Politechnika Warszawska

Ośrodek Kształcenia Na Odległość

Inteligentne Techniki Obliczeniowe (studia magisterskie)

Autor: Włodzimierz Kasprzak

Wersja 2.0 beta

Warszawa, wrzesień 2018

Inteligentne Techniki Obliczeniowe

Przedmiot: Inteligentne Techniki Obliczeniowe / Metody Sztucznej Inteligencji – studia magisterskie

Edycja: 3.0

Słowa kluczowe: e-book, reprezentacja wiedzy, wnioskowanie, przeszukiwanie, planowanie, uczenie, wiedza niepewna, Dynamiczna Sieć Bayesa

Autor: Włodzimierz Kasprzak <W.Kasprzak(at)elka.pw.edu.pl>

Spis treści

Inteligentne Techniki Obliczeniowe (studia magisterskie) ... 1
Spis treści ... 3
1. System agentowy ... 11
1.1. Systemy agentowe ... 12
1.2. Środowisko ... 15
1.3. Rodzaje agentów ... 16
1.4. System z bazą wiedzy ... 21
1.5. Logiczne agenty ... 21
1.6. Pytania ... 23
1.7. Zadania ... 23
2. Logika klasyczna ... 26
2.1. Inżynieria wiedzy ... 26
2.2. Języki logiki ... 27
2.2.1. Rachunek zdań ... 27
2.2.2. Logika predykatów – logika pierwszego rzędu ... 28
2.2.3. Semantyka rachunku zdań ... 29
2.2.4. Semantyka języka predykatów ... 31
2.3. Rachunek sytuacyjny w logice predykatów ... 33
2.4. System logicznego wnioskowania ... 35
2.4.1. Zadania systemu ... 35
2.4.2. Wybrane rodzaje systemów logicznego wnioskowania ... 36
2.5. Ontologia języka ... 36
2.6. Pytania ... 40
2.7. Zadania ... 40
3. Wnioskowanie w logice ... 42
3.1. Reguły wnioskowania ... 42
3.1.1. Wnioskowanie w rachunku zdań ... 42
3.1.2. Wnioskowanie w logice predykatów ... 44
3.2. Postaci normalne formuł ... 46
3.3. Wnioskowanie metodą rezolucji ... 49
3.4. Wnioskowanie z regułą odrywania ... 50
3.4.1. Wnioskowanie w rachunku zdań ... 50
3.4.2. Wnioskowanie w logice predykatów ... 54
3.5. Pytania ... 55
3.6. Zadania ... 55
4. Rozszerzenia logiki klasycznej ... 57

4.1. Logika niemonotoniczna ... 57
4.2. Logika modalna ... 58
4.3. Logika opisowa ... 59
4.4. Zadania ... 60
5. Przeszukiwanie jako realizacja celu ... 65
5.1. Przeszukiwanie przestrzeni stanów ... 65
5.1.1. Problem przeszukiwania ... 66
5.1.2. Przeszukiwanie drzewa ... 67
5.1.3. Przeszukiwanie grafu ... 69
5.2. Przeszukiwanie poinformowane ... 70
5.2.1. Strategia zachłanna („najbliższy celowi najpierw”) ... 71
5.2.2. Przeszukiwanie A* ... 74
5.3. Składowe heurystyczne ... 78
5.3.1. Dominująca heurystyka ... 78
5.3.2. Generowanie heurystyk metodą „złagodzonego problemu” ... 78
5.4. Przeszukiwanie lokalne poinformowane ... 79
5.5. Symulowane wyżarzanie ... 81
5.6. Pytania ... 82
5.7. Zadania ... 82
6. Zaawansowane algorytmy przeszukiwania ... 85
6.1. IDA* („Iterative deepening A*) ... 85
6.2. SMA* („Simplified memory bounded A*”) ... 86
6.3. „Anytime A*” (przeszukiwanie z wynikiem „o dowolnym czasie”) ... 89
6.4. Real-time A* ... 90
6.5. Zadania ... 92
7. Problemy z ograniczeniami ... 93
7.1. Problem CSP ... 93
7.1.1. Przykłady ograniczeń ... 93
7.1.2. Przeszukiwanie przyrostowe lub ze stanem kompletnym dla CSP ... 95
7.2. Przeszukiwanie przyrostowe dla CSP ... 95
7.2.1. Przeszukiwanie z nawrotami ... 95
7.2.2. Usprawnienia przeszukiwania z nawrotami ... 96
7.3. Lokalne przeszukiwanie dla CSP ... 99
7.4. Pytania ... 99
7.5. Zadania ... 100
8. Klasyczne planowanie działań ... 101
8.1. Agent realizujący cel ... 101
8.1.1. Przeszukiwanie przestrzeni planów ... 101

8.1.2. Planowanie ... 102
8.2. Klasyczne planowanie ... 102
8.2.1. Planowanie w języku logiki ... 102
8.2.2. STRIPS ... 103
8.3. Przestrzeń planów częściowych ... 105
8.3.1. Plan częściowo uporządkowany ... 106
8.3.2. Tworzenie planu częściowo uporządkowanego ... 107
8.4. Przykład budowy planu ... 108
8.5. Pytania ... 112
8.6. Zadania ... 112
9. Zaawansowane techniki planowania ... 114
9.1. Graf planujący ... 114
9.2. Algorytm „Graphplan” ... 116
9.3. Planowanie hierarchiczne (*) ... 117
9.4. Zadania ... 122
10. Wiedza niepewna i niedokładna ... 127
10.1. Wiedza niedoskonała ... 127
10.2. Reprezentacja probabilistyczna ... 128
10.3. Wnioskowanie przez wyliczanie ... 130
10.4. Warunkowa niezależność ... 132
10.5. Sieć Bayesa ... 133
10.6. Wnioskowanie w logice rozmytej ... 137
10.7. Pytania ... 140
10.8. Zadania ... 141
11. Wnioskowanie w sieci Bayesa ... 143
11.1. Dokładne wnioskowanie „przez przeliczanie” ... 143
11.1.1. Dokładne wnioskowanie probabilistyczne ... 143
11.1.2. Algorytm dokładnego wnioskowania „przez przeliczanie” zmiennych ... 143
11.2. Dokładne wnioskowanie „z eliminacją zmiennych” ... 145
11.2.1. Idea algorytmu ... 145
11.2.2. Algorytm dokładnego wnioskowania „z eliminacją zmiennych” ... 146
11.3. Przybliżone wnioskowanie metodami „symulacji stochastycznej” ... 147
11.4. Pytania ... 148
11.5. Zadania ... 148
12. „Dynamiczna Sieć Bayesa” ... 149
12.1. Czas i niepewność ... 149
12.1.1. Modele Markowa ... 149
12.1.2. Dynamiczna sieć Bayesa ... 149

12.2. Wnioskowanie dla systemów dynamicznych ... 150
12.2.1. Filtracja ... 151
12.2.2. Predykcja ... 152
12.2.3. Wygładzanie ... 152
12.2.4. Najbardziej prawdopodobna sekwencja stanów ... 154
12.3. Szczególne postacie DBN ... 155
12.3.1. HMM (Ukryty Model Markowa) ... 155
12.3.2. Filtr Kalmana ... 155
12.4. Pytania ... 156
12.5. Zadania ... 156
13. Wnioskowanie w DBN ... 157
13.1. Filtr Kalmana ... 157
13.1.1. Równania filtru Kalmana ... 158
13.1.2. Odmiany filtru Kalmana ... 159
13.2. Wnioskowanie dokładne dla DBN ... 159
13.2.1. „Naiwne” wnioskowanie dla DBN ... 160
13.3. Wnioskowanie przybliżone dla DBN ... 160
13.3.1. Metoda symulacji stochastycznej według „wag wiarygodności” ... 160
13.3.2. Filtr cząstek ... 160
13.4. Pytania ... 162
13.5. Zadania ... 163
14. Uczenie się ze wzmacnianiem ... 166
14.1. Zadanie uczenia ze wzmacnianiem ... 166
14.2. Uczenie się strategii ... 167
14.3. Problem decyzyjny Markowa a funkcja użyteczności ... 168
14.4. Uczenie się funkcji użyteczności ... 170
14.5. Uczenie się strategii ... 170
14.6. Pytania ... 171
14.7. Zadania ... 171
15. Racjonalne decyzje ... 172
15.1. Decyzje pojedyncze ... 172
15.1.1. Klasyfikator numeryczny ... 172
15.2. Podstawowe rodzaje klasyfikatorów numerycznych ... 173
15.3. Sekwencja decyzji prostych ... 174
15.4. Programowanie dynamiczne ... 175
15.5. Pytania ... 176
15.6. Zadania ... 177
16. Uczenie statystyczne ... 178

16.1. Klasyfikator stochastyczny Bayesa 178
16.1.1. Definicja 178
16.1.2. Uczenie się rozkładów prawdopodobieństwa 178
16.1.3. Klasyfikator geometryczny według minimalnej odległości 180
16.1.4. Klasyfikator „według k sąsiadów” 181
16.2. Uczenie modelu HMM 181
16.2.1. Zadanie algorytmu Bauma-Welcha 182
16.2.2. Określenie prawdopodobieństw "wprzód i wstecz" 182
16.2.3. Algorytm estymacji prawdopodobieństw “wprzód i wstecz” 182
16.2.4. Estymacja parametrów modelu HMM 183
16.3. Uczenie bez nadzoru 184
16.3.1. Klasteryzacja “*k*-średnich” 184
16.3.2. Klasteryzacja EM (“Expectation Maximization”) 185
16.3.3. Wyznaczanie liczby klastrów 186
16.4. Pytania 188
16.5. Zadania 188
Bibliografia 190

Część I: Wnioskowanie w logice

1. System agentowy

Próbując zdefiniować czym jest **Sztuczna Inteligencja** przyjęło się wyróżniać cztery podejścia do tego zagadnienia, nazywane [4]:

1. Myśleć jak człowiek
2. Myśleć racjonalnie
3. Działać jak człowiek
4. Działać racjonalnie

W latach 1960-ych pojawiła się dziedzina wiedzy zwana „kognitywistyką". Z początku ograniczona była do „psychologii poznawczej", czyli psychologii przetwarzania informacji. Kognitywistyka tworzy naukowe teorie aktywności mózgu. Powstaje pytanie, jak te teorie należy weryfikować? Możliwe są zasadniczo dwa sposoby:

1. Przewidzieć model i testować zachowania osób (tzw. podejście „top-down"), co czyni psychologia poznawcza,
2. Bezpośrednia identyfikacja modelu na podstawie danych neurologicznych (tzw. podejście „bottom-up") – jest toi przedmiotem „neuro-kognitywistyki".

Oba podejścia („cognitive science" i „cognitive neuroscience") są obecnie oddzielone od Sztucznej Inteligencji.

Podejście zwane „myśleć racjonalnie" bazuje na badaniach logiki myślenia. Już w starożytności Arystoteles badał to, jakie argumenty i procesy myślowe *są prawidłowe*, a greckie szkoły logiczne wprowadziły *zapis* i *reguły wyprowadzania* myśli. Problemy tego podejścia do inteligencji:

1. Nie każde inteligentne zachowanie jest wynikiem logicznego wyprowadzenia.
2. Czy potrafimy odpowiedzieć na pytania" „Jaki jest cel myślenia?", „Jakie myśli są prawidłowe?".

Podejście zwane „działać jak człowiek" sięga słynnego Testu Turinga (1950). Alan Turing tłumaczył pytanie „czy maszyna potrafi myśleć?" na to „czy maszyna działa inteligentnie (tak jak człowiek)?" Test Turinga to test inteligentnego zachowania się maszyny – jest to próba imitowania człowieka przez komputer. Pionierskie prace Turinga zaowocowały zdefiniowaniem głównych zadań tego typowego podejścia do Sztucznej Inteligencji: reprezentacja wiedzy, wnioskowanie, przeszukiwanie, uczenie.

Nasze podejście to: „**działać racjonalnie**". Racjonalne działać oznacza „wykonywać właściwą akcję". Właściwa akcja to taka, która maksymalizuje osiągnięcie celu przy zadanej informacji. Niekoniecznie wymaga to myślenia, np. może to być też nieświadomy refleks, ale myślenie powinno być na usługach racjonalnego działania. Takie podejście wymaga powiększenia zadań Sztucznej Inteligencji zagadnienia percepcji środowiska i wykonywania akcji. Zalety podejścia „działać racjonalnie" w porównaniu do poprzednich to:

- Pojęcie „działania" jest ogólniejsze od pojęcia „myślenia".
- Pojęcie „racjonalny" jest lepiej określone niż „ludzki".

Główne zastosowania metod **sztucznej inteligencji** jak dotąd to:

1. **systemy ekspertowe,** realizujące podejście „myśleć racjonalnie", wspomagające człowieka w problemach logistycznych, diagnostyce medycznej, reagowaniu w nieoczekiwanych sytuacjach lub podejmowaniu decyzji biznesowych;
2. **systemy agentowe**, realizujące podejście "działać racjonalnie", mają posiadać wyższy poziom autonomii, posiadając zdolności percepcji, wykonywania akcji i uczenia się.

Przedmiotem wykładu jest projektowanie **racjonalnie działających agentów,** ograniczone do ich komputerowej symulacji w postaci tzw. „softbot-ów". Materiał podzielony został na cztery zasadnicze części: systemy logicznego wnioskowania, przeszukiwanie i planowanie jako rozwiązanie problemu, działanie w warunkach niepewności i uczenie się.

1.1. Systemy agentowe

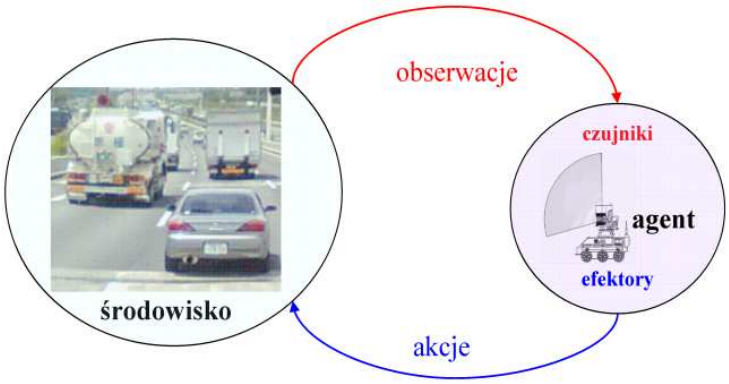
Pojęcie „agenta"

Agentem jest każdy obiekt (jednostka), który obserwuje (odbiera) swoje otoczenie dzięki czujnikom (sensorom) i oddziałuje na to środowisko przy pomocy „efektorów" (wykonuje akcje).

Formalnie: agent to funkcja z dziedziny historii obserwacji (percepcji) w wykonywane akcje:

$$[f: P^* \rightarrow A]$$

Ludzki agent posiada: oczy, uszy i inne organy pełniące role czujników oraz ręce, nogi, usta i inne części ciała będące efektorami. Agent upostaciowiony (robot, maszyna) posiada: kamery, czujniki podczerwieni, skanery laserowe, itp., będące czujnikami oraz różne silniki, napędy i manipulatory będące efektorami. Agent nieupostaciowiony (program komputerowy, *softbot*) to odpowiedni program wykonujący się na fizycznym komputerze (o określonej architekturze) i realizujący funkcję *f* agenta. Głównym wyznacznikiem przy projektowaniu agenta będzie znalezienie najbardziej efektywnego działania w zależności od charakteru środowiska i rodzaju zadania. W przypadku agenta programowego wystąpi jeszcze problem ograniczenia zasobów komputera i wtedy naszym celem będzie: zaprojektowanie najlepszego programu z możliwych przy zadanych zasobach maszynowych.

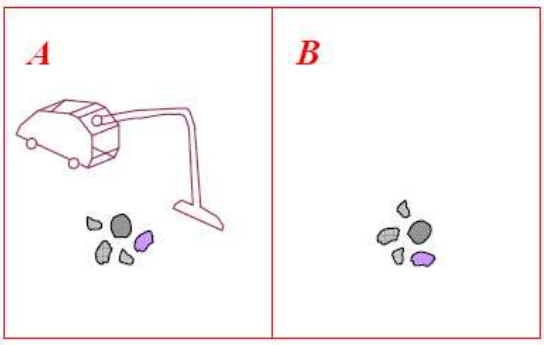


Rys. 1.1: Elementy systemu agentowego.

Zanim przystąpimy do pewnej kategoryzacji agentów zilustrujmy ten problem na przykładzie prostego agenta i możliwych sposobów jego realizacji.

Przykład: „agent odkurzający" (rys. 1.2)

Percepcja tego agenta to 2-elementowy wektor: [położenie w kratce, stan czystości kratki]. Np. możliwe obserwacje to: [*A, kurz*], [*B, czysto*]. Załóżmy, że agent może wykonywać 4 akcje: { *Lewo, Prawo, Odkurzaj, NicNieRób* }. Program prostego agenta „odkurzającego" przedstawia Tablica 1-1.



Rys. 1.2: Agent-odkurzacz i jego środowisko.

Tabl. 1-1. Funkcja prostego agenta-odkurzającego

```
function ProstyAgentOdkurzajacy( [lokalizacja, czystosc]) returns Akcja
{
  if (czystosc == kurz)  return Odkurzaj;
  else        if (lokalizacja == A)  return Prawo;
              else if (lokalizacja == B) return Lewo;
}
```

Zauważmy, że powyższy agent nie posiada pamięci o przeszłych obserwacjach. Tym samym nie będzie on wiedział, kiedy się zatrzymać. Jeśli środowisko jest deterministyczne, to byłoby możliwe zatrzymanie agenta wtedy, gdy w obu kratkach będzie czysto. To tego potrzebne jest jednak zapamiętanie przynajmniej jeszcze przedostatniej obserwacji. Możemy wyobrazić sobie program agenta korzystający z wykazu (tablicy asocjacyjnej), w którym sekwencja obserwacji pełni rolę klucza dla wyboru właściwej akcji agenta (Tab. 1-2).

Tabl. 1-2. Tablica działań agenta w postaci wykazu asocjacyjnego

Percepcja	Akcja
[*A, czysto*]	*Prawo*
[*A, kurz*]	*Odkurzaj*
[*B, czysto*]	*Lewo*
[*B, kurz*]	*Odkurzaj*
[*A, czysto*], [*B, czysto*]	*NicNieRób*
[*A, czysto*], [*B, brudno*]	*Odkurzaj*
...	...

Tabl. 1-3. Program agenta z wykazem (tablicą asocjacyjną)

```
function AgentZWykazem(obserwacja) returns akcja
{ static:
   sekwObserwacji, // obserwacje, początkowo puste
   wykaz; // tablica asocjacyjna indeksowana sekwencją obserwacji, całkowicie wypełniona
```

```
        sekwObserwacji += obserwacja;
        akcja ← POBIERZ(sekwObserwacji, wykaz); // pobierz właściwą akcję z wykazu
        return akcja;
}
```

W praktyce, program agenta nie powinien sprowadzać się do przeglądania wykazu, przynajmniej z dwóch powodów. Po pierwsze dla rzeczywistych problemów będą potrzebne duże tablice - potrzebny będzie długi czas dla wypełniania i dostępu do tablicy. Po drugie, w zasadzie nie będzie to autonomiczny agent, gdyż brak jest mechanizmu uczenia się wiedzy przyjmującej postać wykazu.

W takiej sytuacji lepszym rozwiązaniem będzie „agent z pamięcią", w której magazynowana będzie przetworzona informacja o wszystkich obserwacjach. **Agenta z pamięcią** opiszemy funkcją odwzorowującą skumulowane obserwacje w akcje (Tab. 1-4).

Tabl. 1-4. Funkcja agenta z pamięcią.

```
function AgentZPamięcią(obserwacja) returns akcja
{ static: pamiec, // skumulowane obserwacje agenta
    pamiec ← MODYFIKUJ_PAMIEC(pamiec,obserwacja);
    akcja ← WYBIERZ_AKCJE(pamiec);
    pamiec ← MODYFIKUJ_PAMIEC(pamiec,akcja);
    return akcja;
}
```

Racjonalny agent

Jedna z możliwych funkcji (lub nieduża klasa równoważnościowa funkcji) agenta jest **racjonalna**. Idealnie racjonalny agent w oparciu o dany ciąg obserwacji i wiedzę o środowisku, w którym działa: powinien wykonać działanie, prowadzące do maksymalizacji **miary skuteczności**.

Miara skuteczności (użyteczności) to pewne obiektywne kryterium oceny stopnia sukcesu w zachowaniu agenta. Np. dla „agenta odkurzającego" miarami skuteczności mogą być: usunięta masa zanieczyszczeń, czas pracy, ilość pobranej energii, ilość wygenerowanego szumu, itd.

Racjonalność nie oznacza „nieomylność", gdyż nie mamy pełnej (nieskończonej) wiedzy o wszystkim. Stąd racjonalność powinna być wspierana przez eksplorację środowiska i uczenie się. Agenty mogą wykonywać akcje po to, aby zmodyfikować przyszłe obserwacje i w ten sposób pobrać pożyteczną informację (**eksploracja środowiska**). Agent jest autonomiczny wtedy, gdy na jego zachowanie wpływa własne doświadczenie (ma zdolności **uczenia się** i adaptowania do środowiska), np. poprzez samodzielne nabywanie wiedzy i uczenie się reguł wyboru akcji.

Struktura agenta

Jakie zagadnienia uwzględniamy podczas projektowania agenta:

1. miara skuteczności,
2. środowisko,
3. efektory,
4. czujniki.

Np. gdy naszym celem jest zaprojektowanie automatycznej taksówki to zapewne będziemy rozpatrywać następujące zagadnienia:

- miara skuteczności: bezpieczny, szybki, zarejestrowany, komfortowa podróż, maksymalny zysk;
- środowisko: drogi, inne agenty, piesi, klienci;
- efektory: kierownica, przyspieszenie, hamulce, sygnalizacja, sygnał dźwiękowy;
- czujniki: kamery, sonar, prędkościomierz, GPS, odometria, czujniki silnika, klawiatura.

Tabl. 1-5. Przykłady racjonalnych agentów

Typ	Percepcja	Akcje	Miara skuteczności	Środowisko
Diagnostyka medyczna	Objawy, badania, odpowiedzi pacjenta	Pytania, testy, leczenie	Zdrowie pacjenta, minimalizacja kosztów	Pacjent, szpital
Analiza obrazów satelitarnych	Piksele kolorowe obrazu	Klasyfikacja elementów sceny	Prawidłowa klasyfikacja	Obrazy z satelity
Manipulator sortujący	Obraz, sygnał czujnika	Uchwyć część, sortuj do pojemników	Części w prawidłowych pojemnikach	Taśma produkcyjna z częściami
Sterownik przemysłowy w rafinerii	Temperatura, ciśnienie	Otwórz/zamknij zawory, dopasuj temperaturę	Maksymalizacja stabilności i bezpieczeństwa	Rafineria
Zdalna nauka języka	Słowa pisane	Ćwiczenia, dyskusja, poprawianie	Maksymalizacja punktów z testu	Grupa studentów

1.2. Środowisko

Własności środowiska w znacznym stopniu wpływają na projekt agenta. Dlatego spróbujemy podać kilka najważniejszych kryteriów dla charakteryzowania środowisk.

- W pełni **obserwowalne** lub częściowo obserwowalne

Środowisko jest w pełni obserwowalne, gdy czujniki agenta dostarczają pełnej informacji o stanie środowiska w każdej chwili czasu. Z zasady nie rozpatrujemy przypadku, gdy brak jest jakiejkolwiek możliwości obserwacji środowiska przez agenta.

- **Deterministyczne** lub niedeterministyczne (stochastyczne)

Determinizm środowiska oznacza, że następny stan środowiska jest w pełni (jednoznacznie) określony przez aktualny stan i akcję wykonywaną przez agenta. Jeśli środowisko jest deterministyczne za wyjątkiem akcji innych agentów to mówimy, że jest ono **strategiczne**.

- **Modularne** lub sekwencyjne

Wiedza agenta o środowisku składa się z atomowych "epizodów" - każdy epizod składa się z pojedynczej obserwacji agenta i następującej po niej sekwencji akcji agenta, czyli wybór akcji w każdym epizodzie zależy jedynie od samego (całościowego) epizodu.

- **Statyczne** lub dynamiczne

Statyczne środowisko nie ulega zmianie podczas zastanawiania się agenta – w czasie pomiędzy obserwacją a wykonaniem akcji. Pół-dynamiczne środowisko wprawdzie nie zmienia się w tym czasie ale zmienia się wynik funkcji skuteczności dla akcji agenta.

- **Dyskretne** lub ciągłe

W dyskretnym przypadku występuje ograniczenie liczby możliwych obserwacji i akcji (skończone liczby).

- **O pojedynczym agencie** lub wieloagentowe

Pojedynczy agent operuje w środowisku lub jest ich wiele.

Przykłady środowisk

Agent grający w szachy z zegarem działa w środowisku, które jest: w pełni obserwowalne, strategiczne, sekwencyjne, pół-dynamiczne, dyskretne i wieloagentowe.

Agent grający w szachy bez zegara nie musi uwzględniać upływu czasu i dlatego jego środowisko różni się od poprzedniego tylko tym, że jest statycze.

Realny świat, jakim jest środowisko agenta - kierowcy taksówki, jest: częściowo obserwowalny, niedeterministyczny, niemodularny, dynamiczny, ciągły i wieloagentowy

Do generacji obserwacji agenta-softbota w zadanym środowisku posłużymy się w generatorem środowiska. Jego działanie przedstawia procedura w tablicy 1-6.

Tabl. 1-6. Procedura symulatora środowiska

```
procedure GenerujŚrodowisko(stan,ModFun,agenty,koniec)
/* Parametry: stan – początkowy stan środowiska, ModFun – funkcja modyfikująca stan środowiska, agenty – zbiór agentów, koniec – predykat, warunek zakończenia symulacji */
  do
    foreach (agent in agenty)
        Obserwacje[agent] ← pobierzObserwacje(agent,stan);
    foreach (agent in agenty)
        Akcje[agent] ← Program[agent](Obserwacje[agent];
    stan ←ModFun(Akcje,agenty,stan);
  while koniec(stan);
}
```

1.3. Rodzaje agentów

Wyróżnimy tu następujące rodzaje agentów (w porządku o rosnącej ogólności):

A. Prosty agent reaktywny;

B. Agent reaktywny ze stanem;

C. Agent realizujący cel (przeszukiwanie, planowanie)

D. Agent racjonalny (optymalny);

U. Agent ze zdolnością uczenia.

A. Prosty agent reaktywny

Taki agent reaguje wyłącznie na bieżącą obserwację, gdyż nie przechowuje on informacji o aktualnym stanie świata ani o przeszłych obserwacjach. Agent stosuje reguły o postaci:

warunek → *akcja*.

Np. if *światło_czerwone* then *stój*.

Tabl. 1-7. Funkcja prostego agenta reaktywnego.

```
function ProstyAgentReaktywny(obserwacja) returns akcja
{ static:  reguły; // zbiór reguł „warunek→akcja”

  stan ← ANALIZA_OBSERWACJI(obserwacja);
  reguła ← DOPASUJ_REGUŁĘ(stan,reguły);
  akcja ← AKCJA[reguła];
  return akcja;
}
```

B. Agent reaktywny ze stanem

Reaguje na bieżącą obserwację w oparciu o aktualny stan środowiska (tzn. opis środowiska dostępny mu na podstawie wszystkich obserwacji).

Tabl. 1-8. Funkcja agenta reaktywnego ze stanem.

```
function AgentReaktywnyZeStanem(obserwacja) returns akcja
{ static:  stan, // opis aktualnego stanu środowiska
          reguły; // zbiór reguł „warunek→akcja”
  stan ← MODYFIKUJ_STAN(stan,obserwacja);
  reguła ← DOPASUJ_REGUŁĘ(stan,reguły);
  akcja ← AKCJA[reguła];
  stan ← MODYFIKUJ_STAN(stan,akcja);
  return akcja;
}
```

C. Agent realizujący cel

Agent realizujący cel podejmuje akcje w oparciu o stan środowiska i zadany z góry cel. Problem agenta zwykle reprezentujemy łącznie w postaci:

- **celu** (inaczej warunków zatrzymania - stopu),
- przestrzeni **stanów** problemu i
- zbioru wykonywanych **akcji** odpowiadających operacjom przejścia pomiędzy stanami w powyższej przestrzeni.

Wyróżnimy dwie główne strategie realizacji celu:

1. akcje wybierane są w wyniku **przeszukiwania** przestrzeni stanów problemu;
2. poprzez konstruowanie **planu** (rozumianego jako sekwencja akcji), w szczególności w wyniku procesu przeszukiwania przestrzeni planów dla (pod-)problemu.

