w drugim przypadku drzewo decyzyjne uczono na zbiorach uczących o różnej liczebności, a następnie testowano na wszystkich możliwych wektorach wejściowych (4096). Dla każdej liczebności przeprowadzono po kilka eksperymentów dla losowo wybranych zbiorach wektorów uczących. Liczbę trafnych odpowiedzi przedstawiono na rysunku.

Jak widać z wykresu zwiększanie liczby obrazów uczących powoduje wzrost jakości generalizacji. Przewaga jakosci wyników generowanych przez drzewa decyzyjne wynika w tym przypadku ze specyfiki problemu (funkcja boolowska) bardziej odpowiedniego dla drzew decyzyjnych. ■

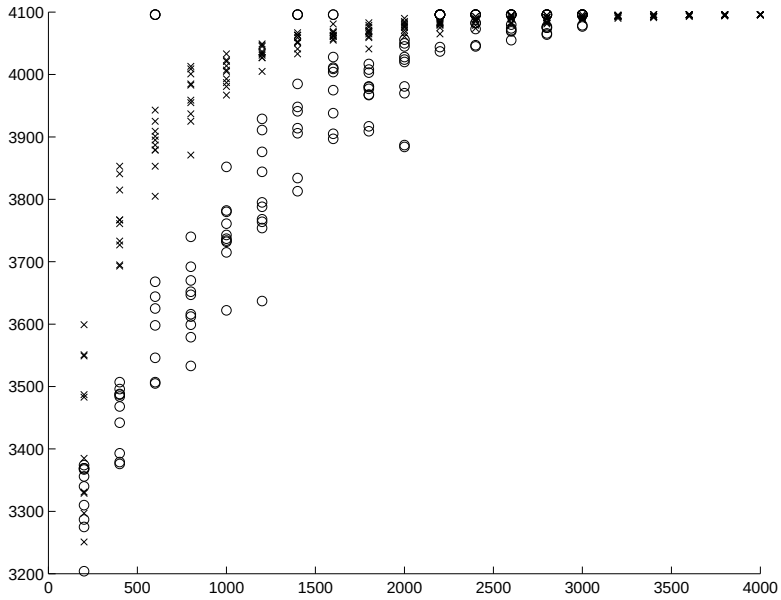
6.7.2. Nierówność Vapnika

Omówimy tu klasyczną nierówność Vapnika pokazującą zależność pomiędzy błędem generalizacji, a błędem uczenia (ryzyka empirycznego), złożonością systemu klasyfikującego oraz rozmiarem zbioru uczącego.

Wymiar VC

Tzw. wymiar VC (Vapnik-Chervonenkis) jest miarą złożoności systemu klasyfikującego lub miarą jego zdolności do klasyfikacji.

Najpierw zdefiniujemy pojęcie rozdzielania (ang. shattering) zbioru.



Rys. 6.48: Zależność liczby poprawnych odpowiedzi od liczby wcześniej użytych obrazów uczących w przykładowym problemie klasyfikacji wartości 12 argumentowej funkcji boolowskiej. Kółka przedstawiają wyniki dla sieci neuronowej, krzyżyki – wyniki dla drzewa decyzyjnego. Poszczególne kółka i krzyżyki dla danych liczebności obrazów uczących przedstawiają eksperymenty dla kilku losowo wybranych zbiorów uczących

Definicja 6.5 *Maszyna* $y = f(\mathbf{x}_i, \mathbf{w})$ *może rozdzielić zbiór punktów* $\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n$ *wtedy i tylko wtedy, gdy dla każdego możliwego zbioru uczącego w postaci*

$$(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \ldots, (\mathbf{x}_n, d_n), \quad d_i \in \{-1, 1\}$$

istnieje pewna wartość wektora $\mathbf{w}$ *która zapewnia zerowy błąd klasyfikacji.*

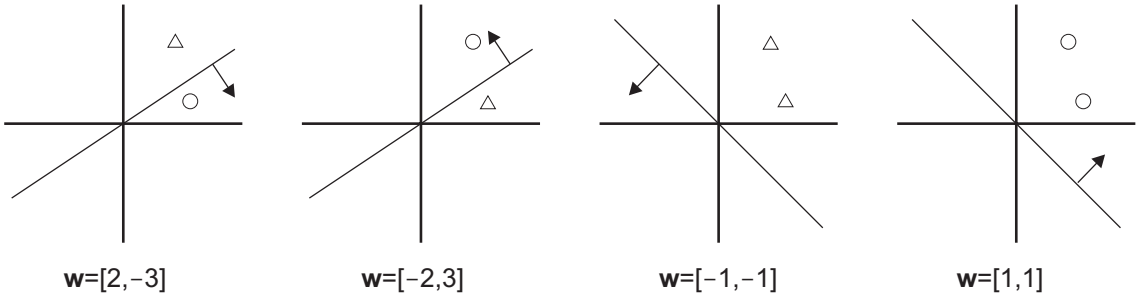
Z definicji wynika, że by maszyna mogła rozdzielić dany zbiór punktów potrzeba aby dla każdej wariacji wartości odpowiedzi $d_1, d_2, \ldots, d_n$ istniał pewien wektor $\mathbf{w}$ zapewniajacy bezbłędną klasyfikację. Takich kombinacji jest oczywiście 2^n.

Przykład 6.14

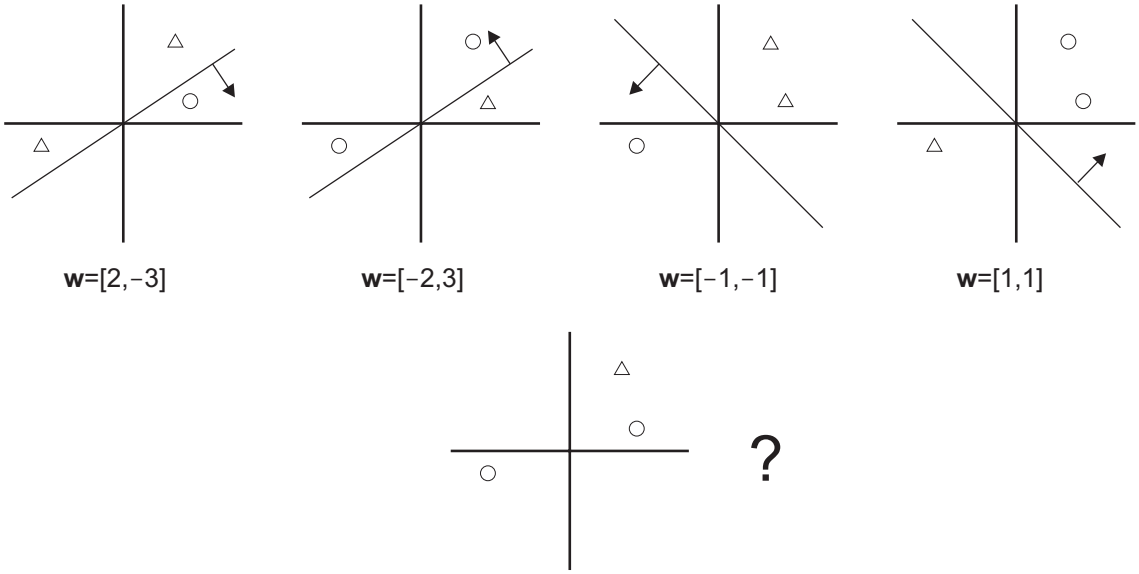
Łatwo pokazać, że maszyna $y = \text{sgn}(w_1 x_1 + w_2 x_2) = \text{sgn}(\mathbf{w}^t \mathbf{x})$ może rozdzielić 2 punkty. Reprezentacją geometryczną tej maszyny (zbór punktów dla których odpowiedź maszyny jest równa zero) jest linia prosta przechodząca przez początek układu współrzędnych. Na rysunku 6.49 pokazano, że dla każdego możliwego przyporządkowania klas dwóm punktom można znaleźć wektor $\mathbf{w}$ zapewniający bezbłędną klasyfikację. Natomiast rysunek 6.50 pokazuje, że ta sama maszyna nie jest w staie rozdzielić 3 punktów. ■

Przykład 6.15

Maszyna $y = \text{sgn}\left[w_1(x_1^2 + x_2^2) + w_2\right] = \text{sgn}\left[w_1 \mathbf{x}^t \mathbf{x} + w_2\right]$ jest w stanie rozdzielić dwa punkty, co ilustruje rys. 6.51. ■



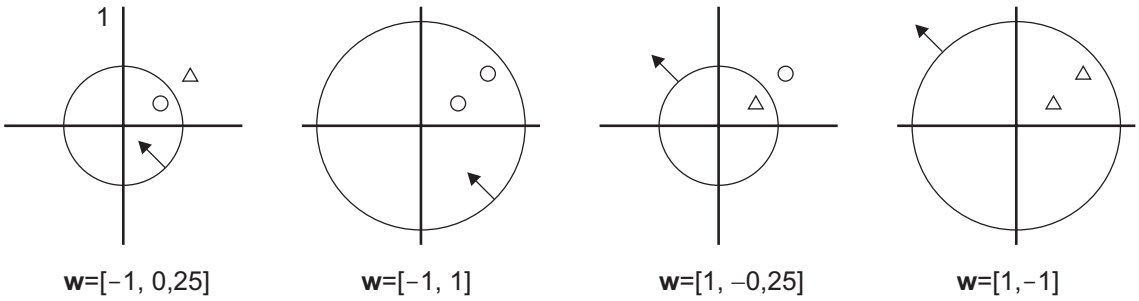
Rys. 6.49: Linie proste reprezentujące maszynę $y = \mathrm{sgn}(w_1x_1 + w_2x_2) = \mathrm{sgn}(\mathbf{w}^t\mathbf{x})$ dla róznych wektorów $\mathbf{w}$. Strzałki wskazują z której strony linii odpowiedź maszyny wynosi $+1$. Jak widać maszyna ta może rozdzielić 2 punkty



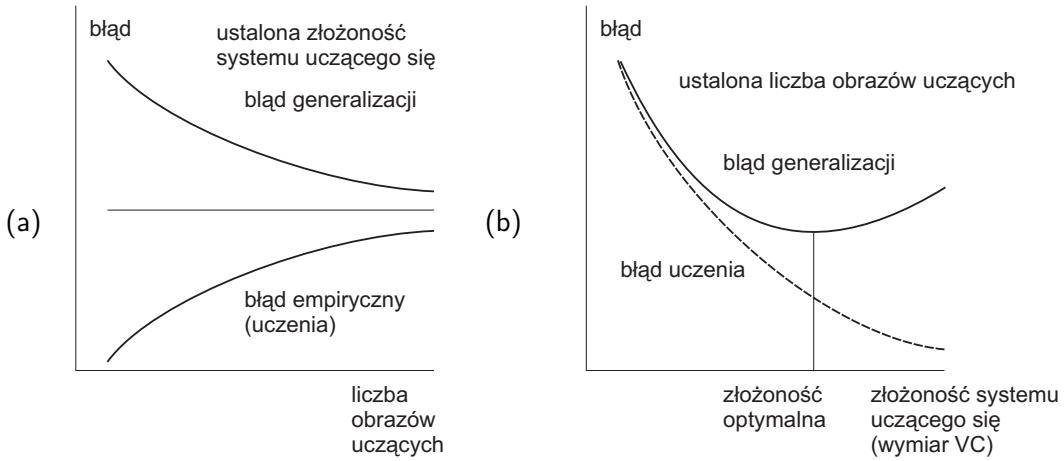
Rys. 6.50: Maszyna $y = \mathrm{sgn}(w_1x_1 + w_2x_2) = \mathrm{sgn}(\mathbf{w}^t\mathbf{x})$ dla róznych wektorów $\mathbf{w}$ nie może rozdzielić 3 punktów

Definicja 6.6 *Wymiar VC maszyny $y = f(\mathbf{x}, \mathbf{w})$ jest maksymalną liczbą punktów, które mogą być tak ustawione aby maszyna mogła je rozdzielić.*

Nierówność Vapnika: niech h będzie wymiarem VC pewnej maszyny $y = f(\mathbf{x}, \mathbf{w})$,



Rys. 6.51: Maszyna $y = \mathrm{sgn}\left[w_1(x_1^2 + x_2^2) + w_2\right] = \mathrm{sgn}\left[w_1\mathbf{x}^t\mathbf{x} + w_2\right]$ może rozdzielić 2 punkty



Rys. 6.52: Zależność błędu generalizacji i błędu uczenia od: (a) liczby obrazów uczących przy ustalonej złożoności systemu uczącego się; (b) złożoności systemu uczącego się przy ustalonej liczbie obrazów uczących

wtedy z prawdopodobieństwem 1-η zachodzi

$$E(\mathbf{w}) \leq E_{emp}(\mathbf{w}) + \sqrt{\frac{h(\log(2n/h)+1) - \log(\eta/4)}{n}} \tag{6.53}$$

gdzie n jest liczbą obrazów uczących.

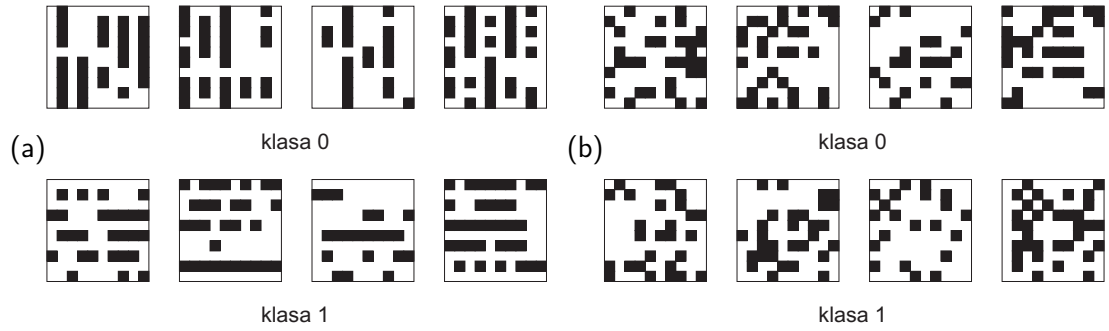
Nierówność (6.53) pozwala oszacować błąd dla danych nie używanych w procesie uczenia opierając się tylko na wymiarze VC i liczebności zbioru uczącego. Widać z niej, że ze zwiększaniem wymiaru h, a tym samym, naogół złożoności systemu zwiększa się wyrażenie pod pierwiastkiem, ale maleje też błąd empiryczny. Z kolei zmniejszanie h powoduje zmniejszenie wyrażenia pod pierwiastkiem, ale powoduje zwiększanie błędu empirycznego. Istnieje więc optymalna złożonośc systemu, co ilustruje rysunek 6.52, gdzie przedstawiono ogólnie zależności blędu generalizacji od liczby obrazów uczących i złożoności systemu uczacego się.

Specyfika klasyfikacji obrazów dwuwymiarowych

Problemy klasyfikacji przedstawione na rysunkach 6.53a i b różnią się jedynie przestawieniem punktów. Sieć neuronowa warstwowa o pełnych połączeniach nie rozróżnia sąsiedztwa przestrzennego sygnałów wejściowych i dlatego nauczenie jej klasyfikacji obrazów przedstawionych na rysunkach 6.53ab stanowi dokładnie taką samą trudność mimo, że dla człowieka problem przedstawiony na rys. 6.53a jest znacznie łatwiejszy.

6.8. Samoorganizacja

Samoorganizacja jest typem uczenia, gdzie nie dysponujemy ani poprawnymi odpowiedziami ani ocenami odpowiedzi. Samoorganizacja sprowadza się zazwyczaj do wyznaczenia centrów klastrów obrazów wejsciowych lub wyznaczaniu innych powiązań pomiedzy obrazami wejsciowymi.



Rys. 6.53: Dwa różne problemy klasyfikacji, których nauczenie się stanowią dla sieci identyczną trudność pomimo, że dla człowieka problem przedstawiony na rysunku (a) jest znacznie prostszy niż na (b)

6.8.1. Problemy klasteryzacji

W wielu zastosowaniach potrzebne jest znalezienie środków skupisk punktów (klastrów). Niektóre algorytmy znajdywania takich punktów można zaliczyć do metod sztucznej inteligencji.

Mamy zbiór N punktów $\mathbf{x}_j$, $j = 1, \ldots, N$ i chcemy je podzielić na M klastrów G_i, $i = 1, \ldots, M$ i znaleźć środki klastrów $\mathbf{c}_i$, $i = 1, \ldots, M$ tak aby zminimalizować następującą funkcje odległosci między punktami

$$J = \sum_{i=1}^{M} J_i = \sum_{i=1}^{M} \left(\sum_{k, \mathbf{x}_k \in G_i} \|\mathbf{x}_k - \mathbf{c}_i\|^2 \right) \tag{6.54}$$

Wprowadźmy macierz $\mathbf{U}$ określająca przynależność punktu do klastra

$$u_{ij} = \begin{cases} 1 & \text{jeżeli } \|\mathbf{x}_j - \mathbf{c}_i\|^2 \leq \|\mathbf{x}_j - \mathbf{c}_k\|^2, \text{ dla każdego } k \neq i, \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Własności macierzy $\mathbf{U}$:

$$\sum_{i=1}^{M} u_{ij} = 1, \quad \forall j = 1, \ldots, N$$

$$\sum_{i=1}^{M} \sum_{j=1}^{N} u_{ij} = N$$

Jeżeli u_{ij} jest ustalone, wtedy środkiem klastra $\mathbf{c}_i$ minimalizujacym wyrażenie (6.54) jest wartość średnia

$$\mathbf{c}_i = \frac{1}{|G_i|} \sum_{k, x_k \in G_i} \mathbf{x}_k \tag{6.55}$$

gdzie $|G_i|$ jest licznością klastra G_i.

Algorym C-średnich (ang. C-means clustering)

Krok 1. Wybierz początkowe środki klastrów $\mathbf{c}_i$, $i = 1, 2, \ldots, M$. Zazwyczaj losowo spośród punktów $\mathbf{x}_i$, $i = 1, \ldots, N$.

Krok 2. Wyznacz macierz przynależności **U**.

Krok 3. Oblicz wartość kryterium z wzoru (6.54).

Krok 4. Oblicz nowe środki klastrów według wzoru (6.55). Przejdź do kroku 2.

Algorytm C-średnich nie zapewnia zbieżności do minimum globalnego.

Algorytm „zwycięzca bierze wszystko"

Algorytm „zwycięzca bierze wszystko" (ang. winner take all) zaproponowany przez Kohonena jest algorytmem iteracyjnym, w którym w kolejnych krokach przesuwane są punkty **w** reprezentujące centra klastrów. Punkty te nazywane są w literaturz neuronami – współrzędne punktu są wagami neuronu[7]. W każdym kroku algorytmu ze zbioru punktów **x** wybierany jest jeden punkt i spośród wszystkich punktów **w** wyznaczany jest punkt najbliższy do punktu **x** – nazywany zostaje „zwycięzcą", Następnie zwyciężający neuron przesuwany jest w kierunku punktu **x**.

Algorytm działa w następujaco:

1. Losowy wybór punktu **x**
2. Wyznaczenie zwycięskiego punktu o indeksie m poprzez obliczenie minimum odległości euklidesowej pomiedzy punktem **x** i wektorem wag neuronu $\mathbf{w}_i$, $i = 1, 2, \ldots, p$
$$m(\mathbf{x}) = \arg \min_{i=1,2,\ldots p} \|\mathbf{x} - \mathbf{w}_i\|$$
3. Przesunięcie wektora wag według reguły Kohonena
$$\mathbf{w}_m^{k+1} = \mathbf{w}_m^k + \alpha^k(\mathbf{x} - \mathbf{w}_m^k),$$
$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k, \quad i \neq m$$
4. Przejście do p. 1

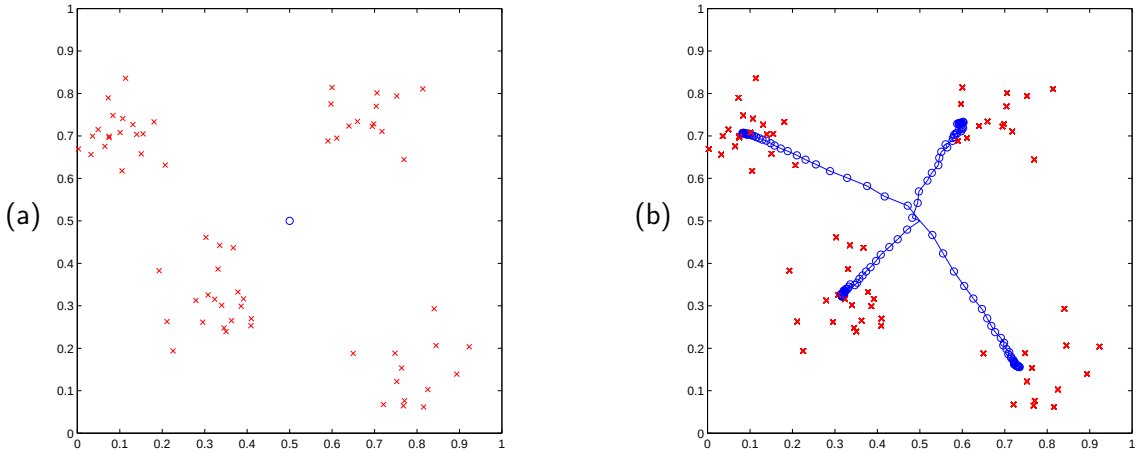
Na rys. 6.54 przedstawiono kolejne kroki przykładowego poszukiwania centrów klastrów przez algorytm „zwycięzca bierze wszystko".

Czasami może się zdarzyć, że pewien punkt **w** nigdy nie wygra. Aby temu zapobiec zazwyczaj stosuje się modyfikacje algorytmu polegającą na tym, że za każdą przegraną punktowi dopisuję się pewną ujemną liczbę reprezentującą odległość. Sumy tych liczb uwzględnia się przy wyznaczaniu odległości w kolejnych iteracjach. Punktowi zwycięskiemu te liczby się kasuje.

Inne metody

Dla poszukiwania centrów klastrów można także stosować metodę symulowanego wyżarzania lub metody ewolucyjne.

[7]Przyjęcie interpretacji, że punkt jest reprezentowany przez wagi neuronu powoduje, że znajdywanie zwycięskiego punktu może odbywać się za pomocą sieci neuronowej – w przypadku znormalizowania wag, tzn. $\|\mathbf{w}\| = 1$ zwycieski neuron to ten, którego odpowiedź na pobudzenie **x** jest największa



Rys. 6.54: Ilustracja działania algorytmu „zwycięzca bierze wszystko": (a) stan początkowy poszukiwania, gdzie krzyżyki reprezentują obrazy wejsciowe, a kółko początkowe cztery identyczne centra klastrów i (b) trajektorie przesuwania się centrów klastrów w czasie działania algorytmu

6.8.2. Samoorganizujące się mapy cech

Self-organizing map is a set of neurons representing center of clusters (cluster nodes). The neurons connected by edges are forming some geometric structure.