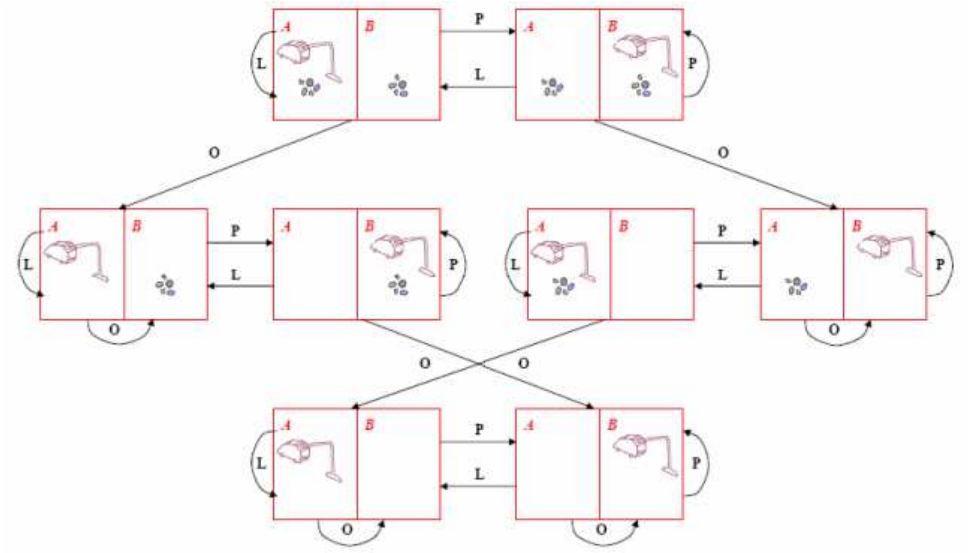
Tabl. 1-9. Funkcja agenta realizującego cel.

```
function AgentRealizującyCel(obserwacja) returns akcja
{ static:  sekwAkcji, // sekwencja akcji, początkowo pusta
           stan, // opis aktualnego stanu środowiska
           cel, // cel agenta, początkowo zerowy
           problem; // sformułowanie (reprezentacja) problemu
  stan ← MODYFIKUJ_STAN(stan,obserwacja);
  if (sekwAkcji jest pusta)
  { cel ← WYZNACZ_CEL(stan);
    problem ← WYZNACZ_PROBLEM(stan, cel);
    sekwAkcji ← PRZESZUKIWANIE(problem);
  }
  akcja ← PIERWSZA(sekwAkcji);
  sekwAkcji ← RESZTA(sekwAkcji);
  return akcja;
}
```

Przeszukiwanie przestrzeni stanów problemu oznacza, że system agentowy wybiera akcje (podejmuje decyzje) w oparciu o aktualny stan problemu i zadany cel. Sposób wyboru zależy od konkretnej strategii przeszukiwania, realizowanej przez agenta.

Przykład: przestrzeń stanów dla problemu „agenta odkurzającego" (rys. 1.3)

Stany wyznaczone są łącznie przez stopnie zabrudzenia obu kratek i pozycję odkurzacza. Wystarczą nam trzy akcje: [*w lewo* (*L*)*, w prawo* (*P*), odkurzanie *(O)*]. Poprzednio wymienioną akcję „NicNieRób" reprezentujemy poprzez akcję „w lewo" będąc w kratce A lub akcję „w prawo" będąc w kratce B. W obu przypadkach efekt jest taki sam, jak efekt akcji „NicNieRób". Zmniejszenie liczby akcji do 3 nie wpływa na liczbę stanów problemu. Warunek stopu to: „obie kratki są czyste (brak brudu)". Niech koszt akcji wynosi zawsze 1 za każdą akcję.



Rys. 1.3: Agent-odkurzacz i stany środowiska.

Przeszukiwanie przestrzeni stanów a planowanie

Na potrzeby przeszukiwania reprezentacja stanów jest zwykle prostą strukturą danych, zawierającą dane potrzebne jedynie dla *generacji następnika, funkcji oceny* i *warunku stopu.* Poza tym z punktu widzenia strategii przeszukiwania stan ma zwykle charakter „czarnej skrzynki". Reprezentacja akcji ma postać procedury, która generuje następne stany problemu. Reprezentacja celu jest to zwykle pośrednia informacja o celu, wyrażona poprzez warunek stopu i funkcję oceny kosztów resztkowych (heurystykę). Takie reprezentacje problemu nie dają agentowi żadnych dalszych wskazówek co do wyboru akcji, np. zależnych od analizy rzeczywistego stanu środowiska.

Planowanie ma na celu „usprawnienie" przeszukiwania przez „otwarcie" reprezentacji stanów, celów i akcji. Reprezentacja planu jest to sekwencja operatorów w przestrzeni planów, która:

(a) rozszerza plan początkowy (prosty, niepełny plan) do końcowego; lub

(b) modyfikuje pełny, ale błędny plan do końcowego.

Algorytmy planujące korzystają z formalnych języków, zwykle logiki predykatów lub podzbiorów tego języka, do opisu stanów, celów i akcji. Stany i cel są reprezentowane poprzez zbiory formuł, które są spełnione w danym stanie (względnie stanie docelowym).

1) Akcje są charakteryzowane poprzez logiczne opisy warunków wykonania i efektów akcji. Pozwala to na określenie bezpośrednich związków pomiędzy stanami a akcjami i na pominięcie bezzasadnych akcji w danym stanie.

2) Akcje są dodawane do planu w dowolnej kolejności wtedy, gdy są potrzebne, a nie w wyniku sztywnej sekwencji prowadzącej od stanu początkowego do końcowego (wykonując *oczywiste* i *ważne* decyzje w pierwszej kolejności, prowadzimy najpierw do ograniczenia przestrzeni rozwiązania a następnie stopniowo uszczegóławiamy plan).

3) Problem jest często formułowany jako koniunkcja pod-problemów. Możemy wtedy wyróżnić podcele i wyznaczać osobne plany dla osiągnięcia kolejnych podcelów.

Dzięki powyższym rozwiązaniom planowanie w porównaniu do przeszukiwania przestrzeni stanów prowadzi do znacznego zmniejszenia alternatywnych akcji i dowolności wyboru akcji.

D. Agent racjonalny (optymalny)

Agent racjonalny realizuje zadany cel w sposób optymalny. Dlatego dla agenta realizującego cel dodatkowo zdefiniowana jest funkcja użyteczności stanu w przestrzeni problemu, która określa

stopień spełnienia celu przez dany stan. Tym samym agent może uzależnić wybór akcji od wartości użyteczności stanu problemu, do którego wykonanie tej akcji go doprowadzi.

Projekt racjonalnego agenta zależy głównie od tego, czy: środowisko jest w pełni obserwowalne lub nie, oraz czy jest ono deterministyczne lub nie. Stąd wynikają szczegółowe rodzaje agentów w ramach kategorii D:

D1. Agent działa w pełni obserwowalnym i deterministycznym środowisku;

D2. Agent działa w deterministycznym, ale nie w pełni obserwowalnym środowisku;

D3. Agent działa w niedeterministycznym środowisku.

D1: środowisko jest deterministyczne i w pełni obserwowalne.

Prowadzi to do prostego problemu jednostanowego - 1 stan problemu odpowiada zawsze 1 stanowi środowiska. Agent dokładnie zna stan przed i po wykonaniu operacji. Rozwiązaniem problemu jest z góry przewidywalna (warunkowana wykonaniem obserwacji) sekwencja akcji prowadząca do zadanego celu.

D2: środowisko jest deterministyczne ale nie w pełni obserwowalne - na skutek braku pewnych czujników.

Prowadzi to do problemu wielostanowego – niepewność agenta co do aktualnego stanu środowiska sprawia, że 1 stan problemu odpowiada wielu możliwym stanom środowiska. Agent nie ma pełnej wiedzy o stanie środowiska, ale zna wpływ wykonania swoich akcji na środowisko. Rozwiązaniem jest taka sekwencja akcji, która najpierw redukuje niepewność stanu środowiska a następnie wykonuje sekwencję akcji prowadzącą z rozpoznanego stanu środowiska do celu.

D3: środowisko jest niedeterministyczne.

Jest to tzw. problem ewentualności – występuje niepewność następnego stanu środowiska. Jednak obserwacje dostarczają nowych informacji o środowisku w następnym stanie. Rozwiązaniem problemu w takiej sytuacji może być dodatkowe warunkowanie wykonania akcji – zawsze uzależnienie akcji od stanu i aktualnej obserwacji.

U. Uczenie się

W potocznym znaczeniu pojęcie „uczenie się” jest często utożsamiane z eksploracją nieznanego środowiska , czyli z nabywaniem wiedzy o świecie dzięki obserwacjom. Jest to nieodzowny element działania agenta wtedy, gdy projektant systemu nie dysponuje pełną informacją o środowisku.

W systemie agentowym przez „uczenie się” rozumiemy uczenie się sposobu wyboru akcji, czyli uczenie funkcji decyzyjnej. Osiągamy to wystawiając system na oddziaływanie realnego środowiska, zamiast od razu w pełni specyfikować jego zasady działania. Zadaniem procesu uczenia jest zmiana mechanizmu decyzyjnego systemu w celu poprawy skuteczności jego działania. Aby rozwiązać problem uczenia też zastosujemy przeszukiwanie przestrzeni problemu i specyficzne funkcje użyteczności dla oceny skuteczności działania systemu po zmianie.

Projekt modułu uczącego zależy od tego jak reprezentujemy funkcję systemu i jaki jej element jest modyfikowany, a także jaki rodzaj sprzężenia zwrotnego stosujemy podczas uczenia. O wybranych funkcjach decyzyjnych i ich procedurach uczenia powiemy później. Natomiast sprzężenie zwrotne przyjmuje zwykle jedną z trzech poniższych postaci:

1. Uczenie z nadzorem (*supervised learning*): istnieją prawidłowe (wzorcowe) odpowiedzi dla każdego przykładu uczącego;
2. Uczenie ze wzmacnianiem (*reinforcement learning*): brak wzorcowej odpowiedzi ale istnieje krytyk nagradzający ”dobre” akcje (zbliżające agenta do prawidłowego stanu);
3. Uczenie bez nadzoru (*unsupervised learning*): brak wzorcowych odpowiedzi, brak krytyka.

1.4. System z bazą wiedzy

Z punktu widzenia informatyki systemy agentowe są realizowane w technologii **systemu z bazą wiedzy**. Głównym elementem takiego systemu jest baza wiedzy (ang. „*knowledge base*", KB), która zawiera zbiór **aksjomatów i faktów** o modelowanym świecie, wyrażony w języku reprezentacji wiedzy, oraz odpowiednie dla tego języka **reguły wnioskowania**. Z punktu widzenia logiki matematycznej aksjomaty i fakty mają postać formuł zapisanych w języku reprezentacji wiedzy a reguły wnioskowania są **tautologiami** tego języka. Tautologia jest formułą zawsze prawdziwą w danym języku. Prawdziwość aksjomatów ograniczona jest do konkretnej dziedziny zastosowania systemu, natomiast prawdziwość formuł wyrażających fakty zależy od obserwacji aktualnego środowiska, od tego czy dane fakty zostały zaobserwowane (lub wywnioskowane z innych faktów).

System agentowy rozpoczyna pracę z początkowym stanem bazy wiedzy (modelem dziedziny zawartym w KB). W wyniku obserwacji dodawane są nowe dane, wyrażające fakty prawdziwe w aktualnym środowisku. Z obserwowanych faktów i aksjomatów dziedziny sterowanie systemu może generować (wnioskować) dalsze ukryte fakty.

Omówimy teraz pokrótce podstawowe **języki reprezentacji wiedzy** w dwóch ważnych aspektach: ontologii i epistemologii. **Ontologia** języka wskazuje na to czemu odpowiadają symbole języka w rzeczywistym świecie. **Epistemologia** języka wyraża to, jak odbierane (oceniane) są te elementy świata - jakie wartości przyjmują jednostki wiedzy. W tabeli 1-10 zebranych jest 5 istotnych języków. W niniejszym wykładzie szczegółowo omówimy projektowanie systemu agentowego wtedy, gdy baza wiedzy wyrażona jest w języku predykatów lub w języku probabilistyki.

Tabl. 1-10. Ontologia i epistemologia języka reprezentacji wiedzy.

Język	Ontologia	Epistemologia
Rachunek zdań	Fakty	true/false/unknown
Logika predykatów (1 rzędu)	Fakty, obiekty, relacje	true/false/unknown
Logika temporalna	Fakty, obiekty, relacje (wiedza niepełna)	true/false/unknown
Teoria probabilistyczna	Fakty (wiedza niepewna)	Rozkład prawdopodobieństwa
Logika rozmyta	Fakty (wiedza niedokładna)	Stopień przynależności <0, 1>

1.5. Logiczne agenty

Zaletą każdego **systemu logicznego** jest to, że udostępnia on język formalny do takiej reprezentacji informacji, z której można wyciągać wnioski. Dlatego też obok składni i semantyki języka logicznego, czyli typowych sposobów opisu każdego języka, w naturalny sposób dołączamy do niego mechanizmy wynikania i wnioskowania.

Składnia języka logiki L podaje reguły tworzenia poprawnych zdań języka (w logice predykatów nazywanych formułami). **Semantyka** języka L definiuje „znaczenie" zdań (formuł):

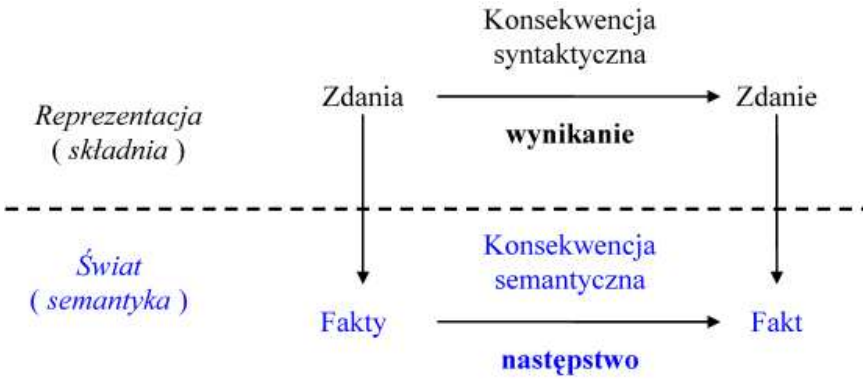
1. podaje ona znaczenie wszystkich symboli **X** języka L (czyli zawiera pewne odwzorowanie: **X**→ {elementy modelowanego świata}) i
2. sposób, w jaki zdaniom (formułom) można przypisać znaczenie, co z kolei pozwala określić ich wartość logiczną.

Dla przykładu język działań arytmetycznych jest językiem logiki. Zasady składni mówią np., że (x+2 ≥ y) jest zdaniem, a (x2+y > {}) nie jest zdaniem tego języka. Z kolei zasady semantyki mówią np., że:

- x+2 ≥ y jest prawdziwe wtw. liczba x+2 jest nie mniejsza niż liczba y;

- x+2 ≥ y jest prawdziwe w świecie, w którym x = 7, y = 1;
- x+2 ≥ y jest fałszywe w świecie, w którym x = 0, y = 6.

Wynikanie (ang. *entailment*) jest związkiem pomiędzy zdaniami. Mówimy, że „ze zbioru zdań X wynika zdanie A" i oznaczamy to jako: X ╞ A , wtedy gdy odzwierciedla to następstwo (*konsekwencję semantyczną*) odpowiadających tym symbolom faktów w modelowanym świecie. Zachodzi więc zależność pomiędzy elementami składni i semantyki języka, przedstawiona na rys. 1.4.



Rys. 1.4: Wynikanie zdań w języku logiki.

Semantyczną relację **następstwa** wyrazimy poprzez relację zawierania się zbiorów reprezentujących realizacje świata, zwane **modelami**. Modele w logice to formalnie zdefiniowane **światy**, względem których można określać to co jest **prawdziwe** a co **nie**. **Model** dla zbioru zdań **X** to każdy **świat**, w którym prawdziwe są wszystkie zdania ze zboru **X**.

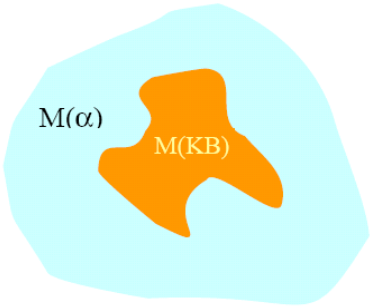
Możemy powiedzieć, że zdanie A **wynika** ze zbioru zdań **X**, co zaznaczamy **X** ╞ A , jeśli A jest **prawdziwe** w każdym modelu dla **X**. Odnosząc to stwierdzenie do bazy wiedzy, która zawiera jedynie zdania uznane za prawdziwe w obserwowanym środowisku (realizacji świata) powiemy, że z bazy wiedzy KB **wynika** zdanie α wtw. gdy α jest **prawdziwe** dla wszystkich **modeli zdań** zawartych w KB (oznaczamy, KB ╞ α).

Związek relacji wynikania (konsekwencji syntaktycznej) i następstwa (konsekwencji semantycznej) zapiszemy następująco (patrz rys. 1.5). Niech M(α) będzie zbiorem wszystkich modeli zdania α. Wtedy:

$$KB \models \alpha \text{ wtw. } M(KB) \subseteq M(\alpha).$$

Np. niech baza wiedzy KB zawiera zdania "Legia *wygrała mecz*" i "Wisła *wygrała mecz*". Z nich wynika zdanie: "Legia w*ygrała mecz lub Wisła wygrała mecz*", a to dlatego, że jest ono bardziej ogólne i jego model obejmuje modele obu poprzednich zdań.

Zdania uznane za **równoważne** w pewnej dziedzinie wiedzy są specyficznym przypadkiem wynikania. Np. ze zdania (x+y= 4) wynika, że (4 = x+y), a w *matematyce* są to zdania równoważne



Rys. 1.5: Konsekwencja semantyczna jako relacja zawierania się modeli zdań.

Celem **procedury wnioskowania** jest ujawnienie lub sprawdzenie zależności pomiędzy zdaniami języka, zwanej wynikaniem. Oczywiście interesują nas tylko takie procedury wnioskowania, które są **poprawne** i **zupełne** :

- Poprawność procedury wnioskowania

  Procedura wnioskowania Π jest poprawna wtedy i tylko wtedy (wtw.) gdy dla każdego zbioru zdań ***X*** i każdego zdania *A*: $\boldsymbol{X} \vdash_{\Pi} A$ pociąga za sobą $X \models A$.

  Innymi słowy, jeśli poprawna procedura wnioskowania wskazuje na istnienie relacji wynikania pomiędzy pewnymi zdaniami, to ona zawsze rzeczywiście zachodzi.

- Zupełność procedury wnioskowania

  Procedura wnioskowania Π jest zupełna wtw. gdy dla każdego zbioru zdań ***X*** i każdego zdania *A*: $\boldsymbol{X} \models A$ pociąga za sobą $\boldsymbol{X} \vdash_{\Pi} A$.

  Tym samym wskazujemy, że zupełna procedura wnioskowania to taka, która jest w stanie wykazać każde rzeczywiście istniejące wynikanie.

Podsumowując, system logiczny obejmuje język reprezentacji wiedzy (charakteryzowany poprzez składnię i semantykę) oraz odpowiedni mechanizm wnioskowania (dedukcji). W następnych rozdziałach zdefiniujemy dwa podstawowe systemy logiczne (*rachunek zdań*, *logika pierwszego rzędu*), które są wystarczająco „mocne", aby wyrazić większość interesujących nas rzeczy i dla których istnieją poprawne i zupełne procedury wnioskowania.

1.6. Pytania

1. Wyjaśnić pojęcie **agenta**.
2. Jakie wyróżniamy główne **rodzaje agentów** ?
3. Jak charakteryzujemy **środowiska działania** agentów?
4. Jakie są główne **sposoby uczenia** agenta?
5. Omówić pojęcia: **przeszukiwanie** i **planowanie**.
6. Wyjaśnić pojęcia: „**wynikanie"** i „**wnioskowanie formuł".** Zilustrować odpowiedź na przykładach ze świata matematyki.

1.7. Zadania

Zad. 1.1

Dla „agenta odkurzającego" (z pamięcią) wyznaczyć sekwencję akcji wiodącą do celu „wszędzie czysto" w trzech różnych sytuacjach:

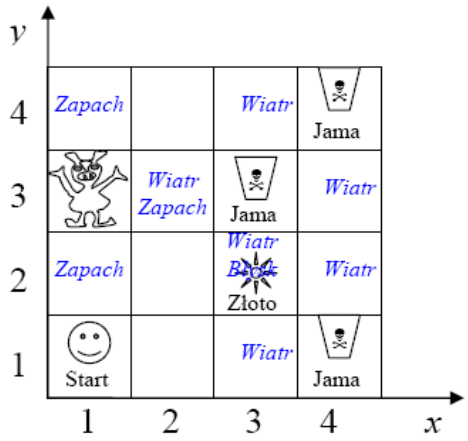
1) Środowisko jest deterministyczne i w pełni obserwowalne → problem jednostanowy (1 stan problemu to 1 stan środowiska).

2) Środowisko jest deterministyczne ale nie w pełni obserwowalne - na skutek braku pewnych czujników → problem wielostanowy – niepewność aktualnego stanu środowiska (1 stan problemu to wiele możliwych stanów środowiska).

3) Środowisko jest niedeterministyczne (problem ewentualności) – niepewność następnego stanu środowiska. Obserwacje dostarczają nowych informacji o środowisku.

Zad. 1.2

„*Świat Wumpusa*" to wczesna gra komputerowa (rys. 1.6). Celem agenta jest znalezienie złota i wydostanie się z jaskini (powrót do kwadratu startowego [1,1]). Miara skuteczności agenta: złoto: +1000 pkt., śmierć: -1.000 pkt, -1 pkt. za każdą akcję, -10 pkt. za skorzystanie z jedynej strzały. Nie zawsze agent może uzyskać dodatni wynik - świat może być „źle" zdefiniowany: złoto może być w jamie lub być niedostępne – otoczone *jamami* i/lub *Wumpusem*.

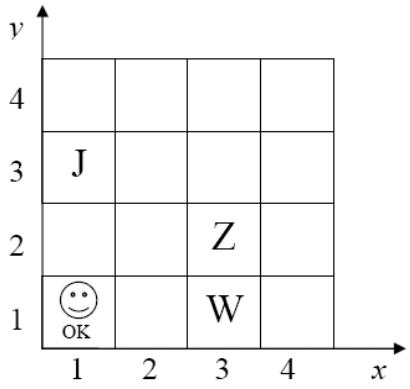


Rys. 1.6: Idea „świata Wumpusa".

Obserwacje tworzą wektor 5-elementowy: [zapach, **wiatr**, **błysk**, **uderzenie**, **krzyk**]. W kwadracie, gdzie mieszka Wumpus oraz w ściśle przylegających agent czuje zapach (*smród Wumpusa*); W kwadratach przylegających do jamy agent czuje wiatr. W kwadracie, gdzie znajduje się złoto agent obserwuje błysk; Jeśli agent wejdzie na ścianę to czuje uderzenie; Kiedy Wumpus zostaje zabity, w całej jaskini słychać krzyk.

Akcje: { *ObrótWLewo,* ObrótWPrawo*, RuchWPrzód, Podnieść, Upuścić, Strzelić, Wyjść, Zginąć* }. *„Strzelić"* - zużywa jedyną strzałę w kierunku patrzenia agenta; *„Podnieść"* złoto, jeśli jest w tej samym kwadracie; *„Upuścić"* złoto, pozostawiając je w tym samym kwadracie; „Wyjść" - pozwala agentowi opuścić jaskinię, o ile znajduje się on w kwadracie startowym. „Zginąć" - agent ginie jeśli wejdzie na kwadrat, w którym znajduje się *Wumpus* lub *jama*.

Przedstawić sekwencję obserwacji i akcji agenta oraz (stanu rozpoznania środowiska) wiodącą go do kratki ze złotem w poniższej wersji świata (rys. 1.7) (na początku zupełnie nieznanego agentowi). Początkowa obserwacja w kwadracie [*x*,*y*]=[1,1]: agent nie odczuwa zapachu ani wiatru.



Rys. 1.7: Stan początkowy „świata Wumpusa".

Oznaczenia: ☺ oznacza położenie agenta. OK – kratka jest „bezpieczna". **J** – jama, **W** – Wumpus, **Z** – złoto.

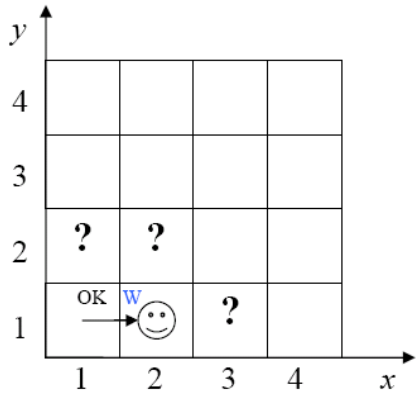
Zad. 1.3

W *świecie Wumpusa* załóżmy sytuację powstałą po:(1) pustej obserwacji w [1,1], ruchu w prawo do [2,1] i (2) obserwacji wiatru w [2,1].

Teraz agent chciałby wiedzieć, czy w sąsiednich kratkach: [1,2], [2,2] i [3,1], występują jamy. Sprawdzić wynikanie, lub nie, poniższych zdań z aktualnej bazy wiedzy, poprzez sprawdzenie zawierania się modeli, dla zdań:

A) α1= „w kratce [1,2] nie ma jamy”

B) α2= „w kratce [2,2] nie ma jamy”



Rys. 1.8: Stan „świata Wumpusa” po 2 obserwacjach.

2. Logika klasyczna

- Inżynieria wiedzy
- Języki logiki
- Rachunek sytuacyjny
- Logiczne systemy wnioskowania
- Ontologia języka
- Pytania

2.1. Inżynieria wiedzy

Inżynieria wiedzy zajmuje się metodologią konstruowania i korzystania z systemów z bazą wiedzy. Jednym z jej przejawów jest „programowanie w logice", uznawane też za jeden z 4 głównych paradygmatów programowania. Poniższa tabelka 2-1 pokazuje zasadnicze różnice pomiędzy inżynierią oprogramowania a inżynierią wiedzy.

Tabl. 2-1. Porównanie inżynierii oprogramowania i inżynierii wiedzy.

Inżynieria oprogramowania	Inżynieria wiedzy
Języki programowania	Języki reprezentacji wiedzy
Dane	Bazy wiedzy
Operacje	Procedury wnioskowania (dowodzenia zdań)
Wykonanie programu	Wnioskowanie (dowodzenie zdania zapytania)

Zgodnie z zasadami inżynierii w procesie tworzenia bazy wiedzy, wyrażonej w języku logiki, wyróżnimy następujące etapy:

1. Identyfikacja zadania.
2. Zebranie odpowiedniej wiedzy o dziedzinie (jakie obiekty i fakty modelowanej dziedziny należy reprezentować w systemie a jakie są zbędne).
3. Wybór słownika dla relacji, funkcji i stałych (ontologia).
4. Zakodować ogólną wiedzę o dziedzinie (aksjomaty).
5. Zakodować opis specyficznego zadania (formuły atomowe).
6. Testowanie - wysyłać zapytania do procedury wnioskowania i odbierać jej odpowiedzi.
7. Przeanalizować wyniki testowania i ewentualnie poprawiać bazę wiedzy .

System agentowy realizowany w technologii systemu z bazą wiedzy powinien posiadać następujące możliwości:

- reprezentować stany i akcje;

- integrować nowe obserwacje;
- modyfikować wewnętrzną reprezentację świata, tzn. wnioskować o ukrytych własnościach świata wynikających z obserwacji;
- wnioskować o odpowiednich akcjach agenta.

Zwykle przyjmujemy, że sterowanie agenta komunikuje się z bazą wiedzy (KB) używając dwóch operacji TELL i ASK:

TELL: Agent → KB (powiedz bazie o nowych obserwacjach / faktach),

ASK : KB → Agent (zapytaj się bazy co robić, sama treść zapytania też może pochodzić z KB).

Działanie agenta dysponującego bazą wiedzy przedstawia tabl. 2-1. Podany tam program odwołuje się do bazy wiedzy za pomocą funkcji TELL i ASK, przekazując im odpowiednie zdania wyrażone w języku reprezentacji wiedzy tej bazy. Generalnie są to zdania trzech rodzajów, tworzone przez poniższe podfunkcje:

- UtwórzZdanieObserwacji() – pobiera obserwację i indeks czasu a następnie generuje zdanie w języku bazy wiedzy reprezentujące obserwację w danej chwili czasu.
- UtwórzZapytanieAkcji() – pobiera indeks czasu i generuje zdanie w języku bazy wiedzy będące zapytaniem o to, jaką akcję należy wykonać.
- UtwórzZdanieAkcji() – generuje zdanie w języku bazy wiedzy reprezentujące wykonaną akcję w danej chwili czasu.

Tabl. 2-2. Program agenta w systemie z bazą wiedzy

```
function AgentZBaząWiedzy(obserwacja) zwraca akcję
{ static:       KB, // baza wiedzy
                t;  // licznik (indeks czasu) – początkowo wynosi 1
        TELL(KB, UtwórzZdanieObserwacji(obserwacja,t));
        akcja ← ASK(KB, UtwórzZapytanieAkcji(t));
        TELL(KB, UtwórzZdanieAkcji(akcja, t));
        t ← t+1;
        return akcja;
}
```

2.2. Języki logiki

Prezentację podstawowego języka logiki, jakim jest logika predykatów, rozpoczniemy od jej prostego podzbioru, zwanego **rachunkiem zdań**.

2.2.1. Rachunek zdań

W skład alfabetu rachunku zdań wchodzą:

- stałe logiczne *True* i *False*,
- symbole zdaniowe (*P*, *Q*, *R*,...),

- spójniki logiczne (koniunkcja, alternatywa, implikacja, równoważność, negacja) $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$
- nawiasy okrągłe **(**, **)**.

Z liter alfabetu tworzone są **zdania** za pomocą następujących zasad:

- *True* , *False* i symbole zdaniowe są zdaniami atomowymi.
- Jeśli *A* i *B* są zdaniami to ($A \wedge B$), ($A \vee B$), ($A \Rightarrow B$) i ($A \Leftrightarrow B$) też są zdaniami języka.
- Jeśli *A* jest zdaniem to $\neg A$ też jest zdaniem.

Mianem **literału** określamy zdanie atomowe lub negację zdania atomowego.

Prostota rachunku zdań sprawia, że posiada on małą siłę wyrazu – każdy kolejny fakt musi być reprezentowany oddzielnym symbolem języka. Dla przykładu zastanówmy się jak reprezentować w rachunku zdań akcje agenta „*świata Wumpusa*" [4] wykonywane w określonym czasie. Wprowadzamy symbole $L_{\mathrm{i,j}}$ dla oznaczenia, że agent działający w 2-wymiarowym środowisku znajduje się w kratce o współrzędnych $[\mathrm{i,j}]$. Wtedy możliwym akcjom odpowiadałyby zdania w rodzaju:

$$L_{1,1} \wedge ZwróconyWPrawo \wedge RuchWPrzód \Rightarrow L_{2,1}$$

Jednak to nie prowadzi do prawidłowego wnioskowania. Po wykonaniu akcji oba zdania $L_{1,1}$ i $L_{2,1}$ będą w bazie danych uważane za prawidłowe, tymczasem już tak nie jest, gdyż świat zmienia się wraz z upływem czasu. Jak reprezentować te zmiany w rachunku zdań? Jedynym sposobem jest odpowiednie indeksowanie symboli. Np.:

$$L^{1}_{1,1} \wedge ZwróconyWPrawo^{1} \wedge RuchWPrzód^{1} \Rightarrow L^{2}_{2,1}$$

$$ZwróconyWPrawo^{1} \wedge ObrótWLewo^{1} \Rightarrow ZwróconyWGórę^{2}$$

Powyższy przykład ilustruje ograniczenia w sile wyrazu właściwe dla rachunku zdań. Baza wiedzy musi zawierać liczne formuły o podobnej formie różniące się „indeksami", gdzie każdy zbiór „indeksów" identyfikuje inne „fizyczne" miejsce świata lub czas (np. kratkę w „świecie Wumpusa" w pewnej chwili czasu). Nie ma możliwości wyrażenia w zwarty sposób wspólnej własności wszystkich „fizycznych" miejsc.

Podobnie jest z reprezentacją możliwych akcji agenta - musimy wprowadzić osobne symbole i zdania dla każdej chwili czasu t i każdego miejsca $[\mathrm{x,y}]$. W „świecie Wumpusa" dla każdego kierunku, każdej kratki i czasu musiałaby istnieć formuły o postaci

$$L^{t}_{\mathrm{x,y}} \wedge ZwróconyWPrawo^{t} \wedge RuchWPrzód^{t} \Rightarrow L^{t+1}_{\mathrm{x+1,y}}$$

Efektem jest „eksplozja" liczby zdań w bazie wiedzy przy rosnącym rozmiarze świata i liczbie wykonanych akcji.

2.2.2. Logika predykatów – logika pierwszego rzędu

Podczas gdy rachunek zdań zakłada, że świat składa się z faktów, **logika pierwszego rzędu** (logika predykatów), podobnie jak język naturalny, pozwala specyfikować te fakty znacznie dokładniej. W logice predykatów zakładamy, że w świecie występują:

1. Obiekty

   Np. osoby, domy, liczby, kolory, gry, wojny, …

2. Relacje

   Np. jest czerwony, jest okrągły, jest liczbą pierwszą, jest bratem, większy niż, jest częścią, jest pomiędzy, …

3. Funkcje

   Np. jego ojciec, jego najlepszy przyjaciel, o jeden więcej, suma, …

Pozwala to wyrazić fakty jako relacje zachodzące na obiektach wyznaczanych przez funkcje. Stąd składnia języka predykatów obejmuje następujące zasadnicze elementy:

- Stałe np. *KrólJan, 2, PW*,...
- Symbole predykatów np. *Brat, >,...*
- Symbole funkcji np. *Sqrt, LewaNoga*,...
- Zmienne *x, y, a, b*,...
- Negacja i spójniki $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- Predykat równości =
- Kwantyfikatory (dla wyrażenia własności zbioru obiektów) $\forall, \exists$

Z powyższych elementów tworzone są wyrażenia języka: termy i formuły. **Termy** wskazują na obiekty i są ogólnej postaci funkcji, stałej lub zmiennej:

funkcja (*term*$_1$,...,*term*$_n$)

lub *stała* lub *zmienna*

Formuła atomowa to wyrażenie zbudowane na pojedynczym predykacie, tzn. o ogólnej postaci:

predykat (*term*$_1$,...,*term*$_n$)

lub

term$_1$ = *term*$_2$

Predykat „=" („równość") został wyróżniony dlatego, gdyż we wszystkich zastosowaniach języka powinien posiadać podobne znaczenie.

Przykłady:

Term: *Brat*(*Jan*)*;*

Formuła atomowa*: >(Długość(LewaNoga(Ryszard)), Długość(LewaNoga(Jan)))*

Formuły złożone powstają z połączenia formuł atomowych spójnikami z możliwością wykorzystania kwantyfikatorów i negacji:

$\neg S, \quad S_1 \wedge S_2, \quad S_1 \vee S_2, \quad S_1 \Rightarrow S_2, \quad S_1 \Leftrightarrow S_2,$

Przykłady formuł

Rodzeństwo(*Jan, Ryszard*) $\Rightarrow$ *Rodzeństwo*(*Ryszard, Jan*)

$>(1,2) \vee \leq (1,2)$

$>(1,2) \wedge \neg >(1,2)$

2.2.3. Semantyka rachunku zdań

Semantyka rachunku zdań może być formalnie ujęta jako struktura algebraiczna: <D, φ >. **Dziedzina** D to zbiór faktów, do których odnoszą się symbole języka. **Interpretacja** φ to funkcja przyporządkowująca symbolom zdaniowym fakty należące do dziedziny i wartościująca zdania jako *Prawdę* (*True*) lub *Fałsz* (*False*). Wartościowanie zdań atomowych zależy od prawdziwości reprezentowanego faktu w aktualnym świecie a przy wartościowaniu zdań złożonych korzystamy z tabelki prawdy dla spójników (0 oznacza *Fałsz* a 1 – *Prawdę*) (rys. 2.1).

Model zdania A (oznaczmy go jako M(A) = <D, φ >) jest wyznaczony przez każdą interpretację φ dla dziedziny D, w której zdanie A jest prawdziwe. **Tautologia** w dziedzinie D jest to zdanie prawdziwe w każdej interpretacji (modelu) dla D.

Rachunek zdań jest **rozstrzygalny**, tzn. istnieje algorytm, który dla dowolnego zdania A stwierdza, czy A jest tautologią. Takim algorytmem jest chociażby sprawdzanie tabelki prawdy dla zdania. Ponieważ każde zdanie zawiera skończoną liczbę symboli i spójników, więc tabelka będzie miała skończony rozmiar co umożliwi sprawdzenie wszystkich wierszy.

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Rys. 2.1: Tabelka prawdy dla spójników logicznych

Dla przykładu, dzięki tabeli prawdy pokazujemy, że zdanie złożone, $((P \vee H) \wedge \neg H) \Rightarrow P$, jest tautologią, tzn. zawsze prawdziwą niezależnie od modelu dla *P* i *H* (rys. 2.2 - zauważmy, że ostatnia kolumna jest zawsze prawdziwa).

P	H	P∨H	(P∨H)∧¬H	((P∨H)∧¬H)⇒P
0	0	0	0	1
0	1	1	0	1
1	0	1	1	1
1	1	1	0	1

Rys. 2.2: Sprawdzanie tautologii przy pomocy tabelki prawdy.

Procedura sprawdzająca wszystkie modele jest procedurą wnioskowania **poprawną** i **zupełną**. Może ona zostać efektywnie zaimplementowana wtedy, gdy formuły dają się uporządkować (np. odpowiednio do odległości miejsca od pozycji startowej agenta, do którego te formuły się odnoszą). Np. funkcja WNIOSKUJTABPRAWDY() (rys. 2.3) jest efektywną implementacją procedury wnioskowania. Korzysta ona z rekursywnej funkcji TABPRAWDY(), wywoływanej dla częściowego i stopniowo rozszerzanego modelu. Podfunkcja CZYMODEL() sprawdza poprawność zdania (lub bazy wiedzy) w częściowym modelu a podfunkcja ROZSZERZ() rozszerza model o wartość dla kolejnego symbolu (zdania).

Tabl. 2-3. Procedura wnioskowania polegająca na sprawdzaniu tabelki prawdy.

funkcja WNIOSKUJZTABPRAWDY(*KB*, α)

zwraca wynik: true lub **false**

{ *symbole* ← *symbole zdaniowe w KB* i α *;*

return TABPRAWDY(*KB*, α , *symbole*, []*);*

}

```
funkcja TABPRAWDY(KB, α, symbole, model)
zwraca wynik: true lub false
{ if (symbole == ∅ ) {
        if CZYMODEL(KB, model) return CZYMODEL(α, model);
        else return true;
  } else {
    P ← Pierwszy(symbole); reszta ← Reszta(symbole);
    return (TABPRAWDY(KB, α, reszta, Rozszerz(P, true, model)
      && TABPRAWDY(KB, α, reszta, Rozszerz(P, false,model)); }
}
```

2.2.4. Semantyka języka predykatów

W przypadku języka predykatów znaczenie formuły określamy ze względu na: ustaloną dziedzinę *D,* funkcję interpretacji *m* i wartościowanie a. Model zbioru formuł nadal jest postaci, *M* = [*D*, *m*], ale aby wyznaczyć wartość formuły należy tez wartościować jej zmienne, a to wymaga podania funkcji wartościowania *a*.

Dziedzina *D* zawiera obiekty (elementy dziedziny) i relacje oraz funkcje pomiędzy nimi.

Funkcja interpretacji *m* przyporządkowuje:

- symbolom stałych → obiekty, czyli elementy dziedziny

- *n*-arg. symbolom predykatów → *n*-argumentowe relacje określone na D^n

- n-arg. symbolom funkcyjnym → funkcje z D^n na *D*

Formuła atomowa o postaci *predykat*($term_1$,...,$term_n$) *jest prawdziwa* (*True*) w interpretacji *m* wtw. gdy obiekty wyznaczone przez $term_1$,...,$term_n$ spełniają relację referowaną przez *predykat.*

W jaki sposób wyznaczamy obiekt będący wartością termu? Przy interpretowaniu formuły w danym modelu nadawanie wartości zmiennym tej formuły określa się mianem **wartościowania**. Wartościowanie zmiennych a w zadanym modelu M to odwzorowanie symboli zmiennych na elementy dziedziny.

Oznaczmy przez $V_a^M(t)$ wartość termu *t* w modelu $M = [D, m]$ względem wartościowania a. Wartość stałej lub zmiennej obliczamy jako:

- $V_a^M(x) = a(x)$, gdzie x jest zmienną.
- $V_a^M(C) = m(C)$, gdzie C jest stałą.

Wartość termu $f(t_1,...,t_n)$ wynosi:

- $V_a^M(f(t_1,...,t_n)) = m(f)(V_a^M(t_1),..., V_a^M(t_n))$.

Oznaczmy przez $V_a^M(\tau)$ wartość formuły τ w modelu M, gdzie $M = [D, m]$, względem wartościowania a.

- Dla predykatu P: $V_a^M(P(t_1,...,t_n)) = m(P)(V_a^M(t_1),..., V_a^M(t_n))$.
- Dla równości: $V_a^M(t_1 = t_2)) = (V_a^M(t_1) = V_a^M(t_2))$.
- Dla formuł złożonych:

$$V_a^M(B \wedge C) = V_a^M(B) \wedge V_a^M(C)$$
$$V_a^M(B \vee C) = V_a^M(B) \vee V_a^M(C)$$
$$V_a^M(B \Rightarrow C) = V_a^M(B) \Rightarrow V_a^M(C)$$
$$V_a^M(B \Leftrightarrow C) = V_a^M(B) \Leftrightarrow V_a^M(C)$$

- Dla kwantyfikatorów:

$$V_a^M(\forall x\ B) = \min_{d \in D}[V_{a(x \leftarrow d)}^M(B)]$$
$$V_a^M(\exists x\ B) = \max_{d \in D}[V_{a(x \leftarrow d)}^M(B)]$$

gdzie *min*(True, False) wynosi False, a *max*(True, False) wynosi True.

Niech $a(x \leftarrow d)$ oznacza wartościowanie identyczne z a dla wszystkich zmiennych poza x, w którym zmiennej x nadawana jest wartość d. Czyli $a(x \leftarrow d)$ różni się od a co najwyżej wartościowaniem zmiennej x.

Uniwersalny kwantyfikator jest ogólnej postaci:

∀<*zmienne*> <formuły>

Np. Zdanie „*Każda osoba studiująca na PW jest inteligentna*” zapiszemy z użyciem kwantyfikatora jako: ∀*x* (*Studiuje*(*x*, PW) ⇒ *Inteligentna*(*x*)).

Wartościowanie formuły z kwantyfikatorem ogólnym: formuła ∀x *P* jest prawdziwa w modelu *M* wtw. gdy *P* jest prawdziwe dla *x* wartościowanego dowolnym obiektem w tym modelu. W przybliżeniu jest to równoważne koniunkcji wszystkich możliwych wartościowań *P:*

(*Studiuje*(Jan, PW) ⇒ *Inteligentna*(Jan)) ∧ (*Studiuje*(Ryszard, PW) ⇒ *Inteligentna*(Ryszard)) ∧ (*Studiuje*(PW, PW) ⇒ *Inteligentna*(PW)) ∧ ...

Egzystencjalny kwantyfikator jest ogólnej postaci

∃<*zmienne*> <formuły>

Np. Zdanie „*Ktoś spośród osób studiujących na PW jest inteligentny*” zapiszemy jako:

∃*x* (*Studiuje*(x, PW) ∧ *Inteligentna*(x))

Formuła ∃*x P* jest prawdziwa w modelu *M* wtw. gdy *P* jest prawdziwe dla *x* wartościowanego jakimś obiektem modelu. W przybliżeniu jest to równoważne alternatywie różnych wartościowań *P.* Np.

(*Studiuje*(Jan, PW) ∧ *Inteligentna*(Jan)) ∨ (*Studiuje*(Ryszard, PW) ∧ *Inteligentna*(Ryszard)) ∨ (*Studiuje*(PW, PW) ∧ *Inteligentna*(PW)) ∨ ...

Własności kwantyfikatorów:

- ∀x ∀y jest taka sama jak ∀y ∀x
- ∃x ∃y jest taka sama jak ∃y ∃x
- ∃x ∀y nie jest taka sama jak ∀y ∃x

Np. formuła, ∃x ∀y *Kocha*(x, y), oznaczająca “*Istnieje osoba, która kocha wszystkich w świecie*”, nie jest tożsama z formułą, ∀y ∃x *Kocha*(x,y), oznaczającą “*Każdy na świecie jest kochany przez przynajmniej jedną osobę*”.

Dualność kwantyfikatorów: każdy kwantyfikator może być wyrażony przez drugi z nich. Np.:

∀x *Lubi*(x, lody) ≡ ¬∃x ¬*Lubi*(x, lody)

∃x *Lubi*(x, brokuły) ≡ ¬∀x ¬*Lubi*(x, brokuły)

W dalszym ciągu badamy znaczenie formuł względem ustalonej dziedziny D. Powiemy, że formuła A jest **spełnialna** jeśli istnieje interpretacja m i wartościowanie a przy których A jest spełniona (tzn. ma wartość *True*). A jeśli jej prawdziwość nie zależy od wartościowania? Formuła A jest **prawdziwa** w interpretacji m jeśli ma wartość *True* w tej interpretacji przy każdym wartościowaniu a. Mówimy wtedy, że $[D, m]$ jest modelem formuły A. A co jeśli jej prawdziwość nie zależy od funkcji interpretacji? Mówimy, że formuła A jest **tautologią** jeśli jest *prawdziwa* przy każdej interpretacji m i wartościowaniu a (tzn. jest zawsze prawdziwa dla zadanej dziedziny D).

Teoria jest to zbiór formuł tworzony dla określonej dziedziny. Teoria jest **niesprzeczna** jeśli istnieją: interpretacja i funkcja wartościująca, dla których prawdziwe są wszystkie formuły teorii. **Aksjomaty** teorii to formuły (uznane za) prawdziwe niezależnie od wartościowania. Interesują nas teorie, których formuły dadzą się wyprowadzić w procesie wnioskowania używając aksjomatów i podzbioru formuł początkowej bazy wiedzy jako „punktów startowych" wnioskowania.

Np. aksjomaty w teorii (w świecie) „Osoby spokrewnione":

„Bracia są rodzeństwem" : $\forall x,y\ Brat(x,y) \Leftrightarrow Rodzeństwo(x,y)$

„Matka jest rodzicem i kobietą" : $\forall m,c\ Matka(c) = m \Leftrightarrow (Kobieta(m) \wedge Rodzic(m,c))$

"Rodzeństwo" jest symetryczną relacją : $\forall x,y\ Rodzeństwo(x,y) \Leftrightarrow Rodzeństwo(y,x)$

Równość jest wyróżnionym predykatem, który w każdej interpretacji ma to samo znaczenie – oznacza on relację identyczności, tzn.

$(\text{term}_1 = \text{term}_2)$ jest *prawdziwe* wtw. gdy term_1 i term_2 referują ten sam obiekt.

Z predykatu równości korzystamy często opisując własności funkcji lub definiując predykaty w terminach innych funkcji lub relacji, poprzez wprowadzenie wymogu równości lub różności obiektów. Np. poniższa definicja predykatu *Rodzeństwo* korzysta z równości termów i prawdziwości relacji *Rodzic*:

$$\forall x,y\ Rodzeństwo(x,y) \Leftrightarrow$$
$$[\neg(x = y) \wedge \exists m,f\ \neg(m = f) \wedge Rodzic(m,x) \wedge Rodzic(f,x) \wedge Rodzic(m,y) \wedge Rodzic(f,y)]$$

2.3. Rachunek sytuacyjny w logice predykatów

Zastanowimy się teraz do jakiej kategorii należy agent w „świecie Wumpusa": czy jest to prosty agent reaktywny, agent ze stanem czy też agent realizujący cel. Wprowadzimy też podzbiór języka predykatów określany jako rachunek sytuacyjny („*situation calculus*") dla modelowania agenta ze stanem.

Prosty agent reaktywny będzie posiadał jedynie reguły wiążące bezpośrednio aktualne obserwacje i akcje. Np. reguła wyrażona w logice predykatów, wiążąca aktualną obserwację błysku złota w chwili *t* z wyborem właściwej akcji „Podnieś" może przyjąć postać dwóch formuł języka:

Formuła dla obserwacji (percepcji)

$$\forall_{s,\ w,\ u,\ k,\ t}\ Percepcja([s,\ w,\ \text{Błysk},\ u,\ k],\ t) \Rightarrow PrzyZłocie(t)$$

Formuła dla wyboru akcji (refleks):

$$\forall_t\ PrzyZłocie(t) \Rightarrow NajlepszaAkcja(\text{Podnieś},\ t)$$

Dzięki wprowadzeniu zmiennych reprezentacja w logice predykatów jest znacznie efektywniejsza niż w rachunku zdań. Jednak prosty agent reaktywny nie poradzi sobie ze *światem Wumpusa*, gdyż: (1) nigdy nie będzie wiedział na pewno, czy lepiej jest wrócić do kwadratu startowego czy dalej szukać złota, (2) często się zapętli, gdyż nie rozróżnia on czy niesie już złoto czy też jeszcze nie. Przyczyny obu trudności leżą w braku informacji o stanie środowiska.

Agent reaktywny ze stanem reaguje na bieżącą obserwację środowiska w oparciu o jego aktualnie stworzony opis (stan problemu). Baza wiedzy takiego agenta ma charakter dynamiczny – akumuluje ona wszystkie obserwacje w postaci modyfikowalnego stanu problemu.

Rachunek sytuacyjny („*situation calculus*" - Hayes, McCarthy 1969) jest to pewien sposób opisywania zmian (w czasie) za pomocą języka logiki pierwszego rzędu, czyli do modelowania dynamicznie zmieniającego się świata. Główne jego założenia to:

1. Świat to ciąg sytuacji, z których każda opisuje stan świata w pewnym momencie czasu.
2. Nowa sytuacja powstaje z bieżącej sytuacji w wyniku wykonania akcji przez agenta.

Zgodnie z tym w rachunku sytuacyjnym każdy symbol predykatu, reprezentujący relację (lub własność) zmieniającą się w czasie, będzie posiadał dodatkowy argument, określający **sytuację**. Sytuacje są to obiekty należące do specyficznej kategorii ***czasu-stanu***.

Konkretnie, dla agenta w „świecie Wumpusa" wprowadzimy predykat o 3 argumentach

$$Jest(\text{Agent}, pozycja, sytuacja),$$

dla wyrażenia pozycji agenta (kratki położenia i kierunku zorientowania) w określonej sytuacji w 2-wymiarowym świecie. Np. możemy zapisać:

$$Jest(\text{Agent}, [(1,1), 90], S_0) \wedge Jest(\text{Agent}, [(1,2), 90], S_1)$$

Kolejne sytuacje (w czasie) nie są jawnie reprezentowane przez predefiniowane obiekty „czasu-stanu" ale są wynikiem poprzedniej sytuacji i wykonanej akcji. W tym celu wprowadzimy funkcję

$$Rezultat(akcja, sytuacja),$$

która wyznaczy sytuację będącą wynikiem wykonania akcji w zadanej sytuacji poprzedniej. Np. wyznaczamy nową sytuację bezpośrednio występująca po sytuacji początkowej S_0 jako:

$$S_1 = Rezultat(\text{RuchWPrzód}, S_0) .$$

W nowej sytuacji będącej wynikiem wykonanej akcji będą zachodzić nowe relacje (własności) reprezentowane predykatami. Wnioskujemy je dzięki **aksjomatom efektów akcji**, zwykle zadanym w postaci **reguł** (formuły połączone spójnikiem implikacji). Np. efektem akcji „*Podnieś złoto*" w pozycji „*x*" i sytuacji „*s*" będzie:

$$\forall_{x,s} \quad PrzyZłocie(s) \wedge Jest(Agent, x, s) \Rightarrow TrzymaZłoto(x, Rezultat(\text{Podnieś}, s))$$

Np. efektem akcji „*Puść*" będzie:

$$\forall_{x,s} \neg TrzymaZłoto(\text{x}, Rezultat(\text{Puść}, s))$$

Agent w „świecie Wumpusa" nie jest tylko agentem ze stanem - agent ma do zrealizowania plan złożony z dwóch **celów**.

- Cel 1: Wprowadzając i stosując reguły wyboru bezpiecznych akcji agent potrafi już rozsądnie poruszać się po jaskini w **celu dotarcia do złota**. Nie ma tu jawnej **lokalizacji celu** lecz jedynie warunek zatrzymania – „pobierz złoto jeśli to możliwe". Agent poznaje środowisko i bezpiecznie przemieszcza się dopóki nie znajdzie złota.
- Cel 2: W momencie dotarcia do złota, jawnym **celem agenta** staje się **bezpieczny powrót** do kwadratu startowego i wydostanie się z jaskini. Teraz istnieje jawna **lokalizacja celu**:

$$\forall_{\mathbf{x},s} \; TrzymaZłoto(\mathbf{x}, s) \Rightarrow LokalizacjaCelu([1,1], s)$$

Agent powinien znaleźć odpowiednią sekwencję akcji **realizującą ten cel**. Istnieją trzy główne techniki dla rozwiązania zadania realizacji celu:

- **Wnioskowanie** (ogólna, ale najmniej efektywna),
- **Przeszukiwanie** przestrzeni stanów problemu,
- **Planowanie** (przeszukiwanie przestrzeni planów) .

Przyjrzyjmy się temu, jak może wyglądać ogólna funkcja agenta z bazą wiedzy wyrażoną w języku logiki, posługująca sie jedynie wnioskowaniem dla wyboru akcji (tab. 2-4). Funkcja agenta posiada podfunkcję „LISTAMOŻLIWYCHAKCJI()". Realizacja tej funkcji jest trywialna – zwraca ona zawsze pełną listę możliwych akcji. Funkcja ASK uruchamia procedurę wnioskowania właściwą dla języka bazy wiedzy i sprawdzającą zachodzenie formuły zapytania utworzonej w funkcji „UTWÓRZZAPYTANIEAKCJI()". Procedura wnioskowania bezpośrednio nie wybiera akcji. Pośredni wybór akcji osiągniemy jedynie wtedy, gdy w bazie wiedzy właściwie zdefiniujemy aksjomaty wyboru akcji.

Tabl. 2-4. Program agenta w systemie z bazą wiedzy o niejawnym wyborze akcji

```
funkcja AGENTZNIEJAWNYMWYBOREMAKCJI(obserwacja)
zwraca: akcja
{ static: KB, // baza wiedzy
          t; // licznik czasu, początkowo wynosi 1
  TELL(KB, UTWÓRZZDANIEOBSERWACJi(obserwacja, t));
  for each (akcja in LISTAMOŻLIWYCHAKCJI(KB, t)) {
          if (ASK(KB, UTWÓRZZAPYTANIEAKCJI(t, akcja) ) {
                    t ← t+1;
                    TELL(KB, UTWÓRZZDANIEAKCJI(akcja, t));
                    return akcja;
          }
}
```

2.4. System logicznego wnioskowania

2.4.1. Zadania systemu

System z bazą wiedzy wyrażoną w języku logiki stanowi implementację systemu logicznego wnioskowania, czyli obok języka do deklaratywnej reprezentacji wiedzy posiada także odpowiedni mechanizm wnioskowania. Zdefiniujmy podstawowe **zadania** realizowane przez taki system na potrzeby jednostki sterowania agenta.

1. WPROWADŹ – operacja wprowadzania nowego faktu do bazy wiedzy – realizacja w postaci funkcji TELL;
2. WNIOSKUJ nowe fakty z połączenia dotychczasowej zawartości bazy wiedzy i nowo wprowadzonych faktów – realizacja w postaci wnioskowanie wprzód jako część funkcji TELL;
3. DECYDUJ czy zapytanie WYNIKA z bazy wiedzy – realizacja w postaci funkcji ASK;
4. DECYDUJ czy zapytanie ISTNIEJE w bazie wiedzy – ograniczona wersja funkcji ASK;
5. USUŃ – usuwa zdanie z bazy wiedzy z różnych przyczyn – zdanie jest nieprawdziwe, już niepotrzebne lub modyfikacja wynika ze zmiany w środowisku.

O efektywności procedur wnioskowania decyduje głównie właściwa implementacja bazy wiedzy i tych części funkcji TELL i ASK, które bezpośrednio odwołują się do KB.

- Implementacja zdań i formuł w KB

Wprowadzamy bazowy typ danych COMPOUND, który reprezentuje powiązanie operatora (czyli predykatu, funkcji lub spójnika logicznego) z listą argumentów (czyli z termami lub zdaniami). Posiada on pola dla reprezentacji operatora (OP) i argumentów (ARGS).

Np. niech istnieje zdanie „c": $P(x) \wedge Q(x)$.

Wtedy: OP[c] = $\wedge$ i ARGS[c] = [P(x), Q(x)].

- Wprowadzamy funkcje dostępu do bazy danych: STORE(KB, S) i FETCH(KB, Q).

Przy nieuporządkowanej bazie wiedzy złożoność obliczeniowa obu funkcji wynosi O(*n*), gdzie *n* jest liczbą elementów w bazie wiedzy. Dla osiągnięcia efektywnego wykonania tych często wołanych funkcji należy uporządkować elementy stosując, np. indeksowanie z wykazem asocjacyjnym lub indeksowanie w postaci drzewa.

2.4.2. Wybrane rodzaje systemów logicznego wnioskowania

Systemy **dowodzenia twierdzeń** (ang. *theorem provers*) (np. AURA, OTTER) stosują wnioskowanie metodą rezolucji (omówienie w rozdziale 3) w logice predykatów i ich celem jest dowodzenia twierdzeń w zadaniach matematycznych oraz realizacja zapytań do bazy wiedzy.

Języki **programowania w logice** (np. Prolog) stosują zwykle wnioskowanie wstecz (patrz rozdział 3), nakładają ograniczenia na język predykatów i dysponują proceduralnymi elementami wejścia/wyjścia. Program w Prologu to uporządkowany ciąg formuł a dane to formuła celu (zapytanie). Wykonanie programu to realizacja wnioskowania wstecz z przeszukiwaniem według zasady „*przeszukiwanie w głąb*", i „*od lewej-do prawej*". Programowanie w Prologu ma zasadniczo charakter deklaratywny (kod programu jest logiczną specyfikacją problemu a jego wykonanie – wnioskowaniem logicznym). Jednak w celu zwiększenia efektywności występuje też wiele mechanizmów pozalogicznych: jest duży zbiór wbudowanych predykatów związanych z arytmetyką i wejściem/wyjściem – sprawdzenie poprawności tych predykatów polega na wykonaniu odpowiedniego kodu procedur a nie na wnioskowaniu logicznym; występuje też szereg funkcji systemowych i obsługujących bazę wiedzy.

Systemy **produkcyjne** (np. OPS5, CLIPS, SOAR) stosują reguły (implikacje wiążące warunki wykonania z opisami akcji) i są podstawą wielu systemów ekspertowych. Systemy produkcji realizują wnioskowanie wprzód (patrz rozdział 3), używając bardzo ograniczonego języka. Typowy system produkcji składa się z: pamięci roboczej – zawiera pozytywne literały bez zmiennych (zdania atomowe); zbioru reguł – wyrażeń postaci:

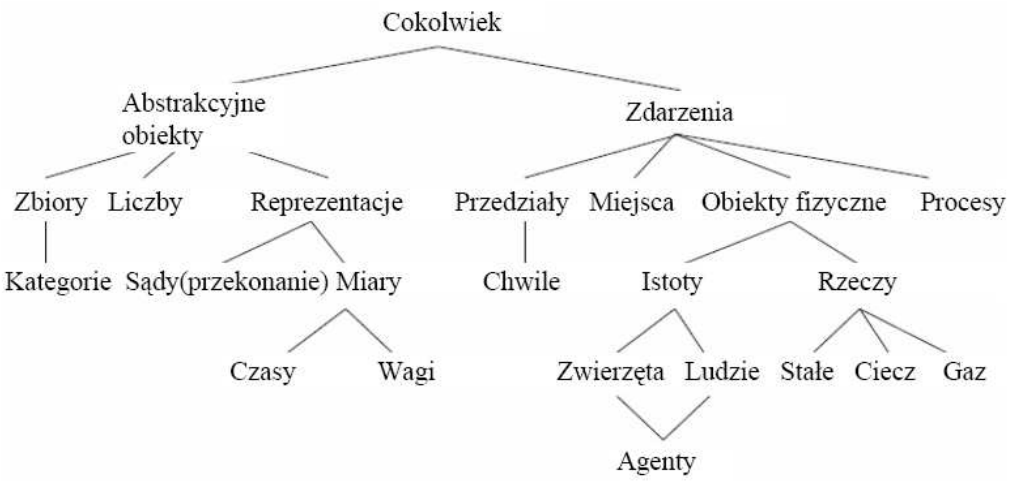
$$P_1 \wedge ... \wedge P_n \Rightarrow akcja_1 \wedge ... \wedge akcja_m ,$$

gdzie (P_1,...,P_n) są literałami (formułami atomowymi), a ($akcja_1$,..., $akcja_m$) są działaniami, które należy wykonać wtedy, jeśli wszystkie zdania P_i są prawdziwe (tzn. znajdują się w pamięci roboczej). Wśród akcji są operacje wprowadzania i usuwania do/z bazy wiedzy i operacje we/wy.