6.9. Uczenie ze wzmocnieniem

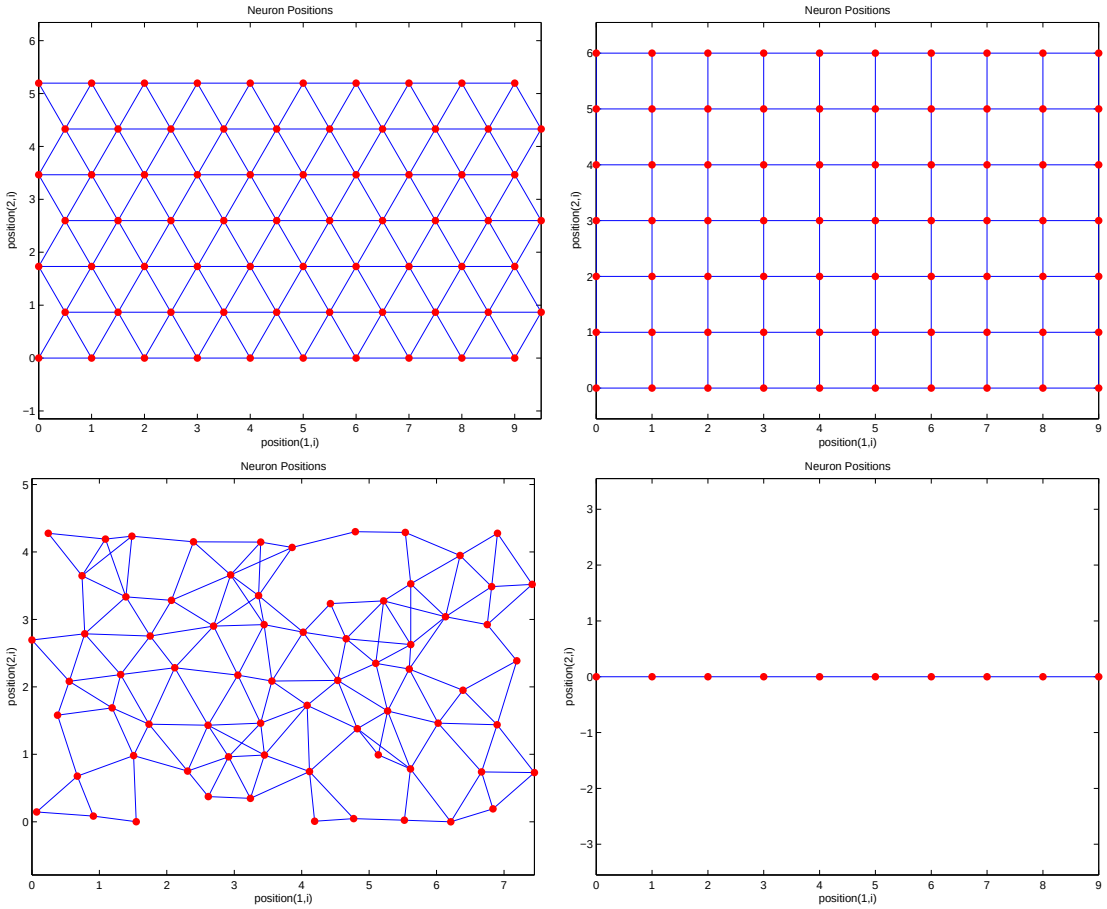
Uczenie z krytykiem ma miejsce wtedy, gdy systemowi uczącemu nie są znane oczekiwane odpowiedzi, a jedynie oceny liczbowe wartościujące dokonane akcje. Najczęściej uczenie z krytykiem stosowane jest do sterowania wieloetapowymi procesami decyzyjnymi, a powszechnie stosowane wtedy algorytmy uczenia noszą nazwę uczenia ze wzmocnieniem (ang. reinforcement learning)

6.9.1. Wieloetapowe procesy decyzyjne

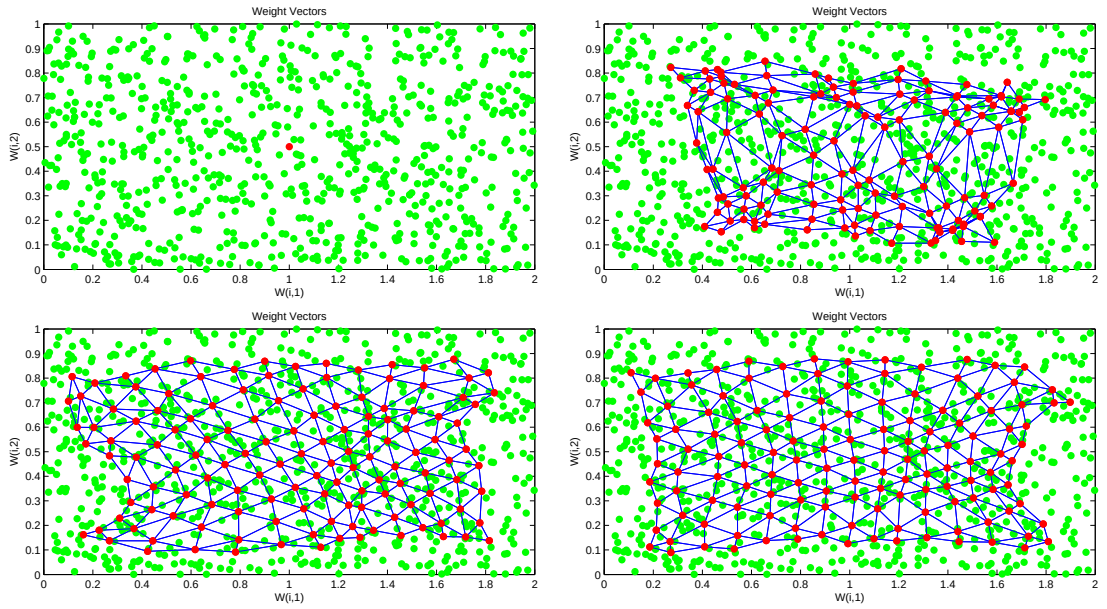
Wieloetapowy proces decyzyjny jest systemem dynamicznym przechodzącym, w wyniku podjętej akcji a_i, z bieżącego stanu do stanu następnego, generując przy tym liczbę r_i uważaną za ocenę wykonanego kroku (*nagrodę*, ang. *reward*). Ocena całego procesu decyzyjnego jest sumą tych ocen po pewnej (dużej) liczbie kroków, $r = \sum_{i=1}^{n} r_i$ (rys. 6.58)

Wieloetapowe procesy decyzyjne mogą być modelami wielu rzeczywistych procesów, w szczególności procesów ekonomicznych, procesów sterowania oraz planowania akcji.

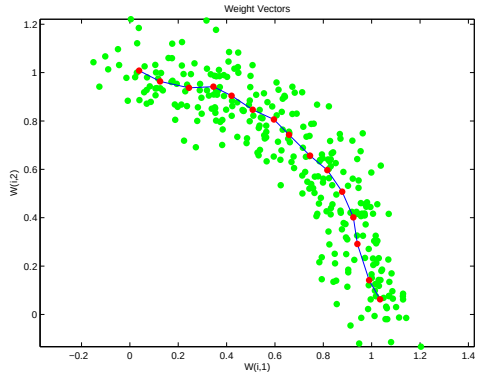
Przykładami wieloetapowych procesów decyzyjnych są: maksymalizacja zysku przedsiębiorstwa w ciągu zadanego okresu czasu, sterowanie zespołem elektrowni wodnych umieszczonych wzdłuż jednej rzeki i jej dopływów (np. kanadyjski system Hydro Quebec) zapewniający maksymalną ilość dostarczanej energii, sterowanie zespołem wind, sterowanie odwróconym wahadłem, gry, itp. Do typowych wieloetapo-



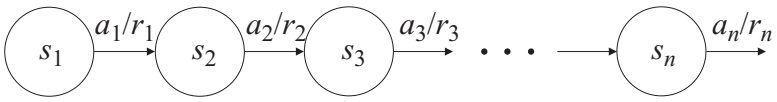
Rys. 6.55: Przykłady siatek stosowanych w samoorganizujących się mapach cech



Rys. 6.56: Przykład działania samoorganizującej się mapy cech



Rys. 6.57: Przykład działania samoorganizującej się mapy cech – siatka jednowymiarowa



Rys. 6.58: Wieloetapowy proces decyzyjny – trajektoria zmian stanu i otrzymywane oceny

wych procesów decyzyjnych zaliczają się też wszelkie zadania alokacji, czyli rozdzielania określonych zasobów pomiędzy określoną liczbę użytkowników zapewniające optymalny zysk.

Charakterystyczne dla wieloetapowych procesów decyzyjnych jest, że akcje składające się na optymalny proces decyzyjny nie muszą być lokalnie najlepsze. Związana z nimi nagroda nie musi być największą z możliwych nagród do otrzymania w danym stanie i czasami należy zrezygnować z wysokich lokalnych nagród dla osiągnięcia globalnego optimum. Przeciwnie, maksymalizacja nagrody na pewnym etapie może prowadzić do pogorszenia globalnej oceny systemu.

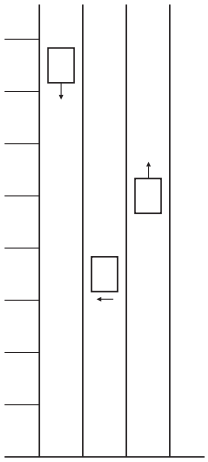
Rozróżnia się systemy ciągłe i dyskretne, *deterministyczne*, w których żądane przejścia stanów zachodzą na pewno, i *niedeterministyczne*, w których żądane zmiany zachodzą tylko z określonym prawdopodobieństwem (mogą nastąpić przejścia inne niż żądane) lub zachodzą spontaniczne zmiany stanów na skutek losowych czynników zewnętrznych. Nagrody mogą być otrzymywane po każdej akcji lub istnieć tylko jednorazowa nagroda po zakończeniu procesu (np. wygrana w grach). Procesy mogą trwać ustaloną liczbę kroków (określony czas) lub nie mieć końca.

Przykład 6.16

Przykład wieloetapowego procesu decyzyjnego – zadanie optymalnego sterowania zespołem wind (rys. 6.59).

Stan systemu:

- położenie wind,
- kierunek ruchu wind (góra, dół, zatrzymanie),
- sygnały z przycisków umieszczonych na piętrach,
- sygnały z przycisków wewnątrz wind,
- (ewentualnie) sygnały z czujników wagi pasażerów i/lub liczby przekroczeń drzwi wejściowych.



Rys. 6.59: Przykładowy stan systemu trzech wind

Ograniczenia: np. winda, której docelowe piętro zostało wskazane przez pasażera wewnątrz windy, jadąca we wskazanym kierunku nie może zawrócić nie zatrzymując się uprzednio na wskazanym piętrze.

Kryterium może kombinacją następujących kryteriów:

- Liczba pasażerów dowieziona do miejsc wskazanych przez nich,
- czas oczekiwania pasażera na windę i czas spędzony w windzie,
- koszt energii zużytej przez system,
- koszt zużycia windy (zależny od jej ruchu).

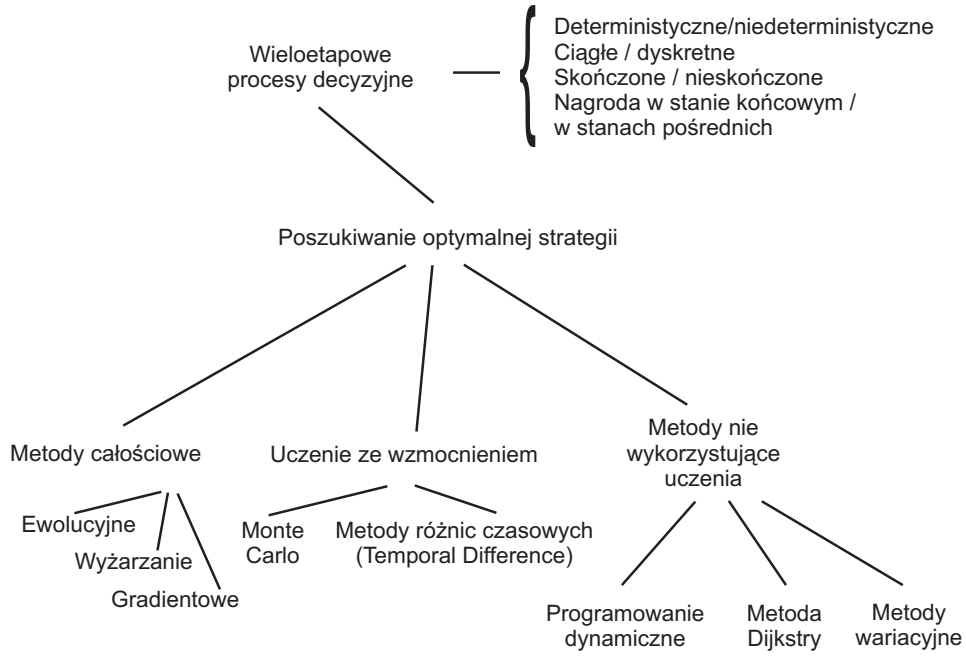
Np. jako kryterium można przyjąć, że w każdym kroku czasowym przychód wynosi -50 punktów za każde piętro na którym oczekują pasażerowie (sygnalizowane przyciskiem) oraz -1 punkt za każdą windę w ruchu.

6.9.2. Optymalizacja wieloetapowych procesów decyzyjnych

Istnieje wiele metod optymalizacji wieloetapowych procesów decyzyjnych i są one aktualnie intensywnie badane i rozwijane (Sutton, Barto 1998, Jang i inni 1997, Cichosz 2000). Oprócz podejścia klasycznego, opartego na rachunku wariacyjnym, stosowane są metody ewolucyjne oraz metody oparte na tzw. zasadzie optymalności Bellmana, do których zalicza się programowanie dynamiczne oraz metodę różnic czasowych. Niektóre z tych metod wymagają znajomości rozkładów prawdopodobieństwa czynników losowych oddziaływujących na system. Zestawienie stosowanych typowych metod i ich podstawowych własności przedstawia rysunek 6.60 oraz tablica 6.4.

6.9.3. Metody ewolucyjne

Metody ewolucyjne poszukują najlepszej polityki operując na populacji polityk i stosując typowe operacje algorytmów genetycznych. Politykę ocenia się na podstawie pewnej liczby kompletnych sekwencji sterowania. Metody ewolucyjne zalicza



Rys. 6.60: Rodzaje i metody optymalizacji wieloetapowych procesów decyzyjnych

metody	wymagana znajom. rozkł. prawdop.	stos. zasady optymalności	występowanie uczenia
wariacyjne i Dijkstry	$\surd$		
prog. dynamiczne.	$\surd$	$\surd$	
całościowe			$\surd$
ze wzmocnieniem		$\surd$	$\surd$

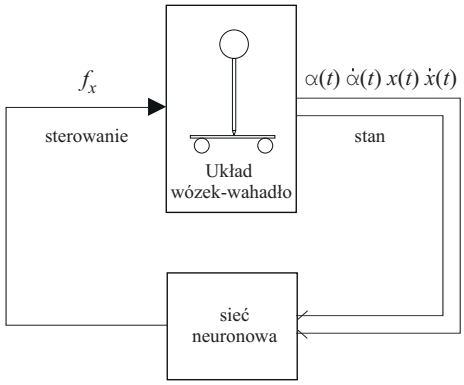
Tablica 6.4: Własności podstawowych grup metod optymalizacji wieloetapowych procesów decyzyjnych

się do tzw. grupy metod całościowych, w których ocenia się całą politykę, a nie poszczególne akcje. Przykłady zastosowania metod ewolucyjnych do wyznaczania neuronowego sterownika odwróconego wahadła przedstawiono poniżej.

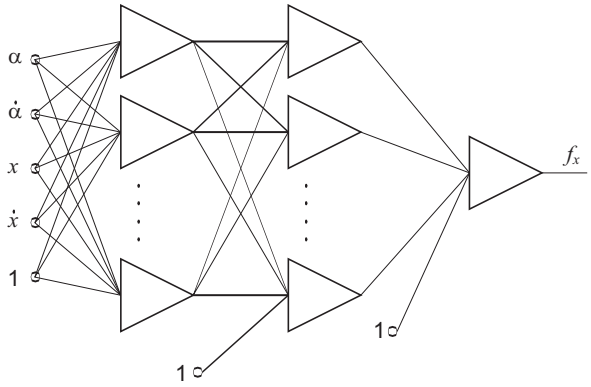
6.9.4. Przykład zastosowania algorytmu genetycznego do uczenia sieci neuronowej

Zadaniem sieci neuronowej jest sterowanie układu składającego się z wózka i odwróconego wahadła w układzie przedstawionym na rys. 6.61. Przyjęto, że na wejście sieci podane są składowe wektora stanu wahadła α, $\dot{\alpha}$, x, $\dot{x}$. Sieć sterująca jest siecią warstwową jednokierunkową o strukturze przedstawionej na rys. 6.62. Na jej pojedynczym wyjściu generowany jest sygnał określający siłę f_x działającą na wózek.

W postawionym zadaniu opracowania algorytmu sterującego odwróconym wahadłem nie można zastosować metody uczenia z nauczycielem, ponieważ nie są znane wartości siły przykładanej do podstawy wahadła zapewniające optymalne stero-



Rys. 6.61: Sieć neuronowa sterująca układem wózek – odwrócone wahadło



Rys. 6.62: Struktura zastosowanej sieci neuronowej

wanie. Zastosowany poniżej sposób jest jednym z przykładów systemów uczenia z krytykiem, gdzie zmiana parametrów systemu (wag sieci neuronowej) następuje po całej sekwencji sterowania na podstawie skumulowanej oceny jakości sterowania.

Zastosowany algorytm genetyczny

System nauki sieci za pomocą algorytmu genetycznego zawiera stałą, parzystą liczbę sieci (oznaczoną w programie komputerowym przez `LiczbaSieci`) dwuwarstwowych o niezmiennej strukturze. Liczba neuronów w warstwie ukrytej wynosi `LiczbNeuronUkr`. W jednym cyklu algorytmu genetycznego generowana jest losowo stała liczba stanów początkowych wahadła (`LiczbaStanPocz`). Ocena jakości sieci w cyklu polega na sterowaniu przez sieć wahadłem, począwszy od każdego z wylosowanych stanów początkowych, przez określoną liczbę kroków (`LiczbaKroTest`), przy czym długość jednego kroku sterowania jest równa długości kroku całkowania `KrokCalk` równania (6.4). W czasie sterowania obliczane są chwilowe wartości błędów sterowania zdefiniowane jako

$$e = \begin{cases} \alpha^2 + 0.25\dot{\alpha}^2 + 0.0025x^2 + 0.0025\dot{x}^2 & \text{jeżeli } \alpha < \pi/2 \\ 10^6 + 0.25\dot{\alpha}^2 + 0.0025x^2 + 0.0025\dot{x}^2 & \text{jeżeli } \alpha \geq \pi/2 \end{cases}$$

które następnie są sumowane po wszystkich krokach sterowania i wszystkich stanach początkowych. Otrzymuje się w ten sposób błąd działania sieci w cyklu

$$E = \sum^{\texttt{LiczbaStanPocz}} \sum^{\texttt{LiczbaKroTest}} e \tag{6.56}$$

W czasie działania algorytmu genetycznego jako kryterium sieci przyjmuje się skumulowany błąd zdefiniowany jako

$$B_i = B_{i-1} + 0.5(E_i - B_{i-1}).$$

gdzie i jest numerem cyklu.

Po zakończeniu sprawdzania działania wszystkich sieci następuje ich selekcja. Przed jej rozpoczęciem dokonuje się projekcji wartości błędów sterowania sieci B na odcinek $(\beta, 1)$ według wzoru

$$c_i = \frac{B_{i,max} - B_i}{B_{i,max} - B_{i,min}}(\beta - 1) + \beta$$

gdzie $B_{i,max}$ i $B_{i,min}$ są odpowiednio wartościami maksymalnymi i minimalnymi skumulowanych błędów, a $0 \leq \beta < 1$ jest arbitralnie wybranym współczynnikiem. Kryterium to rzutuje wartości błędów B_i sterowania sieci na odcinek $(\beta, 1)$.

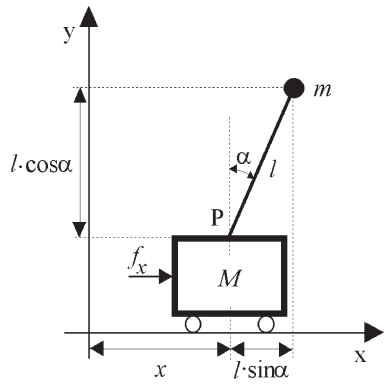
Do następnego etapu algorytmu genetycznego wybiera się losowo sieci z prawdopodobieństwem proporcjonalnym do jakosci c. Liczba wybranych w ten sposób sieci jest parzysta i wynosi $\texttt{LiczbaSelLos} \leq \texttt{LiczbaSieci}$. Dodatkowo do następnego etapu bez losowania wprowadza się `LiczbaSelBez` najlepszych sieci (o największych wartościach kryterium c), przyczym $\texttt{LiczbaSelBez} = (\texttt{LiczbaSieci} - \texttt{LiczbaSelLos})/2$.

Podczas etapu krzyżowania spośród sieci wybranych losowo tworzy się losowo $\texttt{LiczbaSelLos}/2$ par sieci. Następnie przeglądając odpowiadające sobie neurony w parach sieci, wymienia się ich wagi z prawdopodobieństwem `PrawdKrzyz`. Dodatkowo na podstawie każdej z `LiczbaSelBez` sieci wprowadzonej bez losowania tworzy się sieć potomną o wagach zmienionych według wzoru $w = w + \texttt{PrzedzMut} \times (\texttt{Y} - 0.5)$, gdzie `Y` jest zmienną losową o równomiernym rozkładzie prawdopodobieństwa przyjmującą wartości z przedziału $(0, 1)$, a `PrzedzMut` jest arbitralnie dobranym współczynnikiem. Po tych etapach otrzymuje się ponownie liczbę sieci wynoszącą `LiczbaSieci` ponieważ $2 \times \texttt{LiczbaSelBez} + \texttt{LiczbaSelLos} = \texttt{LiczbaSieci}$.

Na etapie mutacji wybranych zostaje losowo $\texttt{CzestMut} \times \texttt{LiczbaWag}$ wag, które następnie zostają zmienione według identycznej zasady jak przy tworzeniu sieci potomnych do wybranych bez losowania, gdzie $\texttt{LiczbaWag} = (\text{liczba zmiennych stanu} \times \texttt{LiczbaNeuronUkr} + 2 \times \texttt{LiczbaNeuronUkr} + 1) \times \texttt{LiczbaSieci}$ jest łączną liczbą wag we wszystkich sieciach, a `CzestMut` jest arbitralnie wybraną stałą.

Model matematyczny odwróconego wahadła[8]

W czasie działania algorytmu genetycznego wymagane jest przeprowadzenie wielkej liczby cykli sterowań wahadła (np. około 10^6). Sterowania takie praktycznie można zrealizować jedynie poprzez symulację. Aby przeprowadzać symulację sterowania



Rys. 6.63: Układ wózek-wahadło

potrzebna jest znajomość modelu matematycznego odwróconego wahadła czyli znajomość reakcji odwróconego wahadła na przyłożoną siłę.

Układ wózek-wahadło przedstawiono schematycznie na rys. 6.63. Opisujący go układ równań różniczkowych ma postać:

$$\dot{x}_1(t) = x_2(t),$$

$$\dot{x}_2(t) = \frac{g(M+m)\sin x_1(t) - \frac{1}{2}ml\,x_2^2(t)\sin 2x_1(t) + bx_4(t)\cos x_1(t) - f_x(t)\cos x_1(t)}{l(M + m\sin^2 x_1(t))}$$

$$\dot{x}_3(t) = x_4(t),$$

$$\dot{x}_4(t) = \frac{ml\,x_2^2(t)\sin x_1(t) - mg\sin x_1(t)\cos x_1(t) - bx_4(t) + f_x(t)}{M + m\sin^2 x_1(t)}.$$

gdzie

$$\mathbf{x}(t) = [\, x_1(t) \;\; x_2(t) \;\; x_3(t) \;\; x_4(t) \,]^t \stackrel{\text{def}}{=} [\, \alpha(t) \;\; \dot{\alpha}(t) \;\; x(t) \;\; \dot{x}(t) \,]^t$$

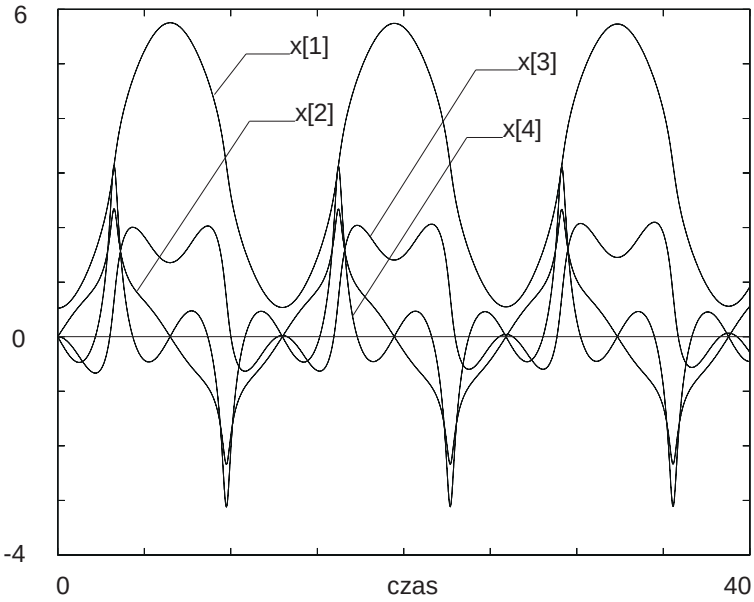
oraz M – masa wózka, m – masa wahadła, l – długość sztywnego (z założenia – nieważkiego) ramienia wahadła, b – współczynnik tarcia lepkiego, x – położenie wózka, α – kąt odchylenia ramienia od pionu, f_x – siła przyłożona do wózka.

Dla rozwiązania powyższego układu równań nieliniowych zastosowano algorytm numeryczny Runge-Kutta drugiego rzędu. Na rys. 6.64 przedstawiono przykładowy przebieg w czasie stanu wahadła począwszy od zadanego stanu początkowego przy braku siły oddziałującej na wózek.