„**Ramy**" (ang. *frames*) i **sieci semantyczne** (np. SNePS, Netl, KL-ONE) są to etykietowane grafy o ściśle zdefiniowanych etykietach łuków. Obiekty i ich kategorie są węzłami grafu a ich relacje binarne reprezentują zależności typu: „*jest częścią*", „*jest specjalizacją*", „*jest instancją*". Każdy węzeł sieci, posiada nazwane atrybuty, gdzie każdy atrybut wiąże obiekt z jakimś termem (w szczególności ze stałą). Atrybuty kategorii ogólnej są dziedziczone przez jej specjalizacje. W zasadzie każda sieć semantyczna może być wyrażona w postaci zbioru formuł logiki predykatów. Jednak sieć semantyczna realizująca dziedziczenie z możliwością wystąpienia **wyjątków** jest potencjalnie **niemonotoniczna** – dodanie nowej przesłanki może unieważnić dotychczas wyprowadzone formuły.

2.5. Ontologia języka

Ontologia to formalizacja (modelowanie) fundamentalnych pojęć wiedzy (obiektów, relacji), tzn. takich, które występują w wielu dziedzinach zastosowań. Np. są to pojęcia: kategorie, miary, czas itp. (rys. 2.3).



Rys. 2.3: Przykład kategoryzacji pojęć

Kategoria

Kategoria - zbiór obiektów o podobnych cechach.

Jak reprezentować kategorie?

1) Przy użyciu 1-argumentowych predykatów. Np. *Pomidor*(*a*).

2) Stosując technikę reifikacji – predykat lub funkcja traktowane są jak obiekt (stała) języka. Np. *Pomidory* - stała interpretowana jako obiekt reprezentujący zbiór wszystkich pomidorów.

Kategorie w logice predykatów

$Pomidor_{12} \in Pomidory$ (obiekt jest elementem kategorii)

$\forall x \;\; x \in Pomidory \Rightarrow x \in Owoce$ (relacja dziedziczenia pomiędzy kategoriami *Pomidory* i *Owoce*)

$\forall x \;\; x \in Pomidory \Rightarrow Czerwone(x) \wedge Okrągłe(x)$ (elementy kategorii posiadają własności)

Miary

Miary – abstrakcyjne obiekty dla wyrażenia własności obiektów fizycznych.

Miary ilościowe przedstawiamy jako połączenie funkcji wyrażającej jednostkę miary z liczbą. Np. $Długość(L_1) = Cale(1.5)$

Konwersje pomiędzy jednostkami, np.: $\forall l \; Centymetry(2.54*l) = Cale(l)$

$\forall t \; Celsiusz(t) = Fahrenheit(32+1.8*t)$

Charakterystyka obiektu za pomocą miary, np.: $Waga(Pomidor_{12}) = Kilogramy(0.16)$

$\forall d \; d \in Dni \Rightarrow (CzasTrwania(d) = Godziny(24))$

Należy rozróżniać pomiędzy jednostkami miary a narzędziami operacji na danej wielkości. Np.

$\forall b \; b \in RachunekWDolarach \Rightarrow (JednostkaRachunku(b) = \$(1.00))$

Miary jakościowe są trudniejsze do użycia niż ilościowe, gdyż nie posiadają ogólnie przyjętej skali liczbowych wartości. Można je jednak porządkować. Np.:

$\forall e_1, e_2 \;\; e_1 \in Ćwiczenia \wedge e_2 \in Ćwiczenia \wedge Autor(e_1, Norvig) \wedge Autor(e_2, Russel)$

$\Rightarrow Trudność(e_1) > Trudność(e_2)$.

Obiekty złożone

Relacja „części-do-całości”. Np.

CzęśćCałość(*Budapeszt, Węgry*),

CzęśćCałość(*Węgry, EuropaŚrodkowa*),

CzęśćCałość(*EuropaŚrodkowa, Europa*)

$\forall x,y,z$ *CzęśćCałość*(x,y) $\wedge$ *CzęśćCałość* (y, z) $\Rightarrow$ *CzęśćCałość*(x,z)

Obiekt złożony – to obiekt, który posiada części. Możemy określić strukturę obiektu, przez podanie jego części i relacji pomiędzy częściami.

Związek obiektów – to obiekt, posiadający części ale bez struktury. Np. obiekt złożony z 3 jabłek:

Związek({$Jabłko_1$, $Jabłko_2$, $Jabłko_3$})

Zdarzenia (*events*)

Do reprezentowania zmian możemy użyć „zdarzeń”. Zdarzenie to element *czasoprzestrzeni* – coś co odbywa się jednocześnie w czasie i przestrzeni – to jakby ciągła wersja rachunku sytuacyjnego.

Przedział (*interval*) – jest to specjalne zdarzenie posiadające jedynie wymiar czasowy – przedział czasu. Zawiera w sobie, jako pod-zdarzenia, wszystkie te zdarzenia, które wystąpiły w tym przedziale czasu. Np.: *PodZdarzenie*(WojnaŚwiatowa2, WiekXX)

Zdarzenia możemy pogrupować w kategorie. Np. skorzystamy z kategorii *Podróże* aby wyrazić zdarzenie odbycia konkretnej podróży:

$\exists j$ $j \in$ *Podróże* $\wedge$ *Start*(*Warszawa*, j) $\wedge$ *Cel*(*NewYork*, j) $\wedge$

Podróżny(*Kowalski*, j) $\wedge$ *PodZdarzenie*(j, *Wczoraj*).

Często interesują nas nie tyle same nazwy kategorii co ich właściwości. Skorzystamy w tym celu z formuł złożonych. Np. symbol *Go* będzie wyznaczał pewien rodzaj podróży - niech *Go*(x,s,c) oznacza, że osoba x podróżuje z miejsca s do miejsca c:

$\forall e,x,s,c$ $e \in Go(x,s,c) \Leftrightarrow [e \in Podróże \wedge Podróżny(x,e) \wedge Start(s,e) \wedge Cel(c,e)]$.

Użyjmy notacji $E(c,i)$, aby wyrazić to, że zdarzenie kategorii c jest pod-zdarzeniem zdarzenia (lub przedziału) i:

$\forall c,i$ $E(c,i) \Leftrightarrow \exists e\ e \in c \wedge PodZdarzenie(e,i)$.

Np: *E*(*Go*(*Kowalski,Warszawa,NewYork*),*Wczoraj*)

Miejsca

Miejsca, podobnie jak przedział, stanowią obszar w czaso-przestrzeni, ale o niezmiennym położeniu „rozciągniętym w czasie. Relacje pomiędzy miejscami wyrazimy np. w postaci predykatu *W* („miejsce zawarte w”). Np. *W* (Warszawa, Polska)

Wprowadźmy funkcje odwzorowującą obiekty i miejsca. Np. funkcja *Miejsce* odwzorowuje obiekt na najmniejsze miejsce, zawierające ten obiekt:

$\forall x,l$ $Miejsce(x) = l \Leftrightarrow Jest(x,l) \wedge \forall l_2\ Jest(x,l_2) \Rightarrow W(l,l_2)$.

Procesy

Proces to zdarzenie kategorii c, którego każde pod-zdarzenie (w mniejszym przedziale czasu) jest zdarzeniem tej samej kategorii c. W ten sposób wyrazimy zdarzenie ciągłe w czasie. Np. *Lot*(*Kowalski*) wzięty jako całość lub jego fragment, są to zawsze zdarzenia tej samej kategorii *Lot*.

W odniesieniu do zdarzenia ciągłego w czasie (procesu) możemy zastosować tę samą notacją co poprzednio do zdarzeń dyskretnych. Np. lot odbył się wczoraj:

$E(Lot(Kowalski),Wczoraj)$.

Często jednak chcemy wyrazić, że proces trwał dokładnie w jakimś przedziale czasu. Użyjemy notacji $T(c,i)$ na wyrażenie faktu, że zdarzenie c odbyło się dokładnie w przedziale i.

Np. $T(Pracuje(Jan), DzisiejszaPoraLunchu)$.

Notacja dla łączenia zdarzeń i czasu

Nie możemy napisać: $T(Jest(A_1,Loc_1) \wedge Jest(A_2,Loc_2), i)$, gdyż pierwszy argument predykatu T nie jest termem tylko formułą. Ogólnie zauważymy, że w wyrażeniu postaci $T(p \wedge q, e)$ - $p \wedge q$ jest formułą, a nie termem wskazującym na kategorię zdarzeń. Dlatego powinniśmy wprowadzić symbol funkcji *And* zdefiniowany aksjomatem:

$\forall p,q,e \;\; T(And(p,q), e) \Leftrightarrow T(p,e) \wedge T(q,e)$,

czyli funkcja *And* zwraca kategorię złożonego obiektu. Możemy teraz napisać:

$T(And(Jest(A_1,Loc_1), Jest(A_2,Loc_2)), i)$.

Podobnie wprowadzimy, np. symbol funkcji *OR* dla połączenia dwóch zdarzeń (wzajemnie wykluczających się) alternatywą:

$\forall p,q,e \;\; T(OR(p,q), e) \Leftrightarrow T(p,e) \vee T(q,e)$.

Czas

Podstawowe pojęcie w tym zakresie to: **przedział** (odcinek) czasu. Z przedziałem związany jest jego czas trwania (długość). Przedział o długości zero nazwiemy **chwilą** czasu.

$\forall i \;\; i \in Przedziały \Rightarrow [i \in Chwile \Leftrightarrow Trwa(i) = 0]$.

Wprowadźmy pewne funkcje, wyznaczające chwile czasu na globalnej osi czasu:

Start(i) – początkowa chwila przedziału,

End(i) – końcowa chwila przedziału,

Trwa(i) – różnica pomiędzy końcową a początkową chwilą czasu,

Czas(j) – wyznacza punkt na osi czasu dla zadanej chwili.

Określmy możliwe położenia wzajemne dwóch odcinków czasu:

$\forall i,j \;\; Spotyka(i,j) \Leftrightarrow Czas(End(i)) = Time(Start(j))$.

$\forall i,j \;\; Przed(i,j) \Leftrightarrow Czas(End(i)) < Time(Start(j))$.

$\forall i,j \;\; Po(j,i) \Leftrightarrow Przed(i,j)$.

$\forall i,j \; Podczas(i,j) \Leftrightarrow Czas(Start(j)) \leq Czas(Start(i)) \wedge Czas(End(i)) \leq Czas(End(j))$

$\forall i,j \;\; Przecina(i,j) \Leftrightarrow \exists k \; Podczas(k,i) \wedge Podczas(k,j)$.

Akcje

Relacje pomiędzy przedziałami czasu wykorzystywane są przy definiowaniu akcji. Zwykle polega to na podaniu przedziału czasu, w którym dane zdarzenie zachodzi.

Przykłady. Kiedy 2 osoby są ze sobą w związku to mogą zajść w przyszłości akcje: *Ślub* lub *RozpadZwiązku*.

$\forall x,y,i_0 \; T(Związek(x,y),i_0) \Rightarrow \exists i_1 \; [Spotyka(i_0,i_1) \vee Po(i_1,i_0)] \wedge T(OR(Ślub(x,y), RozpadZwiązku(x,y)),i_1)$.

Uwaga: OR(*Ślub*(x,y),*RozpadZwiązku*(x,y)) jest termem wskazującym na kategorię zdarzeń.

Po akcji *Ślub* dwoje ludzi staje się małżonkami:

$\forall x,y,i_0 \;\; T(\text{Ślub}(x,y),i_0) \Rightarrow \exists i_1 \; T(\text{Małżonek}(x,y),i_1) \wedge \text{Spotyka}(i_0,i_1)$

Wynikiem akcji *Go* jest zalezienie się w drugim miejscu:

$\forall x,a,b,i_0 \; \exists i_1 \; T(\text{Go}(x,a,b),i_0) \Rightarrow \mathrm{T}(W(x,b),i_1) \wedge \text{Spotyka}(i_0,i_1)$

Fluenty

Fluenty są to obiekty, których właściwości zmieniają się w czasie. Np. *Powierzchnia*(*Polska*) – obiekt przyjmujący różne wartości w różnych okresach czasu, *Prezydent*(*USA*) – obiekt wskazujący na różne osoby w różnych momentach czasu.

Fluenty pozwalają na zwarte wyrażenie wspólnych cech. Np. dla wyrażenia faktu, że prezydenci byli mężczyznami w 19 wieku napiszemy: *T*(*Mężczyzna*(*Prezydent*(*USA*))*,19tyWiek).* Np. dla wyrażenia powierzchni Polski w określonym roku napiszemy:

T(*Wartość*(*Powierzchnia*(*Polska*)*, KmKw*(*621000*))*,AD1426*)

T(*Wartość*(*Powierzchnia*(*Polska*)*, KmKw*(*312000*))*,AD1970*)

Sądy (przekonania)

Modelujemy przekonania (*sądy*) agenta za pomocą relacji typu: *Wierzy*(Agent, *x*), *Wie*(Agent, *x*), *Chce*(Agent, *x*). Np. chcemy tym wyrazić, że: „*a* wie, że *p", „a* wierzy, że *p", „a* chce aby, *p"*. Czyli chcemy wyrazić relację pomiędzy agentem i formułą (lub zdaniem).

Ale możemy napisać, *Wierzy*(*Agent, x*), jedynie pod warunkiem, że *x* jest termem. Np. *Wierzy*(*Agent, Lata*(*Superman*)) jest prawidłowe pod warunkiem, że *Lata*(*Superman*) jest termem. Dlatego należy dokonać najpierw *reifikacji* predykatu *Lata*(*Superman*) do postaci *obiektu-fluentu*.

Związek pomiędzy „przekonaniem" a „wiedzą" może być rożnie definiowany. Filozofowie definiują wiedzę jako „udowodniona prawdziwość przekonania". Wyrazimy to następująco:

$\forall a,p \;\; \text{Wiedza}(a,p) \Leftrightarrow \text{Wierzy}(a,p) \wedge T(p) \wedge T(KB(a) \Rightarrow p)$

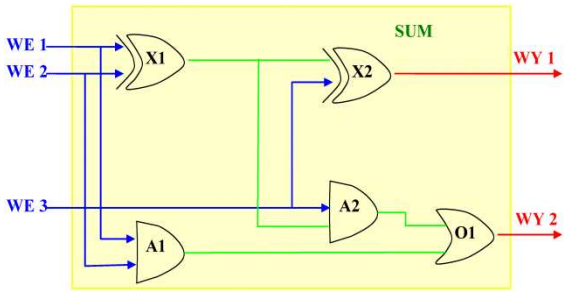
2.6. Pytania

1. Omówić pojęcie „inżynieria wiedzy" i zasady realizacji agenta w postaci systemu z bazą wiedzy.
2. Omówić elementy składni rachunku zdań i logiki predykatów.
3. Omówić semantykę rachunku zdań i języka predykatów.
4. Wyrazić działanie agenta w rachunku sytuacyjnym.
5. Omówić systemy logicznego wnioskowania spotykane w praktyce.
6. Omówić „ontologię" języka i podać przykłady typowych pojęć.

2.7. Zadania

Zad. 2.1

Zaprojektować bazę wiedzy dla reprezentacji układu cyfrowego sumatora jedno-bitowego (rys. 2.4).



Rys. 2.4:

Zad. 2.2

Zapisać następujące zdania w języku logiki predykatów, wprowadzając niezbędne symbole i ustalając ich interpretację:

1. ojciec każdego człowieka jest jego bezpośrednim przodkiem,
2. jeśli ktoś jest przodkiem bezpośredniego przodka pewnej osoby, to jest także przodkiem tej osoby,
3. każdy jest spokrewniony z każdym swoim przodkiem,
4. każdy jest spokrewniony ze swoim bratem i siostrą,
5. każdy jest spokrewniony z braćmi i siostrami wszystkich osób spokrewnionych ze sobą.

Zad. 2.3

W rachunku sytuacyjnym logiki predykatów zdefiniować przykładowe predykaty zmienne w czasie, dla agenta „świata Wumpusa”.

Zad. 2.4

W rachunku sytuacyjnym zdefiniować reguły wyboru akcji dla agenta „świata Wumpusa”.

3. Wnioskowanie w logice

Reguły wnioskowania

Postaci normalne formuł

Wnioskowanie metodą rezolucji

Wnioskowanie wprost

Pytania

3.1. Reguły wnioskowania

3.1.1. Wnioskowanie w rachunku zdań

Wiemy już, że zdanie logiczne (formuła) jest tautologią wtw. gdy jest prawdziwa we wszystkich modelach języka. Np. tautologiami są następujące zdania:

$True, \qquad A \vee \neg A, \qquad A \Rightarrow A, \qquad (A \wedge (A \Rightarrow B)) \Rightarrow B$

Tautologia jest powiązana z wnioskowaniem poprzez **twierdzenie o dedukcji**, które mówi, że:

$KB \models \alpha$ wtw. gdy formuła $(KB \Rightarrow \alpha)$ jest tautologią.

To twierdzenie prowadzi do metod wnioskujących wprost.

Wiemy też, że formuła A jest spełnialna wtw. gdy posiada przynajmniej jeden model. Wtedy niespełnialną formułą jest taka, która nie posiada żadnego modelu. Np.:

$A \wedge \neg A$ jest niespełnialne.

Spełnialność formuły jest powiązana z wnioskowaniem poprzez **drugie twierdzenie o dedukcji**:

$KB \models \alpha$ wtw. gdy formuła $(KB \wedge \neg\alpha)$ jest niespełnialna.

To twierdzenie prowadzi do wnioskowania nie wprost – do dowodu przez zaprzeczenie.

Na potrzeby procedur wnioskowania zdania w bazie wiedzy powinny przyjmować wymagane postacie *normalne*. Istnieje szereg równoważnościowych przekształceń zdań logicznych. Mówimy, że dwa zdania są **logicznie równoważne** (oznaczamy ≡) wtw. gdy są poprawne w tych samych modelach. Innymi słowy logiczną równoważność możemy powiązać z wynikaniem:

$\alpha \equiv ß$ wtw. $(\alpha \models \beta)$ i $(\beta \models \alpha)$

Oto podstawowe pary logicznie równoważnych zdań:

1. $(\alpha \wedge ß) \equiv (ß \wedge \alpha)$ – przemienność $\wedge$
2. $(\alpha \vee ß) \equiv (ß \vee \alpha)$ – przemienność $\vee$
3. $((\alpha \wedge ß) \wedge \lambda) \equiv (\alpha \wedge (ß \wedge \lambda))$– łączność $\wedge$
4. $((\alpha \vee ß) \vee \lambda) \equiv (\alpha \vee (ß \vee \lambda))$– łączność $\wedge$
5. $\neg(\neg \alpha) \equiv \alpha$ – eliminacja podwójnej negacji
6. $(\alpha \Rightarrow ß) \equiv (\neg ß \Rightarrow \neg \alpha)$ – kontrapozycja
7. $(\alpha \Rightarrow ß) \equiv (\neg \alpha \vee ß)$ – eliminacja implikacji
8. $(\alpha \Leftrightarrow ß) \equiv ((\alpha \Rightarrow ß) \wedge (ß \Rightarrow \alpha)$ – eliminacja równoważności

9. $\neg(\alpha \wedge ß) \equiv (\neg\, \alpha \vee \neg\, ß)$ – prawo de Morgana
10. $\neg(\alpha \vee ß) \equiv (\neg\, \alpha \wedge \neg\, ß)$ – prawo de Morgana
11. $(\alpha \wedge (ß \vee \lambda)) \equiv ((\alpha \wedge ß) \vee (\alpha \wedge \lambda))$ – rozdzielczość $\wedge$ względem $\vee$
12. $(\alpha \vee (ß \wedge \lambda)) \equiv ((\alpha \vee ß) \wedge (\alpha \vee \lambda))$ – rozdzielczość $\vee$ względem $\wedge$

Dla rachunku zdań istnieją zupełne i poprawne **procedury wnioskowania** (dowodzenia formuł). Stosują one **reguły wnioskowania** o ogólnej postaci:

$$\frac{poprzednik}{nastepnik}$$

Reguła wnioskowania mówi, że jeśli spełniony jest warunek zadany *poprzednikiem* reguły to wnioskowana jest poprawność *następnika*. Oczywiście, aby móc stosować jakąś regułę wnioskowania musimy upewnić się, że jest ona poprawna we wszystkich modelach języka, tzn. że jest tautologią języka. Powiemy, że reguła wnioskowania o postaci

$$\frac{\alpha_1, \alpha_2, \cdots, \alpha_n}{\beta}$$

jest **poprawna**, jeśli zdanie β jest prawdziwe w każdej interpretacji (modelu), w której prawdziwe są zdania: $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Wybór poprawnych i najczęściej stosowanych reguł wnioskowania:

1. Reguła **odrywania** (*modus ponens*):

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta}$$

2. Reguła **eliminacji koniunkcji**:

$$\frac{\alpha_1 \wedge \cdots \wedge \alpha_n}{\alpha_i}$$

3. Reguła **wprowadzania koniunkcji**:

$$\frac{\alpha_1, \cdots, \alpha_n}{\alpha_1 \wedge \cdots \wedge \alpha_n}$$

4. Reguła **rezolucji**:

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Sprawdźmy za pomocą tablicy prawdy poprawność powyższej reguły rezolucji (rys. 3-1).

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	1	1

Rys. 3-1: Tabela prawdy dla reguły rezolucji.

W tabeli wyróżniono na różowo 4 wiersze, dla których spełniony jest poprzednik tej reguły. Widzimy, że w tych wierszach następnik również jest zawsze spełniony. Tym samym reguła rezolucji jest poprawna.

3.1.2. Wnioskowanie w logice predykatów

W logice pierwszego rzędu również zachodzą oba znane nam już **twierdzenia o dedukcji**:

1. $KB \models \alpha$ wtw. gdy $(KB \Rightarrow \alpha)$ jest tautologią.
2. $KB \models \alpha$ wtw. gdy formuła $(KB \wedge \neg\alpha)$ jest niespełnialna.

Ze względu na potencjalnie nieprzeliczalną liczbę wartościowań dla badanej teorii, która jest możliwa w języku logiki pierwszego rzędu, pojawia się problem z generalną rozstrzygalnością procesu wnioskowania. W logice predykatów zachodzi następujące twierdzenie:

„Problem stwierdzenia, czy dana formuła jest tautologią czy nie, jest problemem nierozstrzygalnym".

Można jedynie pokazać, że:

„istnieje algorytm, który dla zadanej formuły *A* stwierdza, że *A* jest tautologią, pod warunkiem, że tak istotnie jest."

Podstawienie

Rozszerzymy teraz reguły wnioskowania z rachunku zdań do odpowiednich postaci wymaganych przez logikę predykatów.

Przez **podstawienie** rozumiemy zbór par postaci $\{x_1/t_1,...,x_n/t_n\}$, gdzie $x_1,...,x_n$ są różnymi zmiennymi, natomiast, $t_1,...,t_n$, są termami. Dopuszczamy podstawienie puste ε.

Niech θ będzie podstawieniem i niech α będzie wyrażeniem (tzn. formułą lub termem). Przez **SUBST**(θ , α) oznaczamy wyrażenie powstałe w wyniku jednoczesnego zastąpienia wszystkich wolnych wystąpień zmiennych, $x_1,...,x_n$, termami, $t_1,...,t_n$.

Np. jeśli $\theta = \{x/\text{Jan}, y/\text{Ewa}\}$, to

$$\text{SUBST}(\theta, Lubi(x, y)) = Lubi(\text{Jan}, \text{Ewa}).$$

Przypomnijmy, że literał jest to formuła atomowa lub negacja formuły atomowej, a zdanie - formuła bez zmiennych.

Unifikacja (uzgadnianie) zmiennych

Obecność symboli zmiennych w logice predykatów powoduje to, że nie możemy poprzestać na prostym wymaganiu identyczności formuł tworzących poprzednik reguły wnioskowania. Stosujemy łagodniejsze wymaganie takie, że pierwsza i druga formuła wejściowa (tworzące poprzednik reguły wnioskowania) są **unifikowalne**, czyli mogą zostać sprowadzone do postaci identycznej przez zastosowanie odpowiednich podstawień dla zmiennych - przypisanie im obiektów lub pewnych termów. W istocie mamy do czynienia z dwoma przypadkami **unifikacji (uzgadniania zmiennych)**:

1. Ujednolicanie zmiennych – wtedy, gdy dwie formuły różnią się jedynie tym, że w odpowiednich miejscach występują w nich konsekwentnie inne symbole zmiennych,

2. Uszczegółowienie – wtedy, gdy w miejscach, gdzie w jednej z nich występuje pewien symbol zmiennej (związany kwantyfikatorem ogólnym), w drugiej konsekwentnie występuje pewien term nie będący zmienną (stała albo zastosowanie symbolu funkcyjnego).

Mówimy, że podstawienie θ jest unifikatorem wyrażeń, $E_1,...,E_n$, jeśli

$$SUBST(\theta, E_1) = ... = SUBST(\theta, E_n).$$

Np. podstawienie {x/A} jest unifikatorem wyrażeń P(x), P(A). Ale wyrażenia P(x), Q(b) nie są unifikowalne.

Reguły wnioskowania w logice predykatów stosują **algorytm unifikacji** (nazwijmy go UNIFY(p,q)), który zwraca takie podstawienie (tzw. **najogólniejszy unifikator**) dla literałów p i q, które czyni je *identycznymi*, lub zwraca *błąd*, jeśli podstawienie nie istnieje. Czym jest najogólniejszy unifikator? **Złożeniem** podstawień θ_1, θ_2 jest takie podstawienie (oznaczmy je przez COMPOSE(θ_1, θ_2)), które polega na wykonaniu po kolei obu podstawień, czyli:

$$SUBST(COMPOSE(\theta_1, \theta_2), E) = SUBST(\theta_2, SUBST(\theta_1, E)).$$

Podstawienie θ jest **najogólniejszym** unifikatorem dla $\{E_1,..., E_n\}$, jeśli dla każdego unifikatora γ dla tych wyrażeń istnieje podstawienie λ takie, że: γ = COMPOSE(θ, λ). Innymi słowy najogólniejszy unifikator zawiera jedynie to co niezbędne dla unifikacji dwóch literałów, na których działa. Ważną dodatkową obserwacją jest to, że unifikowalne wyrażenia posiadają dokładnie jeden najogólniejszy unifikator.

Przykład. Wyrażenia P(Jan, x), P(y, z) posiadają nieskończenie wiele unifikatorów. Niektóre z nich to: {y/Jan, x/z}, {y/Jan, x/z, w/Fred}, {y/Jan, x/Jan, z/Jan}. Najogólniejszym unifikatorem jest:

{y/Jan, x/z}

Problem uzgadniania zmiennych w wyrażeniach, $E_1,..., E_n$, polega najpierw na zbadaniu, czy dane wyrażenia są unifikowalne, a jeśli tak to należy znaleźć ich najogólniejszy unifikator.

Problem unifikacji jest rozstrzygalny. Wynika to ze skończonej liczby zmiennych w literałach.

Przykład unifikacji literałów:

p = Zna(Jan, x);	q = Zna(Jan, Ewa);	UNIFY(p,q) =	{x/Ewa}
p = Zna(Jan, x)	q = Zna(y, Ewa)		{x/Ewa, y/Jan}
p = Zna(Jan, x)	q = Zna(y, Matka(y))		{y/Jan, x/Matka(Jan)}
p = Zna(Jan, x)	q = Zna(x, Ewa)		*błąd*

Standaryzacja rozłączna

Możemy otrzymać równoważne warianty zadanej formuły zmieniając nazwy zmiennych tej formuły (*przemianowanie zmiennych*). Powiemy, że formuła A jest **wariantem** formuły B, jeśli A można otrzymać z B zastępując niektóre zmienne w B nowymi zmiennymi nie występującymi w B.

Np. *Lubi* (x, Jan) jest wariantem formuły *Lubi* (y,Jan) .

W zasadzie każda formuła w bazie wiedzy może zostać zastąpiona swoim wariantem (przemianowana) i otrzymamy bazę wiedzy równoważną z poprzednią postacią. Jednak dlaczego należy przemianowywać zmienne w formule? Jest to motywowane potrzebą uniknięcia błędów unifikacji, które powodowane są jedynie przez powtarzanie się nazw zmiennych w obu literałach.

Np. załóżmy, że w istnieją dwie formuły: $Zna(x, Ewa)$, $Zna(Jan, x) \Rightarrow Nienawidzi(Jan, x)$. Powinniśmy móc wywnioskować na podstawie tej pary, stosując ogólną regułę *Modus Ponens*, że zachodzi $Nienawidzi(Jan, Ewa)$. W praktyce nie jest to jednak możliwe, gdyż formuły, $Zna(Jan, x)$ i $Zna(x, Ewa)$, nie są unifikowalne. Dopiero potem, jak przemianujemy jedną ze zmiennych x (np. formułę $Zna(x, Ewa)$, zamienimy na, $Zna(y, Ewa)$) to będziemy mogli wyprowadzić $Nienawidzi(Jan, Ewa)$.

Proces przemianowania zmiennych występujących w formułach, tak, aby każda formuła posiadała unikalne zmienne nazywamy **standaryzacją rozłączną** formuł. Możemy założyć, że przesłanki ogólnej reguły *Modus Ponens* stosowanej w logice predykatów nie zawierają wspólnych zmiennych, gdyż zostały wcześniej poddane standaryzacji rozłącznej.