Przykładowe wyniki

Przeprowadzono szereg symulacji starając się dobrać parametry systemu zapewniające możliwie szybkie uczenie sieci. Okazało się, że uczenie następuje ale jest wolno zbieżne i krytyczna jest moc obliczeniowa komputera i parametry algorytmu genetycznego. Poniżej zaprezentowano wyniki jednej z symulacji, w której przyjęto następujące parametry:

[8] Dla dociekliwych



Rys. 6.64: Trajektoria stanu odwróconego wahadła o parametrach $M = 10$kg, $m = 30$kg, $l = 30$m, $b = 0$, stanie początkowym $x_1(0) = \pi/6$, $x_2(0) = 0$, $x_3(0) = 0$, $x_4(0) = 0$ i sile $f_x = 0$. Krok całkowania `KrokCalk` = 0.01s. Czas w sekundach, wychylenie kątowe x_1 w radianach, prędkość kątowa x_2 w rad/ś, przesunięcie wózka x_3 w metrach, prędkość wózka x_4 w m/ś

układ wózek-wahadło: $M = 10$kg, $m = 30$kg, $l = 30$m, $b = 0$;

sieć neuronowa: `LNeuronUkr`=10–5), `WzmNeu`=1000, co zapewniało możliwość podniesienia wahadła odchylonego od pionu o około 73°;

algorytm genetyczny: `LSieci`=30, `LSelBez`=2(8), `LStanPocz`=20, `LKroTest`=1000, $\beta = 0.1$, `PrawdKrzyz`=0.2, `PrzedzMut`=0.1, `CzestMut`=0.01.

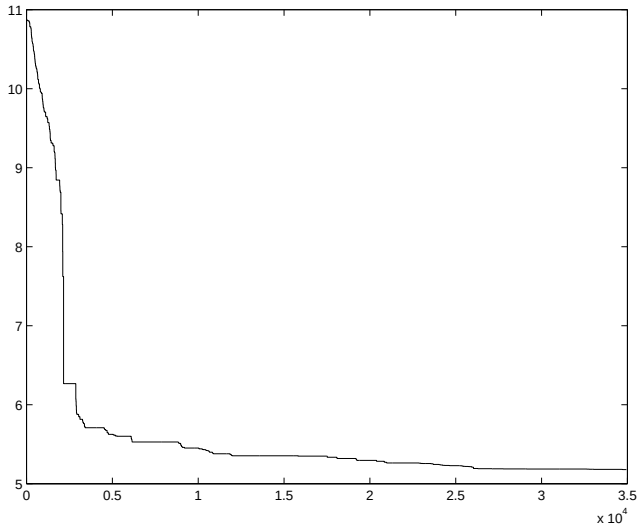
W czasie uczenia generowano losowo 20 stanów początkowych o równomiernym rozkładzie prawdopodobieństwa z zakresu: $\alpha \in (-\pi/6, \pi/6)$, $\dot{\alpha} \in (-\pi/3, \pi/3)$, $x \in (-10, 10)$,, $\dot{x} \in (-10, 10)$ i dla każdego z nich sieć sterowała wahadłem przez 1000 kroków.

Na rys. 6.65 przedstawiono otrzymany dla tych parametrów przebieg minimalnego błędu podczas uczenia sieci. Błąd E obliczany był według wzoru (6.56) dla najlepszej sieci w danym cyklu, dla 100 wybranych losowo i niezmiennych testowych stanów początkowych, różnych od tych dla których określana była jakość sieci podczas działania algorytmu genetycznego.

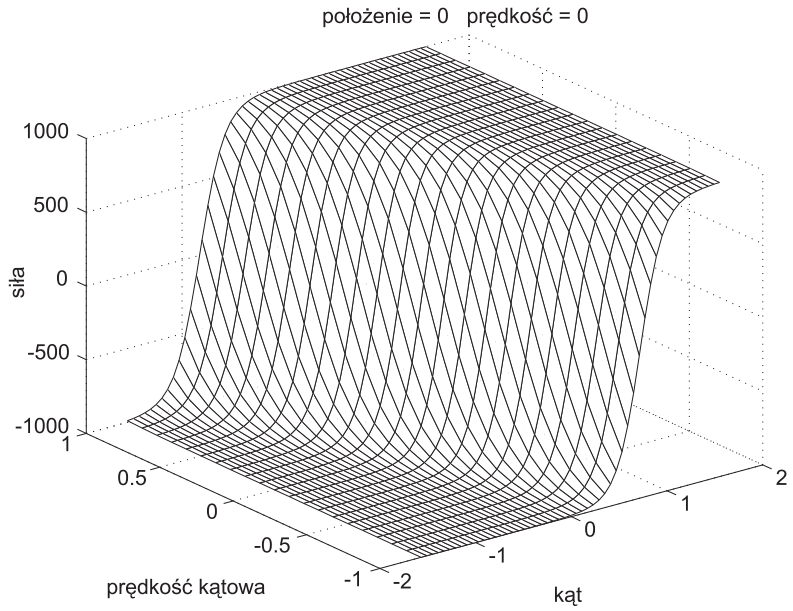
W przedstawionym przykładzie przeprowadzono ponad 30000 cykli uczenia – realizacja jednego cyklu algorytmu genetycznego wykonywana przez komputer z procesorem Athlon 850MHz trwała około 20s.

Własności otrzymanej sieci neuronowej zilustrowano na rysunku 6.66 gdzie przedstawiono sygnał wyjściowy sieci w funkcji dwóch sygnałów wejściowych, kąta i prędkości kątowej wahadła przy przyjęciu zerowego położenia wózka i zerowej prędkości wózka.

Przebieg sterowania wahadłem za pomocą nauczonej sieci pokazany jest na rys. 6.67 dla dwóch różnych stanów początkowych. Widać na nich stopniową (z przeregulo-

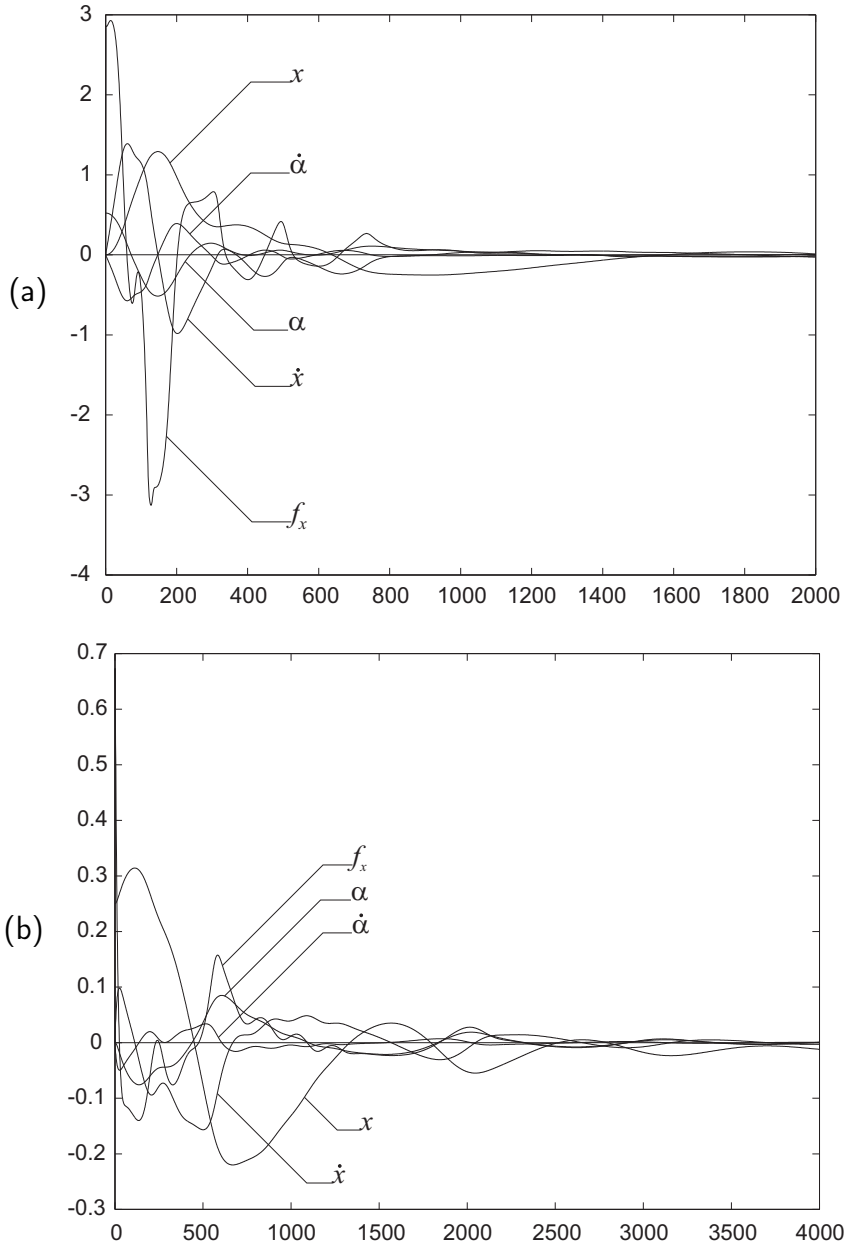


Rys. 6.65: Przebieg błędu uczenia sieci neuronowej algorytmem genetycznym – skala log-lin



Rys. 6.66: Sygnał wyjściowy sieci neuronowej w funkcji kąta i prędkości kątowej wahadła przy przyjęciu zerowego położenia wózka i zerowej prędkości wózka

waniami) stabilizację wahadła. Na szczególną uwagę zasługuje działanie sterownika doprowadzające wózek z wahadłem do położenia zerowego ze stanu początkowego $[\,0\ \ 0\ \ 5\ \ 0\,]^t$. Następuje przy tym zaburzenie równowagi wahadła i chwilowe zwiększenie wartości kryterium (pogorszenie jakości) po to aby efekt końcowy byl lepszy. Jest to typowy test poprawności działania wieloetapowych procesów decyzyjnych.



Rys. 6.67: Sterowanie wahadłem przy pomocy sieci neuronowej od stanu: (a) $[\pi/6 \;\; 0 \;\; 0 \;\; 0]^t$, (b) $[0 \;\; 0 \;\; 5 \;\; 0]^t$. Na rysunkach przedstawiono wykresy α, $\dot{\alpha}/2$, $x/20$, $\dot{x}/20$ i $f_x/300$

6.9.5. Programowanie dynamiczne

W przypadku znajomości rozkładów prawdopodobieństw czynników losowych optymalizację wieloetapowego procesu decyzyjnego można przeprowadzić stosując metody rachunku wariacyjnego. Jednak ze względu na od dawna znane trudności uzyskiwania użytecznych rozwiązań w przypadku systemów nieliniowych lub dyskretnych jedną z konstruktywnych metod jest programowanie dynamiczne Bellmana (Bellman, Dreyfus 1960). Rozwiązuje ona zadania poszukiwania ekstremum funkcji wielu zmiennych przy istnieniu ograniczeń stosując tzw. zasadę optymalności:

Zasada optymalności

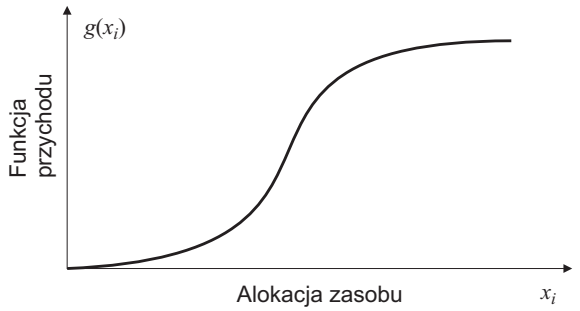
Polityka optymalna ma tę własność, że niezależnie od początkowego stanu i pierwszej decyzji pozostałe decyzje muszą stanowić politykę optymalną ze względu na stan wynikający z pierwszej decyzji.

Obliczenie maksimum funkcji N zmiennych:

$$R(x_1, x_2, \ldots, x_N) = g_1(x_1) + g_2(x_2) + \ldots + g_N(x_N)$$

przy warunkach:

$$x_1 + x_2 + \ldots + x_N = x,$$
$$x_i \geq 0, \qquad i = 1, 2, \ldots, N$$



Rys. 6.68: Typowa funkcja przychodu

Wprowadzamy ciąg funkcji $f_N(x)$, $N = 1, 2, \ldots$, $x \geq 0$ zdefiniowanych następująco:

$$f_N(x) = \max_{\{x_i\}} R(x_1, x_2, \ldots, x_N)$$

$$x_i \geq 0, \qquad \sum_{i=1}^{N} x_i = x$$

Początkowa alokacja zasobu x_N dla N-tej działalności daje przychód całkowity wynoszący

$$g_N(x_N) + f_{N-1}(x - x_N)$$

dla $N = 2, 3, \ldots,$ $x \geq 0$, przy czym $f_1(x) = g_1(x)$.

$$f_N(x) = \max_{0 \leq x_N \leq x} [g_N(x_N) + f_{N-1}(x - x_N)]$$

dla $N = 2, 3, \ldots,$ $x \geq 0$, przy czym $f_1(x) = g_1(x)$.

Kolejne kroki algorytmu

$$f_1(x) = g_1(x)$$
$$f_2(x) = \max_{0 \le x_2 \le x} [g_2(x_2) + f_1(x - x_2)]$$
$$f_3(x) = \max_{0 \le x_3 \le x} [g_3(x_3) + f_2(x - x_3)]$$
$$\vdots$$
$$f_N(x) = \max_{0 \le x_N \le x} [g_N(x_N) + f_{N-1}(x - x_N)]$$

6.9.6. Metoda różnic czasowych

Inną koncepcją rozwiązywania problemu optymalizacji, dominującą w ostatnich latach, jest uczenie ze wzmocnieniem, a ściślej jeden z wynikających z tej koncepcji algorytmów, zwany *metodą różnic czasowych* (ang. *temporal difference* – TD).

Metoda TD wprowadza obok pojęć *optymalna polityka* i *nagroda* uzyskiwana za akcję w danym stanie, również pojęcie *wartości* lub *użyteczności* pary stan–akcja (ang. *value, utility*) mówiącej o oczekiwanej sumie nagród za przyszłe decyzje:

$$v(s_t, a) = \mathcal{E}\left[r(s_t, a) + \max_{\text{polityka}} \sum_{j=1}^{N-1} r_{t+j}\right] = \mathcal{E}\left[r(s_t, a) + \max_b v(s_{t+1}, b)\right]$$

gdzie: $r(s_t, a)$ oznacza nagrodę powstałą na skutek akcji a podjętej gdy system znajdował się w stanie s_t, $\max_{\text{polityka}} \sum_{j=1}^{N-1} r_{t+j}$ jest sumą nagród za przyszłe sterowania według najlepszej polityki, zaś $\mathcal{E}$ jest operatorem uśredniania po możliwych czynnikach losowych.

Podczas gdy nagroda wskazuje co jest korzystne w danym kroku, wartość stanu określa co jest korzystne w przeciągu długiego czasu jeśli proces przejdzie do tego stanu.

W trakcie działania algorytmu TD stanom procesu decyzyjnego nadaje się wartości zgodnie z zasadą

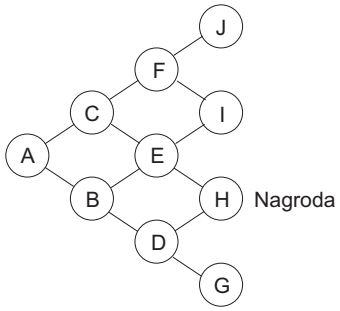
$$v(s_t, a) \leftarrow v(s_t, a) + \alpha(r(s_t, a) + \max_b v(s_{t+1}, b) - v(s_t, a))$$

gdzie s_t jest stanem bieżącym, s_{t+1} stanem następnym, zaś α jest wspolczynnikiem z przedziału (0,1). Dzięki temu, w wyniku wielokrotnego powtarzania procesu decyzyjnego, wartości stanów wchodzących w skład optymalnej trajektorii wzrastają, a innych maleją.

Po zakończeniu uczenia postępowanie polega na podejmowaniu w stanie s akcji a zapewniającej maksymalną wartość $v(s, a)$.

Przykład 6.17

Omówimy tu zadanie znajdywania drogi w grafie prowadzącej do nagrody (rys. 6.69). Droga zaczyna się w węźle A, a nagroda znajduje się w węźle H. Przedstawimy tu jedną z metod znajdywania optymalnej strategii w wielu próbach. Przyjmijmy, że w każdym z węzłów od A do F znajduje się pudełko, w którym znajduje się początkowo jednakowa liczba kul białych i czarnych. Będąc w węźle wyciągamy losowo



Rys. 6.69: Zadanie znalezienia drogi do nagrody

jedną kule z pudełka i pozostawiamy obok węzła. Wyciągnięcie kuli białej powoduje wykonanie kroku w kierunku prawo-góra, a kuli czarnej krok w kierunku prawo-dół. Po trzech krokach osiągnięty zostaje węzeł w końcowej warstwie. Wtedy, gdy osiągnięty został węzeł z nagrodą cofamy się do przebytych węzłów wrzucając do pudełka leżącą obok kulę oraz dodatkowo kulę o takim samym kolorze. Gdy osiągnięto węzeł nie zawierający nagrody cofamy się do przebytych węzłów zabierając leżące obok nich kule. Postępowanie to jak można wykazać (i co jest intuicyjnie oczywiste) doprowadza do ustalania się liczby kul w pudełkach (z możliwymi fluktuacjami) zapewniającymi dotarcie do wygranej. W omawianym przykładzie uzyskuje się granicznie: węzły C, E – same kule czarne, węzeł A znaczna przewaga kul czarnych, węzły B, F – jednakową liczbę kul czarnych i białych, węzeł D – kule białe.

Analogicznie, optymalną strategię wyznaczyć można usuwając z grafu kolejno drogi prowadzące bezpośrednio do pustych węzłów końcowych lub węzłów, z których nie ma drogi do przodu. ■

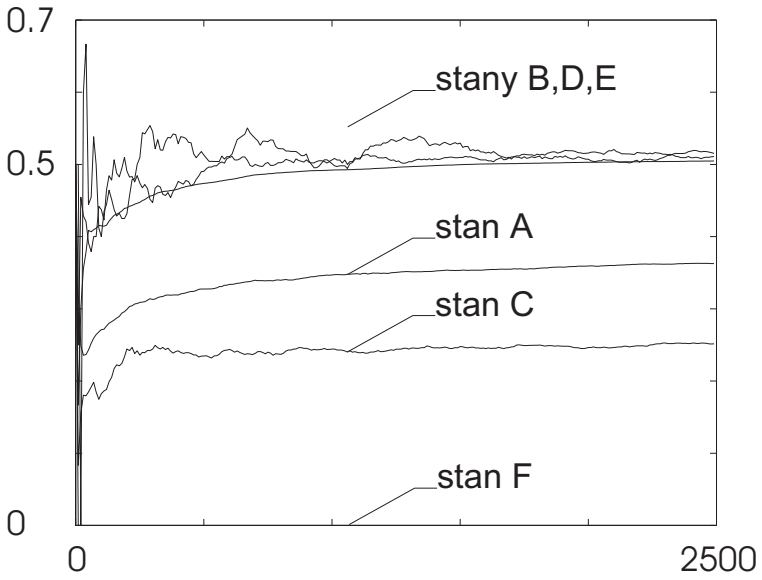
Poniżej podany zostanie przykład uaktualniania wartościowania stanow zadania po każdym kroku. Po zakończeniu uczenia strategia postępowania polega na wyborze następnego stanu o największej wartości. Zaproponowany algorytm uczenia jest szczególnym przypadkiem metody różnic czasowych.

Przykład 6.18

Rozpatrzymy zadanie z poprzedniego przykładu (rys. 6.69). Niech kierunki ruchów (góra, dół) będą wybierane losowo. Wtedy algorytm wartościowania stanów ma postać:

$$w_t = \alpha(w_{t+1} - w_t)$$

gdzie w_t i w_{t+1} oznaczają wartości przypisane stanom osiągniętym w chwili t i chwili następnej, $t+1$, zaś α jest stałym współczynnikiem. Przykład działania tego algorytmu jest przedstawiony na rysunku 6.70. ■



Rys. 6.70: Przebieg procesu nadawania wartości stanom

Rozdział 7

Systemy ekspertowe

System ekspertowy jest systemem komputerowym pełniącym rolę eksperta lub rozwiązującym zadania wymagające profesjonalnej ekspertyzy.

Inna definicja – za system ekspertowy uważa się system komputerowy posiadający wiedzę pobraną od eksperta i przechowywaną w formie umożliwiającej dostarczenie przez system inteligentnej porady lub podjęcie inteligentnej decyzji. Pożądana dodatkową cechą (przez wielu uważanych za podstawową) jest zdolność systemu do przedstawienia, na żądanie, drogi rozumowania w sposób zrozumiały dla użytkownika.

Rozróżnia się systemy ekspertowe współpracujące i komunikujące się z człowiekiem (systemy doradcze i diagnostyczne) i systemy sterujące złożonym procesami (np. platforma wiertnicza 500 sygnałów analogowych i 2500 cyfrowych, Hubble Space Telescope – 6000 czujników). Te ostatnie przyjęto nazywać systemami ekspertowymi, gdy okazało się, że system sterujący złożonymi procesami musi zawierać bazę wiedzy i przeprowadzać wnioskowanie i jego struktura nie różni się od systemów ekspertowych współpracujących z człowiekiem.

7.1. Historia

MYCIN – (połowa lat 70-tych, Stanford) podaje diagnozę i sposoby leczenia bakteryjnych infekcji krwi i zapalenia opon mózgowych. Zawiera ponad 500 reguł. Budowa systemu zajęła ponad 50 osobo lat.

DENDRAL – (połowa lat 60-tych, Stanford) wyznacza opis strukturalny złożonych organicznych związków chemicznych na podstawie ich spektrogramów masowych i innych danych.

MACSYMA – (rozpoczęcie budowy: 1969, MIT) rozwiązywanie zadań matematycznych poprzez przekształcenia symboliczne. Obecnie zawiera ponad 300000 linii kodu w języku LISP. 100 osobo lat.

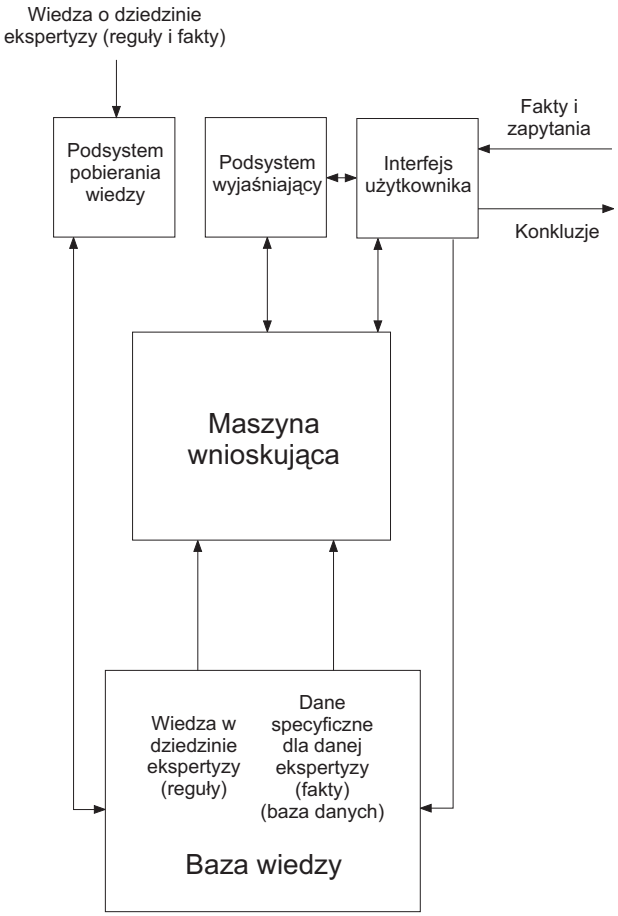
PROSPECTOR – (późne lata 70-te, Stanford) na podstawie danych sejsmicznych i innych generuje wnioski dotyczące obecności złóż rud mineralnych.

XCON – (1979, Digital Equipment Corporation i Carnegie Mellon University). Firma DEC miała problemy przy konfiguracji kom-

puterów VAX składanych z ponad 400 zespołów wybieranych przez klientów. Przy zastosowaniu systemu szkieletowego OPS4, w ciągu trzech miesięcy zbudowano system prototypowy zawierający 250 reguł. Rezultaty uzyskiwane przy zastosowaniu prototypu przekonały DEC do zainwestowania w budowę kompletnego systemu, który został zakończony w 1981 roku. Stał się wielkim sukcesem komercyjnym.

7.2. Struktura systemu ekspertowego

Na rys. 7.1 przedstawiono strukturę typowego systemu ekspertowego.



Rys. 7.1: Struktura typowego systemu ekspertowego

7.3. Reprezentacja wiedzy

W standardowym systemie ekspertowym wiedza może być reprezentowana przez jedną z metod omówionyvh w rozdziale 3.

7.4. Neuronowy system ekspertowy

Jednym z obiecujących zastosowań sieci neuronowych są oparte na nich systemy ekspertowe. Najczęściej służą do diagnostyki i sterowania. Reprezentacja wiedzy w neuronowym systemie ekspertowym ma zupełnie inny charakter niż w dotychczas omawianych systemach regułowych – jest zawarta w wagach i strukturze nauczonej sieci.