3.2. Postaci normalne formuł

Procedury wnioskowania zakładają, że formuły występują we właściwej postaci **normalnej**. Pozwala to zdefiniować obliczeniowo efektywną procedurę wnioskowania.

Klauzula Horna

Dla procedur stosujących regułę *Modus Ponens* (zwaną także *regułą odrywania*) zdania (lub formuły) przyjmują postać tzw. **klauzul Horna**. Klauzula Horna to pojedynczy literał lub implikacja o postaci:

(*koniunkcja prostych literałów*) $\Rightarrow$ prosty literał .

„Prosty" oznacza „pozytywny", nie zanegowany. Np. w poniższym zdaniu występują trzy klauzule Horna: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Modus Ponens

Reguła odrywania (*Modus Ponens*) stosowana jest w procedurach wnioskowania dla rachunku zdań wtedy, gdy zdania w bazie wiedzy występują w postaci klauzul Horna. Tworzą one poprzednik reguły:

$$\frac{\alpha_1, \cdots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Z reguły odrywania korzystają procedury wnioskowania wprost: progresywna (wnioskowanie wprzód) i regresywna (wnioskowanie wstecz).

Uogólniona reguła odrywania (*General Modus Ponens*) stosowana jest w procedurach wnioskowania dla logiki predykatów. Jest to następująca reguła:

$$\frac{p_1', \cdots, p_n', \qquad p_1 \wedge \cdots \wedge p_n \Rightarrow q}{SUBST(\theta, q)}$$

gdzie (p_1', … ,p_n‘ ,p_1 , ... ,p_n ,q) są literałami, a θ jest podstawieniem takim, że dla każdego $i = 1,2,...,n$:

$\mathrm{SUBST}(\theta, p') = \mathrm{SUBST}(\theta, p)$.

Np. $p_1' = Król(Jan)$, $p_1 = Król(x)$, $p_2' = Chciwy(y)$, $p_2 = Chciwy(x)$, $q = Zło(x)$

$\theta = \{x/Jan, y/Jan\}$, $\mathrm{SUBST}(\theta, q) = Zło(Jan)$

Zastosowanie uogólnionej reguły odrywania poprzedzone jest zamianą formuł w bazie wiedzy do postaci klauzul Horna, wykonywaną w następujących krokach:

1. Przemianowanie zmiennych w formule - każdy zmienna w formule związana kwantyfikatorem musi być unikalnie nazwana.
2. Zamiana każdej formuły na postać iloczynu klauzul;
3. Przekształcanie iloczynu klauzul do postaci zbioru klauzul przez eliminację koniunkcji
4. Eliminacja kwantyfikatorów egzystencjalnych
5. Opuszczenie kwantyfikatorów uniwersalnych
6. Standaryzacja rozłączna klauzul

Eliminacja uniwersalnego kwantyfikatora

Każde wartościowanie formuły związanej uniwersalnym kwantyfikatorem wynika z tej formuły. Możemy to zapisać w postaci reguły:

$$\frac{\forall_v \alpha}{SUBST(\{v/g\},\alpha)}$$

dla każdej zmiennej v i termu g.

Tym samym kwantyfikator uniwersalny nie nakłada ograniczeń na wartościowanie związanej nim zmiennej. Jeśli nie budzi to wątpliwości, po uprzednim unikalnym przemianowaniu zmiennych i po eliminacji ewentualnych kwantyfikatorów egzystencjalnych, opuszczamy kwantyfikatory uniwersalne.

Eliminacja egzystencjalnego kwantyfikatora

Eliminacja egzystencjalnego kwantyfikatora nosi nazwę **skolemizacji formuły**. Rozróżnimy tu dwa przypadki skolemizacji, zależnie od tego czy w formule występuje kwantyfikator uniwersalny poprzedzający kwantyfikator egzystencjalny, czy też nie.

Eliminacja kwantyfikatora egzystencjalnego nie poprzedzonego żadnym kwantyfikatorem uniwersalnym polega na zastosowaniu następującej reguły wnioskowania: „dla każdej formuły α, zmiennej v i pewnego symbolu stałej K, który nie występuje nigdzie indziej w bazie wiedzy, zachodzi":

$$\frac{\exists_v \alpha}{SUBST(\{v/K\},\alpha)}$$

Eliminacja kwantyfikatora wiąże się z jednoczesnym przemianowaniem zmiennej v w nim związanej na pewną unikalną stałą K.

Np. z formuły, $\exists x\ Korona(x) \wedge NaGłowie(x, Jan)$, wynika:

$$Korona(C_1) \wedge NaGłowie(C_1, Jan),$$

pod warunkiem, że C_1 jest nowym symbolem stałej zwanej **stałą Skolema**.

Jeśli kwantyfikator egzystencjalny formuły poprzedzony jest kwantyfikatorem uniwersalnym dla pewnej zmiennej x to w miejsce zmiennej za v podstawiamy unikalny symbol funkcji zwanej **funkcją Skolema** o parametrze x. Wyrażamy to w postaci reguły wnioskowania jako:

$$\frac{\forall_x \exists_v \alpha}{SUBST(\{v/F(x)\},\alpha)}$$

Postać normalna CNF i reguła rezolucji

Drugą postacią normalną dla zdań (lub formuł) jest **koniunkcyjna postać normalna** (ang. *Conjunctive Normal Form*, **CNF**) będąca koniunkcją alternatyw literałów.

Np.: w rachunku zdań poniższe zdanie jest w postaci CNF:

$$(\mathrm{A} \vee \neg \mathrm{B}) \wedge (\mathrm{B} \vee \neg \mathrm{C} \vee \neg \mathrm{D})$$

Postać CNF jest odpowiednia dla procedur wnioskowania korzystających z reguły rezolucji, czyli dla wnioskowania nie wprost, inaczej mówiąc: dowodzenia formuły zapytania przez zaprzeczenie.

Reguła rezolucji w rachunku zdań jest postaci:

$$\frac{l_1 \vee \ldots \vee l_k\ ,\ m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

gdzie l_i i m_j są komplementarnymi literałami ($l_i = \neg m_j$).

Np.: jeśli zachodzą zdania: $P_{13} \vee P_{22}$ i $\neg P_{22}$ to wnioskujemy stąd, że zachodzi P_{13}.

Pokażemy poprawność reguły rezolucji stosując jedynie przekształcenia zdań.

1) Zakładamy, że zachodzi poprzednik. Stosując równoważność, $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$, przejdziemy z postaci po prawej stronie do lewej strony dla obu zdań w poprzedniku. Wyłączamy przy tym komplementarne literały od reszty literałów. Otrzymujemy:

$\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i$

$$\neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

2) Z założenia, $l_i = \neg m_j$, i z przechodniości implikacji, $\alpha \Rightarrow \beta \Rightarrow \gamma$, wynika:

$\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$

3) Ponownie stosujemy równoważność z pkt. 1) ale teraz przekształcamy implikację z pkt. 2) na postać alternatyw literałów:

$$l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n .$$

Tym samym uzyskaliśmy postać następnika reguły rezolucji co kończy dowód.

Przykład . Konwersja zdania do postaci normalnej CNF. Przekształcimy zdanie, $B_{11} \Leftrightarrow (P_{12} \vee P_{21})$.

1. Usuwamy równoważność $\Leftrightarrow$, zamieniając $\alpha \Leftrightarrow \beta$ na $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Usuwamy obie implikacje $\Rightarrow$, zamieniając $\alpha \Rightarrow \beta$ na $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Wprowadzamy $\neg$ do środka nawiasów stosując reguły de Morgana i ewentualnie eliminujemy podwójną negację:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Stosujemy prawo rozdzielczości ($\wedge$ nad $\vee$) i rozpisujemy:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Uzyskaliśmy zdanie w postaci CNF.

Uogólniona reguła rezolucja, właściwa dla logiki predykatów, jest postaci:

$$\frac{l_1 \vee \ldots \vee l_k \, , \, m_1 \vee \ldots \vee m_n}{SUBST(\theta, (l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n))}$$

gdzie $\mathrm{UNIFY}(l_i, \neg m_j) = \theta$.

Zakłada się, że formuły są w postaci klauzul Horna, tzn. wszystkie l_i, m_i są literałami, a dwie formuły w poprzedniku zostały standaryzowane rozłącznie, tzn. nie mają wspólnych zmiennych.

Np. Z pary formuł w postaci CNF, $\neg Bogaty(x) \vee Nieszczęśliwy(x)$, i , $Bogaty(Jan)$, o unifikowalnych komplementarnie literałach $\neg Bogaty(x)$ i $Bogaty(Jan)$, wnioskujemy zachodzenie formuły: $Nieszczęśliwy(Jan)$, przy zastosowaniu podstawienia: $\theta = \{x/Jan\}$.

Wnioskowanie z regułą rezolucji prowadzone jest nie wprost i polega na sprawdzeniu warunków niespełnialności formuły będącej zaprzeczeniem formuły zapytania. Jeśli α jest formułą zapytania to jej zaprzeczenie dodawane jest do bazy wiedzy (jest ona wtedy postaci CNF(KB $\wedge \neg\alpha$)) a następnie stosuje się iteracyjnie regułę rezolucji. Jeśli zostanie wygenerowana formuła pusta to będzie oznaczać, że formuła zapytania zachodzi.

Przykład. Konwersja formuły do postaci CNF. Zdanie „*każdy kto kocha wszystkie zwierzęta jest kochany przez kogoś*" wyrazimy w postaci formuły logiki predykatów jako:

$$\forall x \, [\forall y \, Zwierz(y) \Rightarrow Kocha(x,y)] \Rightarrow [\exists y \, Kocha(y,x)].$$

1. Eliminacja obu implikacji:

$$\forall x\ [\neg\forall y\ \neg Zwierz(y) \vee Kocha(x,y)] \vee [\exists y\ Kocha(y,x)]$$

2. Przesuwamy $\neg$ w prawo: $\neg\forall x\ p \equiv \exists x\ \neg p,\ \neg\ \exists x\ p \equiv \forall x\ \neg p$

$$\forall x\ [\exists y\ \neg(\neg Zwierz(y) \vee Kocha(x,y))] \vee [\exists y\ Kocha(y,x)]$$

$$\forall x\ [\exists y\ \neg\neg Zwierz(y) \wedge \neg Kocha(x,y)] \vee [\exists y\ Kocha(y,x)]$$

$$\forall x\ [\exists y\ Zwierz(y) \wedge \neg Kocha(x,y)] \vee [\exists y\ Kocha(y,x)]$$

3. Standaryzacja rozłączna zmiennych: każdy kwantyfikator korzysta z innej zmiennej

$$\forall x\ [\exists y\ Zwierz(y) \wedge \neg Kocha(x,y)] \vee [\exists z\ Kocha(z,x)]$$

4. Skolemizacja: każda egzystencjalna zmienna (i jej kwantyfikator) jest zastępowana przez funkcję Skolema dla zmiennej należącej do poprzedzającego kwantyfikatora uniwersalnego:

$$\forall x\ [Zwierz(F(x)) \wedge \neg Kocha(x,F(x))] \vee Kocha(G(x),x)$$

5. Pomijamy uniwersalny kwantyfikator:

$$[Zwierz(F(x)) \wedge \neg Kocha(x,F(x))]\ \vee Kocha(G(x),x)$$

6. Rozdzielamy $\vee$ względem $\wedge$:

$$[Zwierz(F(x)) \vee Kocha(G(x), x)] \wedge [\neg Kocha(x,F(x)) \vee Kocha(G(x),x)]$$

Uzyskaliśmy postać CNF, w tym przypadku jest to iloczyn dwóch klauzul.

7. Ewentualnie z klauzul usuwamy każdy literał o postaci $\neg$True i False. Usuwamy też klauzule zawierające literał $\neg$False lub True.

3.3. Wnioskowanie metodą rezolucji

Procedury wnioskowania (dowodzenia) formuł stosujące regułę rezolucji prowadzą dowód przez zaprzeczenie, tzn. aby dowieść, że (KB $\models \alpha$) pokazują one, że zdanie (KB $\wedge\neg\alpha$) jest niespełnialne. Schemat algorytmu wnioskowania przez rezolucję pokazuje tabela 3-1.

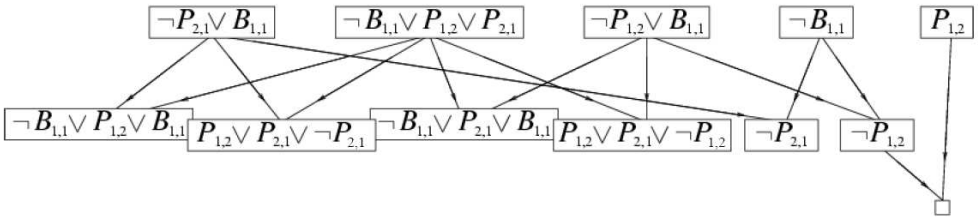
Tab. 3-1. Procedura wnioskowania przez rezolucje w rachunku zdań.

```
funkcja Rezolucja(KB, α) zwraca wynik: True lub False
{
  zdania ← zbiór zdań w postaci CNF dla (KB ∧ ¬α);
  nowe ← {};
  while ( True ) {
     for each (Ci, Cj ∈ zdania ) {
        rezolwenty ← KrokRezolucji(Ci, Cj);
        if (rezolwenty zawierają zdanie puste) return True;
        nowe ← nowe ∪ rezolwenty;
     }
     if (nowe ⊆ zdania ) return False;
     zdania ← zdania ∪ nowe;
  }
}
```

W funkcji Rezolucja() najpierw zamieniamy zdania w bazie wiedzy rozszerzonej o negację zapytania, (KB $\wedge\neg\alpha$), na postać CNF. Następnie w pętli, dla każdej odpowiedniej pary zdań, funkcja KrokRezolucji() generuje *rezolwentę* swoich 2 argumentów, tzn. zdanie w następniku reguły rezolucji, które jest pozbawione pary komplementarnych symboli. Są możliwe dwa warunki zakończenia procedury: 1) nie można dodać nowych zdań, wtedy α jest fałszywe w modelu M(KB) lub 2) w

wyniku rezolucji powstaje zdanie puste, co wynika ze sprzeczności w bazie; a to oznacza, że α jest prawdziwe w modelu M(KB).

Przykład wnioskowania metodą rezolucji. Niech baza wiedzy agenta zawiera m.in..: $B_{11} \Leftrightarrow (P_{1,2} \vee P_{2,1})$, i $\neg B_{11}$. Chcemy dowieść, że zachodzi zdanie: $\alpha = \neg P_{12}$. Na rys. 3.2. górny rząd zawiera zdania powstałe po normalizacji $(KB \wedge \neg \alpha)$ do postaci CNF. Dolny rząd zawiera *rezolwenty* par zdań zawierających komplementarne literały. Kolejny rząd zawiera zdanie puste powstałe z rezolucji pary P_{12} i $\neg P_{12}$. Dowodzi to, że zdanie zapytania jest spełnione w modelu bazy wiedzy, tzn. $KB \models \alpha$.



Rys. 3.2: Przykład kroków rezolucji [6]. Wygenerowana zostanie formuła pusta.

3.4. Wnioskowanie z regułą odrywania

Procedury wnioskowania korzystające z reguły odrywania stosują dowodzenie wprost. Wyróżnimy tu zasadniczo dwa sposoby takiego wnioskowania:

- procedura progresywna (wnioskowanie w przód) i
- procedura regresywna (wnioskowanie wstecz).

3.4.1. Wnioskowanie w rachunku zdań

Idea procedury wnioskowania **progresywnego (w przód)**:

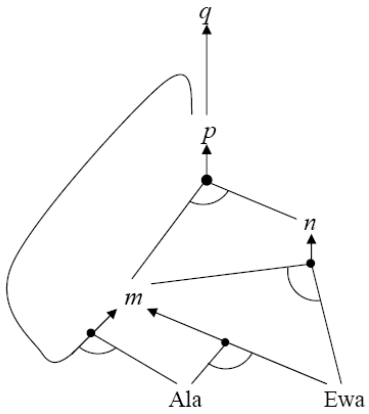
1. wykonaj każdą regułę, której warunek (poprzednik) jest spełniony w KB,
2. dodaj wynik wyprowadzenia (następnik reguły) do KB,
3. kontynuuj kroki 1-2 aż do znalezienia zdania zapytania lub niemożliwości wygenerowania nowych zdań.

Progresywna procedura wnioskowania jest poprawna i zupełna dla bazy wiedzy o postaci klauzul Horna.

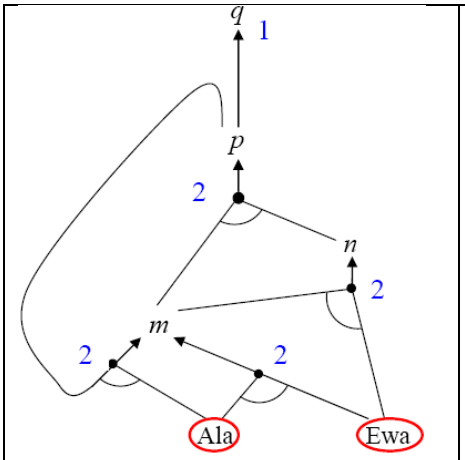
Przykład wnioskowania w przód. Dane są zdania w KB w postaci klauzul Horna:

$$p \Rightarrow q$$
$$m \wedge n \Rightarrow p$$
$$Ewa \wedge m \Rightarrow n$$
$$Ala \wedge p \Rightarrow m$$
$$Ala \wedge Ewa \Rightarrow m$$
$$Ala$$
$$Ewa$$

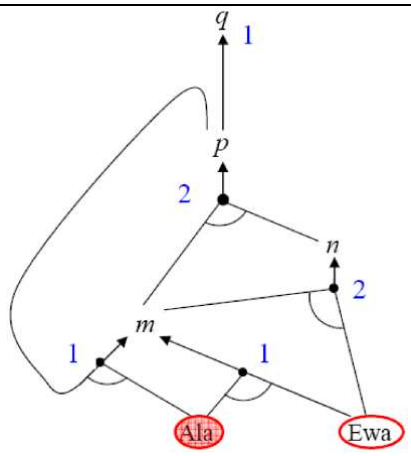
Odpowiada im reprezentacja graficzna w postaci grafu I-LUB (rys. 3.3). Każdy węzeł typu „I" posiada etykietę – odpowiada ona liczbie warunków w poprzedniku reguły pozostałych jeszcze do spełnienia.



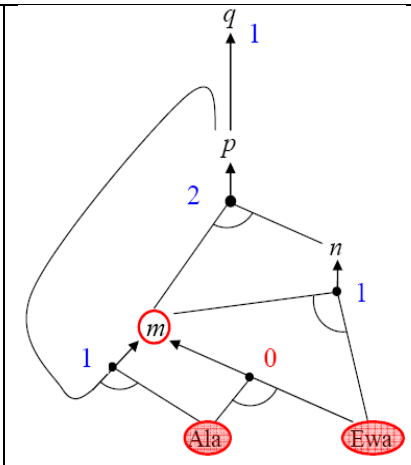
Rys. 3.3: Graf l-LUB dla reprezentacji zbioru klauzul Horna.



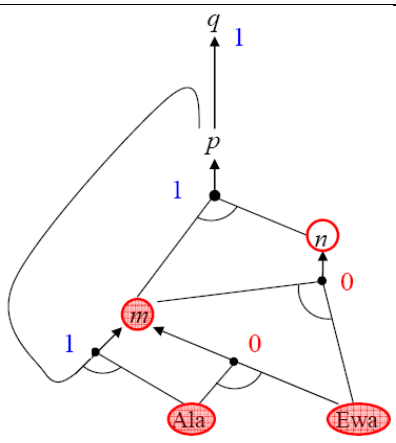
(a) Początkowe etykiety



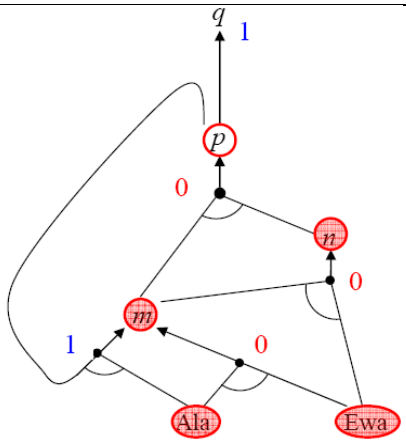
(b) Dodajemy symbol „Ala”



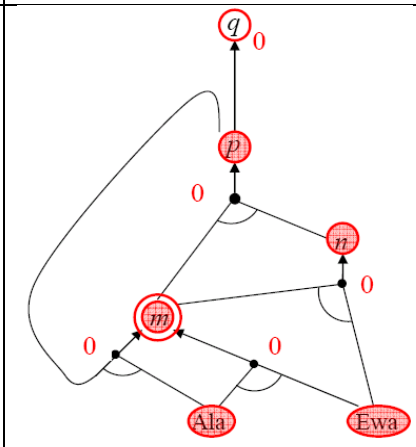
(c) Po dodaniu „Ewa”



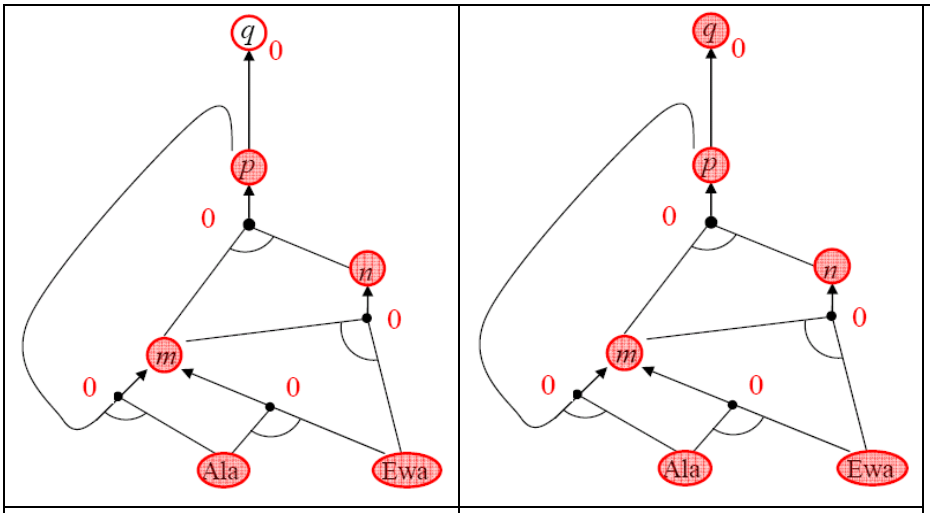
(d) Dodaj *m*



(e) Dodaj *n*



(f) Dodaj *p*



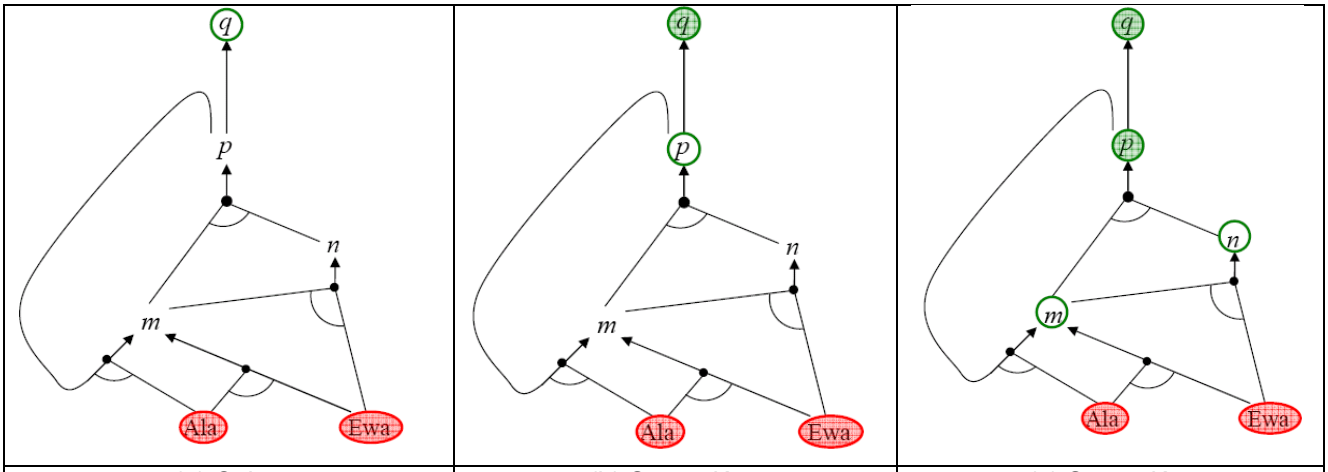
(g) Próbuj ponownie dodać *m* | (h) Dodaj *q*

Rys. 3.4: Przykład wnioskowania „w przód”

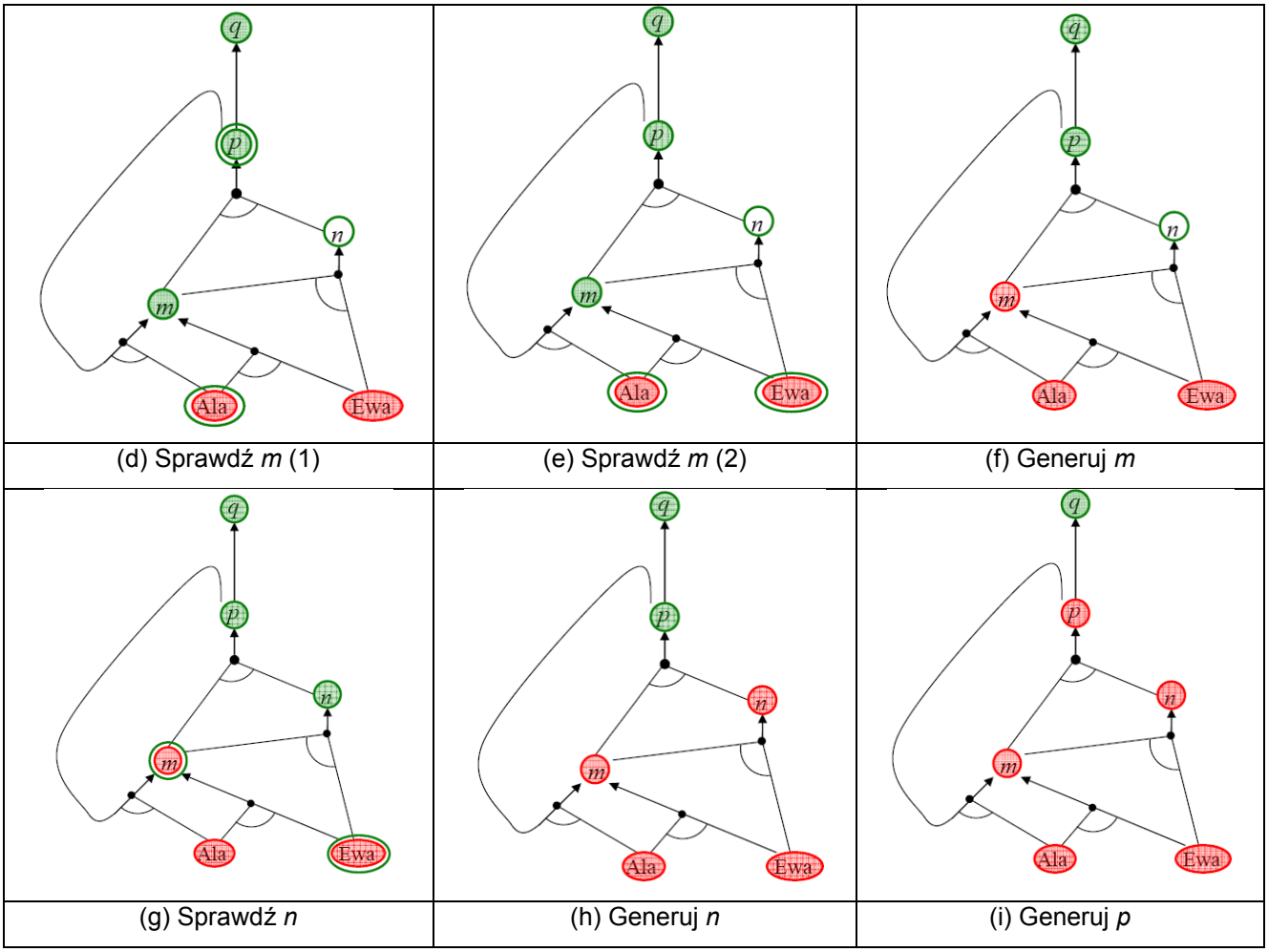
Kolejno wykonywane kroki wnioskowania w przód ilustruje rys. 3.4. Idea procedury wnioskowania „wstecz” w rachunku zdań wygląda następująco:

1. Funkcja rozpoczyna od zdania zapytania (celu) *q*.
2. Aby sprawdzić prawdziwość *q* procedura sprawdza, czy *q* już występuje w KB a jeśli nie, to sprawdza czy istnieje przynajmniej jedna implikacja wyprowadzająca zdanie *q*. Jeśli tak, to literały stanowiące warunek tej implikacji stają się „pod-celami” i rekurencyjne badana będzie ich prawdziwość z punktu widzenia aktualnego modelu KB, podobnie jak poprzednio główny cel.
3. Unikanie zapętleń: procedura sprawdza, czy aktualny „pod-cel” nie znajduje się już na stosie wygenerowanych „pod-celów”.
4. Unikanie powielania przejść: sprawdza, czy nowy „pod-cel” został już sprawdzony i pokazano to, czy jest prawdziwy lub fałszywy.

Przykład wnioskowania „wstecz”. Załóżmy, że baza danych zawiera zdania z poprzedniego przykładu (rys. 3.3). Wśród nich występują też 2 fakty: „Ala” i „Ewa”. Niech zdanie zapytania to „*q*”. Kolejne kroki w procesie wnioskowania wstecz dla tej sytuacji ilustruje rys. 3-5.