Przykład 7.1

Przykład neuronowego systemu ekspertowego diagnozy chorób skóry przedstawiono na rys. 7.2. System ten o nazwie DESKNET zaprojektowany został głównie w celu wspomagania nauczania studentów medycyny Zbudowano go w postaci sieci neuronowej o strukturze 96–20–10 zawierającej ciągłe neurony unipolarne.

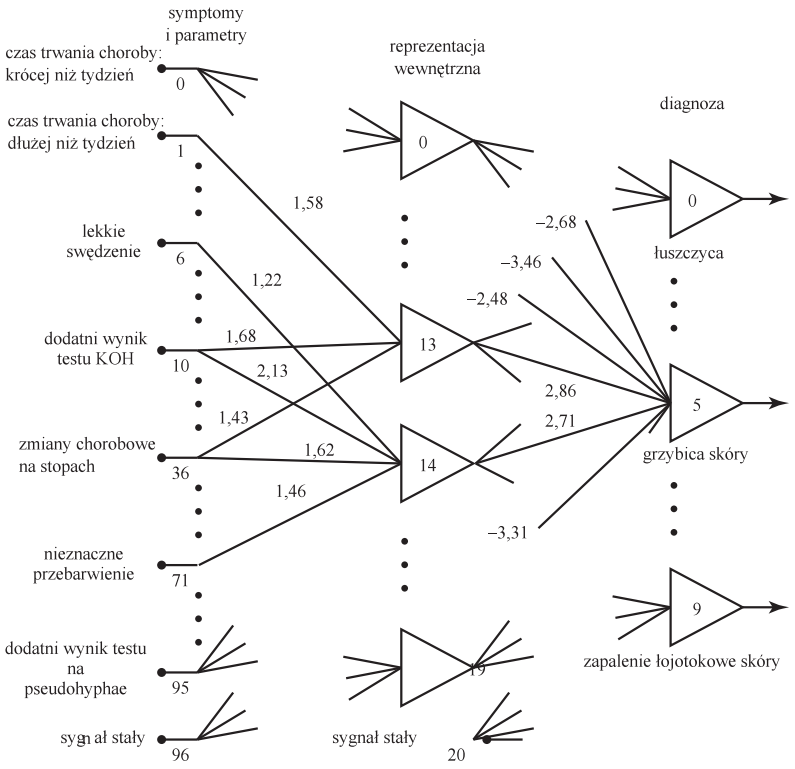
Sygnałami wejściowymi sieci były między innymi następujące symptomy i ich parametry: umiejscowienie, rozmieszczenie, kształt, liczba uszkodzeń, obecność aktywnych granic, liczba złuszczeń, wysokość brodawek, barwa, zmiana pigmentacji, swędzenie, krostowatość, limfadenopatia skóropochodna, zgrubienia na dłoniach, wyniki badań mikroskopowych, obecność wykwitu pierwotnego oraz wyniki KOH przy teście mikologicznym. Ponadto na wejście sieci podawano czas trwania objawów chorobowych, rozróżniając dnie i tygodnie.

System miał łącznie 97 wejść, w tym jedno wejście z sygnałem stałym, oraz 10 wyjść. Sygnały wejściowe przyjmowały trzy wartości: 0, 1 i 0,5 reprezentujące odpowiednio: występowanie poszczególnych symptomów, ich niewystępowanie oraz brak danych o symptomie. Sygnały wyjściowe poszczególnych neuronów w warstwie wyjściowej przedstawiają w reprezentacji lokalnej następujące 10 schorzeń: łuszczyca, łupież mieszkowy czerwony, liszaj płaski, łupież różowy, łupież pstry, grzybica skóry, chłoniak z komórek T, wtórna kiła, przewlekłe kontaktowe zapalenie skóry oraz zapalenie łojotokowe skóry.

Dane uczące pochodziły od 250 pacjentów. Sieć nauczano standardową metodą propagacji wstecznej. Jakość nauczonej sieci była testowana za pomocą danych nie używanych w procesie uczenia, zebranych od 99 pacjentów. Poprawną diagnozę otrzymano w 70% przypadków. Wyłączając łuszczycę, uzyskano dokładność ponad 80%. Pacjenci z łuszczycą diagnozowani byli poprawnie tylko w 30%. Jednak jak stwierdzają lekarze, objawy łuszczycy często są podobne do innych chorób typu grudkowo-złuszczającego, a tym samym trudne do diagnozy nawet dla lekarza specjalisty.

Jednym z istotnych wymagań jest zdolność systemu ekspertowego do przedstawiania łańcucha wnioskowania prowadzącego do wytworzonej przez niego konkluzji. W systemach regułowych, w których wnioskowanie polega na wykazaniu spełniania warunków w określonych regułach *jeżeli — to* spośród istniejących w systemie, przedstawienie łańcucha wnioskowania prowadzącego do danej konkluzji jest możliwe do realizacji. Systemy neuronowe dochodzą do konkluzji na drodze wielu złożonych, nieliniowych i równoległych transformacji i wpływ na konkluzję mają wszystkie neurony. W związku z tym odtworzenie w nich i wyrażenie w postaci werbalnej łańcucha wnioskowania, wskazującego wpływ indywidualnego wejścia lub grupy wejść na ostateczną konkluzję, jest bardzo trudne. Oczywiście, można badać eksperymen-

talnie wpływ wartości wejść na konkluzję, ale nie jest to równoważne ze znajomością łańcucha wnioskowania. Ostatnie badania w dziedzinie sieci neuronowych próbują rozwiązać zadanie wytworzenia bazy wiedzy w postaci reguł i faktów na podstawie wag nauczonej sieci neuronowej. ■



Rys. 7.2: Neuronowy system ekspertowy do diagnostyki chorób skóry (na podstawie [68])

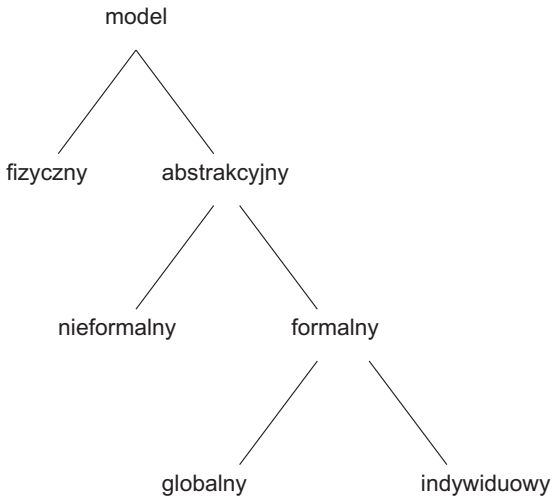
Rozdział 8

Modelowanie systemów i sztuczne życie

8.1. Koncepcje modelowania systemów

Model układu fizycznego – układ fizyczny lub opis, podobny (izomorficzny, analogiczny) do układu badanego, ale prostszy i łatwiej dostępny badaniom

Na rysunku 8.1 przedstawiono taksonomię modeli. Poszczególne rodzaje modeli omówiono w dalszej części rozdziału.



Rys. 8.1: Taksonomia modeli

8.1.1. Modele fizyczne

Modele fizyczne oprócz opisów w języku naturalnym były najdawniejszymi typami modeli. Przykłady modeli fizycznych – patrz rys. 8.2

Rys. 8.2: Przykłady modeli fizycznych

8.1.2. Modele abstrakcyjne

Modele abstrakcyjne są opisami modelowanego systemu w pewnym języku. Nie są modelami materialnymi jak modele fizyczne. Opis może być nieformalny, np. opis w języku naturalnym lub rysunek odręczny lub może być formalny, np. formuła matematyczna lub program komputerowy.

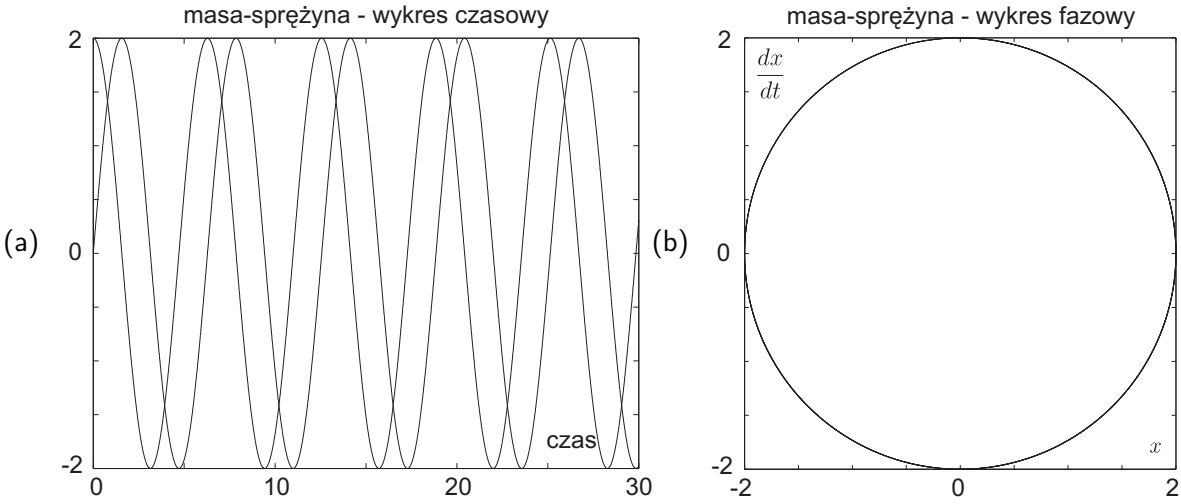
Modele matematyczne

Przykład 8.1

Układ masa–sprężyna bez tarcia. Równanie ruchu ma następującą postać.

$$\frac{d^2x}{dt^2} = -x \tag{8.1}$$

Na rys. 8.3 przedstawiono rozwiązanie powyższych równań ruchu. ■



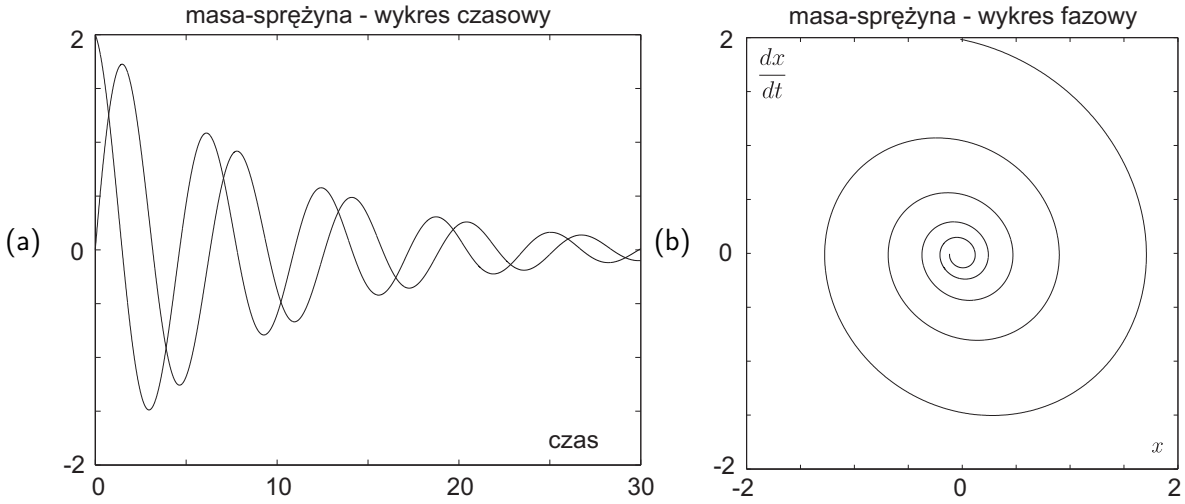
Rys. 8.3: Rozwiązania równań 8.1. Przebiegi czasowe położenia i prędkości masy umieszczonej na sprężynie: (a) we współrzędnych kartezjańskich i (b) fazowych

Przykład 8.2

Układ masa–sprężyna z tarciem

$$\frac{d^2x}{dt^2} = -c\frac{dx}{dt} - x \tag{8.2}$$

Na rys. 8.4 przedstawiono rozwiązanie powyższych równań ruchu. ■



Rys. 8.4: Rozwiązania równań 8.2. Przebiegi czasowe położenia i prędkości masy umieszczonej na sprężynie w przypadku występowania tarcia: (a) we współrzędnych kartezjańskich i (b) fazowych

Przykład 8.3

System drapieżnik–ofiara (równania Lotki-Volterry)

$$\begin{aligned} \frac{dx}{dt} &= k_1 x - k_2 xy \\ \frac{dy}{dt} &= k_2 xy - k_3 y \end{aligned} \tag{8.3}$$

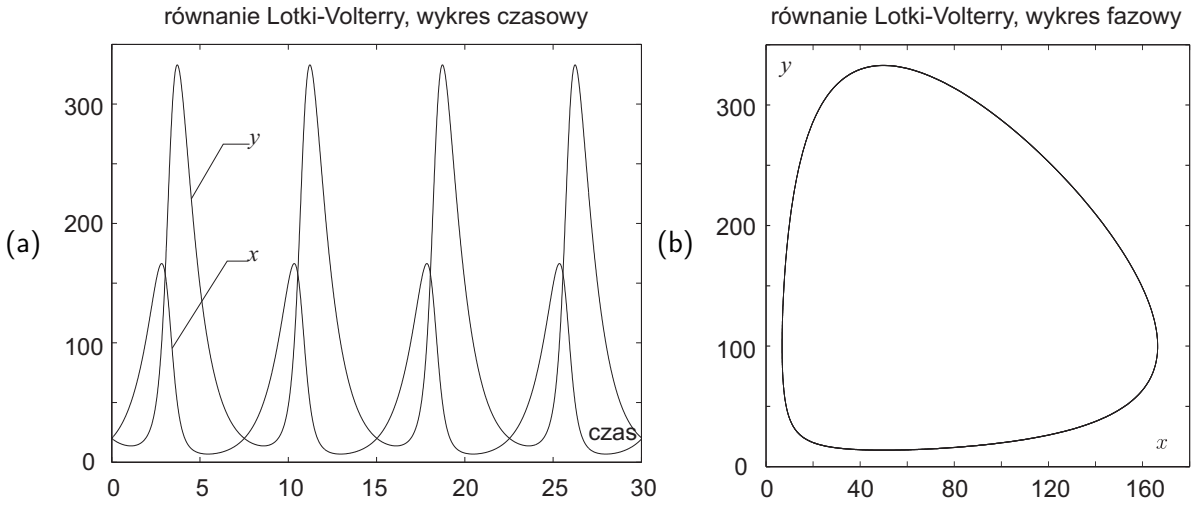
Na rys. 8.5 przedstawiono rozwiązanie numeryczne powyższych równań. ■

Przykład 8.4

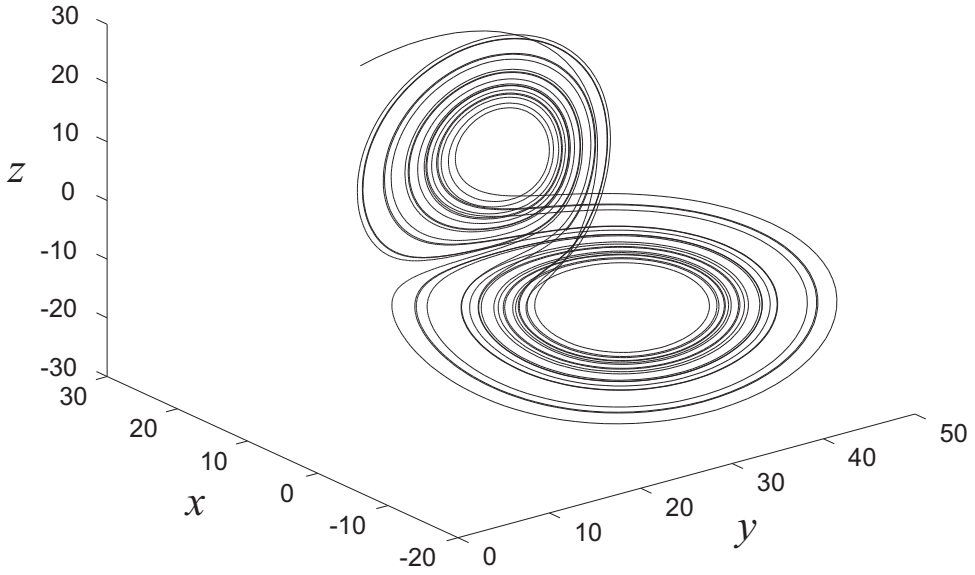
Równanie Lorenza. Klasyczny przykład równania, którego rozwiązaniem są przebiegi chaotyczne.

$$\begin{aligned} \frac{dx}{dt} &= -\frac{8}{3}x + yz \\ \frac{dy}{dt} &= -10y + 10z \\ \frac{dz}{dt} &= -xy + 28y - z \end{aligned} \tag{8.4}$$

Trajektorie rozwiązania numerycznego równania Lorenza przedstawiono na rys. 8.6. Trajektorie te nigdy się nie przecinają – przebieg aperiodyczny. ■



Rys. 8.5: Rozwiązania równań 8.3. Przebiegi czasowe liczby drapieżników i ofiar: (a) we współrzędnych kartezjańskich i (b) fazowych



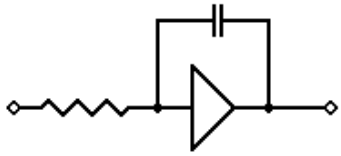
Rys. 8.6: Rozwiązanie równania Lorenza

Modele matematyczno–fizyczne – maszyny analogowe

W modelowaniu matematyczno-fizycznym tworzy się model matematyczny badanego procesu (najczęściej w postaci równań różniczkowych). Następnie budujemy się układ fizyczny (model) opisany identycznymi równaniami jak układ badany. Obserwując zachowanie modelu uzyskuje się wiedzę o zachowaniu badanego układu. W takim modelowaniu, najczęściej stosowanym układem fizycznym jest układ elektryczny zestawiany z oporności, indukcyjności, pojemności, elementów nieliniowych i wzmacniaczy. Zestawiając odpowiednio te elementy można w łatwy sposób uzyskać

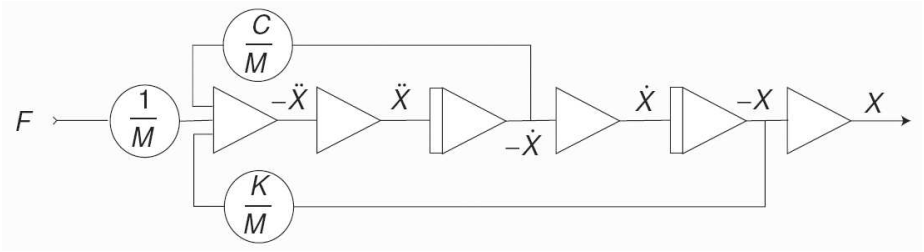
obwód elektryczny opisany zadanym równaniem różniczkowym.

Tego typu modelowania były bardzo popularne do lat 70. ubiegłego wieku. Produkowano uniwersalne zestawy elementów z wyprowadzonymi końcówkami na specjalnych panelach. Zestawy takie nazywały się maszynami (komputerami) analogowymi i przez wiele lat skutecznie rywalizowały z maszynami cyfrowymi w dziedzinie uzyskiwania rozwiązań równań różniczkowych. Dopiero pojawienie się tanich i szybkich maszyn cyfrowych wyparło tą technikę. Na rys. 8.7 przedstawiono okładkę czasopisma Radio-Electronics z roku 1959 ze zdjęciem ilustrującym programowanie maszyny analogowej oraz podstawowy obwód maszyny analogowej, układ całkujący.



Rys. 8.7: Programowanie maszyny analogowej, a obok układ całkujący

Na rysunku 8.8 przedstawiono układ elektryczny modelujący układ mechaniczny masy i sprężyny w przypadku występowaniu tarcia.



Rys. 8.8: Układ elektryczny równoważny układowi masy i sprężyny

Modele matematyczno–komputerowe – symulatory

W modelowaniu matematyczno komputerowym tworzy się model matematyczny modelowanego procesu, a następnie symuluje się jego zachowanie poprzez numeryczne rozwiązywanie opisujących go równań w komputerze. Sam opis matematyczny często od razu powstaje jako program komputerowy. Symulatory komputerowe złożonych obiektów czy procesów wsparte często przez urządzenia mechaniczne symulujące ruch obiektów (np kabiny pilota samolotu) stanowią bardzo ważne zastosowania informatyki w wielu dziedzinach techniki i nauki.

Na rys. 8.9 przedstawiono widok ekranu symulatora panelu sterowania silnika okrętowego (Szkoła Morska w Gdyni) i jeden z ekranów popularnego symulatora lotu firmy Microsoft.

Rys. 8.9: Symulator silnika okrętowego i symulator lotu

8.2. Modelowanie indywiduowe

Można wyróżnić dwa poniżej podane podejścia do modelowania formalnego systemów.

Modele globalne Model matematyczny globalnego zachowania się systemu (oparty najczęściej o równania różniczkowe). Symulacja działania systemu poprzez rozwiązanie równania różniczkowego: numeryczne lub w postaci zamkniętej.

Modele indywiduowe Określenie lokalnych reguł funkcjonowania i wzajemnego oddziaływania elementów systemu. Symulacja komputerowa ewolucji systemu składającego się z wielu takich elementów. Złożone globalne zachowanie się modelu uzyskuje się jako rezultat oddziaływań o zasięgu lokalnym.

Modele omawiane do tej pory należały do grupy modeli globalnych. Obecnie zajmiemy się modelami indywiduowymi. Klsycznymi metodami modelowania indywiduowego są *dynamikę molekularną* i *automaty komórkowe*.

Na rysunku 8.10 przedstawiono przykład systemu, który może być modelowany globalnie (równanie różniczkowe wynikające z praw mechaniki płynów) i indywiduowo (modelowanie ruchów i zderzeń wszystkich cząsteczek).

8.2.1. Dynamika molekularna

Metoda dynamiki molekularnej (MD od ang. molecular dynamics) polega na symulacji ruchu cząsteczek w polu sił przez nie wytwarzanych lub sił zewnętrznych.

W przypadku układu N cząsteczek, siła $\mathbf{F}_i$ działająca na cząsteczkę i wynosi

$$\mathbf{F}_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{F}_{ij} + \mathbf{F}_i^z$$

gdzie $\mathbf{F}_{ij}$ jest siłą, z którą cząsteczka j oddziałuje na cząsteczkę i, a $\mathbf{F}_i^z$ jest siłą pochodzącą z zewnątrz układu. Znając energię potencjalną cząsteczki, Ψ_i, siłę na nią oddziałującą wyznacza się ze wzoru

$$\mathbf{F}_i = -\nabla_{\mathbf{r}_i} \Psi_i$$

gdzie $\nabla_{\mathbf{r}_i}$ oznacza wartość gradientu w punkcie $\mathbf{r}_i$.

Często, w symulacjach przyjmuje się aproksymację energii potencjalnej pomiędzy dwoma atomami za pomocą funkcji

$$\psi_{ij} = 4\epsilon \left[\left(\frac{\sigma}{|\mathbf{r}_i - \mathbf{r}_j|} \right)^{12} - \left(\frac{\sigma}{|\mathbf{r}_i - \mathbf{r}_j|} \right)^{6} \right]$$

zwanej potencjałem Lennarda–Jonesa, gdzie ϵ i σ są stałymi.

Dla wyznaczenia ruchu cząsteczek w polu sił należy rozwiązywać układ równań różniczkowych drugiego rzędu

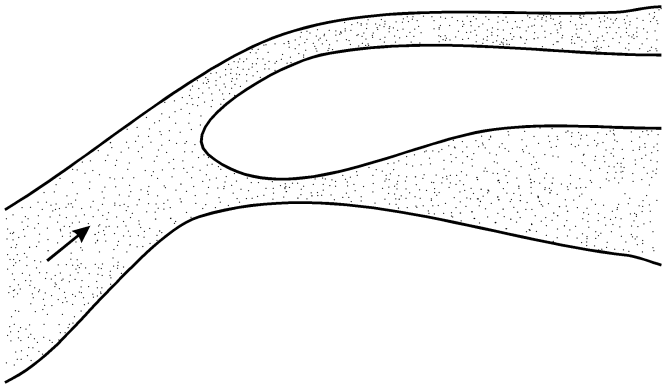
$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i, \qquad i = 1, 2, \ldots, N$$

gdzie m_i jest masą cząsteczki i, a $\mathbf{F}_i$ jest siłą działającą na cząsteczkę. Równania powyższe można zastąpić równoważnym układem dwóch równań rzędu pierwszego

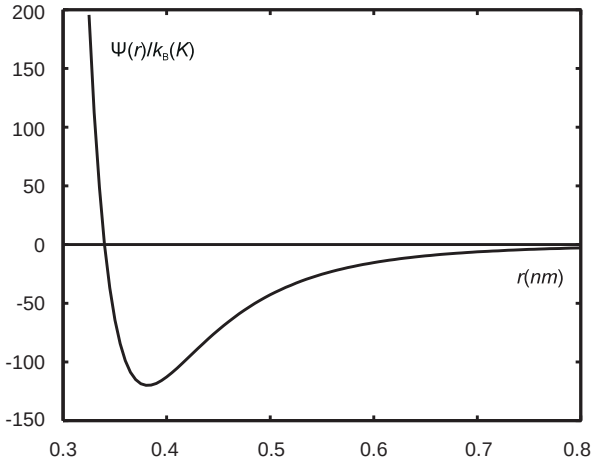
$$\begin{array}{rcl} \dot{\mathbf{r}}_i & = & \mathbf{v}_i \\ \dot{\mathbf{v}}_i & = & \mathbf{F}_i \end{array}, \qquad i = 1, 2, \ldots, N$$

Tak więc, aby przeprowadzać symulację ruchu N cząsteczek należy rozwiązywać układ $3N$ równań różniczkowych rzędu drugiego lub $6N$ równań rzędu pierwszego.