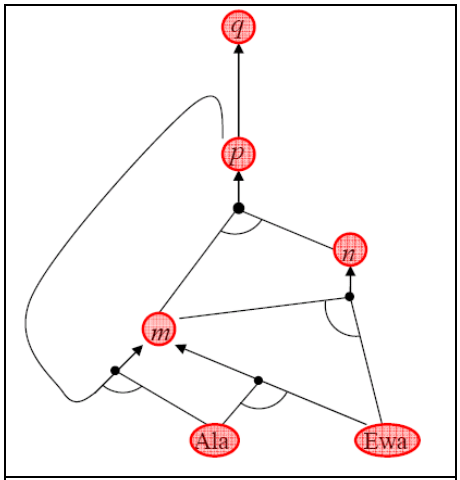
(a) Cel: *q* | (b) Sprawdź: *q* | (c) Sprawdź *p*



(d) Sprawdź *m* (1) | (e) Sprawdź *m* (2) | (f) Generuj *m*

(g) Sprawdź *n* | (h) Generuj *n* | (i) Generuj *p*

Rys. 3.5: Przykład wnioskowania „wstecz” (kroki a-j)



(j) Generuj *q*

Rys. 3.5 (c.d.): Przykład wnioskowania „wstecz” – krok końcowy

Procedura wnioskowania w przód jest sterowana danymi – odpowiada to automatycznemu, „nieświadomemu” przetwarzaniu danych. Np. rozpoznawanie obiektów, rutynowe decyzje. Może wykonywać „nadmiarową” pracę, która nie zmierza bezpośrednio do celu.

Procedura wnioskowania wstecz jest sterowana celem – jest to odpowiednie dla rozwiązywania zadanego problemu. Np. dostarczenie odpowiedzi na pytanie: „Gdzie są moje klucze?" Złożoność procedury regresywnej może w praktyce być poniżej liniowej względem rozmiaru KB.

3.4.2. Wnioskowanie w logice predykatów

W rachunku zdań procedura wnioskowania sprawdza jedynie, czy podane zdanie zapytania wynika z bazy wiedzy czy też nie. Np. ASK(KB, *Brat*(Jan,Piotr)) , Ask(KB, *Lubi*(Anna,Jan)).

W logice predykatów możemy spytać bazę wiedzy o prawdziwość formuły dla wielu obiektów na raz. Mając formułę S pytamy się, czy istnieje podstawienie θ dla którego S wynika z bazy wiedzy. Wywołanie funkcji ASK(KB, S) zwraca wszystkie podstawienia θ takie, że: $KB \models SUBST(\theta, S)$.

Pytania kierowane do bazy wiedzy są typu: kto, gdzie, kiedy?

Np. „*Kto jest bratem Piotra*?": ASK(KB, $\exists$x Brat(x, Piotr)) . Oczekiwana odpowiedź to zbiór podstawień, np.: {{x/Jan}, {x/Stefan}} .

Wnioskowanie wprzód zwykle stosujemy wtedy, gdy nowa formuła p zostaje dodana do bazy wiedzy *KB* (implementujemy to wnioskowanie jako element funkcji **TELL**). Poniższa funkcja wnioskowania WPRZOD() (tabela 3.2) korzysta z reguły „*uogólnione modus ponens*" i znajduje ona nowe literały, które wynikają z $KB \cup \{ p \}$.

Tab. 3-2. Funkcja wnioskowania wprzód w logice predykatów.

```
function WPRZOD(KB, p) zwraca Tak/Nie
{
        if (w KB istnieje już wariant formuły p) then return Tak;
        Dodaj p do KB;
        for ((p1 ∧ p2 ∧ … ∧ pn ⇒q) ∈ KB takie, że zachodzi ∃i UNIFY(pi, p) = θ ) do
                WNIOSKUJ(KB, [p1,..., pi-1,pi+1,…,pn ], q, θ );
        return Nie; // nie było jeszcze zadanej formuły w bazie wiedzy
}
```

```
procedure WNIOSKUJ(KB, warunki, konkluzja, θ)
{
        if (warunki == []) then
        WPRZOD(KB, SUBST(θ, konkluzja));
        else
        for (każde p' ∈ KB takie, że  UNIFY(p', SUBST(θ , Pierwszy(warunki)))) = θ2  do
                        WNIOSKUJ(KB, Reszta(warunki), konkluzja, COMPOSE(θ, θ2));
}
```

Wnioskowanie wstecz stosujemy wtedy, gdy chcemy otrzymać odpowiedź na pytanie zadane do bazy wiedzy (implementujemy funkcję **ASK**). W tab. 3-2 podano schemat procedury wnioskowania wstecz, w którym wywoływana jest rekurencyjna podfunkcja. Parametry funkcji WSTECZ_LISTA() to: baza wiedzy *KB*, lista celów [q], szukane podstawienie {}. Jest to funkcja rekurencyjna, wywoływana najpierw dla celu (formuły zapytania) a następnie dla kolejnych podcelów, przy jednoczesnym rozszerzaniu zbioru podstawień.

Tab. 3-3. Funkcja wnioskowania wstecz w logice predykatów.

```
function WSTECZ(KB, q) : zwraca zbór podstawień;
{
        WSTECZ_LISTA(KB, [q], {});
}
```

```
function WSTECZ_LISTA(KB, cele, θ) : zwraca zbiór podstawień wynik (i odpowiednio rozszerza
                                                    zbiór podstawień θ);
{
        wynik ← ∅;
        if (lista cele jest pusta) then return {θ};
        q←SUBST(θ, Perwszy(cele));
        for (każdy r ∈ KB taki, że STAND_ROZŁĄCZNA(r) = (p1 ∧ p2 ∧ … ∧ pn ⇒q') ∈ KB
                i gdy zachodzi θ' ←UNIFY(q, q') ) do
        wynik ← WSTECZ_LISTA(KB, SUBST(θ', [p1,...,pn|Reszta(cele)], COMPOSE(θ, θ'))
                ∪ wynik;
        return wynik;
}
```

Wnioskowanie wstecz jest potencjalnie niepełne z powodu możliwości istnienia pętli zależności w grafie I-LUB. Dla zabezpieczenia przed taką sytuacją należy sprawdzać aktualny cel z każdym celem pamiętanym na stosie i unikać wielokrotnego umieszczania celu na stosie. Procedura jest tez potencjalnie nieefektywna na skutek możliwego powtarzania celów już opracowanych (zarówno z pozytywnym jak i negatywnym wynikiem). Zabezpieczeniem przed tym jest pamiętanie w procedurze poprzednich wywołań celów (co wiąże się z potrzebą dodatkowej pamięci).

3.5. Pytania

1. Co to są reguły wnioskowania? Wymienić najważniejsze z nich.
2. Na czym polega eliminacja kwantyfikatorów w języku predykatów.
3. Omówić problematykę uzgadniania zmiennych.
4. Na czym polega reguła rezolucji w rachunku zdań? Zilustrować odpowiedź na przykładzie.
5. Co to jest postać normalna CNF. Przedstawić konwersję dowolnego zdania (formuły) do postaci CNF w rachunku zdań i logice predykatów.
6. Omówić regułę odrywania i postać klauzul Horna?
7. Na czym polega uogólniona reguła odrywania. Zilustrować odpowiedź na przykładzie.
8. Przedstawić zasadę wnioskowania metoda rezolucji.
9. Przedstawić ideę procedury wnioskowania w przód i wstecz.

3.6. Zadania

Zad. 3.1

Dane są zdania w bazie wiedzy (uznane za prawdziwe):

*Zdanie R*1: $\neg$ J11

Zdanie R2: W11 $\Leftrightarrow$ (J12 $\vee$ J21)

Zdanie R3: W21 $\Leftrightarrow$ (J11 $\vee$ J22 $\vee$ J31)

*Zdanie R*4: $\neg$W11

*Zdanie R*5: W2,1

Metodą tablicy prawdy wyznaczyć model aktualnej bazy wiedzy ograniczonej do zdań *R*1 – *R*5.

Zad. 3.2

Zdefiniować w rachunku zdań aksjomaty ukrytych własności dla *świata Wumpusa.* Aksjomaty „ukrytych własności" łączą widoczne obserwacje z ukrytymi przyczynami (zasadami budowy świata).

Zad. 3.3

Sprawdzić, czy z bazy wiedzy KB wynikają zapytania q_1 i q_2.

KB reguły:
$$P(x,y) \wedge Q(y,z) \Rightarrow R(x,y)$$
$$R(x,y) \wedge S(z,v) \wedge W(y,v) \Rightarrow W(x,z)$$
$$\neg Q(x,y) \Rightarrow Q(y,x)$$

KB obserwacje:
$$P(Ania, Beata), \quad \neg Q(Czarek, Beata)$$
$$S(Darek, Ewa), \quad W(Czarek, Ewa)$$

Zapytania:
$$q_1: \quad W(a,d)$$
$$q_2: \quad W(d,e) \Rightarrow W(a,d)$$

Zad. 3.4

Przeprowadzić skolemizację następującej formuły:

$$\forall x \forall y \, (P(x,y) \rightarrow \exists z(\forall y \, Q(y,z) \wedge \neg R(x,z))) \vee Q(x,y,z)$$

Zad. 3.5

Zakładamy regułę modelowanego świata: „*Według prawa przestępstwo popełnia Amerykanin, który sprzedaje broń wrogim narodom"*.

Znamy fakty: *Kraj Nono, wróg Ameryki, posiada rakiety, które sprzedał im pułkownik West, będący Amerykaninem.*

Sprawdzić, czy *„pułkownik West jest przestępcą",* stosując procedurę wnioskowania w przód w logice predykatów.

Zad. 3.6

Zilustrować działanie funkcji wnioskowania wstecz w logice predykatów na przykładzie formuł utworzonych dla problemu opisanego w zadaniu 3.5 i zapytania „ ∃x *Przestępca*(x)"?

Zad. 3.7

Zilustrować działanie funkcji wnioskowania metodą rezolucji w logice predykatów na przykładzie formuł utworzonych dla problemu w zadaniu 3.5 i zapytania „*Przestępca*(West)"?

4. Rozszerzenia logiki klasycznej

Logika niemonotoniczna

Logika modalna

Logika opisowa

Zadania

4.1. Logika niemonotoniczna

Logika klasyczna jest **monotoniczna**:

„jeśli formuła *A* wynika ze zbioru formuł ***X***, to *A* wynika także z każdego nadzbioru zbioru ***X*** :

$$\text{if KB} \models \alpha \text{ then (KB} \wedge \beta) \models \alpha .$$

Własność monotoniczności jest powiązana z założeniem o **zupełności** bazy wiedzy (**zamknięty świat**):

- brakujące literały są niespełnione (False),
- unikamy reprezentowania w bazie wiedzy wielu negatywnych literałów.

Problem: jednak w praktyce w wielu dziedzinach wiedza ma charakter **niemonotoniczny** – dodanie nowej przesłanki (obserwacji) może unieważnić dotychczas przyjęte założenia.

Jest to zgodne z założeniem „**otwartego świata**”:

- spełnianie brakujących literałów nie jest z góry znane;
- **niemonotoniczność** zachodzi wtedy, gdy mogą być wyjątki od ogólnej reguły.

Przykład logiki niemonotonicznej: **logika „domyślna”** (ang. „default logic”)

Np. wiedza domyślna o „świecie zwierząt” może polegać na tym, że jeśli obiekt jest kotem to ma cztery nogi. Wprowadzamy tę własność świata do bazy wiedzy w postaci **reguły** definiującej relację, nazwaną np. *Rel4 :*

$$\forall r,a,b\ Rel4(r,a,b) \Leftrightarrow [r,a,b] \in \{[Nogi, Koty, 4], \ldots\}.$$

Jeśli dopuszczamy, że od tej reguły mogą być wyjątki – to logika jest potencjalnie ***niemonotoniczna*** – dodanie nowej przesłanki może unieważnić dotychczas wyprowadzone zdania. Np. dodanie obserwacji *Nogi*(*kotek, 3*) unieważnia wyprowadzony poprzednio fakt, że *kotek* ma 4 nogi.

W logice „domyślnej” wprowadza się dwa typy reguł wnioskowania:

- *zwykłe*: mają one znaną nam postać

$$\frac{Przeslanka}{Konkluzja},$$

  czyli są uporządkowaną parą zdań:

- *domyślne*: mające postać uporządkowanej trójki zdań:

$$\frac{Przeslanka : Uzasadnienie}{Konkluzja}$$

gdzie zdanie *Uzasadnienie* odgrywa rolę uprawomocnienia (ang. *justification*).

Normalna postać reguły *domyślnej* to:

<A : B /B>,

co oznacza, że: „możemy wyprowadzić B z A o ile B jest niesprzeczne z tym, co już jest wiadome.

4.2. Logika modalna

Logiki modalne stanowią rozszerzenie logiki predykatów o tzw. *operatory modalne*, takie jak operator „przekonania" („jest możliwe") B (ang. „believes") i „wiary" („jest konieczne") K (ang. „knows"), których argumentami są *zdania logiczne* a nie *termy.*

Wnioskowanie w logice modalnej ogranicza możliwość podstawiania pod zmienne w kontekście takich wyrażeń modalnych.

W logie predykatów (L1R) modelujemy przekonania (*sądy*) agenta za pomocą relacji typu: *Wierzy* (Agent, *x*), *Wie*(Agent, *x*), *Chce*(Agent, *x*).

Np. chcemy tym wyrazić, że: „*agent* wie, że *p", „agent* wierzy, że *p", „agent* chce aby, *p"*.

Czyli chcemy wyrazić relację pomiędzy agentem i formułą (lub zdaniem). Ale możemy napisać, *Wierzy* (Agent*, x*), jedynie pod warunkiem, że *x* jest **termem.**

Np. *Wierzy*(Agent*, Lata*(*Superman*)) jest prawidłowe pod warunkiem, że *Lata*(*Superman*) jest termem.

Związek pomiędzy „przekonaniem" a „wiedzą" może być rożnie definiowany. Filozofowie definiują wiedzę jako „udowodniona prawdziwość przekonania". Wyrazimy to następująco:

$$\forall a,p \;\; Wiedza(a,p) \Leftrightarrow Wierzy(a,p) \wedge Prawda(p) \wedge Prawda(KB(a) \Rightarrow p)$$

Problem: logika klasyczna jest **ekstensjonalna** tzn. wartość wyrażenia jest funkcją wartości jego podwyrażeń.

Prowadzi to np. do następującego wynikania: z faktu, że

(*złodziej* = Kowalski)

i z przekonania agenta,

Wie(*Agent, TymOknemWszedł(złodziej)*)

wynikałoby, że

Wie(*Agent, TymOknemWszedł(*Kowalski*)*).

Taka cecha logiki może prowadzić do generowania zupełnie niepotrzebnych zdań.

Intensjonalność w praktyce: zdarza się, że wartość logiczna zdania złożonego, w którym występuje czyjeś „przekonanie", nie jest jednoznacznie wyznaczona przez wartości logiczne zdań składowych.

Rozwiązanie powyższego problemu może być: zastosowanie logiki modalnej, która jest intensjonalna lub też ograniczenie zakresu termów reprezentujących obiekty-własności tylko do działania na symbolach stałych.

Przykład logiki modalnej

W modalnym rachunku zdań wprowadzimy dwa dodatkowe jednoargumentowe operatory (spójniki):

- „jest możliwe, że" – spójnik możliwości, ozn. ◊
- „jest konieczne, żeby" – spójnik konieczności, ozn. □.

Są one wzajemnie ze sobą powiązane (**aksjomaty**):

$$\Diamond A \Leftrightarrow \neg \Box \neg A$$

$$\Box A \Leftrightarrow \neg \Diamond \neg A$$

Za ich pomocą zdefiniujemy spójnik ścisłej implikacji:

$$(A \rightarrow B) \Leftrightarrow \Box(A \Rightarrow B)$$

$$(A \rightarrow B) \Leftrightarrow \neg\Diamond(A \wedge \neg B)$$

Zachodzi także:

$$\Box A \rightarrow A$$

$$\Diamond A \rightarrow \Box \Diamond A$$

$$\Box (A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B),$$

Reguły wnioskowania:

$$\frac{A}{\Box A},$$

$$\frac{(A \rightarrow B),\ A}{B}$$

4.3. Logika opisowa

Logika opisowa jest pewnym rozstrzygalnym podzbiorem logiki predykatów. Wyróżnia się w niej dwa zbiory wyrażeń:

TBox i **ABox.**

TBox obejmuje terminologię ontologii, w skład której wchodzą:

1. zbiór pojęć (konceptów) – kategorie obiektów
2. zbiór ról (atrybuty obiektów) – binarne relacje
3. zbiór aksjomatów definiujących ograniczenia nałożone na koncepty i role.

ABox obejmuje aktualny opis świata.

Każdy język z rodziny języków logiki opisowej zawiera:

- Pojęcia reprezentują kategorie, czyli zbiory obiektów.
- Pojęcia (koncepty) atomowe, w tym koncept uniwersalny $\top$ (Top), reprezentujący uniwersum, oraz koncept pusty $\bot$ (Bottom), który nie może mieć żadnych wystąpień; pojęcia atomowe są nazwami elementarnych pojęć.
- Role atomowe (atrybuty); reprezentują relacje między parami obiektów.
- Konstrukcjie (operatory) służące do tworzenia złożonych konceptów i ról. Można użyć wielu własności (tzn. pojęć nadrzędnych lub ograniczeń atrybutów) równocześnie w definicji pojęcia.

Przykład logiki opisowej: język ALC

Konstruktor	Znaczenie
$\neg C$	negacja pojęcia
$C \sqcap D$	część wspólna pojęć
$C \sqcup D$	suma pojęć
$\exists R.C$	Kwantyfikacja egzystencjalna: zbiór takich osobników, które są powiązane przynajmniej jeden raz rolą R z osobnikiem należącym do konceptu C
$\forall R.C$	Kwantyfikacja ogólna: zbiór takich osobników, których wszystkie istniejące powiązania rolą R dotyczą osobników należących do konceptu C (obejmuje także takie osobniki, które nie są powiązane rolą R z żadnymi osobnikami)

Przykład opisu świata „rodzina”.

Pojęcia: Osoba, Mężczyzna, Kobieta i Rodzic.

Role: maDziecko, maSyna

Tbox	ABox
Mężczyzna $\sqsubseteq$ Osoba	Kobieta (Anna)
Kobieta $\sqsubseteq$ Osoba	Kobieta (Joanna)
Kobieta $\sqcap$ Mężczyzna $\equiv \bot$	Mężczyzna (Karol)
Rodzic $\equiv$ Osoba $\sqcap \exists$maDziecko.Osoba	maDziecko (Anna, Joanna)
Ojciec $\equiv$ Mężczyzna $\sqcap$ Rodzic	maDziecko (Anna, Karol)
Matka $\equiv$ Kobieta $\sqcap$ Rodzic	

Aksjomaty:

1. $\text{maSyna} \sqsubseteq \text{maDziecko}$

2. „Syn zawsze jest mężczyzną”: $\exists \text{maSyna}.\neg\text{Mężczyzna} \equiv \bot$

3. „Tylko rodzice mogą mieć synów”: $\exists \text{maSyna}.\top \sqsubseteq \text{Rodzic}$

Fakty: $\text{maSyna}(\text{Karol}, \text{Jan})$

Wnioskowanie

Pytanie: types(Jan)

Odpowiedź: Mężczyzna, Osoba (podaje wszystkie pojęcia z terminologii z wystąpieniem Jan.

Pytanie: types(Karol). Odpowiedź: Rodzic, Mężczyzna, Osoba.

4.4. Zadania

Zad. 4.1

Załóżmy istnienie poniższych definicji w logice opisowej:

1. Kobieta ≡ Osoba ∩ PłećŻeńska

2. Mężczyzna ≡?

3. Matka ≡ Kobieta ∃ maDziecko.Osoba

4. Ojciec ≡?

5. Rodzic ≡?

6. Babcia ≡?

7. MatkaBezCórki ≡ ?

8. Żona ≡?

Uzupełnić brakujące definicje.

Zad. 4.2

W definicjach z rozwiązania zad. 4.1 zastąpić definicje złożone definicjami ich elementarnych składowych. Np. jeśli po prawej stronie w definicji występuje pojęcie „Kobieta" zastąpimy je odpowiednią definicją „Osoba ∩ PłećŻeńska".

Po rozwinięciu wszystkich definicji sprawdź czy występują w nich cykle, tzn. czy pojęcie nie jest zdefiniowane w sposób zależny od samego siebie.

Np. załóżmy, że zdefiniujemy Osobę jako:

Osoba ≡ Mężczyzna ∪ Kobieta

Co wtedy zmieni się?

Część II: Przeszukiwanie i planowanie

5. Przeszukiwanie jako realizacja celu

Przeszukiwanie przestrzeni stanów

Przeszukiwanie poinformowane

Składowe heurystyczne

Przeszukiwanie lokalne poinformowane

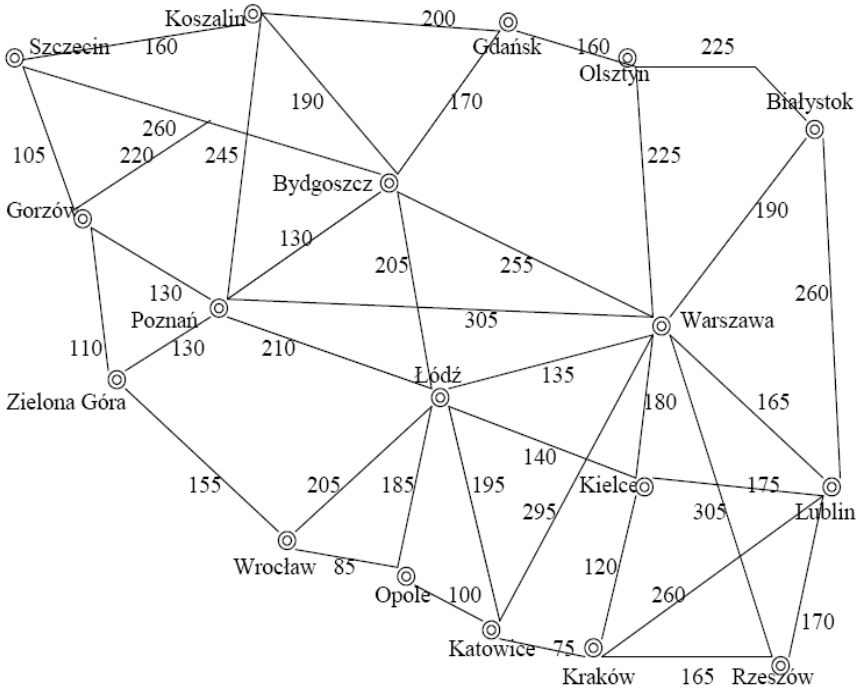
Symulowane wyżarzanie

Pytania

5.1. Przeszukiwanie przestrzeni stanów

W tym rozdziale omówimy metodykę „przeszukiwania przestrzeni stanów" jako generalny (abstrakcyjny) sposób działania agenta realizującego cel. O ile tylko uda nam się wyrazić konkretny problem z konkretnej dziedziny w terminach tej metodyki to będziemy mogli natychmiast skorzystać z istniejącego sposobu rozwiązania problemu.

Przykład. Problem przejazdu ze Szczecina do Krakowa. Załóżmy, że aktualnie jesteśmy w Szczecinie i chcemy pojechać do Krakowa transportem drogowym. Celem działania jest: *dotrzeć do Krakowa*. Kluczową sprawą jest wyrażenie możliwych rozwiązań problemu w postaci **grafu stanów**, gdzie pojedynczy stan oznacza „*agent przebywa w danym mieście*". Dla uproszczenia wyróżnimy tylko większe miasta w Polsce (rys. 5.1). Możliwe **akcje** (operatory w przestrzeni stanów) oznaczają: „*przejazd z miasta A do miasta B*". Rozwiązaniem będzie każda ścieżka w grafie stanów prowadząca od stanu początkowego do stanu końcowego. W tym ostatnim spełniony jest **warunek zatrzymania** działania. Rozwiązaniem może być np. sekwencja akcji dla przejazdu: (Szczecin, Bydgoszcz, Łódź, Katowice, Kraków).



Rys. 5.1: Przykład grafu stanów dla problemu „przejazdu z miasta A do miasta B".

5.1.1. Problem przeszukiwania

Teraz podamy definicję **problemu przeszukiwania**. Problem wyrażony jest w postaci 4 pojęć (rodzajów danych):

1. Zbiór stanów S z wyróżnionym stanem początkowym; np. Szczecin $\in$ S, o znaczeniu: „*agent jest w Szczecinie*" ;
2. Akcje (operacje w przestrzeni stanów), $A = \{a_1, a_2, \ldots a_n\}$, i funkcja następnika stanu:

   $$[(stan\_we, a) \rightarrow stan\_wy] \in S \times A \times S$$

   Np.: (Szczecin, „*ze Szczecina do Bydgoszczy*") $\rightarrow$ Bydgoszcz
3. Warunek osiągnięcia celu T, który jest spełniony w każdym stanie końcowym. Może on bezpośrednio referować stan, np., T: s = „*Kraków*", lub badać własność stanu, np. T: *Szachmat*(s) = true .
4. Koszt każdej akcji i sposób obliczenia kosztu rozwiązania (najczęściej jest to suma kosztów akcji tworzących rozwiązanie); np., suma odległości w km pomiędzy miastami, liczba wykonanych akcji, itp.

   Niech, $c(x, a, y)$, oznacza koszt akcji. W algorytmach realizujących różne strategie przeszukiwania zazwyczaj postuluje się, aby koszt akcji był nieujemny, $c \geq 0$.

Rozwiązaniem problemu jest sekwencja akcji (ścieżka w przestrzeni stanów) prowadząca od stanu początkowego do końcowego.

Uwaga: przedstawiliśmy tu definicję problemu **jednostanowego**, w którym każdy stan przeszukiwania odpowiada jednemu stanowi problemu. Taki przypadek zachodzi wtedy, gdy środowisko działania agenta jest w pełni obserwowalne. Jak pamiętamy z rozdz. 1, w sytuacji, gdy środowisko nie jest w pełni obserwowalne mamy do czynienia z problemem wielostanowym.

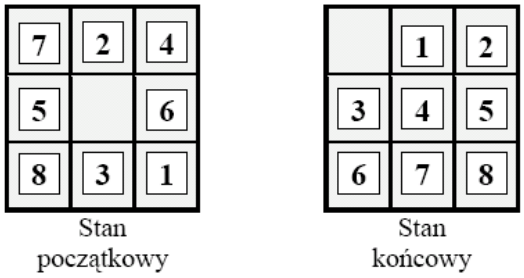
Wybór przestrzeni reprezentacji problemu

Rzeczywisty problem (świat) jest bardzo złożony a podana powyżej definicja problemu przeszukiwania obejmuje jedynie niezbędne elementy opisu tego świata. Podczas projektowania systemu sztucznej inteligencji trzeba stworzyć uproszczony (abstrakcyjny) model świata, w którym:

- Abstrakcyjny stan odpowiada wielu sytuacjom rzeczywistym;
- Abstrakcyjna akcja odpowiada wielu rzeczywistym akcjom; np., „(Szczecin, „przejazd") $\rightarrow$ Bydgoszcz" reprezentuje zbiór możliwych dróg, objazdów, miejsc odpoczynku, itd.
- Abstrakcyjne rozwiązanie odpowiada zbiorowi rzeczywistych dróg, które w rzeczywistym świecie prowadzą do celu.

Tym samym każde z abstrakcyjnych pojęć stanowi zwykle uproszczenie oryginalnego pojęcia w rzeczywistym świecie.

Przykład. Definicja problemu przeszukiwania dla świata „8-puzzli" (rys. 5.2).



Rys. 5.2: Przykład stanów dla problemu „8-puzzli".

Stany problemu reprezentują konfiguracje 8 numerowanych kafelków (płytek) w kwadratowym obszarze o rozmiarze 3×3. Pojedyncza akcja polega na „przemieszczeniu" pustego miejsca w lewo, prawo, na górę lub na dół (dualnie: jest to przesunięcie kafelka sąsiadującego z pustym miejscem). Warunek zatrzymania (stopu) jest podany jawnie w postaci stan końcowego, w którym numery kafelków uporządkowane są w kolejności rosnącej. Koszt każdej akcji wynosi 1.

Rozmiary różnych wersji „świata puzzli" mogą być dowolnie duże (oznaczmy rozmiar przez N). Problem polega na optymalnym (w sensie minimalnego kosztu potrzebnych akcji) uporządkowaniu „świata N-puzzli". Rozwiązanie takiego problemu jest NP-trudne, czyli o złożoności obliczeniowej powyżej wielomianowej względem N. Z naszego punktu widzenia problem „świata N-puzzli" stanowi dogodną ilustrację dla porównywania różnych **strategii przeszukiwania** przestrzeni stanów.

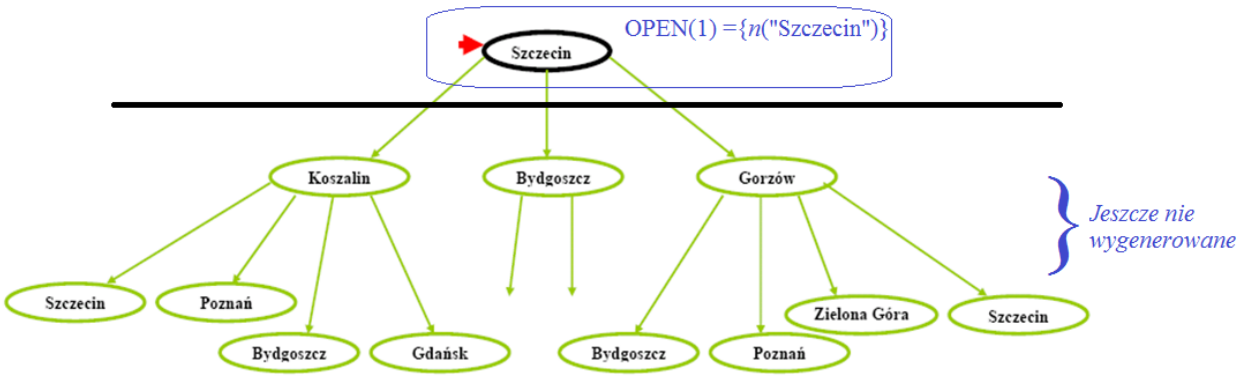
5.1.2. Przeszukiwanie drzewa

W zasadzie każdą strategię przeszukiwania dyskretnej przestrzeni problemu możemy wyrazić jako wizytowanie (przejście) pewnego **drzewa** lub **grafu**. Oczywiście struktura drzewa jest uproszczoną formą grafu. Dlatego najpierw ograniczymy nasze rozważania dotyczące strategii przeszukiwania do postaci przejścia po drzewie a następnie rozszerzymy je o dodatkowe elementy wymagane przy przejściu grafu.

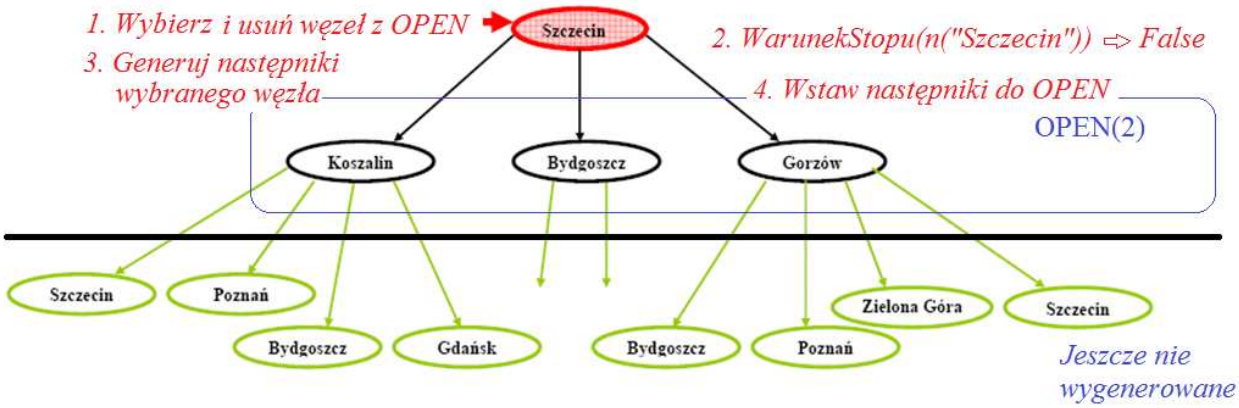
Podczas przeszukiwania przestrzeni reprezentujemy wyniki częściowe w postaci węzłów **drzewa przeszukiwania**, a alternatywne akcje przeprowadzają aktualny węzeł w jeden z możliwych węzłów-następników tego drzewa. Rozwijamy drzewo przeszukiwania począwszy od korzenia, reprezentującego *stan początkowy*, poprzez węzły pośrednie do liści drzewa, z których przynajmniej jeden reprezentuje *stan końcowy,* czyli stanowi cel przeszukiwania.

W problemie „jedno-stanowym" *węzeł* drzewa przeszukiwania odpowiada pojedynczemu stanowi problemu. Aktualny zbiór węzłów-liści, posiadających jeszcze niewizytowane następniki, to zbiór węzłów gotowych do rozwinięcia, utożsamiany z tzw. **skrajem** drzewa (często w algorytmach przeszukiwania nazywany zbiorem OPEN). Sposób porządkowania węzłów w skraju i tym samym sposób wyboru następnego rozwijanego węzła wyraża określoną strategię przeszukiwania.

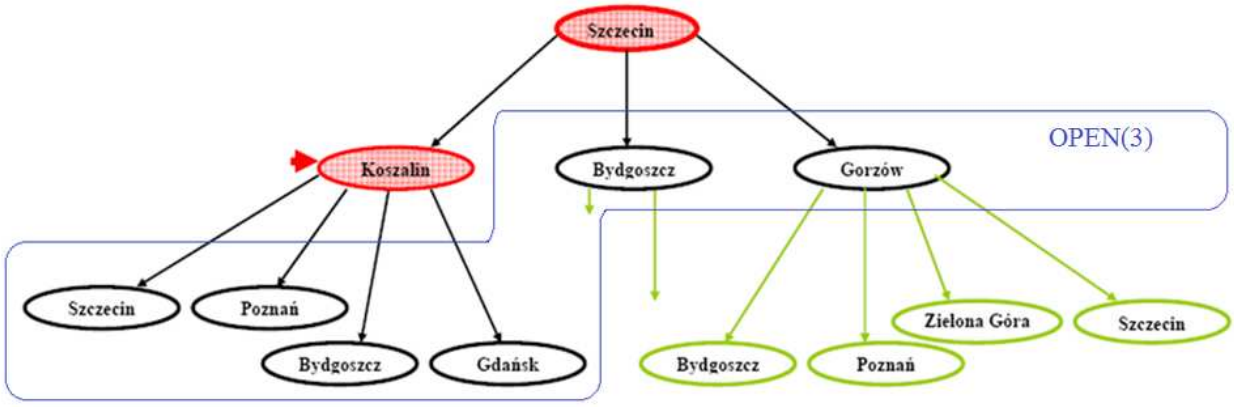
Przykład. Początkowe drzewo decyzyjne dla problemu „powrót ze Szczecina do Krakowa" (rys. 5.3). Węzeł początkowy o etykiecie „Szczecin" reprezentuje sytuację (jest to także jeden stan problemu) „*agent jest w Szczecinie*". Jest to na początku jedyny węzeł w skraju, który jest w pierwszym kroku wybierany do rozszerzenia. Pozostałe widoczne węzły reprezentują możliwe następniki, które zostaną wygenerowane na podstawie opisu problemu (na podstawie grafu problemu) z chwilą wykonania określonej akcji. W pierwszym kroku generowane są następniki węzła „Szczecin", czyli węzły: Koszalin, Bydgoszcz, Gorzów (rys. 5.4.). Załóżmy, że w kolejnym kroku, zgodnie z przyjętą strategią przeszukiwania, wybierany jest węzeł „Koszalin". Nie spełnia on warunku celu więc generowane są jego następniki reprezentujące stany: Szczecin, Poznań, Bydgoszcz, Gdańsk (rys. 5.5). W kolejnych krokach postępujemy podobnie, wybierając węzły znajdujące się w skraju drzewa, sprawdzając dla nich warunek zatrzymania i w przypadku niespełnienia warunku – generując następników węzła.



Rys. 5.3: Przykład początkowego drzewa przeszukiwania dla problemu „powrót ze Szczecina do Krakowa".



Rys. 5.4: Postać drzewa przeszukiwania po wybraniu węzła „Szczecin".

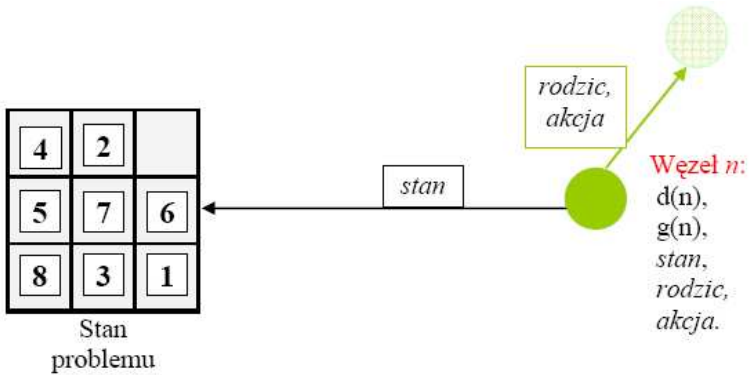


Rys. 5.5: Postać drzewa przeszukiwania po wybraniu węzła „Koszalin".

Zauważmy, ze w powyższym przykładzie możemy wielokrotnie generować węzły reprezentujące ten sam stan problemu. Nie jest to sytuacja pożądana, gdyż może prowadzić do zamkniętych cykli, a te z kolei mogą być powtarzane nieskończenie wiele razy. Dla takich problemów będziemy musieli rozszerzyć problem przeszukiwania drzewa do problemu przeszukiwania grafu.

Wyjaśnijmy jeszcze różnicę pomiędzy **węzłem drzewa** przeszukiwania a **stanem** problemu. Stan jest abstrakcyjną reprezentacją fizycznej konfiguracji świata. Węzeł n jest strukturą danych i elementem drzewa przeszukiwania. Dla przykładu węzeł może zawierać (rys. 5.6):

- odniesienie do stan lub zbioru stanów problemu,
- wskaźnik do węzła rodzica,
- akcję, wykonaną w sytuacji reprezentowanej węzłem rodzica,
- koszt sekwencji akcji prowadzącej do tego węzła $g(n)$ i głębokość węzła $d(n)$.



Rys. 5.6: Przykład struktury węzła w drzewie przeszukiwania.

Charakterystyka strategii przeszukiwania drzewa

Przyjęta strategia przeszukiwania drzewa wyraża się w sposobie wyboru kolejno rozwijanych (wizytowanych) węzłów. Oczywiście dobrze byłoby móc porównać działanie różnych strategii. Podamy tu cztery podstawowe kryteria oceny strategii:

1. **zupełność**;

   Zupełność strategii przeszukiwania oznacza, że jeżeli istnieje rozwiązanie problemu, to zostanie ono zawsze znalezione.

2. **złożoność czasowa**: czas przeszukiwania mierzony całkowitą liczbą węzłów wizytowanych podczas przeszukiwania;
3. **złożoność pamięciowa**: maksymalna liczba węzłów jednocześnie rezydujących w pamięci programu przeszukiwania;
4. **optymalność**;

   Strategia optymalna to taka, która zawsze znajduje najlepsze rozwiązanie.

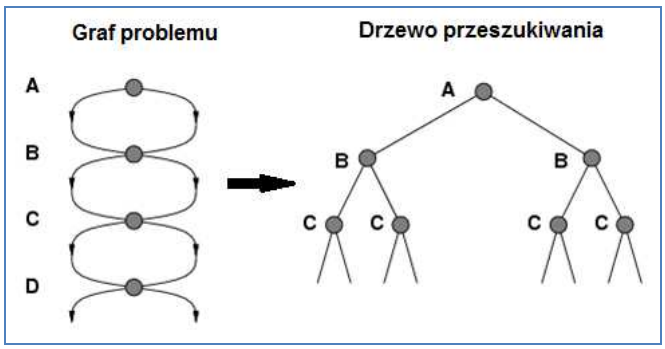
W przypadku kryteriów złożoności czasowej i pamięciowej interesują nas nie tyle złożoności dla konkretnego problemu, wyznaczone po procesie przeszukiwania, lecz oczekiwane złożoności, szacowane dla danej strategii na podstawie znajomości rozmiaru problemu. Przyjęło sie wyrażać rozmiar problemu za pomocą następujących parametrów:

- *b:* maksymalne rozgałęzienia drzewa poszukiwań,
- *d:* głębokość, na której znajduje się najtańsze rozwiązanie,
- *m*: maksymalna głębokość drzewa poszukiwań.

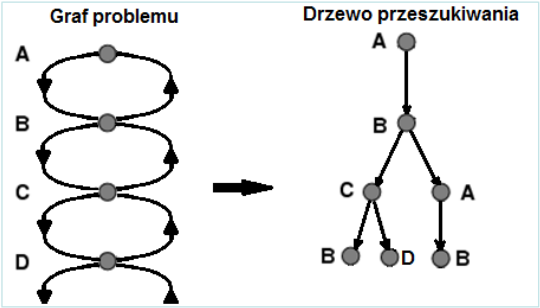
5.1.3. Przeszukiwanie grafu

Jak zauważyliśmy we wcześniejszym przykładzie w procesie przeszukiwania drzewa może dochodzić do sytuacji ponownego generowania węzłów reprezentujących te same stany problemu (rys. 5.7, 5.8). Brak wykrywania takich sytuacji może prowadzić do wielokrotnego powtarzania tej samej sekwencji akcji, czasem nieskończoną ilość razy. Taka sytuacji oznacza konieczność zastąpienia bazowego mechanizmu przeszukiwania drzewa przez przeszukiwanie grafu. Osiągamy to łatwo dokonując następujących zmian i uzupełnień:

1. należy zapamiętać wszystkie wcześniej rozwijane węzły (w tym celu wprowadzamy zbiór węzłów CLOSED),
2. każdy nowo generowany węzeł jest porównywany z węzłami znajdującymi się w zbiorach OPEN i CLOSED; po ewentualnym stwierdzeniu identyczności stanów reprezentowanych dwoma różnymi węzłami, jeden z węzłów (domyślnie ten „gorszy”) jest eliminowany.



Rys. 5.7: Wielokrotne węzły w drzewie – różne ścieżki drzewa przeszukiwania.



Rys. 5.8: Wielokrotne węzły w drzewie – różne ścieżki drzewa przeszukiwania.

5.2. Przeszukiwanie poinformowane

Strategia przeszukiwania jest wyznaczona sposobem wyboru kolejno rozwijanych węzłów w drzewie (grafie) przeszukiwania, tworzonym dla rozwiązania problemu. Istnieje kilka „ślepych" strategii poszukiwania, takich jak: przeszukiwanie wszerz, przeszukiwanie w głąb, przeszukiwanie z jednorodną funkcją kosztu, przeszukiwanie z ograniczoną głębokością i iteracyjnym pogłębianiem. Określenie „ślepy" oznacza w tym przypadku, że w żadnym przypadku na wybór następnego węzła nie wpływa informacja o stanie końcowym. Jest to ich zasadnicza niedogodność, która sprawia, że zwykle nie osiągają one wystarczającej efektywności obliczeniowej czy pamięciowej, a nawet niektóre z nich nie są optymalne. Te strategie przeszukiwania omawiane są podczas studiów I stopnia.

Z analizy słabości strategii przeszukiwania ślepego wynika wniosek: należy użyć funkcji oceny dla węzłów drzewa przeszukiwania, która uwzględniałaby (abstrakcyjnie zdefiniowaną) „odległość" węzła od węzła dla stanu docelowego. Założenie o istnieniu takiej miary kosztów „resztkowych" na ścieżce rozwiązania pozwala zaproponować strategie tzw. przeszukiwania **poinformowanego**. Oszacowanie kosztów resztkowych pozwoli ocenić na ile „obiecujący" z punktu widzenia celu jest dany węzeł i wybrać (rozwinąć) najbardziej „obiecujący" nierozwinięty węzeł.

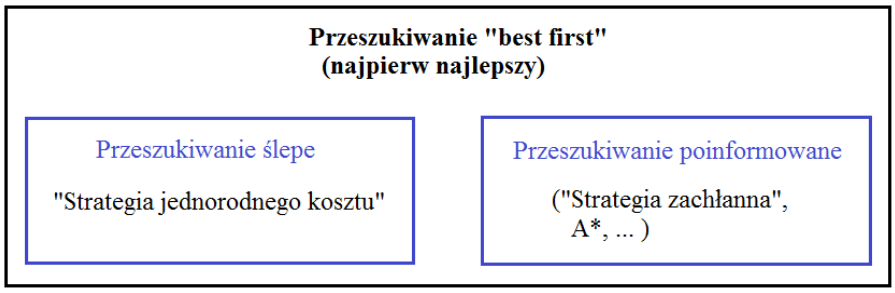
Strategie „najpierw najlepszy" (*best first*)

W ogólności rozpatruje się funkcję oceny węzła (*koszt*) złożoną z 2 części: $f(n) = g(n) + h(n)$, gdzie $g(n)$ oznacza koszt dotychczasowej ścieżki a $h(n)$ oznacza przewidywany koszt resztkowy pozostałej drogi z węzła n do celu. Implementacja każdej strategii, należącej do kategorii „najpierw najlepszy", polega na porządkowaniu węzłów w skraju według zmniejszającej się oceny węzła $f(n)$.

W zależności od postaci funkcji oceny rozpatrzymy trzy odmiany strategii „najpierw najlepszy":

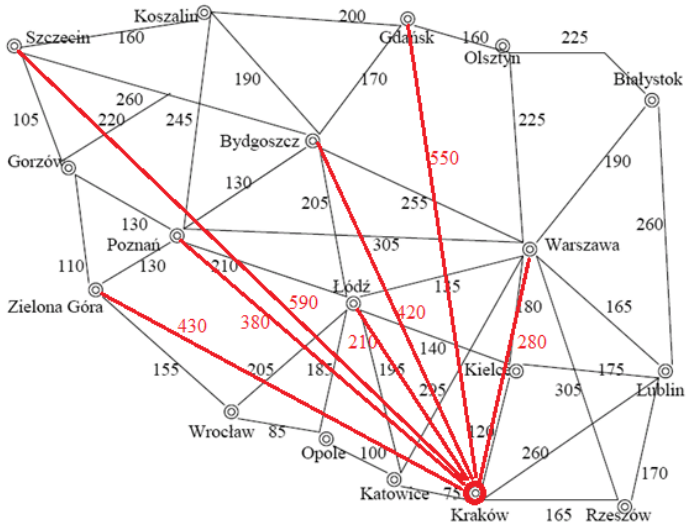
1. strategia jednorodnego kosztu: $f(n) = g(n)$,
2. strategia zachłanna („najbliższy celowi najpierw"): $f(n) = h(n)$,
3. przeszukiwanie A* : $f(n) = g(n) + h(n)$.

Pierwsza strategii z powyższych należy do grupy strategii niepoinformowanych („ślepych”), gdyż nie uwzględnia ona składowej resztkowego kosztu $h(n)$ (tzw. składowej *heurystycznej* funkcji kosztu) (rys. 5.9). Pozostałe dwa przypadki przeszukiwania „*best first*” należą do **strategii poinformowanych**, gdyż korzystają z oszacowań kosztów resztkowych dla każdego węzła przestrzeni przeszukiwania.



Rys. 5.9: Klasyfikacja strategii przeszukiwania typu „best first”.

Przykład składowej heurystycznej kosztu $h(n)$ dla problemu „powrót do Krakowa”: odległość w linii prostej od danego miasta do Krakowa (rys. 5-10).



Białystok	440
Bydgoszcz	420
Gdańsk	550
Gorzów	500
Katowice	70
Kielce	110
Koszalin	580
Kraków	0
Lublin	230
Łódź	210
Opole	150
Poznań	380
Rzeszów	150
Olsztyn	460
Szczecin	590
Warszawa	280
Wrocław	240
Zielona Góra	430

Rys. 5.10. Przykład składowej heurystycznej dla problemu „powrót do Krakowa”.

5.2.1. Strategia zachłanna („najbliższy celowi najpierw”)

W tej strategii funkcja oceny węzła przyjmuje postać:

$$f(n) = h(n) ,$$

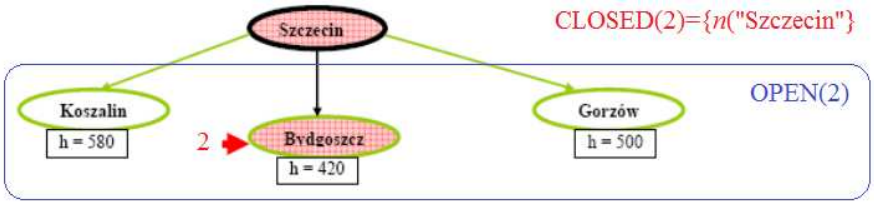
czyli składa się wyłącznie ze składowej heurystycznej (oszacowania kosztów resztkowych przejścia z węzła n do celu). Zakładamy w niej, że najlepszy węzeł to ten, który jest „najbliższy do celu”. Dotychczas poniesione koszty dojścia do aktualnego węzła nie mają żadnego znaczenia dla decyzji wyboru (rys. 5.11).

1	INIT: Pobierz węzeł startowy s i umieść go w zbiorze OPEN. Ustaw $f(s) = h(s)$	
2	Pobierz z OPEN najlepszy węzeł n (o najmniejszym koszcie $f(n)$) i przenieś go do CLOSED	
3	JEŚLI (n jest węzłem końcowym) TO zakończ i zwróć g(n) oraz całą ścieżkę od s do n.	
4	Znajdź węzły następców n - niech będą nimi: $n_1' \ldots n_k'$.	
5	Dla każdego z następców $n_1' \ldots n_k'$ wyznacz jego koszt $f_i = h(n'_i)$	
6	Dla każdego z węzłów $n_1' \ldots n_k'$:	
	a	JEŚLI (n_i' nie należy do zbioru OPEN ani do CLOSED) TO dodaj go do zbioru OPEN i ustaw: $f(n_i') = f_i$.
	b	JEŚLI (n_i' należy do zbioru OPEN lub CLOSED i $f(n_i') > f_i$) TO usuń dotychczasową ścieżkę od s do n_i' i ustaw $f(n_i') = f_i'$. Jeśli n_i' był w zbiorze CLOSED, to umieść go w zbiorze OPEN.
7	Powtórz od kroku 2.	

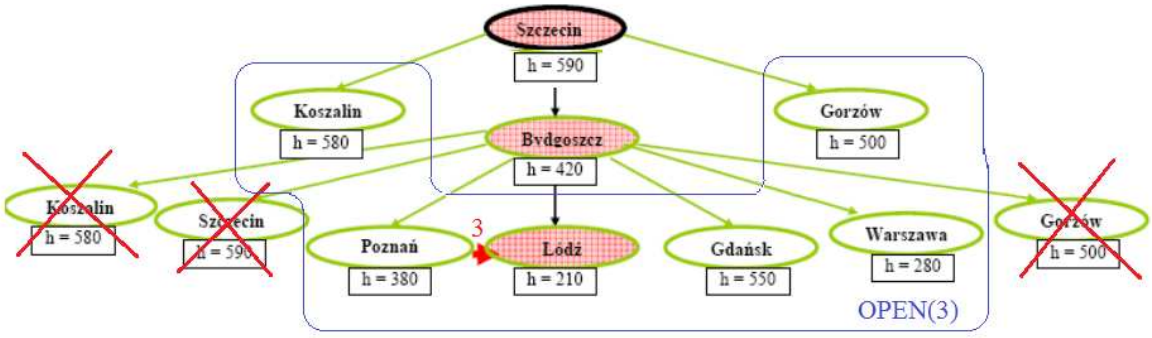
Rys. 5-11. Strategia „zachłanna” poinformowanego przeszukiwania grafu.

Przykład. Strategia „zachłanna” dla problemu „powrót ze Szczecina do Krakowa”. Załóżmy, że graf problemu podano na rys. 5.1, a liczby przy łukach reprezentują koszty akcji przejazdów z miasta do miasta. Dla strategii zachłannej nie mają one żadnego znaczenia. Kieruje się ona jedynie oszacowaniem kosztów resztkowych podanych na rys. 5.10. Kolejne kroki przeszukiwania wykonywane zgodnie z tą strategią, dla podanego problemu, ilustruje rys. 5.12.

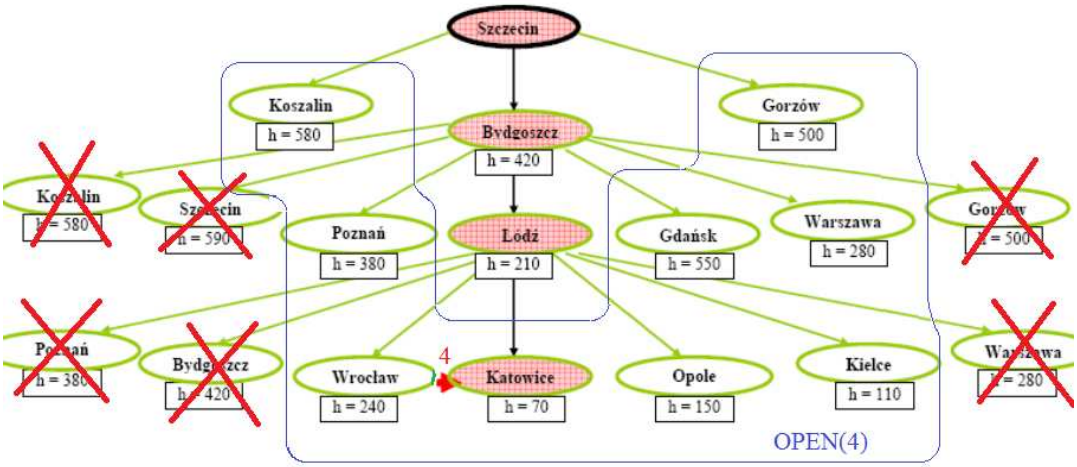
(a)



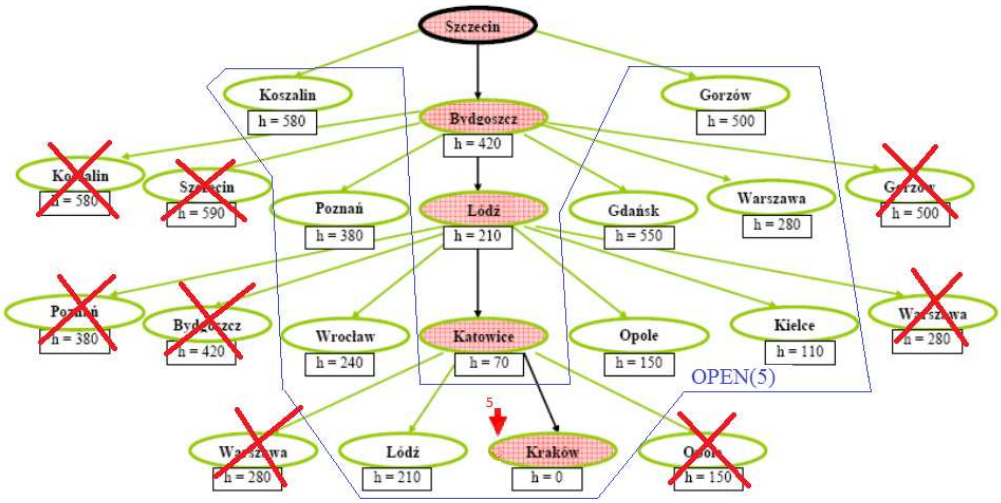
(b)



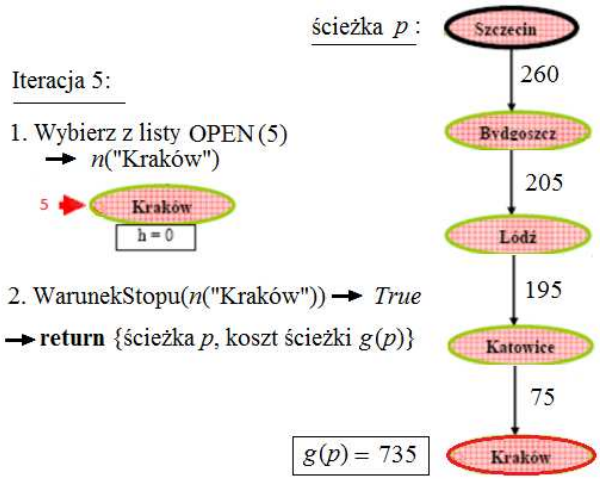
(c)



(d)



(e)



(f)

Rys. 5.12: Ilustracja drzewa przeszukiwania dla strategii zachłannej.

Zbadajmy cechy strategii „zachłannej”:

1. zupełność? Nie. Algorytm może utknąć w pętli;
2. czas? $O(b^m)$, ale dobra heurystyka może dać znaczącą poprawę;

3. pamięć? $O(b^m)$, gdyż utrzymuje wszystkie wizytowane węzły w pamięci.
4. optymalność? nie (np. wybrano drogę przez *Bydgoszcz* zamiast drogi optymalnej przez *Gorzów*).

Zasadniczą wadą strategii zachłannej jest brak gwarancji uzyskania optymalnego rozwiązania (w sensie minimalizacji rzeczywistego sumarycznego kosztu ścieżki rozwiązania). Tej wady nie posiada następna omawiana strategia A*.

5.2.2. Przeszukiwanie A*

Idea strategii A* to: unikać rozwijania ścieżek, które już dotąd są kosztowne a poza tym niezbyt „obiecujące" pod względem możliwości szybkiego dotarcia do celu. Jej realizacja jest możliwa dzięki stosowaniu pełnej funkcji kosztu węzła:

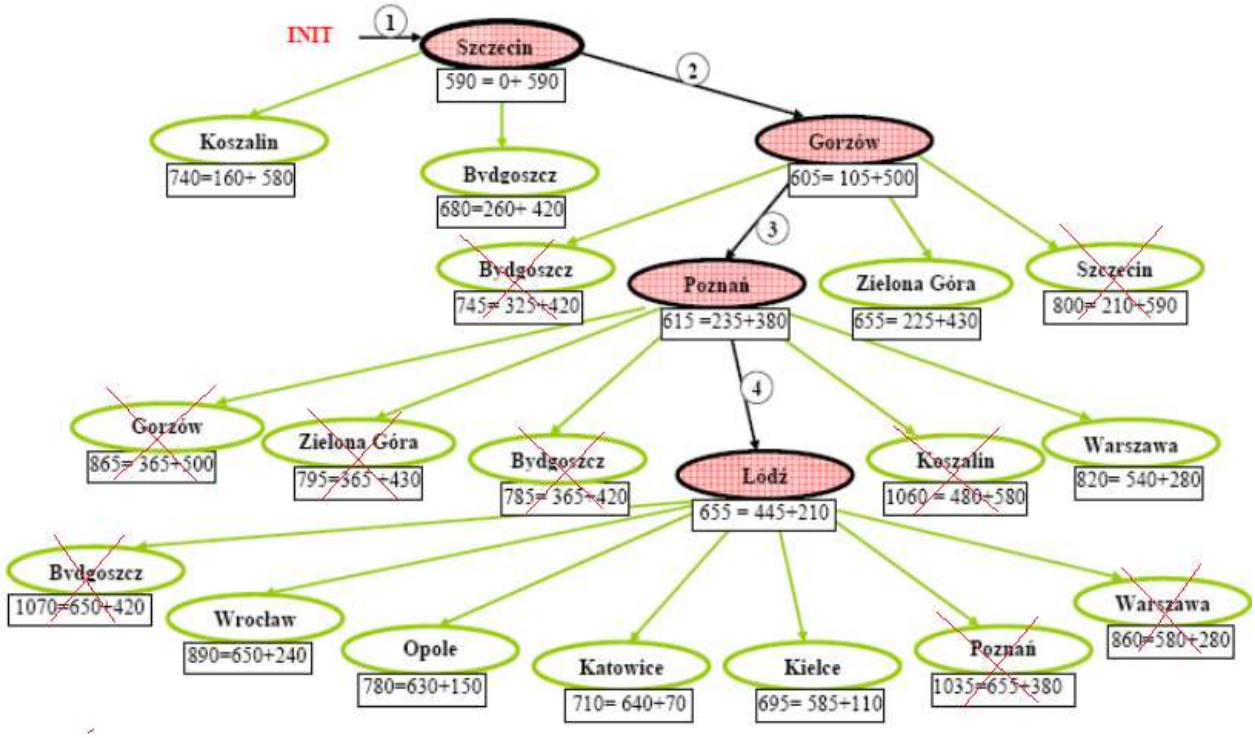
$$f(n) = g(n) + h(n)$$

uwzględniającej zarówno koszt dotarcia do węzła n, (*składowa g(n)*), jak i przewidywany koszt przejścia z n do celu (składowa $h(n)$).