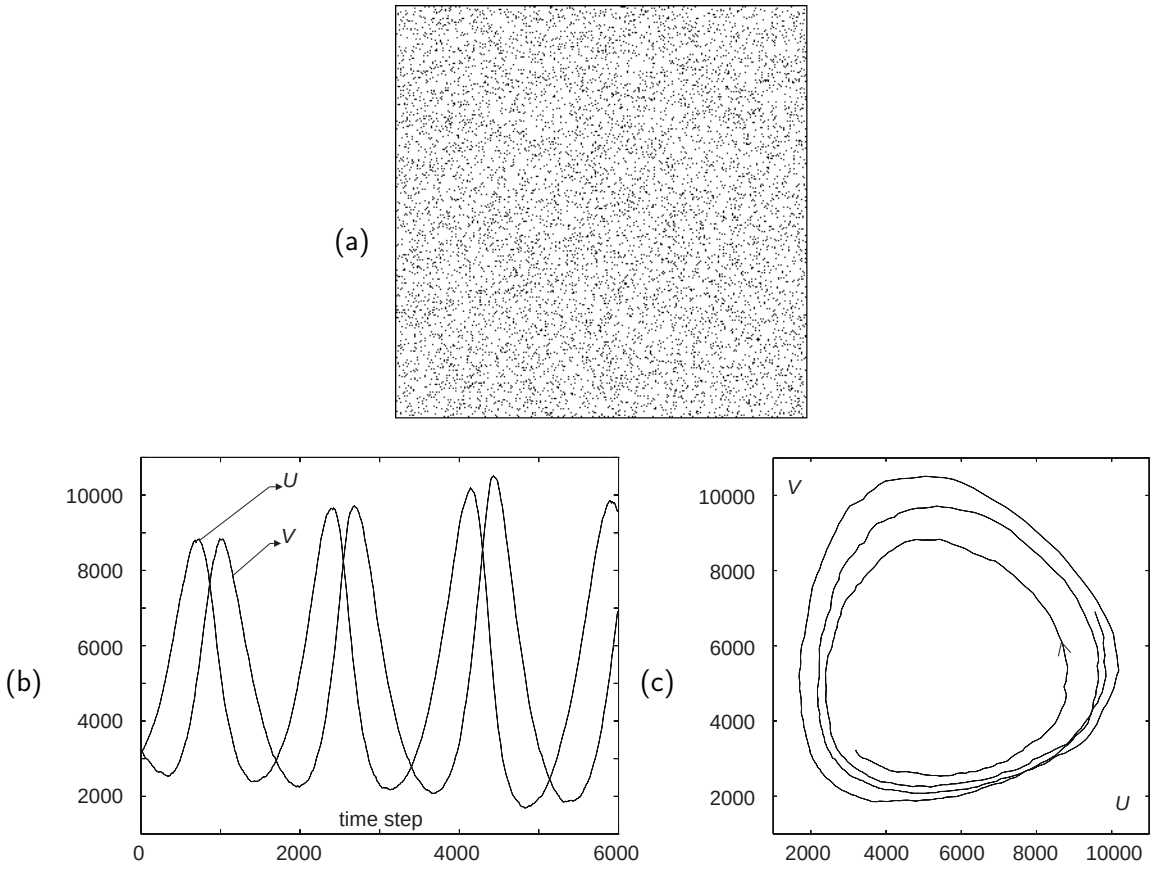
Na rysunku 8.12abc przedstawiono wyniki indywiduowego modelowania systemu drapieżnik-ofiara. Punkty na rysunku 8.12a oznaczają drapieżniki i ofiary. Zetkniecię się drapieżnika z ofiarą powoduje zamianę ofiary w drapieżnika. Otrzymane liczebności drapieżników i ofiar w czasie przedstawione są na rys. 8.12bc. Porównaj te wyniki z rozwiązaniem równania Lotki-Volterry przedstawionym na rys. 8.5.



Rys. 8.10: Przykład systemu, który można modelować globalnie i indywiduowo



Rys. 8.11: Potencjał Lennarda–Jonesa dla wartości $\epsilon/k_B = 120$K i $\sigma = 0.34$nm (przyjmowanych dla ciekłego argonu)



Rys. 8.12: System drapieżnik ofiara modelowany indywiduowo: (a) punkty reprezentujące poruszające się drapieżniki i ofiary, (b,c) wykresy liczbności w czasie drapieżników (V) i ofiar (U)

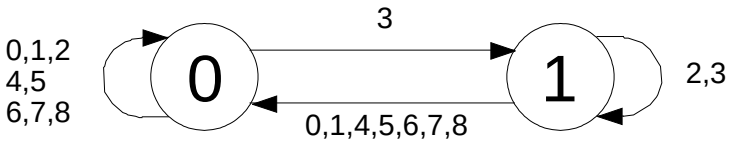
8.2.2. Automaty komórkowe

Zaproponowany przez Neumanna i Ulama w końcu lat 40. automat komórkowy jest zbiorem komórek rozłożonych regularnie w nieograniczonej przestrzeni (np. dwu-

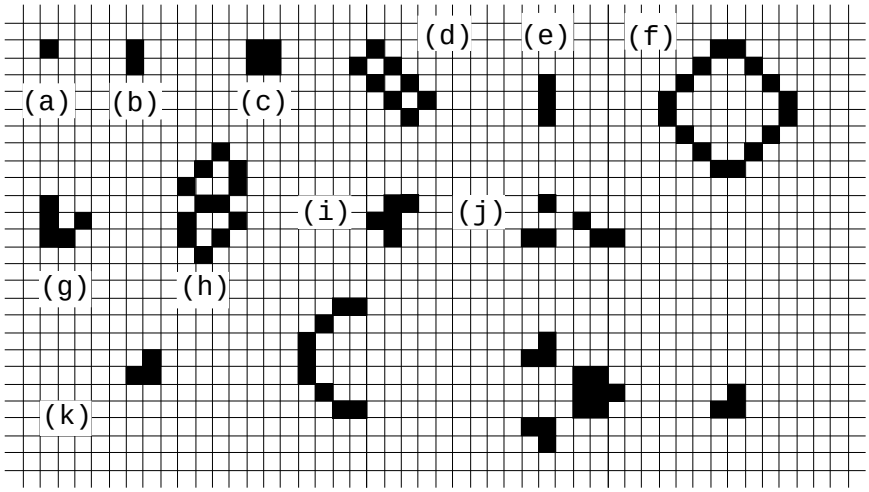
wymiarowej). W komórkach znajdują się identyczne automaty skończone, których stany zależą od stanów określonego zespołu innych automatów, zwanych sąsiednimi.

Zachowanie się komórki jest zdeterminowane przez jej funkcję przejścia, która odwzorowuje stan komórki i wszystkich komórek do niej sąsiednich na nowy stan komórki. Funkcja przejścia komórki automatu komórkowego jest identyczna dla wszystkich komórek i jest niezmienna w czasie. Stan całego automatu komórkowego jest zbiorem stanów poszczególnych komórek.

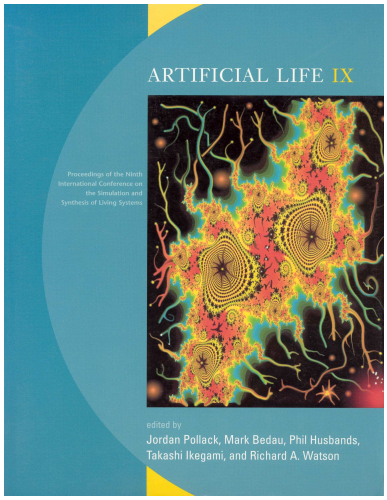
Ważną cechą, którą mogą posiadać automaty komórkowe, jest ich uniwersalność obliczeniowa.



Rys. 8.13: Graf przejść stanu komórki w grze LIFE. Przy łukach podano liczbę sąsiednich komórek w stanie „1"



Rys. 8.14: Przykłady charakterystycznych struktur gry LIFE: (a,b) konfiguracje ginące po jednym takcie, (c,d) konfiguracje stabilne, (e,f) konfiguracje oscylujące – pionowa linia zamienia się po każdym takcie w poziomą i odwrotnie, figura w kształcie kwadratu oscyluje z okresem pięciu taktów, (g) konfiguracja oscylująca i przesuwająca się, zwana „szybowcem", (h) konfiguracja rozpadająca się na dwa „szybowce" przesuwające się w przeciwnych kierunkach, (i,j) konfiguracje długowieczne osiągające stan końcowy po 1103 i 5206 taktach (stan końcowy składa się z konfiguracji stabilnych, oscylujących oraz oscylujących i przesuwających się), (k) „działo" – konfiguracja generująca szybowce co 30 taktów i powracająca do stanu początkowego



Rys. 8.15: Okładka materiałów światowej konferencji na temat sztucznego życia

8.3. Postulaty sztucznego życia

Postulaty podane przez Langtona (1989) w artykule programowym pierwszej konferencji poświęconej zagadnieniom *sztucznego życia* (ang. artificial life), dotyczące środowiska modelowania systemów biologicznych:

- powinno składać się z populacji prostych programów lub specyfikacji;
- nie powinien w nim występować program sterujący pozostałymi programami;
- każdy program powinien zawierać opis reakcji jednostki na zaistniałe lokalne sytuacje w swoim otoczeniu włączając w to spotkania z innymi jednostkami;
- w systemie nie powinno być reguł sterujących jego globalnym zachowanie się.

Postulaty Langtona mogą być spełniane przez środowiska modelowania indywiduowego.

Z dziedziną sztucznego życia blisko związane jest pojęcie inteligencji roju

Swarm Intelligence is the property of a system whereby the collective behaviours of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge. SI provides a basis with which it is possible to explore collective (or distributed) problem solving without centralized control or the provision of a global model.

8.4. Przykładowe środowisko modelowania indywiduowego – DigiHive

Opisane zostanie tutaj środowisko dla modelowania indywiduowego rozwijane na Wydziale ETI Politechniki Gdańskiej [62, 67]. Środowisko to ukierunkowane jest na modelowanie systemów, których zachowanie się jest w istotny sposób związane z ruchem i wzajemnym oddziaływaniem swoich elementów. W szczególności umożliwia modelowanie systemów złożonych, przejawiających się występowaniem w nich

Rys. 8.16: A huge swarm of red-billed queleas returns to the communal roost at dusk, Okavango Delta, Botswana.

procesów wzrostu, samoreprodukcji, samoorganizacji i samomodyfikacji. Środowisko jest nastawione na modelowanie i wykrywanie podstawowych ogólnych własności systemów złożonych z wielu elementów, a w mniejszym stopniu na modelowanie konkretnych procesów fizycznych, chemicznych czy biologicznych.

W środowisku tym zdefiniowanym w przestrzeni dwuwymiarowej poruszają się i zderzają cząsteczki. Na skutek zderzeń cząsteczki mogą losowo łączyć się w kompleksy cząsteczek, lub kompleksy cząsteczek rozpadać. Na wyższym poziomie oddziaływań, co otwiera zupełnie nowe możliwości, kompleksy cząsteczek mogą selektywnie oddziaływać na inne cząsteczki, łącząc je lub rozrywając wiązania. Rodzaj oddziaływania zapisany jest w strukturach kompleksów w specjalnie zdefiniowanym języku.

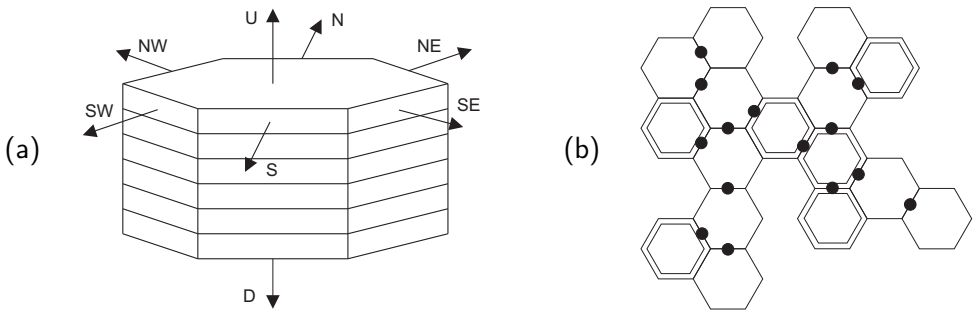
Opisywane środowisko wykorzystuje niektóre założenia, wcześniejszego środowiska rozwijanego od połowy lat osiemdziesiątych [**?**, **?**, **?**].

8.4.1. Podstawowe definicje

Środowisko zdefiniowane jest w przestrzeni dwuwymiarowej o periodycznych warunkach brzegowych. Podstawowymi obiektami środowiska są *cząsteczki* i *fotony*. Cząsteczki są obiektami trwałymi (nie są tworzone ani niszczone w trakcie symulacji) reprezentowanymi przez sześciokąty o ustalonych wymiarach. Cząsteczki występują w 256 różnych typach. Każdy typ charakteryzowany jest następującymi właściwościami: masa, energia wiązań (energia potrzebna do rozerwania wiązania) oraz energia aktywacji (minimalna energia potrzebna do zainicjalizowania reakcji). Poza wymienionymi cechami stałymi każda cząsteczka jest charakteryzowana przez aktualne wartości prędkości, położenia i energii wewnętrznej.

Podczas symulacji pomiędzy cząsteczkami mogą powstawać wiązania – dwie lub więcej powiązanych cząsteczek tworzy *kompleks cząsteczek*. Cząsteczki mogą łączyć się na kierunkach: góra (U) i dół (D), tworząc stos. Cząsteczki znajdujące się na spodzie stosu mogą łączyć się na kierunkach poziomych: północ (N), północny zachód (NW), południowy zachód (SW), południe (S), południowy wschód (SE) i północny

wschód (NE). Przykładowe wyglądy kompleksów przedstawia rysunek 8.17.



Rys. 8.17: Przykładowe kompleksy: (a) stos cząsteczek – widok z boku, (b) kompleks utworzony przez wiązania poziome – widok z góry. Sześciokąty narysowane podwójną linią oznaczają stosy, czarne punkty oznaczają wiązania pomiędzy stosami.

Wszystkie procesy w systemie są synchronizowane dyskretnym zegarem. Na początku każdego cyklu czasowego, położenie każdej cząsteczki jest zmieniane odpowiednio do jej prędkości. Jeżeli w trakcie ruchu dwie cząsteczki, nie należące do tego samego kompleksu, znajdą się w odległości mniejszej niż pewna ustalona wartość, to przyjmowane jest, że doszło do zderzenia. Zderzenia cząsteczek mogą być zarówno sprężyste, jak i niesprężyste.

Zderzenie sprężyste modelowane jest zgodnie z zasadami mechaniki klasycznej, przy czym cząsteczki traktowane są tu jako koła o ustalonym promieniu. Podczas zderzenia sprężystego zachowana jest zarówno energia kinetyczna, jak i pęd cząsteczek. Podczas zderzenia niesprężystego prędkości zderzających się cząsteczek stają się równe. Wtedy zachowany jest jedynie łączny pęd cząsteczek. Powstający przy tym deficyt energii kinetycznej zrekompensowany zostaje emisją *fotonu*.

Fotony są obiektami nietrwałymi przenoszącymi energie. Powstają w trakcie reakcji rozpraszających energię: zderzeń niesprężystych cząsteczek, tworzenia wiązań pomiędzy cząsteczkami i spontanicznej redukcji energii wewnętrznej cząsteczki. Magazynują one energie tracone przez cząsteczki, dzięki czemu energia całkowita środowiska podczas symulacji pozostaje stała. Fotony są modelowane jako punkty o zerowej masie, poruszające się z ustaloną prędkością.

Fotony mogą zderzać się z cząsteczkami. Podobnie jak w przypadku zderzeń cząsteczek, zderzenia fotonu z cząsteczką mogą być zarówno sprężyste jak i niesprężyste. Zderzenia sprężyste zmieniają jedynie kierunek fotonu, natomiast zderzenia niesprężyste prowadzą do jednej z następujących reakcji: odbicie cząsteczki uderzonej przez foton od cząsteczki sąsiedniej (znajdującej się w odległości nie większej niż pewna ustalona wartość), utworzenie wiązania pomiędzy uderzoną cząsteczką a cząsteczką sąsiednią, zerwanie wiązania pomiędzy cząsteczką trafioną a dowolną cząsteczką związaną, absorpcja fotonu (konwersja energii fotonu w energię wewnętrzną cząsteczki).

Wszystkie wymienione reakcje muszą zachowywać całkowitą energię: jeżeli foton ma zbyt małą energię do zainicjowania reakcji, reakcja nie zachodzi. W przypadku gdy energia fotonu jest większa niż zużyta na przeprowadzenie reakcji lub reakcja oddaje energię, to po zakończeniu reakcji wygenerowany zostaje foton o energii zapewniającej stałość energii systemu.

Powyżej opisane właściwości środowiska pozwalają, poprzez ustawienie zerowego prawdopodobieństwa zderzeń niesprężystych, na przeprowadzenie symulacji procesów zgodnych z mechaniką klasyczną – jest to wtedy równoważne prostej dynamice molekularnej. Z kolei ustawiając niezerowe prawdopodobieństwo zderzeń niesprężystych, w środowisku pojawią się fotony, co w konsekwencji prowadzi do powstawania różnych złożonych kompleksów cząsteczek na skutek losowego tworzenia i zrywania wiązań.

8.4.2. Oddziaływania funkcyjne

Poza omówionymi powyżej interakcjami wynikajacymi ze zderzeń cząsteczek pomiędzy sobą jak i zderzeń cząsteczek z fotonami, w środowisku zachodzą oddziaływania innego rodzaju. Poszczególne kompleksy są zdolne do rozpoznawania specyficznych struktur cząsteczek znajdujących się w ich otoczeniu (okrąg o zdefiniowanym promieniu) oraz selektywnego tworzenia i rozrywania wiązań pomiędzy cząsteczkami. Funkcja realizowana przez kompleks zakodowana jest przez typy i rozmieszczenie cząsteczek w kompleksie.

Funkcje kompleksów wyrażone są w opisanym poniżej języku, o strukturze zbliżonej do języka Prolog. Występują w nim wyłącznie następujące predykaty wbudowane: program, search, action, structure, exists, bind, unbind, move, not. Predykaty program, search, action, structure służą do organizowania struktury programu. Natomiast funkcje rozpoznawania struktur i realizacje zmian w środowisku pełnią predykaty: exists (selektywne rozpoznawanie cząsteczek), bind (utworzenie wiązania), unbind (rozrywanie wiązań) oraz move (przesuwanie cząsteczek).

Przykładowy program został przedstawiony w tablicy 8.1. Jak można zauważyć program składa się z sekwencji dwóch predykatów search i action, przy czym pierwszy z nich grupuje predykaty służące do rozpoznawania struktur, natomiast drugi, predykaty umożliwiające manipulowanie rozpoznanymi cząsteczkami.

Każdy predykat structure składa się z sekwencji wywołań predykatu exists oraz zanegowanego predykatu structure. Pozwala to na wykrycie określonej struktury przy warunku nie występowania innych określonych struktur. W przykładowym programie cząsteczka o typie 10101010 jest inhibitorem reakcji. Jej występowanie zablokowałoby zajście przemiany.

Rys. 8.18: Cząsteczka i kompleks cząsteczek rozpoznawany przez program z tablicy 8.1 (a), struktura po wykonaniu programu (b).

Za pomocą predykatu exists możliwe jest sprawdzenie istnienienia cząsteczki o określonym typie (np. exists([0,0,0,0,1,1,1,1] ...)), sprawdzenie czy cząsteczka jest związana z inną cząsteczką na danym kierunku (np. exists(... bound to V2 on N ...)), oraz czy jest przylegająca do innej cząsteczki (np. exists(... adjacent to V3 on N)).

Tablica 8.1: Przykładowy program rozpoznający strukturę przedstawioną na rys. 8.18.

```
program():–
        search(),
        action().
search():–
        structure(0).
structure(0):–
        exists ([0,0,0,0,0,0,×,×], mark V1),
        exists ([1,1,1,1,1,1,1,1] bound to V1 on N, mark V2),
        exists ([0,0,0,0,0,0,0,0], mark V5),
        not(structure(1)),
        not(structure(2)).
structure(1):–
        exists ([1,1,1,1,0,0,0,0] bound to V2 on NW, mark V3),
        exists ([1,1,1,1,0,0,0,0] bound to V3 on SW, mark V4),
        not(structure(3)).
structure(3):–
        exists ([0,0,0,0,1,1,1,1] bound to V4 on S).
structure(2):–
        exists ([1,0,1,0,1,0,1,0]).
action():–
        bind (V2 to V5 on SW).
```

Możliwe jest również oznaczenie cząsteczki spełniającej określone kryteria jedną z 15 etykiet, od V1 do V15 (np. exists(... mark V1)). Przykładowo – polecenie exists([1,1,1,1,0,0,0,0] bound to V2 on NW, mark V3) oznacza: odszukaj cząsteczkę o typie 11110000, związaną na kierunku NW z cząsteczką identyfikowaną przez zmienną V2 i zapamiętaj wynik w zmiennej V3 (zmienna V3 będzie od tego momentu identyfikowała znalezioną przez polecenie cząsteczkę).

Polecenia zgrupowane w predykacie action wykonują sekwencje poleceń umożliwiających zmianę istniejących wiązań oraz położeń cząsteczek. Ze względu na modyfikację właściwości cząsteczek, istotna jest kolejność w jakiej występują poszczególne predykaty, odwrotnie niż w części rozpoznającej cząsteczki. W przykładowym programie polecenie bind(V2 to V5 on SW) oznacza: połącz cząsteczkę identyfikowaną przez zmienną V2 z cząsteczką identyfikowaną przez zmienną V5 na kierunku SW.

W odróżnieniu od Prologu, składnia języka nie determinuje jakie informacje przekazywane są do poszczególnych predykatów. Przyjęta została następująca interpretacja przekazywania parametrów: do predykatów search i action oraz głównego predykatu structure (structure(0) w programie przykładowym) przekazywana jest pełna lista zmiennych, natomiast do pozostałych predykatów structure przekazywane są tylko te zmienne które zostały użyte w predykacie nadrzędnym (tj. zostały wyszczególnione w części mark predykatu exists).

Główną ideą wprowadzonego języka funkcji kompleksów było uzyskanie własności

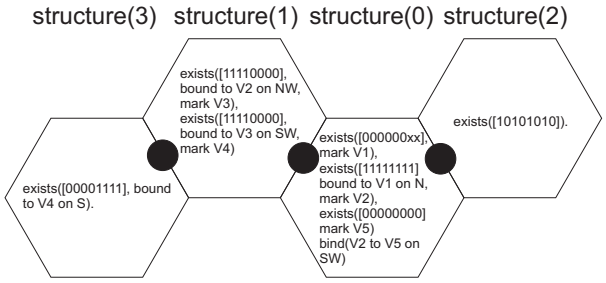
aby niewielkie zmiany w kodzie programu (tzn. w kompleksie cząsteczek kodujących program) na ogół prowadziły do małych zmian w algorytmach części rozpoznającej i wykonawczej programów. Taka własność języka jest podstawowa w przypadku modelowania spontanicznego powstawania złożonych struktur i ich dalszej ewolucji. Brak takich własności był przeszkodą takich symulacji w systemie [**?**, **?**] – patrz [**?**].

W opisanym języku małe zmiany w strukturze kompleksów cząsteczek prowadzą do usunięcia lub zmiany pewnych predykatów, co zazwyczaj prowadzi do zmian zdolności rozpoznawczych programu, np. do redukcji jego precyzji.

8.4.3. Kodowanie

Kompleks cząsteczek może kodować program. Każdy predykat structure jest reprezentowany przez pojedynczy stos cząsteczek w którym zakodowana jest lista predykatów exists. Stos kodujący predykat structure(0) także koduje predykaty akcji. Stosy związane z tym stosem kodują negację predykatów structure. Program przedstawiony w tablicy 8.1 jest reprezentowany przez strukturę cząsteczek pokazaną na rys. 8.19. Zauważmy, że predykaty not(structure) są kodowane przez stosy cząsteczek przylegających do stosu kodującego predykat structure(0).

Stosy cząsteczek kodujących predykaty są etykietowane specyficznymi typami cząsteczek, więc nie każdy stos zawiera w sobie funkcję.



Rys. 8.19: Program z tablicy 8.1 zakodowany w kompleksie cząsteczek.

8.4.4. Interpreter

Programy zakodowane w strukturach cząsteczek są wykonywane przez specjalizowany uproszczony interpreter Prologu. Bezpośrednio przed wykonaniem interpreter tworzy listę faktów o cząsteczkach widzianych przez program.

Jeżeli cześć rozpoznająca strukturę i część wykonawcza programu powiodły się oraz bilans energii przemian jest dodatni interpreter zmienia odpowiednio stan środowiska i wykonanie się kończy. W przeciwnym wypadku wszystkie kierunki poziome zostają przesunięte o kąt 45^0 i program wykonany zostaje ponownie, aż do zakończenia się sukcesem lub wyczerpania wszystkich kierunków.

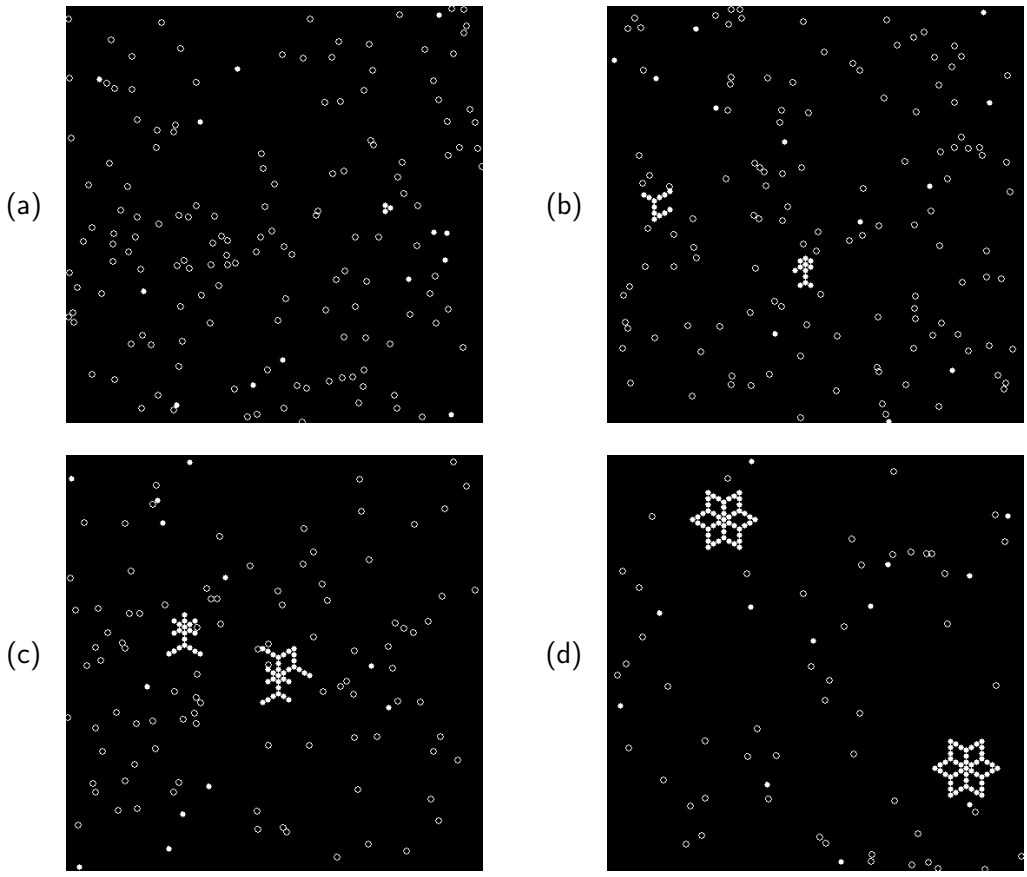
8.4.5. Etapy symulacji

Symulacja środowiska jest realizowana w epokach. Każda epoka składa się z trzech faz. W pierwszej fazie realizowany jest ruch i zderzenia cząsteczek. W fazie dru-

giej ruch i zderzenia fotonów z cząsteczkami, zaś w fazie trzeciej wykonywane są programy kompleksów. Kompleksy wybierane są w losowej kolejności, a w czasie wykonania programu jednego kompleksu przemiany pozostałej części systemu są zatrzymane – tylko jedna funkcja może być realizowana w danej chwili czasu.

8.4.6. Przykładowa symulacja

Ilustracją potencjalnych możliwości środowiska może być prosty system, w którym zbiór kilku typów programów umieszczonych wśród losowo rozłożonych cząsteczek dwóch typów może skonstruować regularną strukturę, „śnieżynkę" [28]. Programy działały po kolei – stan powstającej struktury pozostawionej przez jeden program był rozpoznawany i rozbudowywany przez następny program, itd. Ewolucję systemu w wybranych epokach symulacji przedstawiono na rys. 8.20. Bardziej szczegółowe opisy podobnych symulacji można znaleźć na stronie [67].



Rys. 8.20: Proces tworzenia się „śnieżynek" po 10, 270, 1670 i 4500 krokach symulacji [28].

8.4.7. Przewidywane symulacje

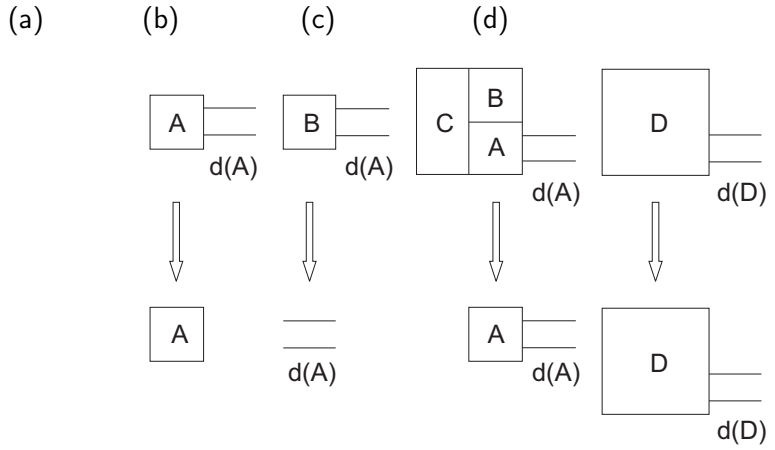
Omówione środowisko może modelować różnorodne procesy samoorganizacji i samomodyfikacji bez potrzeby jawnego określania jakości powstających struktur, jak

to ma miejsce przy modelowaniu procesów rozwoju korzystając z algorytmów genetycznych. Jakość jest bezpośrednio weryfikowana przez reguły działania środowiska, jego „fizykę".

Przewidywane jest użycie środowiska w dwóch kierunkach. Po pierwsze w modelowaniu systemów samoreprodukujących się. Różne strategie reprodukcji mogą być porównywane – ich dynamika i rywalizacja i ewolucja. Po drugie, przewidywane jest modelowanie ewolucji systemu rozpoczynając od przypadkowo rozmieszczonych cząsteczek i oczekując na powstające struktury i wyłaniające się ich funkcje.

Dalszy rozwój środowiska będzie się odbywał w kierunku zwiększania złożoności oddziaływań elementarnych cząstek systemu, co powinno ułatwić samoistne powstawanie bardziej złożonych struktur, przy jednoczesnym utrzymywaniu zasady braku zewnetrznej ingerencji w środowisko.

8.5. Samoreprodukcja – von Neumann

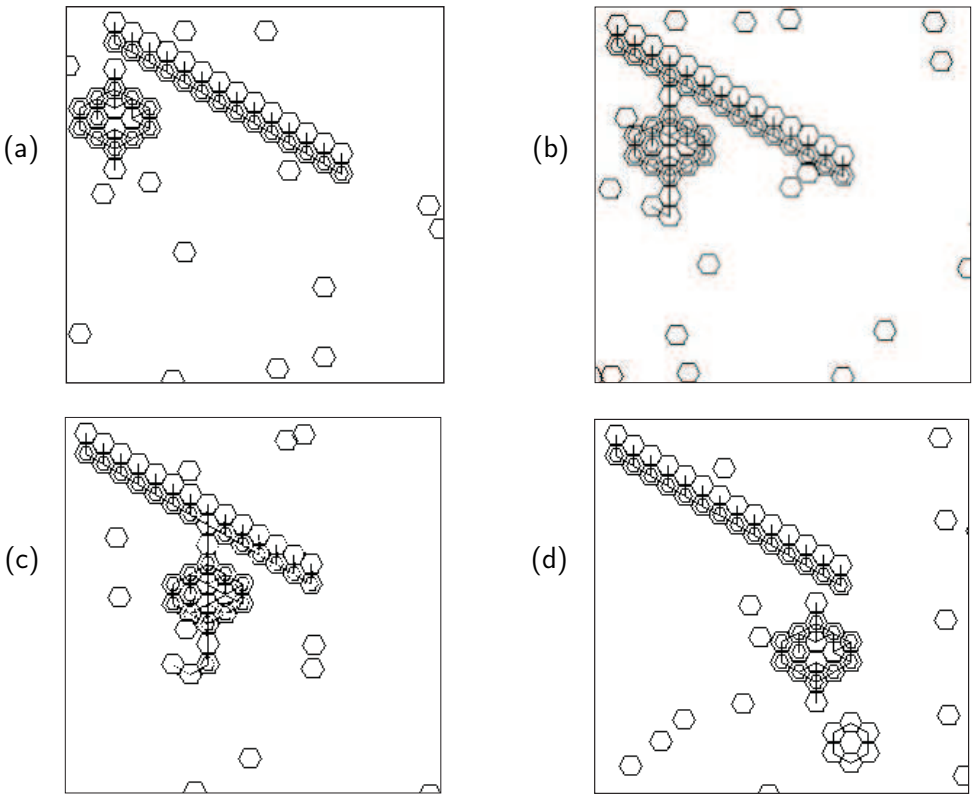


Rys. 8.21: Etapy konstrukcji maszyny samoreprodukującej się

Rozważając ogólnie proces samoreprodukcji von Neumann pokazał [49], że istnienie maszyny samoreprodukującej się wynika logicznie z przyjętego istnienia maszyny zwanej uniwersalnym konstruktorem. Uniwersalny konstruktor, oznaczony przez A, potrafi na podstawie opisu $d(X)$ maszyny X skonstruować maszynę X. Jeżeli podstawimy $X = A$, to uniwersalny konstruktor A na podstawie opisu $d(A)$ skonstruuje A (patrz rys. a). Ale nie jest to jeszcze pełny proces samoreprodukcji, ponieważ skonstruowana maszyna A nie zawiera $d(A)$. Wprowadźmy więc maszynę B, która kopiuje opis $d(A)$ (rys. b). Łącząc maszynę A z maszyną B i dołączając maszynę C, która steruje kolejnością operacji A i B, otrzymujemy kompleks maszyn $A + B + C$ nazwany D, który na podstawie opisu $d(D)$ wykonuje swoją kopię łącznie z kopią swojego opisu (rys. c). Ostatecznie oznaczając $D + d(D)$ jako E, możemy stwierdzić, że E jest maszyną samoreprodukującą się (rys. d). W ten sposób pokazano, że możliwy jest proces samoreprodukcji, gdy maszyna samoreprodukująca się posiada wyróżnioną część stanowiącą jej opis.

W ramach sprawdzenia działania uniwersalnego konstruktora w środowisku DigiHive, zakodowano łańcuch będący instrukcją złożenia zamkniętego pierścienia. Wy-

niki symulacji przedstawiono na rysunku 8.22. W rezultacie, po 24 cyklach, powstała założona struktura.



Rys. 8.22: Konstruktor z łańcuchem informacyjnym po 0 (a), 5 (b), 10 (c) oraz 90 (d) cyklach symulacji.

Rozdział 9

Języki programowania sztucznej inteligencji

9.1. Prolog

Powstały w 1972 roku język PROLOG (PROgramming in LOGic) (A.Colmerauer i P.Roussel – Uniwersytet Marsylski), będący rozwinięciem idei teoretycznych R. Kowalskiego z Uniwersytetu w Edynburgu dotyczących programowania logicznego, jest uniwersalnym językiem programowania bardzo wysokiego poziomu opartym o rachunek predykatow i metody dowodzenia twierdzeń logicznych. Język ten znajduje zastosowanie w rożnorodnych dziedzinach sztucznej inteligencji i robotyki, a w szczegolności w problemach przetwarzania językow naturalnych, w planowaniu akcji, w systemach ekspertowych i w relacyjnych bazach danych, w systemach reprezentacji wiedzy oraz we wszelkich zadaniach przetwarzania symbolicznego. Stosowany w tych dziedzinach umożliwia, w porównaniu z innymi językami programowania, bardzo łatwe, szybkie i wygodne przygotowanie programów. Język PROLOG obok Lispu jest podstawowym

językiem stosowanym w sztucznej inteligencji (Ligęza i inni 1991, Jędruch 1989, LPA 1990).

9.1.1. Struktura danych i programów

Podstawowymi obiektami języka są termy. Istnieje siedem typów termów: liczby całkowite, liczby rzeczywiste, atomy, nazwy zmiennych, termy złożone, listy i łańcuchy.

Atomy. Są używane do nazywania danych, predykatów, programów, plików, itp. Wyróżnia się następujące rodzaje atomów:

- **Atomy alfanumeryczne**. Ciągi liter, cyfr i podkreśleń zaczynającym się od małej litery. Przykłady symboli alfanumerycznych: `monika`, `pije_kakao`.
- **Cytaty**. Ciągi dowolnych znaków otoczone pojedynczymi znakami apostrofów. Przykłady cytatów: `'Cyprian Norwid, Sen'`, `'300 000 mil'`.

Liczby całkowite. Np. `123`, `-1200`.

Liczby rzeczywiste. Np. `1.0`, `-128.3`, `1.7e-12`.

Nazwy zmiennych. Ciągi liter, cyfr i podkreśleń zaczynające się od dużej litery. Np.: `Autor`, `W`. Wyróżnia się też pojedynczy znak podkreślenia „_" nazywany zmienną anonimową.

Termy złożone. Ogólną postacią termu zlożonego jest

$$\mathtt{f}(\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_n)$$

gdzie `f` nazywa się *funktorem*. Funktor jest atomem lub nazwą zmiennej.

Listy. Ciągi o postaci:

$$[\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_n]$$

gdzie $\mathtt{t}_i$, $\mathtt{i} = 1, 2, \ldots \mathtt{n}$ są termami.

Przykłady list: `[dom,as,sok]`, `[2,4,6,3,b,c]`, `[[a,b],[c,f],[g,h,s,w]]`, `[]`, gdzie `[]` oznacza listę pustą.

W liście wyróżniony jest pierwszy jej element, zwany głową i lista, która pozostaje po usunięciu pierwszego elementu, zwana ogonem. Np. głową listy `[1,4,5]` jest `1`, zaś ogonem `[4,5]`; głową listy `[a]` jest `a`, zaś ogonem lista pusta `[]`. W Prologu listę można także zapisać w postaci: `[G|O]`, gdzie `G` jest glową listy zaś `O` jest ogonem. Np. przyrównując listę `[X|Y]` do listy `[a,b,c]`, zmiennej `X` odpowiada element `a`, zaś zmiennej `Y` lista `[b,c]`.

Łańcuchy. Ciągi znaków otoczone podwójnymi apostrofami. Np. `"Rok 1812"`.

Uwzgledniajac powyższe przykładami termów są wyrażenia: `dom`, `[G|w,x,y,z]`, `student(S,politechnika,gdansk)`, `gniazdo(gniazdo(gniazdo(X)))`.

Liczby całkowite, liczby rzeczywiste, atomy i łańcuchy nazywane są stałymi.

Programy w Prologu tworzone są z *formuł atomowych* zdefiniowanych następująco:

Definicja 9.1 *Jeżeli* $\mathtt{p}$ *jest nazwą n-argumentowego predykatu oraz* $\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_n$ *są termami, to wyrażenie* $\mathtt{p}(\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_n)$ *jest formułą atomową. Żadne inne wyrażenie nie jest formułą atomową.*

W Prologu za pomocą klauzul Horna zapisuje się program, który jest zbiorem faktów i reguł dotyczących obiektów jakiejś dziedziny. Klauzule w Prologu mają postać:

$$\mathtt{p}. \tag{9.1}$$

lub

$$\mathtt{p} :\!- \ \mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n. \tag{9.2}$$

gdzie $\mathtt{p}, \mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n$ są formułami atomowymi. Wyrażenie (9.1) nazywa się faktem, zaś wyrażenie (9.2) regułą.

Znaki „`:-`" i przecinek „`,`" reprezentują odpowiednio spójniki `jeżeli` (`if`) i `i` (`and`). Nagłówkiem klauzuli nazywa się formuła atomowa stojąca w niej na pierwszym miejscu. Klauzule posiadające w nagłówku tą samą nazwę predykatu noszą nazwę procedury i muszą być zgrupowane razem.

Fakty i reguły są poprzedzone kwantyfikatorami ogólnymi, czego jednak jawnie się nie zapisuje.

Prolog dopuszcza stosowanie w regułach spójnika `lub` (`or`) (suma logiczna), zapisywanego za pomocą średnika „`;`”. Użycie spójnika `or` pozwala zastąpić kilka reguł o identycznych nagłówkach jedną regułą, czyli reguły

$$\begin{array}{lll} \mathtt{p} & :\!- & \mathtt{p_1}, \mathtt{p_2}, \ldots, \mathtt{p_n}. \\ \vdots & & \vdots \\ \mathtt{p} & :\!- & \mathtt{r_1}, \mathtt{r_2}, \ldots, \mathtt{r_m}. \end{array}$$

można zastąpić regułą

$$\mathtt{p} \quad :\!- \quad \mathtt{p_1}, \mathtt{p_2}, \ldots, \mathtt{p_n}; \ldots; \mathtt{r_1}, \mathtt{r_2}, \ldots, \mathtt{r_m}.$$

Komentarze w Prologu zawiera się pomiędzy parami znaków „`/*`’’ i „`*/`” zaś znak „`%`” oznacza, że jakikolwiek tekst pomiedzy nim a końcem linii, w której znak „`%`” się znajduje, jest ignorowany.

9.1.2. Przykład programu

Rozpatrzmy następujący przykładowy program

```
/* Program 1 */

posiada(jan,auto).
posiada(ewa,radio).
posiada(tomasz,wideo).
posiada(monika,auto).
posiada(jan,radio).

ma_dostep(monika,wideo).
ma_dostep(monika,radio).
ma_dostep(monika,ksiazki).

lubi(ewa,radio).
lubi(monika,ksiazki).
lubi(tomasz,wideo).
lubi(ewa,wideo).
lubi(_,auto).

korzysta(ewa,plyty).
korzysta(X,Y) :- posiada(X,Y), lubi(X,Y).
korzysta(X,Y) :- ma_dostep(X,Y), lubi(X,Y).
```

Wyrażenia występujące w programie są klauzulami Horna. Występują one w dwóch postaciach przedstawionych wzorami (9.1) i (9.2). Postać pierwsza – fakt – zawiera bezpośrednią informację o własności jakiegoś obiektu lub o relacji między

obiektami. Postać druga – reguła – zawiera informacje o zależnościach pomiędzy faktami. Klauzule kończą się kropką.

Prolog nie dopuszcza występowania predykatów zanegowanych, ale istnieje predykat standardowy `not`, którego argumentem są predykaty i który spełnia podobną rolę jak operacja negacji.

Argumentami predykatów w faktach i regułach, a także w pytaniach stawianych programowi są stałe oznaczające konkretne obiekty (np. w Programie 1 stałymi są `monika`, `auto`, `wideo`), zmienne, oznaczające nieznane obiekty, które w trakcie wykonania programu wiążą sie z konkretnymi obiektami (np. w Programie 1 zmiennymi są X,Y) oraz zmienne anonimowe, oznaczające dowolny obiekt (np. w Programie 1 w predykacie `lubi(_,auto)`).

Zapisany zbiór klauzul nabiera znaczenia po nadaniu mu interpretacji. Na przykład występujące w Programie 1 klauzule można by następująco interpretować w języku polskim:

> jan posiada auto,
> ewa posiada radio, ...
>
> monika ma dostęp do wideo, ...
>
> każdy lubi auta,
>
> X korzysta z obiektu Y jeżeli X posiada ten obiekt i go lubi,
> X korzysta z obiektu Y jeżeli X ma dostęp do tego obiektu i go lubi.

9.1.3. Zapytania

Wykonanie programu w Prologu polega na odpowiedzi, czy z zawartych w programie faktów i reguł wynika pozytywna odpowiedź na postawione przez użytkownika zapytanie, oraz na podstawieniu za zmienne zawarte w zapytaniu konkretnych obiektów, dla których ta odpowiedź jest prawdziwa. Zapytanie (kwerenda) ma postać

$$\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n.$$

gdzie $\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n$ są formułami atomowymi.

Całe zapytanie jest poprzedzone kwantyfikatorem szczegółowym, który jednak nie jest zapisywany jawnie.

Z punktu widzenia rachunku predykatów zapytanie ma postać kwantyfikowanego egzystencjalnie wyrażenia, będącego iloczynem logicznym predykatów, zaś wykonanie programu polega na odpowiedzi czy zapytanie wynika logicznie ze zbioru faktów i reguł programu, oraz na związaniu zmiennych zawartych w zapytaniu z konkretnymi obiektami, dla których to wynikanie zachodzi.

Po podaniu zapytania program się wykonuje, i po zakończeniu wykonania oczekuje na nowe pytanie.

Przykład 9.1

Wykonajmy program 1 podając następujące zapytania:

1. `lubi(ewa,wideo).`
   na co program odpowiada `yes` (tak).

2. `ma_dostep(tomasz,radio).`
   na co program odpowiada `no` (nie).
3. `korzysta(ewa,radio).`
   odpowiedź: `yes`. Zauważmy, że przygotowując tę odpowiedź program nie mógł wprost skorzystać z faktu `korzysta(ewa,radio)` bo takiego w programie nie ma i musiał skorzystać z reguły wiążącej korzystanie z posiadaniem i lubieniem.
4. `korzysta(Osoba,wideo).`
   odpowiedź: `Osoba=tomasz` Zauważmy, że używając zmiennej `Osoba` sformułowano zadanie znalezienia wszystkich osób korzystających z wideo.
5. `korzysta(monika,Przedmiot).`
   odpowiedź: `Przedmiot=auto`, `Przedmiot=ksiazki`. Forma pytania podobna do poprzedniej, jednak tym razem Prolog znajduje 2 rozwiązania.
6. `korzysta(monika,X), korzysta(jan,X).`
   odpowiedź: `X=auto`. Jest to przykład pytania złożonego – zapytano, czy z zawartych w programie klauzul wynika, że istnieje jakiś przedmiot z którego korzystają `monika` i `jan`.
7. `korzysta(O,P).`
   odpowiedź:

   ```
   O=ewa P=plyty, O=jan P=auto, O=ewa P=radio,
   O=tomasz P=wideo, O=monika P=auto, O=monika P=ksiazki
   ```

   Użycie w pytaniu dwóch zmiennych zmusiło program do wyznaczenia wszystkich par obiektów spełniających relację korzysta.

■

9.1.4. Przykłady uzycia termów złozonych

Przykład 9.2

Przypuśćmy, że chcemy zawrzeć w programie spis przedmiotów znajdujących się w magazynie wraz z pewnymi danymi je charakteryzującymi, ale dane te (ich liczba i charakter) zależą od przedmiotu. Przypuśćmy, że w magazynie znajdują się książki opisywane przez podanie autora i tytułu, płyty dzielące się na płyty z muzyką klasyczną i rozrywkową przy czym płyty z muzyką klasyczną opisywane są przez nazwisko kompozytora, tytuł utworu i nazwisko solisty lub dyrygenta, zaś plyty rozrywkowe przez zespół lub solistę i tytuł płyty. Wreszcie w magazynie są piłki jednego tylko rodzaju. Ponadto, każdy przedmiot charakteryzowany jest liczbą egzemplarzy i ceną. Wszystkie te dane chcielibyśmy umieszczać w programie korzystając tylko z trzyargumentowego predykatu magazyn(przedmiot,cena,ilosc).

Rozpatrzmy spełniający te wymagania kompletny program:

```
/* Program 2 */

magazyn(ksiazka('Sienkiewicz H.','Potop'),60,17).
magazyn(ksiazka('Blixen K','Pozegnanie z Afryka'),41,30).
```

```
magazyn(plyta(klasyczna('Bruckner A.','Symfonia nr 9',
        'Jochum E.')),40,40).
magazyn(plyta(rozrywkowa('Cat Stevens',
        'Tea for the tillerman')),45,5).
magazyn(pilka,8,100).
```

Wykonajując Program 2 z następującymi pytaniami:

```
magazyn(plyta(klasyczna('Bruckner A.','Symfonia nr 9',
      'Jochum E.')),X,Y).
magazyn(plyta(X),Y,Z).          magazyn(X,Y,Z).
magazyn(X,Y,Z), Z<=200, Z>=30.
```

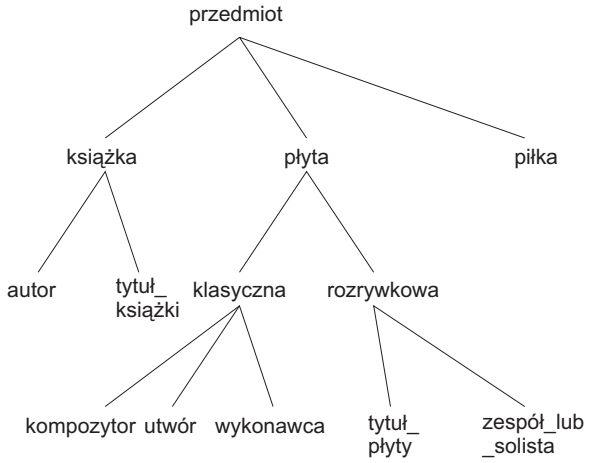
otrzymujemy odpowiednio

```
X=40, Y=40

X=klasyczna('Bruckner A.','Symfonia nr 9','Jochum E.'),
      Y=40, Z=40
X=rozrywkowa('Cat Stevens','Tea for the tillerman'),
      Y=45, Z=5

X=ksiazka('Sienkiewicz H.','Potop'), Y=60, Z=17
X=ksiazka('Blixen K','Pozegnanie z Afryka'),
      Y=41, Z=30
X=plyta(klasyczna('Bruckner A.','Symfonia nr 9','Jochum E.')),
      Y=40, Z=40
X=plyta(rozrywkowa('Cat Stevens','Tea for the tillerman')),
      Y=45, Z=5)
X=pilka, Y=8, Z=100
```

Strukturę obiektu złożonego występującego w programie 2 ilustruje drzewo przedstawione na rys. 9.1.



Rys. 9.1: Struktura obiektu złożonego występującego w Programie 2

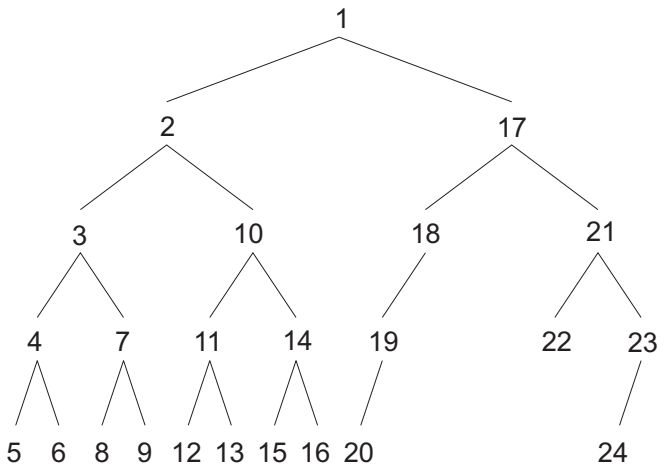
Szczególnym przypadkiem termów złożonych są tzw. *termy rekurencyjne*, czyli termy w których argumentami funktorów są te same funktory. Mogą one służyć do przedstawiania list i drzew.

Przykład 9.3

W Programie 3 zdefiniowano drzewo przedstawione na rys. 9.2.

```
                    /* Program 3 */

drzewo(g(1,g(2,g(3,g(4,g(5,n,n),g(6,n,n)),
                   g(7,g(8,n,n),g(9,n,n))),
               g(10,g(11,g(12,n,n),g(13,n,n)),
                    g(14,g(15,n,n),g(16,n,n)))),
          g(17,g(18,g(19,g(20,n,n),n),
                    n),
               g(21,g(22,n,n),
                    g(23,g(24,n,n),
                    n))))).
```



Rys. 9.2: Drzewo z Programu 3

9.1.5. Wykonanie programu

W rozdziale tym zajmiemy się przebiegiem realizacji programu w Prologu. Wykonanie programu polega na sprawdzeniu, czy zadane pytanie, wyrażone za pomocą predykatu lub ciągu predykatów, wynika logicznie z zawartych w programie faktów i reguł. Znajomość algorytmu tego sprawdzania jest potrzebna dla właściwego przygotowania programu, pomimo że programując w Prologu znacznie mniej trzeba myśleć o sposobie wykonywania programu niż w innych językach. Uważa się czasami, że Prolog jest jednym z pierwszych języków programowania, zawierających zalążki metody programowania polegającej na tym, że podaje się fakty i reguły o jakiejś dziedzinie i precyzuje cele nie zastanawiając nad sposobem wykonania. Podaje się maszynie, „co" ma zrobić nie zastanawiając sie, „jak" ma to zrobić.

Wykonanie programu – unifikacja

W czasie wykonania programu w Prologu następuje wielokrotne próby unifikacji predykatów zawartych w pytaniu lub predykatów występujących po prawych stronach klauzul z predykatami w nagłówkach klauzul. Ten proces sprawdzania zgodności jest podstawową operacją przeprowadzaną w czasie realizacji programu. W trakcie unifikacji zmienne mogą zostać przyporządkowane do konkretnych obiektów i wtedy nazywane są zmiennymi ukonkretnionymi. Zmienne przed przyporządkowaniem nazywane są swobodnymi.

Przykład 9.4

Przeprowadźmy unifikację predykatów:

```
test(dom,X,f(g(deska)))  i  test(Z,f(g(las),f(U))).
```

Łatwo widać, że po podstawieniach: `dom/Z`, `f(g(las))/X` i `g(deska)/U` argumenty obydwu predykatów stają się identyczne i otrzymujemy: `test(dom,f(g(las)),f(g(deska)))`. Zauważmy też, że występujące w predykatach zmienne wolne zostały w procesie unifikacji związane.

Natomiast nie udaje się zunifikować predykatów:

`test(dom,X,f(g(deska)))` i `test(dom,Y,h(g(deska)))`

ponieważ nazwy funktorów w trzecim argumencie są różne.

Unifikacja predykatów zawierających listy np.

`test([G|O])` i `test([[1,3,4],[2,4],[33,66],[123]])`

następuje dla `G=[1,3,4]` i `O=[[2,4],[33,66],[123]]`. ■

Przykład 9.5

Rozpatrzmy Program 5 dołączający element (pierwszy argument predykatu `dolacz_element`) do listy (drugi argument) tworząc nową listę (trzeci argument).

```
/* Program 4 */

dolacz_element(X,Y,[X|Y]).
```

Wykonajmy powyższy program podając zapytanie

```
dolacz_element(5,[1,2,3],L)
```

Unifikuje się ono z predykatem `dolacz_element(X,Y,[X|Y])` dla podstawień `X=5`, `Y=[1,2,3]` oraz `L=[X|Y]=[5,1,2,3]` i takie rozwiązanie dla `L` otrzymujemy. Podajmy teraz zapytanie: `dolacz_element(X,Y,[11,33,55])`. Otrzymujemy: `X=11`, `Y=[33,55]`, co oczywiście znów wynika z unifikacji zadanego zapytania z klauzulą programu. Ten ostatni przykład wywołania ilustruje częste zjawisko występujące w Prologu, a mianowicie, że zakres działania otrzymanego programu wykracza poza wymagania stawiane przy jego tworzeniu (celem programu było dołączanie elementu do listy, zaś otrzymany program może także rozdzielić listę na pierwszy jej element i pozostałą listę). ■

Wykonanie programu – nawracanie

Omówienie przebiegu wykonania programu stanie się proste, jeżeli wprowadzimy pojęcie spełniania predykatu.

Pojedynczy predykat nazywamy spełnionym, jeżeli:

(a) unifikuje się z jakimkolwiek faktem lub

(b) unifikuje się z nagłówkiem jakiejkolwiek reguły i spełniony jest ciąg predykatów po prawej stronie nagłówka dla zmiennych ukonkretnionych podczas unifikacji nagłówka.

Ciąg predykatów nazywamy spełnionym, jeżeli:

(a) spełniony jest każdy predykat w ciągu i

(b) sprawdzenie spełnienia poszczególnych predykatów przebiegało w kolejności ich występowania w ciągu, a zmienne ukonkretniane w trakcie sprawdzania przekazywane były pozostałym, niesprawdzonym jeszcze predykatom.

Predykat nazywamy niespełnionym, jeżeli nie unifikuje się z żadnym faktem programu i nie unifikuje się z żadnym nagłówkiem reguły lub unifikuje się z nagłówkiem reguły i nie spełniony jest którykolwiek z predykatów jej prawej strony.

Korzystając z wprowadzonych definicji można powiedzieć, że wykonanie programu w Prologu polega na sprawdzeniu spełnienia ciągu predykatów występujących w zapytaniu i ponadto na podaniu w odpowiedzi wszystkich wartości zmiennych, które powodują takie spełnienie.

W przypadku niespełnienia któregokolwiek z predykatów program cofa się do predykatu poprzednio sprawdzanego, przywraca wszystkim zmiennym stan, jaki miały przed sprawdzeniem spełnienia tego predykatu i stara się wykazać na nowo jego spełnienie poprzez unifikację z dalszymi klauzulami programu (uzyskując w ten sposób inne ukonkretnienia zmiennych mogące spowodować, tym razem, spełnienie następnego predykatu). Jeżeli predykat nie daje się na nowo zunifikować, to następuje nowe cofnięcie się i postępowanie się powtarza. Jeżeli poprzez kolejne sprawdzania, spełnienie zapytania zostanie ostatecznie wykazane, to Prolog drukuje odpowiedź `yes`, (jeżeli argumentami zapytania były wyłącznie obiekty), lub podaje wartość zmiennych (gdy argumentami zapytania były zmienne). Jeśli jednak w procesie cofania się program dojdzie do pierwszego predykatu zapytania i wszystkie możliwości jego unifikacji się wyczerpią bez wykazania spełnienia zapytania, to proces poszukiwania rozwiązania kończy się z wynikiem negatywnym i system wysyła odpowiedź `no`. Omówiony mechanizm cofania się nosi nazwę nawracania lub nawrotu (ang. backtracking) i jest podstawowym mechanizmem Prologu. Po wykazaniu spełnienia danego zapytania Prolog nie poprzestaje na tym, lecz kontynuując proces nawracania, znajduje wszystkie rozwiązania (tak jak gdyby spełnienie nie zostało wykazane).

Przykład 9.6

Rozpatrzmy działanie programu 5 dla zapytania:

```
p(X), p(Y), p(Z), X<>Y, X<>Z, Y<>Z
```

```
/* Program 5 */
```

```
p(a). p(b). p(c). p(d).
```

Działanie programu rozpoczyna się od sprawdzenia spełnienia pierwszego predykatu zapytania, czyli od predykatu `p(X)`. Unifikuje się on z pierwszą klauzula programu, przy czym zmienna `X` związana zostaje z obiektem `a`, czyli predykat `p(X)` jest spełniony dla `X=a`. Następuje sprawdzenie spełnienia predykatu `p(Y)`. Unifikuje się on z pierwszą klauzulą i jest spełniony dla `Y=a`. Trzeci predykat zapytania też unifikuje się z pierwszą klauzulą programu dla `Z=a`. Czwarty predykat `X<>Y` jest predykatem systemowym i jest spełniony, gdy `X` i `Y` są związane z różnymi obiektami. Ponieważ `X` i `Y` zostały związane z tym samym obiektem, predykat ten nie jest spełniony. Następuje więc nawrót do predykatu `p(Z)`, uwolnienie zmiennej `Z` i próba jego unifikacji z dalszymi klauzulami. Unifikuje się on z klauzulą `p(b)` dla `Z=b`. Następuje ponowne przejście do predykatu `X<>Y`, który oczywiście dalej nie jest spełniony.

Kolejne nawroty i unifikacje `p(Z)` z pozostałymi klauzulami trwają do wyczerpania się wszystkich klauzul. Jeżeli nie ma już innej możliwości unifikacji `p(Z)`, nastepuje nawrócenie do predykatu `p(Y)` i ponowna jego unifikacja. Unifikuje się on z klauzulą `p(b)` dla `Y=b`. Następuje teraz przejście do `p(Z)`, jego unifikacja z `p(a)`, przejście do predykatu `X<>Y`, który tym razem jest spełniony i przejście do predykatu `X<>Z`. Predykat `X<>Z` jest niespełniony, ponieważ zarówno `X` jak i `Z` są związane z obiektem `a`. Następuje nawrócenie bezpośrednio do `p(Z)` (z ominięciem `X<>Y`), ponieważ predykaty systemowe są omijane przy nawracaniu. Predykat `p(Z)` unifikuje się teraz z `p(b)`. Ponieważ ta unifikacja powoduje niespełnienie `Y<>Z`, następuje kolejny nawrót do `p(Z)` i jego unifikacja z `p(c)`. Związanie `Z` z `c` powoduje, że wszystkie predykaty nierówności są już spełnione i w ten sposób spełnione jest całe zapytanie i program wyświetla rozwiązanie: `X=a`, `Y=b`, `Z=c`. Program nie poprzestaje na jednym rozwiązaniu i stosując mechanizm nawracania kontynuuje proces poszukiwania nowych spełnień zapytania i tym samym generacji nowych rozwiązań. W omawianym przykładzie następuje teraz nawrócenie do predykatu `p(Z)`, unifikacja z `p(d)`, sprawdzenie spełnienia nierówności i wyświetlenie nowego rozwiązania: `X=a`, `Y=b`, `Z=d`. Dalej już łatwo widać, że program ten uzyska 24 rozwiązania, czyli wszystkie wariacje bez powtórzeń 3 elementów spośród 4. ■

Rozpatrzmy teraz, na innym przykładzie, proces poszukiwaia rozwiązania, gdy program zawiera zarówno fakty jak i reguły.

Przykład 9.7

Rozpatrzmy proces poszukiwania rozwiązania w programie 1 dla zapytania

```
korzysta(monika,Rzecz)
```

Pierwsza unifikacja predykatu `korzysta(monika,Rzecz)` następuje z nagłówkiem klauzuli:

```
korzysta(X,Y) :- posiada (X,Y), lubi(X,Y).
```

W trakcie unifikacji zmienna `X` zostaje związana z obiektem `monika`, zaś zmienna `Rzecz` pozostaje wolną. Następuje teraz sprawdzenie spełnienia prawej strony reguły, które rozpoczyna się od sprawdzenia spełnienia predykatu `posiada(monika,Rzecz)`. Predykat ten unifikuje się z faktem `posiada(monika,auto)` dla `Rzecz=auto`, jest

więc spełniony. Sprawdzeniu podlega teraz predykat `lubi(monika,auto)`. Unifikuje się on z faktem `lubi(_,auto)`, jest więc także spełniony. W ten sposób wykazano, że kwerenda `korzysta(monika,Rzecz)` jest spełniona dla `Rzecz=auto`. Program wyświetla to rozwiązamnie i kontynuuje proces poszukiwania nowych rozwiązań.

Ponieważ predykat `lubi(monika,auto)` nie unifikuje się żadnym innym nagłówkiem, więc następuje nawrót do poprzedniego predykatu (`posiada(monika,Rzecz)`), z oswobodzeniem zmiennej `Rzecz`. Ponieważ nie unifikuje się on z nagłówkiem żadnej klauzuli, następuje kolejne nawrócenie tym razem do `korzysta(monika,Rzecz)`. Kolejna unifikacja predykatu `korzysta(monika,Rzecz)` następuje więc z nagłówkiem klauzuli:

```
korzysta(X,Y) :- ma_dostep(X,Y), lubi(X,Y).,
```

przy czym `X` zostaje związane z `monika`, zaś zmienna `Rzecz` pozostaje swobodną. Dalej, `ma_dostep(monika,Rzecz)` unifikuje się z `ma_dostep(monika,wideo)` i następuje sprawdzanie spełnienia predykatu `lubi(monika,wideo)`. Predykat ten nie daje się zunifikować z nagłówkiem żadnej klauzuli, więc następuje nawrót do predykatu `ma_dostep(monika,Rzecz)`, który teraz unifikuje się z `ma_dostep(monika,radio)`, ale ponieważ `lubi(monika,radio)`, jest niespełniony, więc następuje ponowny nawrót do predykatu `ma_dostep(monika,Rzecz)`, który tym razem unifikuje się z `ma_dostep(monika,ksiazki)`. Ponieważ `lubi(monika,ksiazki)` jest spełniony, to tym samym oba predykaty prawej strony klauzuli są spełnione, więc spełniony jest także nagłowek `korzysta(X,Y)` dla `X=monika` i `Y=ksiazka` i wobec tego spełnione jest zapytanie `korzysta(monika,Rzecz)` dla `Rzecz=ksiazka`. Program wyświetla na ekranie to nowe rozwiązanie i ponieważ wszystkie możliwości nawracania zostały wyczerpane wykonanie programu zostaje zakończone. ■

9.1.6. Rekursje

Procedurę, która bezpośrednio lub pośrednio wywołuje sama siebie, nazywamy rekurencyjną. Zastosowanie rekursji często prowadzi do bardziej zrozumiałych i zwięzłych algorytmów, ale z drugiej strony, wykonanie algorytmów rekurencyjnych może trwać dłużej i zajmować więcej pamięci niż algorytmów iteracyjnych.

Prolog dopuszcza rekursję i stanowi ona w nim ważną technikę programowania. Pojawiające się przy jej stosowaniu problemy z przepełnieniem stosu i metody unikania takiego przepełnienia omawiane są w rozdziale 11.

Jako przykład programu zawierającego rekursje rozpatrzymy klasyczny program obliczania silni.

Przykład 9.8

Rozpatrzmy działanie Programu 7 dla zapytania silnia(4,S).

```
                    /* Program 6 */

silnia(1,1).
silnia(N,Roz) :-
          N > 1,
          N1 = N-1,
```

```
silnia(N1,X),
Roz = N*X.
```

W programie dwukrotnie występuje standardowy predykat równości: `N1=N-1` oraz `Res = N*X`. Pierwszy z nich jest spełniony przy równości obu stron wyrażenia (znak - oznacza odejmowanie), przy czym jeżeli zmienna `N1` jest wolna a zmienna `N` związana, to `N1` zostaje związana z liczbą zapewniającą spełnienie predykatu. Analogicznie zdefiniowany jest drugi predykat, gdzie znak `*` oznacza mnożenie. ■

Poniżej przedstawionych jest kilka typowych programów rekurencyjnych operujących na listach.

Przykład 9.9

Rozpatrzmy działanie programu sprawdzającego, czy dany obiekt jest elementem listy.

```
/* Program 7 */

element(Nazwa,[Nazwa|_]).
element(Nazwa,[_|Ogon]) :- element(Nazwa,Ogon).
```

Pierwsza klauzula programu stwierdza, że obiekt jest elementem listy, jeżeli jest jej głową, zaś druga klauzula stwierdza, że obiekt jest elementem listy, jeżeli jest elementem jej ogona. Klauzula druga powoduje rekurencyjne przeglądanie listy, jeżeli niespełniona jest klauzula pierwsza. Obydwie klauzule są niespełnione, jeżeli lista jest pustą.

Wykonajując program dla zapytania `element(1,[4,3,4,2,8,1,5])` otrzymujemy odpowiedź `yes`. Wykonując go z zapytaniem `element(X,[1,2,3,4])` otrzymuejmy 4 rozwiązania: `X=1`, `X=b`, `X=c`, `X=d`. Te ostatnie rozwiązania otrzymujemy dzięki procesowi nawracania, w którym zmienna `X` wiąże się z kolejnymi elementami listy. Taki sposób wywołania stanowi interesujący i często wykorzystywany „efekt uboczny" działania tego programu. ■

Przykład 9.10

Rozpatrzmy program dołączajacy listę (pierwszy argument predykatu `dolacz_liste`) do listy (drugi argument), tworząc nową listę (trzeci argument).

```
/* Program 8 */

dolacz_liste([],L,L).
dolacz_liste([X|L1], Lista2, [X|L3]) :-
                    dolacz_liste(L1,Lista2,L3).
```

Na zapytanie `dolacz_liste([osa,komar],[czerwony,czarny],X)`. program odpowiada `X=[osa,komar,czerwony,czarny]`.

Zauważmy, że klauzulom programu można przypisać interpretację: Połaczenie dwu list, gdy pierwsza lista jest pusta daje listę drugą, zaś gdy lista pierwsza nie jest pusta połączenie list polega na dopisaniu głowy listy pierwszej do połączenia

ogona listy pierwszej z listą drugą Łącznie obie klauzule programu wyrażają definicję operacji dołączania list. ■

Przykład 9.11

Program 9 odwraca kolejność elementów listy. Wywołajmy program zapytaniem `odwr([a,b,c,d],L)`.

```
                        /* Program 9 */

dolk(A,[],[A]).
dolk(A,[G|O],[G|Op]):- dolk(A,O,Op).

odwr([],[]).
odwr([G|O],Lo):- odwr(O,Oo),dolk(G,Oo,Lo).
```

Wywołanie predykatu `odwr` powoduje wywołanie trzy argumentowego predykatu `dolk`, który dopisuje element do końca listy.

Zauważmy, że klauzule procedury `dolk` mozna interpretować następująco: dopisanie elememtu na końcu listy pustej tworzy listę jednoelentową zawierającą ten element. Gdy lista nie jest pusta dopisanie elementu na koniec listy polega na dodaniu głowy listy do ogona, do którego końca dopisano element. ■

9.1.7. Programowanie deklaratywne

Chcemy tu zwrócić uwagę na specyficzny dla Prologu sposób pisania lub interpretowania programów. Rozpatrzmy najpierw dwa poniższe przykłady.

Przykład 9.12

Program 10 (przedstawiany jako przykład potwierdzenia efektywności i elegancji Prologu) sprawdza, czy dwie listy mają wspólny element. Wywołajmy go np. zapytaniem: `przeciecie([1,2,3,4],[8,7,6,5,4])`.

```
                        /* Program 10 */

przeciecie(L1,L2) :-element(X,L1),element(X,L2).

element(Nazwa,[Nazwa|_]).
element(Nazwa,[_|Ogon]) :- element(Nazwa,Ogon).
```

Mechanizm nawracania powoduje, że program sprawdza istnienie wspólnego elementu obu list. Patrząc na pierwszą klauzulę programu widać, że stanowi ona definicję wspólnego elementu, którą można następująco odczytać: jeżeli element `X` należy do listy `L1` i należy do listy `L2`, to listy `L1` i `L2` mają wspólny element. ■

Przykład 9.13

Przyjrzyjmy się działaniu Programu 12 obliczającego permutację zadanej listy. Wykonajmy program np. dla zapytania `perm([a,b,c,d],L)`.

```
                    /* Program 11 */

perm([],[]).
perm(L,[G|O]) :- usun(G,L,L1),
                 perm(L1,O).

usun(E,[E|O],O).
usun(E,[G|O],[G|O1]) :- usun(E,O,O1).
```

Procedura `usun(E,L1,L2)` usuwa element `E` z listy `L1` tworząc listę `L2`. Procedura `perm(L1,L2)` sprawdza, czy lista `L2` jest w relacji permutacją z listą `L1`. Pierwsza jej klauzula stwierdza, że permutacją listy pustej jest lista pusta. Druga klauzula mówi, że lista `[G|O]` jest permutacją listy `L`, jeżeli lista `O` jest permutacją listy powstałej z listy `L` po usunięciu z niej elementu `G`. Obie te klauzule definiują permutację. Łączne działanie rekurencji i nawracania powoduje, że program znajduje wszystkie permutacje elementów zadanej listy. ∎

Powyższe dwa przykłady a także 9.10 pokazują, że pisząc program w Prologu można koncentrować sie na podawaniu definicji operacji, które mają być wykonane, nie zajmując się tym, jak one będą wykonane. Takie podejście do programowania nosi nazwę deklaratywnego, w odróżnieniu od podejścia proceduralnego, w którym pisząc program musimy wiedzieć, jakie konsekwencje na wykonanie programu będzie miała każda wprowadzona procedura i w jakiej kolejności będą one wykonywane. Oczywiście, nie da się pisać programów w Prologu wyłącznie stosując podejście deklaratywne, szczególnie stosując sterowanie jego wykonaniem (następny punkt), tym niemniej możliwe w Prologu podejście deklaratywne znacznie ułatwia pisanie i czytanie programów.

9.2. Sterowanie wykonaniem programu

Prolog umożliwia ograniczone sterowanie przebiegiem wykonania programu przez wymuszanie nawracania oraz przez stosowanie tzw. odcięcia. W tej części omówimy obie te metody sterowania, a ponadto omówimy działanie predykatu standardowego `findall` tworzącego listę rozwiązań uzyskanych w procesie nawracania, działanie predykatu standardowego `not` i tzw. założenie o zamkniętości świata.

9.2.1. Wymuszanie nawracania – predykat fail

Jedyną własnością standardowego predykatu `fail` jest to, że nigdy nie jest on spełniony. Jeżeli napotkany zostanie w czasie wykonywania programu, to jego niespełnienie wymusza nawracanie. Stosowany jest więc do wymuszania nawracania w sytuacjach, w których nawracanie samoistnie by nie zaszło.

Przykład 9.14

Zbadamy działanie predykatu `fail` w poniższym programie.

```
/* Program 12 */

wykonaj :- kolor(X), write(X), write(' '), fail.
kolor(czerwony).
kolor(zielony).
kolor(niebieski).
kolor(czarny).
```

W programie oprócz predykatu `fail` występuje standardowy predykat `write(X)`. Predykat `write(X)` jest zawsze spełniony i przez swoje działanie uboczne drukuje nazwę obiektu, z którym jest związane `X`.

Aby zbadać działanie predykatu `fail`, usuńmy go z pierwszej klauzuli programu i wykonajmy program z zapytaniem `wykonaj`. Otrzymujemy wydruk: `czerwony yes`. Program nie wydrukował wszystkich obiektów spełniających zapytanie, ponieważ predykat `wykonaj` nie zawiera argumentów, a w takim przypadku Prolog zatrzymuje się po stwierdzeniu, że zapytanie zostało spełnione. Stwierdzenie spełnienia nastąpiło poprzez spełnienie `kolor(X)` dla `X=czerwony` i wydrukowanie `czerwony` przez predykat `write(X)`. Umieśćmy teraz z powrotem predykat `fail` i ponownie wykonajmy program dla dyrektywy wykonaj. Tym razem program drukuje:

```
czerwony zielony niebieski czarny no
```

Wydruk wszystkich obiektów spowodowany został przez predykat `fail`. Program napotykając na `fail` stwierdzał niespełnienie zapytania `wykonaj`, co wymuszało nawracanie do predykatu `kolor(X)`, który unifikowany był z kolejnymi klauzulami. Zauważmy, że ostatecznie zapytanie wykonaj nie zostało spełnione, co sygnalizowane jest przez komunikat `no`. ■

9.2.2. Odcięcie

Prolog umożliwia sterowanie mechanizmem nawracania. Dokonuje się tego przez zastosowanie standardowego predykatu *odcięcia* (ang. cut), oznaczonego przez wykrzyknik (`!`). Predykat `!` jest zawsze spełniony i jego działanie uboczne polega na tym, że blokuje wszystkie możliwe, pozostałe do wykonania, sprawdzenia spełnienia predykatów znajdujących się w danej klauzuli pomiędzy jej nagłówkiem i predykatem `!` oraz predykatu wywołującego nagłówek. Tym samym predykat `!` uniemożliwia nawracanie w obrębie klauzuli w której występuje i uniemożliwia ponowną unifikację predykatu wywołującego nagłówek klauzuli.

Przykład 9.15

Rozpatrzmy kilka wariantów użycia predykatu `!` w Programie 14 wywoływanym kwerendą `pp(P,Q,R)`.

```
/* Program 13 */

pp(X,Y,Z) :- p(X), p(Y), p(Z).
pp(aa,bb,cc) :- p(c).
```

```
p(a).
p(b).
p(c).
p(d).
```

Wykonując program uzyskujemy 65 rozwiązań (64 rozwiązania pierwszej klauzuli – `PQR=aaa, aab,..., ddd`, i jedno dawane przez drugą – `PQR=aabbcc`). Jeżeli teraz zmienimy czwartą klauzulę Programu 10 na `p(b) :- !.` i ponownie go wykonamy, to otrzymamy 9 rozwiązań (8 rozwiązań klauzuli pierwszej, `PQR=aaa, aab, aba, abb, baa, bab, bba, bbb` i jedno dawane przez drugą `PQR=aabbcc`). Ograniczenie liczby rozwiązań spowodowane zostało przez dodanie predykatu `!`, który wywołany po unifikacji predykatu `p(Z)` z nagłówkiem klauzuli `p(b) :- !.`, odcina wszystkie dalsze możliwe unifikacje tego predykatu (czyli odcina możliwość unifikacji `p(Z)` z `p(c)` i `p(d)`). Identycznie się dzieje po unifikacjach z klauzulami `p(Y)` i `p(X)`. Zauważmy, że działanie predykatu `!` ma charakter lokalny (ograniczony do danej klauzuli, w której występuje, i do danego jej wywołania) i nie przeszkadza w spełnieniu predykatu `pp(aa,bb,cc)` uwarunkowanego spełnieniem `p(c)`.

Powróćmy znów do Programu 14 i zamieńmy tym razem klauzulę pierwszą na:

```
pp(X,Y,Z) :- p(X), p(Y), !, p(Z).
```

Efekt umieszczenia `!` jest tu oczywisty. Po unifikacji `p(X)` z `p(a)` i `p(Y)` z `p(a)` następuje wywołanie predykatu `!`, który odcina dalsze próby unifikacji `p(X)` i `p(Y)` a także samej kwerendy `p(P,Q,R)`. Następuje jeszcze przejście do predykatu `p(Z)`, który unifikuje się czterokrotnie i na tym program się kończy dając cztery rozwiązania `PQR=aaa,aab,aac,aad`.

Kończąc te rozważania łatwo przewidzieć, że Program 14 wywołany zapytaniem `pp(P,Q,R), !.`, odpowie `PQR=aaa`. ■

Typowymi zastosowaniami predykatu odcięcia jest przekształcanie procedur niedeterministycznych w deterministyczne oraz eliminacja zbędnych przeszukiwań przy wykonywaniu programu. Obydwa te zastosowania zilustrowane są w poniższych przykładach.

Przykład 9.16

Rozpatrzmy działanie Programu 15, który stanowi niewielką modyfikację programu 8, polegającą na dodaniu do pierwszej klauzuli programu predykatu `!`.

```
/* Program 14 */

element(Nazwa,[Nazwa|_]) :- !.
element(Nazwa,[_|Ogon]) :- element(Nazwa,Ogon).
```

Dodanie predykatu odcięcia zamieniło niedeterministyczną procedurę element w procedurę deterministyczną. Widać to wyraźnie w przypadku wywołania programu zapytaniem `element(X,[a,b,c,d])`. Program 7 w odpowiedzi na to zapytanie generował rozwiązania będące kolejnymi elementami zadanej listy. Program 15 generuje tylko jedno rozwiązanie, `X=a`. Dzieje się tak dlatego, że po unifikacji zapytania z nagłowkiem pierwszej klauzuli i związaniu zmiennej `X` z `a` program przechodzi do

prawej strony reguły, gdzie napotyka predykat `!`, którego działanie blokuje ponowną możliwość unifikacji zapytania z następną klauzulą programu i w związku z tym działanie programu się kończy. ■

Przykład 9.17

Rozpatrzmy zadanie polegające na wyborze z bazy danych osób spełniających wymagane kryterium. Baza danych składa się z dwóch zbiorów faktów reprezentowanych przez predykaty `student(Numer)` i `dochod(Numer,Wielkość_Dochodu)`, przyczym argumenty `Numer` sa niepowtarzajacym się identyfikatorem osób. Na zapytanie `rejestracja(X)` program powinien podać numery wszystkich studentów o dochodach mniejszych od 10000.

Oto program spełniający postawione zadanie.

```
                        /* Program 15 */

rejestracja(X) :- student(X), dochod(X,Y),
                  Y<=10000.

  student(321).
  student(444).
  student(222).
  student(123).
  student(124).

  dochod(321,12000).
  dochod(541,18000).
  dochod(444,9000).
  dochod(222,2400).
  dochod(123,13000).
  dochod(124,11500).
  dochod(311,8800).
```

Przyglądając się działaniu programu widać, że sprawdzając spełnienie predykatu `dochod` będzie on niepotrzebnie przebiegał wszystkie fakty `dochod` niezależnie od tego, czy unifikacja już nastąpiła czy nie. Program nie wykorzystuje założenia, że numery osób w bazie danych nie powtarzają się.

Użycie predykatu odcięcia zapobiega zbędnemu przeglądaniu zespołu faktów wiek. Zrealizujemy to zastępując pierwszą klauzulę programu 16 dwoma klauzulami

```
  rejestracja(X) :- student(X), pom(X,Y),
                    Y<=10000.

  pom(X,Y) :- dochod(X,Y),!.
```

Użycie w programie klauzuli `pom(X,Y) :- dochod(X,Y),!.` powoduje, że poszukiwanie spełnienia predykatu `pom(X,Y)` kończy się w chwili jakiejkolwiek unifikacji predykatu `dochod` z faktem `dochod`, ponieważ pociąga to za sobą wywołanie predykatu odcięcia i tym samym zablokowanie dalszego przeglądania faktów `dochod`. Z

drugiej strony takie użycie predykatu odcięcia nie przeszkadza programowi w znalezieniu wszystkich odpowiedzi na postawione zapytanie (zamiana wszystkich faktów `dochod` na reguły `dochod :- !.` skróciłaby także proces przeglądania bazy danych, ale spowodowałaby to, że program znajdował by tylko jedną odpowiedź). ■

Przykład 9.18

Ostatnim przykładem zastosowania predykatu `!` jest Program 17 usuwający z listy powtarzające się elementy. Program wywoływany jest zapytanie `usun_pow(L1,L2)`, gdzie `L1` jest listą zadaną, zaś `L2` listą po usunięciu powtórzeń. Np. na zapytanie `usun_pow([a,b,a,b,c,a],L)` program odpowiada `L=[b,c,a]`.

```
/* Program 16 */

usun_pow([],[]).
usun_pow([X|Y],W) :- element(X,Y), !,
                     usun_pow(Y,W).
usun_pow([X|Y],[X|W]) :- usun_pow(Y,W).

element(Nazwa,[Nazwa|_]).
element(Nazwa,[_|Ogon]) :- element(Nazwa,Ogon).
```

Predykat `!` umieszczono dla zapobieżenia unifikacji zapytania `usun_pow(L1,L2)` z klauzulą trzecią, po unifikacji z klauzulą drugą i stwierdzeniu spełnienia predykatu `element(X,Y)`.

9.2.3. Tworzenie listy obiektów – predykat `findall`

Prolog oferuje szereg predykatów standardowych, których argumentami są predykaty. Stanowi to odstępstwo od składni klauzul Horna, którymi posługuje się Prolog i predykaty takie nazywa się metajęzykowymi. Istnienie predykatów metajęzykowych zakłóca obraz Prologu jako systemu dowodzenia metodą rezolucji twierdzeń logicznych zadanych w postaci klauzul Horna, ale z drugiej strony znacznie zwiększa efektywność języka.

Predykat standardowy

```
findall(NazwaZmiennej,<atom>,Lista)
```

powoduje wywołanie predykatu `<atom>`, a następnie gromadzi na liście `Lista` wszystkie uzyskane w procesie nawracania wartości zmiennej `NazwaZmiennej`. Predykat `<atom>` musi zawierać zmienną o nazwie `NazwaZmiennej`, zaś dziedzina listy `Lista` musi być zadeklarowana w bloku `domains`.

Przykład 9.19

W Programie 17, stanowiącym rozszerzenie Programu 11, predykat `findall` powoduje zebranie w listę wszystkich permutacji elementów zadanej listy.

```
                    /* Program 17 */

perm([],[]).
perm(L2,[G|O]) :- usun(G,L2,L),
                  perm(L,O).

usun(E,[E|O],O).
usun(E,[G|O],[G|O1]) :- usun(E,O,O1).

permutacja(L,LL) :- findall(P,perm(L,P),LL).
```

Po zapytaniu `perm([a,b,c,d],P).` program znajduje wszystkie permutacje listy `[a,b,c,d]`, zaś po zapytaniu `permutacja([a,b,c,d],L).` program podaje listę (listę list) wszystkich permutacji. ∎

9.2.4. Predykat `not` i zamkniętość świata

Standardowy predykat `not`, którego argumentem jest predykat, jest spełniony, jeżeli niespełniony jest predykat będący jego argumentem. W Prologu wymagane jest, aby w chwili wywołania predykatu `not` nie zawierał on jako argumentów zmiennych niezwiązanych

Przykład 9.20

Program 20 umożliwia sprawdzenie, czy dana osoba może być zatrudniona w firmie Moda, zatrudniającej wyłącznie kobiety niezamężne.

```
                    /* Program 18 */

zatrudnia('Moda',X) :- not(s_c(X,z)), not(plec(X,m)).

s_c(jan,w).
s_c(barbara,z).
s_c(ewa,w).

plec(jan,m).
plec(barbara,k).
plec(ewa,k).
```

Zadając programowi zapytania: `zatrudnia('Moda',jan).`, `zatrudnia('Moda',barbara).`, `zatrudnia('Moda',ewa).` otrzymamy odpowiednio odpowiedzi `no`, `no`, `yes`. Po zapytaniu `zatrudnia('Moda',X).` zostanie zasygnalizowany błąd spowodowany swobodną zmienną w `not`).

Odpowiednikiem użycia predykatu standardowego `not(s_c(X,z))` może być użycie predykatu `not_z` zdefiniowanego następująco:

```
not_z(X) :- s_c(X,z), !, fail.
not_z(_).
```

Zauważmy, że Program 20 zapytany `zatrudnia("Moda",zenon).` odpowiada `yes`, co jest konsekwencją niespełnienia predykatów `s_c(zenon,z)` i `plec(zenon,m)`. Niespełnienie tych predykatów wynika z przyjętego w Prologu założenia, że wszystkie predykaty, których spełnienia nie daje się wykazać, są niespełnione. Założenie to nazywa się *założeniem o zamkniętości świata* (ang. closed world assumption). Alternatywą dla założenia o zamkniętości jest założenie o otwartości świata, które także łatwo można zrealizować w Prologu. W świecie otwartym istnieją dane mówiące o prawdziwości pewnych faktów i dane mówiące o nieprawdziwości. Program bada prawdziwość lub fałszywość zadanych faktów, a nie znajdując potwierdzenia ich prawdziwości lub fałszywości sygnalizuje brak danych.

Przykład 9.21

Program 21 realizuje założenie o świecie otwartym. Przeprowadza on ocenę pracowników biorąc pod uwagę predykaty `d_p (dobry pracownik)` i `z_p (zły pracownik)`, a po spełnieniu jednego z nich wyświetla uzyskaną ocenę pracownika. Gdy żaden z predykatów nie jest spełniony, program wyświetla tekst: „`brak danych`“. Zauważmy, że klauzula `ocena(_) :- write("brak danych").` jest równoważna klauzuli

```
ocena(X) :- not(d_p(X)), not(z_p(X)),
            write("brak danych").
```

Wykonajmy program podając zapytania: `ocena(jan).`, `ocena(monika).`, `ocena(henryk).`, `ocena(zenon).`

```
                /* Program 19 */

ocena(X) :- d_p(X),
            write("dobry pracownik "), !.
ocena(X) :- z_p(X),
            write("zly pracownik "), !.
ocena(_) :- write("brak danych ").

d_p(X) :- punktualny(X), staz(X,L), L>2.
d_p(jan). d_p(marek). d_p(ewa).

punktualny(monika).  punktualny(maria).

staz(monika,3).  staz(mariusz,5).
z_p(henryk).  z_p(mieczyslaw).
```

Literatura

[1] Adami Ch., Introduction to artifial life, Springer, New York, 1998.

[2] Arabas J., (2001) Wykłady z algorytmów ewolucyjnych. WNT. Warszawa 2001.

[3] Bedau M.A., McCaskill J.S., Packard N., Rasmussen S., Adami C., Green D.G., Ikegami T., Kaneko K., Ray T.S.: Open Problems in Artificial Life, Artificial Life, v. 6, pp 363-376, 2000.

[4] Bishop M.C., (1996) Neural Networks for Pattern Recognition, Clarendon Press, Oxford.

[5] Bolc L, Cytowski J., (1989, 1991) Metody przeszukiwania heurystycznego, t.I i t.II, PWN, Warszawa 1989, 1991

[6] Bolc L, Zaremba J., (1992), Wprowadzenie do uczenia się maszyn, Akademicka Oficyna Wydawnicza RM, Warszawa.

[7] Bonabeau E., Dorigo M., Theraulaz G., Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York 1999.

[8] Brachman R.J., Levesque H.J., (2004), Knowledge representation and reasoning, Morgan Kaufman Publishers, Amsterdam.

[9] Brit (1998), Encyclopedia Britannica CD 99, Multimedia Edition, Shockwave Macromedia, Inc., Encyclopedia Britannica, Inc.

[10] Buller A., (1998), Sztuczny mózg, Prószyński i S-ka, Warszawa.

[11] Callan R. (2003), Artificial Intelligence, Palgrave Macmillan, Houndmills, Hampshire.

[12] Campbell M., Hoane J.A.Jr., Hsu F., (2002), Deep Blue, Artificial Intelligence v. 134, pp. 57–83.

[13] Cichosz P., (2000), Systemy uczące się, WNT, Warszawa.

[14] Engelbrecht A., Fundamentals of computational swarm intelligence, J. Wiley, Chichester, 2006.

[15] Engelbrecht A., Computational intelligence, J. Wiley, Chichester, 2007.

[16] Franchi S., Guzeldere G., (1995)(ed.), Construction of the Mind: Artificial Intelligence and the Humanities, Stanford Humanities Review. v.4. issue 2, 1995, internet: http://shr.stanford.edu/shreview/4-2/toc.htm.

[17] Goldberg D.E., (1995), Algorytmy genetyczne i ich zastosowania, WNT, Warszawa.

[18] Gray P., (1984), Logic, algebra and Databases, Ellis Horwood, Chichester.

[19] Gurba K., (1995), Sztuczna inteligencja z naturalnymi ograniczeniami, Znak, t. XLVII, nr 9, z. 484, wrzesień, Kraków.

[20] Heckerman D., Wellman M.P., (1995), Bayesian networks, Communications of the ACM, vol. 38, No. 3, p. 27-30, March 1995

[21] Hippe Z., (1993), Zastosowanie metod sztucznej inteligencji w chemii, PWN, Warszawa.

[22] Hsu F., Anantharaman T, Campbell M., Nowatzyk A., (1990), A Grandmaster Chess Machine, Scientific American, October 1990.

[23] Jang J-S.R., Sun C-T., Mizutani E., (1997), Neuro-Fuzzy and Soft Computing, Prentice Hall, Upper Saddle River.

[24] Jędruch W., (1989), Turbo Prolog, Wyd. Pol. Gd.

[25] Jędruch W., (1997), Środowisko programowe dla modelowania cząsteczkowego systemów złożonych, Zeszyty Naukowe Politechniki Gdańskiej – monografie, Elektronika, Nr 84, Gdańsk.

[26] Jędruch W., Sienkiewicz R., Inteligencja zespołowa, w: (Kowalczuk Z., Wiszniewski B., red.) Inteligentne wydobywanie informacji w celach diagnostycznych, Pomorskie Wydawnictwo Naukowo Techniczne, Gdańsk, 2007, s. 413-432

[27] Kennedy J., Eberhardt R.C.: Swarm intelligence, Morgan Kaufmann Publishers, San Francisco, 2001.

[28] Knitter S.: *Badanie dynamiki systemów samoreprodukujących się w sztucznym środowisku.* (Praca dyplomowa), WETI, Pol. Gdańska, Gdańsk 2006.

[29] Kosiński R.A., (2002), Sztuczne sieci neuronowe, dynamika nieliniowa i chaos, WNT, Warszawa.

[30] Kowalski R., (1989), Logika w rozwiązywaniu zadań, WNT, Warszawa.

[31] Koza J., (1992), Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, Massachusetts.

[32] Koza J., (1994) Genetic Programming II: Automatic Discovery of Reusable Programs (Complex Adaptive Systems), MIT Press, Cambridge, Massachusetts.

[33] Koza J., Bennett F.H.III, Bennett F.H., Keane M., Andre D., (1999) Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis, Morgan Kaufmann Publishers.

[34] Koza J.R., Keane M.A., Streeter M.J., Mydlowec W., Yu J., Lanza G., Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer Academic Publishers, 2003.

[35] Kubale M., (1994), Introduction to computational comlexity, Wydawnictwo Politechniki gdańskiej, Gdańsk.

[36] Langton C.G. (1989), Artificial life. w: Langton C.G, (wyd.) Artificial life, Proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems, Los Alamos, New Mexico, September, 1987, Redwood City, Cal.: Addison-Wesley.

[37] Lem S., (1996), Tajemnica chińskiego pokoju, Universitas, Kraków.

[38] Levin R. (1992), Complexity: science on the edge of chaos. Macmillan.

[39] Li H., (1998), An introduction to belief networks, Technical research report, The Center for Satellite and Hybrid Communication Network, internet: http://www.isr.umd.edu/CSHCN/

[40] Ligęza A., Czarkowski D., Marczyński P., Włodarczyk M., (1991), Turbo Prolog na IBM PC, Wydawnictwo AGH, Kraków. Warszawa.

[41] LPA, (1990), LPA Prolog Peofessional Compiler, programming reference manual, Logic Programming Associates, London.

[42] Malpas J. (1987), Prolog: a relational language and its applications, Prentice-Hall, Englewood Cliffs, NJ.

[43] Mettrie La, J.O., (1984), Człowiek-Maszyna, PWN, Warszawa.

[44] Michalewicz Z., (1996), Algorytmy genetyczne + struktury danych = programy ewolucyjne, WNT, Warszawa.

[45] Michalski R.S., Kubat M., Bratko I., (1998) Machine Learning & Data Mining: Methods & Applications, J. Wiley & Sons, Chichester.

[46] Mulawka J.J., (1996) Systemy ekspertowe, WNT, Warszawa.

[47] NEP (1995), Nowa Encyklopedia Powszechna PWN, Wydawnictwo Naukowe PWN, Warszawa.

[48] Newell A., Simon H. (1976), Computer science as empirical inquiry: symbols and search, Communication of the ACM, march 1976, v. 19, n. 3, p. 113-126.

[49] Neumann J., von, *Theory of Self-reproducing Automata*. Burks A. (wyd.), Champaign-Urbana, 1966, University of Illinois Press.

[50] Nilsson N.J., (1980), Principles of Artificial Intelligence, Tioga, Palo Alto, Cal.

[51] Nilsson N.J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York 1971.

[52] Pearl J., (1988), Probabilistic reasoning in intelligent systems, Morgan Kauffmann, San Mateo, Cal.

[53] Penrose R., (1995), Nowy umysł cesarza, PWN, Warszawa.

[54] Penrose R., (2000), Cienie umysłu, Poszukiwanie naukowej teorii świadomości, Zysk S-ka. Poznań.

[55] Preston J., Bishop M., (ed.), (2002), Views into the chinese room, New essays on Searle and artificial intelligence, Oxford University Press, Oxford.

[56] Russel S., Norvig P., (2003), Artificial Intelligence, Prentice-Hall, London.

[57] Rutkowska D., Piliński M., Rutkowski L., (1997), Sieci neuronowe, algorytmy genetyczne i systemy rozmyte, PWN, Warszawa-Łódź.

[58] Rutkowski L.: Metody i techniki sztucznej inteligencji, Wydawnictwo Naukowe PWN, Warszawa 2006.

[59] Searle J., (1992), The rediscovery of the mind, MIT Press, Cambridge, [wyd. pol.: Umysł na nowo odkryty, PIW, Warszawa, 1999].

[60] Searle J., The mystery of consciousness, A New York Review Book, New York 1997.

[61] Shannon C.E., (1950), A Chess-Playing Machine, Scientific American, February, 1950.

[62] Sienkiewicz R., Jedruch W.: *Artificial Environment for Simulation of Emergent Behaviour*. – Beliczynski B., Dzielinski A., Iwanowski M., Ribeiro B. (red.): Adaptive and natural computing algorithms. Proc. of the ICANNGA, Warsaw 2007. Lecture Notes in Computer Science, no. 4431, part 1, s. 386-393. Springer, Berlin, 2007.

[63] Sutton R.S., Barto A.G., (1998), Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, A Bradford Book.

[64] Turing A., (1950), Computing machinery and intelligence, Mind, p. 433-460, wyd. pol.: Maszyna licząca a inteligencja, w: B. Chwedczuk (wyb.), Filozofia umysłu, Fundacja „Alatheia“, Warszawa 1995.

[65] Wolfram S., (2002) A new kind of science, Wolfram Media.

[66] Wójcik M., (1991), Zasada rezolucji, PWN, Warszawa.

[67] http://www.swarm.eti.pg.gda.pl/.

[68] Yoon Y.O., Brobst R.W., Bergstresser P.R., Peterson L.L. A desktop neural network for dermatology diagnosis, J. of Neural Network Computing, 1989, Summer, 43–52.

[69] Żurada J., Barski M., Jędruch W., (1996), Sztuczne sieci neuronowe, PWN, Warszawa.