Dzięki temu, że $f(n)$ reprezentuje przewidywany **całkowity koszt** ścieżki prowadzącej od węzła startowego przez węzeł n do celu, a wizytowanie węzłów odbywa się w kolejności rosnącej wartości kosztu, to pod warunkiem **optymistycznego** oszacowania kosztów resztkowych, pierwsze napotkane rozwiązanie będzie jednocześnie najlepszym, czyli optymalnym.

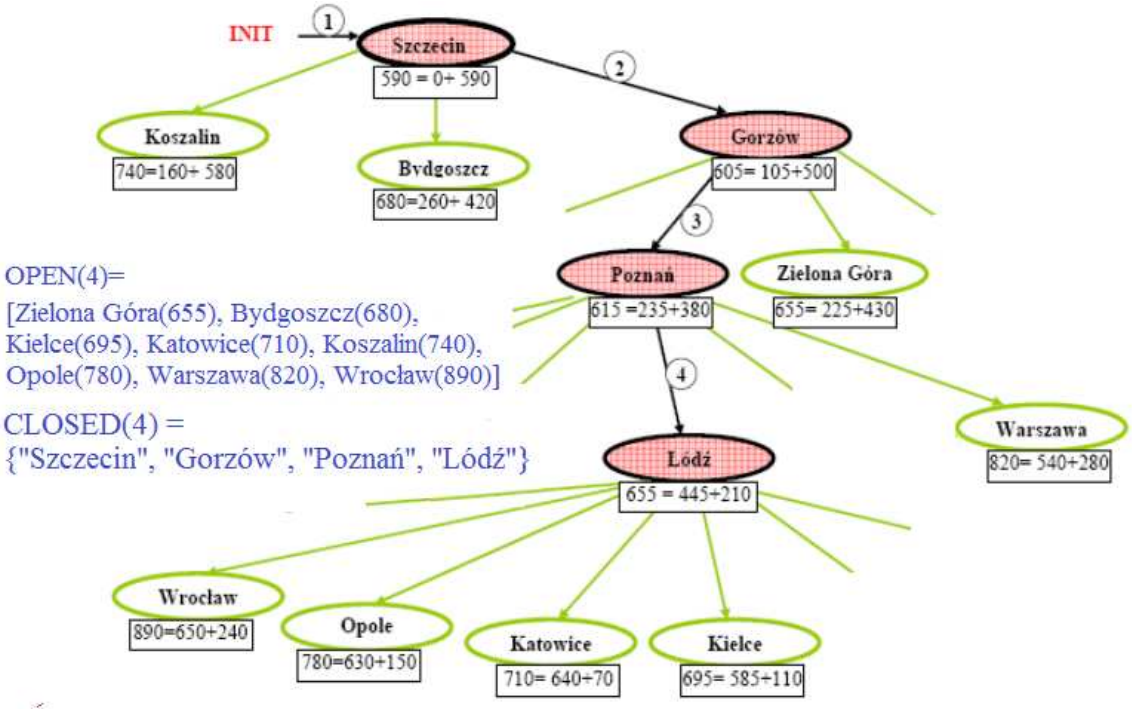
Przykład.

Ilustracja przeszukiwania A* dla problemu „powrót ze Szczecina do Krakowa" (rys. 5.13). Algorytm A* znajduje optymalną ścieżkę o koszcie 705. Dla przypomnienia, algorytm zachłanny znalazł ścieżkę o koszcie 735 (rys. 5.12).

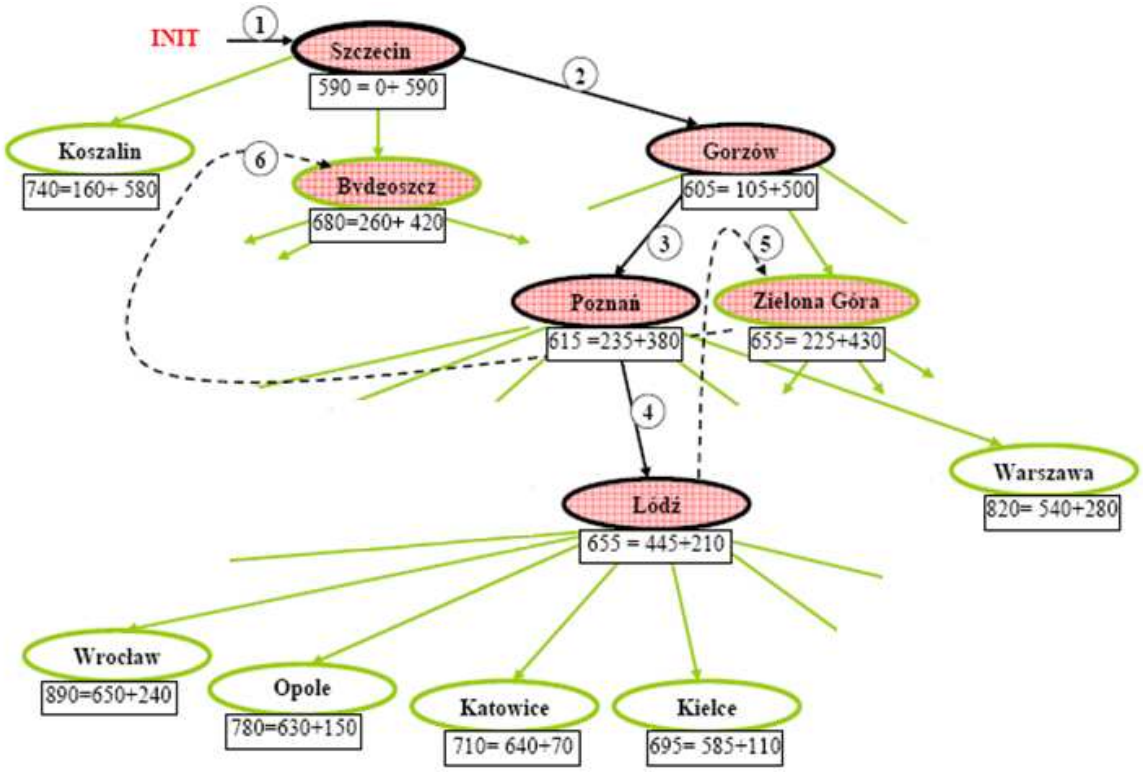


(a) Potencjalne drzewo przeszukiwania (bez usuwania równoważnych węzłów) po czterech iteracjach

Rys. 5.13: Ilustracja drzewa przeszukiwania dla strategii A*.

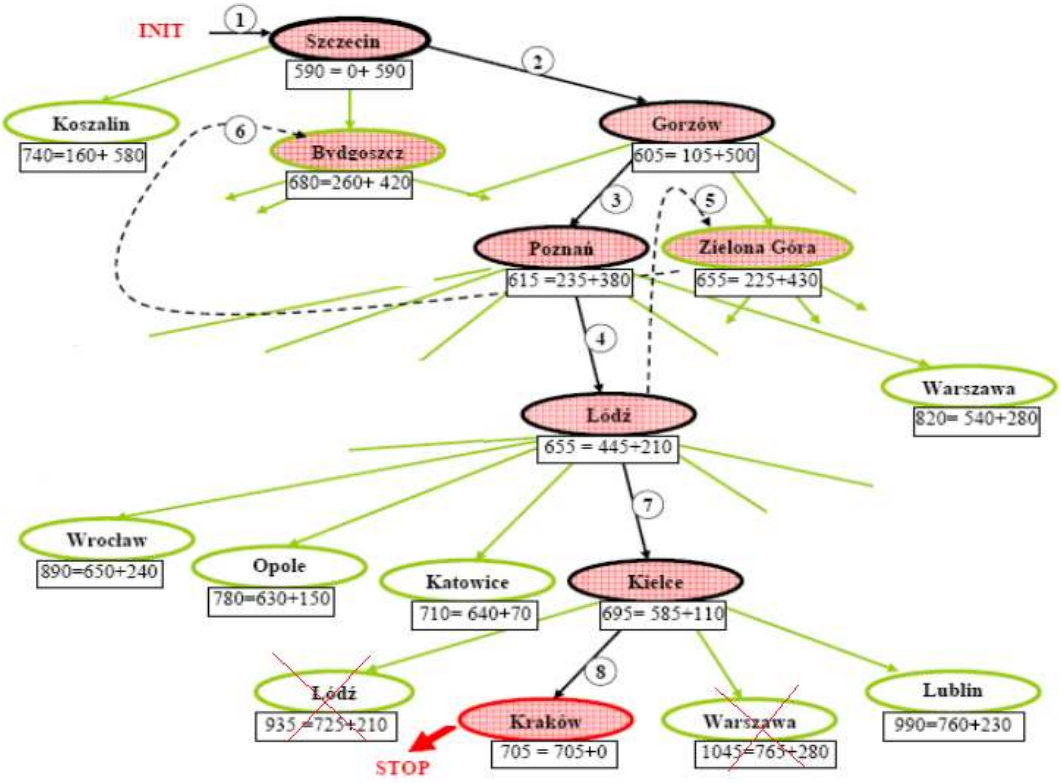


(b) Rzeczywiste drzewo przeszukiwania po czterech iteracjach

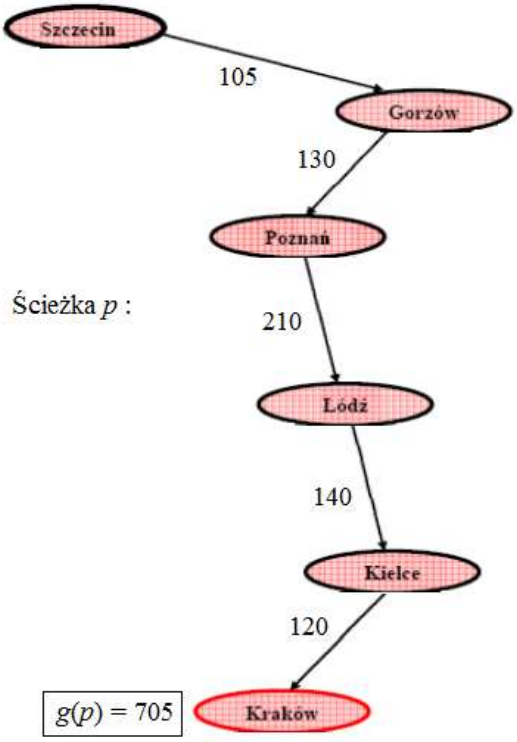


(c) Drzewo przeszukiwania po sześciu iteracjach

Rys. 5.13 (c.d.) Ilustracja drzewa przeszukiwania dla strategii A*.



(d) Po ośmiu iteracjach cel został osiągnięty



(e) Znaleziona ścieżka p o koszcie g(p) = 705.

Rys. 5.13 (c.d.) Ilustracja drzewa przeszukiwania dla strategii A* - końcowe drzewo przeszukiwania i ścieżka będąca rozwiązaniem.

Implementacja strategii A*

Szczegółowy opis algorytmu implementującego strategię przeszukiwania A* podano na rys. 5.14. Dzięki istnieniu zbioru CLOSED, do którego wstawiane są wizytowane węzły, i dzięki krokowi 6.b) uniemożliwiającemu wielokrotne wizytowanie równoważnych węzłów (tego samego stanu problemu), algorytm realizuje przeszukiwanie grafu.

1	INIT: Pobierz węzeł startowy s i umieść go w zbiorze OPEN. Ustaw $f(s)=0$, $g(s)=0$.	
2	Pobierz z OPEN węzeł n o najmniejszej wartości funkcji $f(n)$ i umieść go w zbiorze CLOSED.	
3	JEŚLI (n jest węzłem końcowym) TO zakończ i zwróć g(n) oraz całą ścieżkę od s do n.	
4	Znajdź węzły następców n - niech będą nimi: $n_1' \ldots n_k'$.	
5	Dla każdego z następców $n_1' \ldots n_k'$ oblicz koszt dojścia do niego: $g_i' = g(n) + c(n, ni')$.	
6	Dla każdego z węzłów $n_1' \ldots n_k'$:	
	a	JEŚLI (n_i' nie należy do zbioru OPEN ani do CLOSED) TO dodaj go do zbioru OPEN i ustaw: $g(n_i') = g_i'$, $f(n_i') = g_i' + h(n_i')$.
	b	JEŚLI (n_i' należy do zbioru OPEN lub CLOSED i $g(n_i') > g_i'$) TO ustaw $g(n_i') = g_i'$, $f(n_i') = g_i' + h(n_i')$, usuń ścieżkę od s do n_i'. Jeśli n_i' był w zbiorze CLOSED, to umieść go w zbiorze OPEN.
7	Powtórz od kroku 2.	

Rys. 5.14. Algorytm A* dla przeszukiwania grafu.

Strategia A* posiada bardzo pożyteczne cechy:

- Zupełność? Tak, dla skończonej przestrzeni (nie istnieje nieskończenie wiele węzłów n, dla których, $f(n) \le f(G)$, gdzie G jest optymalnym celem).
- Czas? Potencjalnie wykładniczy ale znaczne zmniejszenie czasu przeszukiwania jest możliwe przy istnieniu dobrej heurystyki (bliskiej rzeczywistym kosztom).
- Pamięć? Wszystkie rozwijane węzły są pamiętane, z uwagi na możliwość wystąpienia cykli w grafie.
- Optymalność? Tak, jeśli heurystyka jest **dopuszczalna** to A* zawsze znajduje najlepsze rozwiązanie. Dalej wyjaśnimy, co oznacza dopuszczalna heurystyka - w skrócie mówiąc oznacza to, że oszacowanie musi być optymistyczne.

Przeszukiwanie A* dysponuje jeszcze jedną ciekawą cechą, jest nią **optymalna efektywność**: przy istnieniu *spójnej* heurystyki dla problemu, żadna inna strategia poinformowanego przeszukiwania nie rozwija mniej węzłów niż algorytm A* dla dotarcia do celu.

Dopuszczalna heurystyka

Heurystyka $h(n)$ jest dopuszczalna (ang. *admissible*), jeżeli dla każdego węzła n zachodzi

$$h(n) \le h^*(n),$$

gdzie $h^*(n)$ jest prawdziwym kosztem osiągnięcia celu z węzła n. Dopuszczalna heurystyka nigdy nie przecenia kosztu osiągnięcia celu, jest więc optymistycznym oszacowaniem rzeczywistego

kosztu. Dla przykładu, heurystyka $h(n)$ podana na rys. 5-10 dla grupy problemów *„powrót do Krakowa”* nigdy nie przecenia faktycznej odległości liczonej jako suma odcinków drogi.

Możemy pokazać, że jeżeli $h(n)$ jest *dopuszczalna* heurystyką, to strategia przeszukiwania A* jest optymalną strategią przeszukiwania grafu. Dowód wynika z kilku obserwacji:

1. Strategia A* rozwija węzły w kolejności nie malejących wartości funkcji oceny (kosztu) f:
$$f_i \leq f_{i+1}$$
2. Strategia A* nie może wybrać węzła o określonym koszcie f_i zanim nie wybierze uprzednio wszystkich węzłów o niższym koszcie.
3. Jeśli heurystyka jest dopuszczalna to koszt każdego częściowego rozwiązania (ścieżki) nie przekracza rzeczywistego kosztu rozwiązania zawierającego tę ścieżkę.
4. Stąd pierwsze wybrane rozwiązanie końcowe nie może być gorsze od żadnego innego rozwiązania. Gwarantuje to osiągnięcie optymalnego rozwiązania

Spójna heurystyka

Heurystyka jest spójna (ang. *consistent*), jeżeli jest dopuszczalna i dla każdego węzła n i dla każdego następnika $n‘$, wygenerowanego przez akcję a, spełniony jest „warunek trójkąta”:

$$h(n) \leq c(n,a,n') + h(n'),$$

gdzie $c(n,a,n')$ oznacza koszt przejścia z n do $n’$ za pomocą akcji a. O koszcie każdej akcji $c(n,a,n’)$ z góry zakładamy, że jest nieujemny. Wtedy spójność h oznacza, że:

1) $h(n) \leq h^*(n)$, czyli heurystyka jest dopuszczalna;

2) $f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$.

Stąd, $f(n') \geq f(n)$, czyli ocena $f(n)$ jest niemalejąca (monotoniczna) wzdłuż dowolnej ścieżki.

Można pokazać, że jeżeli heurystyka $h(n)$ jest spójna, to strategia A* jest efektywnościowo optymalną strategią poinformowanego przeszukiwania grafu. Oznacza to, że żadna inna strategia optymalna, aby osiągnąć cel nie rozwija mniej węzłów niż strategia A*.

5.3. Składowe heurystyczne

5.3.1. Dominująca heurystyka

Przykład. Wybrane składowe heurystyczne dla problemu „8-puzzli”: $h_1(n)$ - liczba kafelków nie na swoim (docelowym) miejscu, $h_2(n)$ - całkowita odległość kafelków od swoich (docelowych) miejsc wyrażona w metryce Manhattan. Dla stanu początkowego na rys. 5.2, wartości obu heurystyk wynoszą:

$$h_1(\text{stan początkowy}) = 8; \qquad h_2(\text{stan początkowy}) = 3+1+2+2+2+3+3+2 = 18$$

Heurystyki możemy ze sobą porównywać. Wprowadźmy określenie **dominującej** heurystyki.

Jeżeli $h_2(n) \geq h_1(n)$ dla każdego węzła n (i obie heurystyki są dopuszczalne) to h_2 dominuje nad h_1.

Wtedy h_2 jest bliższa rzeczywistym kosztom ale pozostaje optymistycznym oszacowaniem i dlatego też jest lepszą heurystyką niż h_1 dla poinformowanej strategii przeszukiwania.

5.3.2. Generowanie heurystyk metodą „złagodzonego problemu”

Zastanówmy się w jaki sposób „zautomatyzować” proces generacji dobrych heurystyk. Dla bardzo złożonych problemów człowiek jest w stanie podać analityczną postać (wzór matematyczny) jedynie dla obliczenia stosunkowo słabych heurystyk, np. dla heurystyki h_1 w problemie „N-puzzli”. Istotne jest też, aby mogła to zrobić maszyna dysponująca ograniczonym zestawem metod obli-

czeniowych. Stąd bierze się idea, aby wyznaczenie dobrej heurystyki potraktować jako oddzielny problem przeszukiwania, ale duży prostszy niż oryginalny problem.

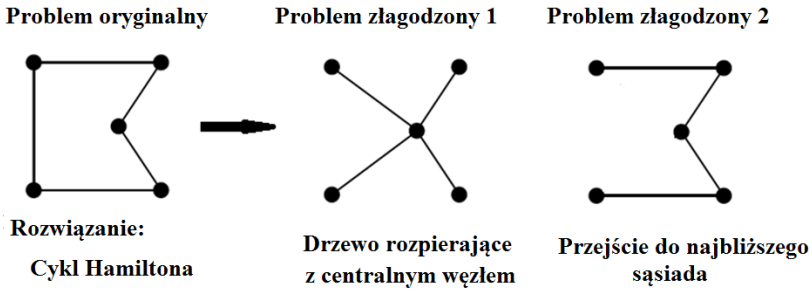
Problem „uproszczony”, o zmniejszonych w porównaniu z oryginalnym problemem wymaganiach nakładanych na przejścia pomiędzy stanami, nazywamy problemem **złagodzonym**. W takiej sytuacji znalezienie optymalnego rozwiązania złagodzonego problemu (dla stanu początkowego n) stanowi jednocześnie dobrą, dopuszczalną heurystykę dla stanu n w oryginalnym problemie.

Np. jeśli złagodzimy problem „8-puzzli” tak, że kafelek może zostać przesunięty w dowolne miejsce, to heurystyka $h_1(n)$ dla oryginalnego problemu powstaje z rozwiązania problemu złagodzonego i wynosi tyle co koszt optymalnego rozwiązania rozpoczynanego w stanie n.

Podobnie, jeśli złagodzimy problem „8-puzzle” tak, że kafelek może zostać przesunięty w dowolną sąsiednią pozycję (tzn. nawet zajętą), to heurystyka $h_2(n)$ dla oryginalnego problemu odpowiada kosztowi optymalnej ścieżki w tym problemie złagodzonym.

Przykład.

Generowanie heurystyk metodą „złagodzonego problemu”. Dany jest „problem komiwojażera” – należy znaleźć najkrótszą trasę odwiedzenia *n* miast, odwiedzając każde miasto jedynie raz (jest to problemem o wysokiej złożoności $O(n!)$) (rys. 5.15). Problem złagodzony wobec oryginalnego problemu – wyznaczyć minimalne drzewo rozpierające dla (pod-)zbioru węzłów pozostałych do odwiedzenia (jest to problem o złożoności jedynie $O(n^2)$). Znalezione rozwiązanie jest optymistycznym oszacowaniem kosztu resztkowego w problemie oryginalnym, gdy pozostaje jeszcze wizytowanie zadanego (pod-)zbioru miast.



Rys. 5.15 Problem oryginalny i dwa problemy złagodzone względem niego.

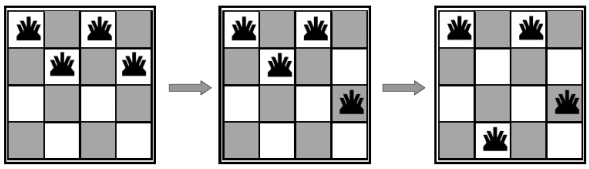
5.4. Przeszukiwanie lokalne poinformowane

W wielu problemach ścieżka prowadząca do celu sama w sobie nie jest taka ważna, jak sam fakt osiągnięcia celu. Np. w grach, przestrzeń stanów definiowana jest jako zbiór „pełnych" konfiguracji pionków. Celem gry (poszukiwanym rozwiązaniem) jest znalezienie konfiguracji spełniającej pewne warunki. W takich sytuacjach możemy zastosować strategie przeszukiwania **lokalnego**, które zapamiętują jedynie pojedynczy stan „aktualny” i starają się go w sposób iteracyjny poprawiać.

W przeciwieństwie do dotychczas przez nas rozpatrywanych strategii globalnych, w których skraj drzewa obejmował wszystkie aktualne węzły niekońcowe, strategie lokalne operują na lokalnych skrajach, złożonych wyłącznie z następników aktualnie wybranego węzła. Dlatego też rozwiązanie znalezione w przeszukiwaniu lokalnym nie musi być globalnie optymalne.

Przykład.

Problem *n*-królowych - ustawić *n* królowych na szachownicy o rozmiarze *n* × *n* tak, aby żadne dwie królowe nie znalazły się w tym samym wierszu, kolumnie i przekątnej (w zasięgu bicia) (rys. 5.16).



Rys. 5.16: Ilustracja pojęć „stan” i „przejście pomiędzy stanami” w problemie 4 królowych.

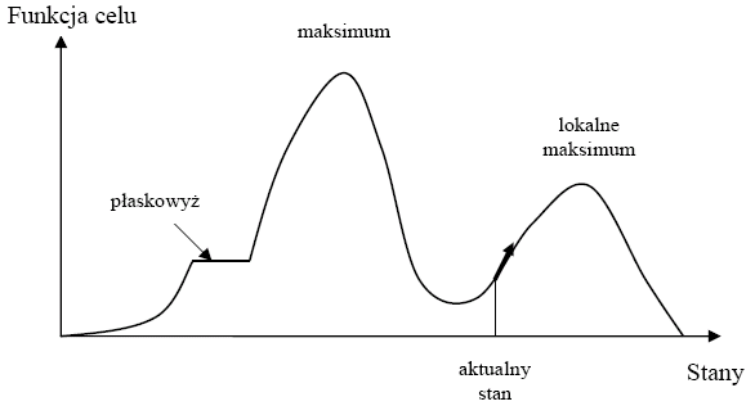
Przeszukiwanie przez „wspinanie”

Strategia lokalnego poinformowanego przeszukiwania określana jest poglądowo zdaniem: „*Wspinanie się na Mt. Everest w gęstej mgle będąc dotkniętym amnezją*” [4]. „Wspinanie się” oznacza wybór najlepiej ocenionego węzła następnika, „gęsta mgła” określa lokalny charakter skraju drzewa przeszukiwania, „amnezja” oznacza brak pamiętania przeszłych akcji i wcześniej odwiedzanych węzłów. Implementacja tej strategii nosi nazwę „przeszukiwanie przez wspinanie” (ang. *hill climbing*) i została zarysowana w tabeli 5-1.

Tab. 5-1. Algorytm „przeszukiwania przez wspinanie”.

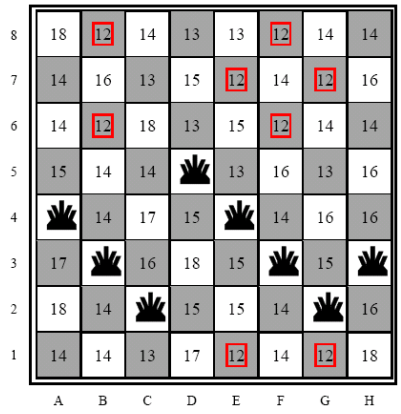
```
function HillClimbing (problem) returns stan;
{
    węzełAktualny ← Węzeł(StanPoczątkowy([problem]));
    while (True)
    {
            sąsiad ← NajlepszyNastępca(węzełAktualny);
            if (Ocena(sąsiad) ≤ Ocena(węzełAktualny)
            then    return Stan(węzełAktualny);
            węzełAktualny ←sąsiad;
    }
}
```

Na rys. 5-17 zilustrowano zasadniczą wadę każdej strategii lokalnej - możemy utknąć w lokalnym optimum (zamiast minimalizacji kosztu zastosowano tu dualnie maksymalizację funkcji celu).



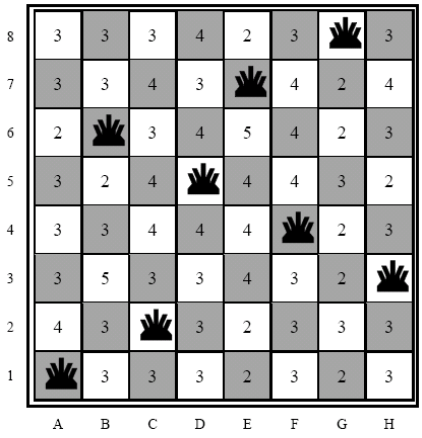
Rys. 5.17: Ilustracja lokalnego optimum znalezionego w przeszukiwaniu lokalnym.

Przykład. Przeszukiwanie przez „wspinanie” w zastosowaniu do problemu 8-królowych. Niech h oznacza liczbę atakujących się wzajemnie par hetmanów (bezpośrednio lub pośrednio).



Rys. 5.18: Przeszukiwanie lokalne zastosowane w problemie „8 królowych”. Przykład stanu początkowego o koszcie 17 i koszty możliwych stanów następnych po przesunięciu jednej królowej.

Ponieważ stan początkowy (rys. 5.18) zawiera po jednej królowej w każdej kolumnie, więc uprościmy problem ograniczając ruch każdej królowej jedynie do swojej kolumny. Ocena stanu początkowego wynosi: $h = 17$. Liczba w każdym wolnym polu podaje ocenę h dla stanu powstałego ze stanu początkowego po przesunięciu do niego królowej znajdującej się w danej kolumnie. Czerwoną ramką zaznaczone zostały najlepsze oceny następników stanu początkowego, $h = 12$. Wybierana jest akcja przesunięcia królowej na najlepszą pozycję, czyli jedną z tych o ocenie 12. Iteracyjnie powtarzamy ten cykl (generowanie następników, ich ocena, wybór najlepszego) dopóki możliwa jest poprawa funkcji oceny. Jak wiemy, strategia lokalna nie gwarantuje osiągnięcia globalnego optimum. W tym przykładzie końcowy wynik, odpowiadający lokalnemu minimum funkcji oceny, jakie można osiągnąć z zadanego stanu początkowego to $h = 1$, podczas gdy idealne rozwiązanie posiada ocenę h=0. Na rys. 5.19 podano stan końcowy osiągany w tym przykładzie. Nie ma już możliwości poprawy oceny, gdyż wszystkie stany następne mają gorszą ocenę od aktualnego.



Rys. 5.19: Wynik przeszukiwania lokalnego dla stanu początkowego z rys. 5.18.

5.5. Symulowane wyżarzanie

Zasygnalizujemy tu inną grupę strategii rozwiązywania problemów, które zawierają niedeterministyczne (stochastyczne) elementy. Jedną z nich jest strategia „symulowanego wyżarzania”. Można ją podsumować jako próbę poprawienia wad lokalnego poinformowanego przeszukiwania poprzez wysoce prawdopodobne wydostanie się z lokalnego optimum przez początkowo częste wykonywanie przypadkowych „złych” akcji, stopniowe zmniejszane wraz z obniżaniem się parametru „temperatury”.

Strategia „symulowanego wyżarzania” realizuje niedeterministyczne przejścia pomiędzy stanami. Jeśli losowe przejście poprawia sytuację to jest na pewno wykonywane, w przeciwnym razie wykonywane jest z pewnym prawdopodobieństwem mniejszym niż 1, zależnym od aktualnej wartości

parametru *T* (globalnej „temperatury”). Wartość ta stale maleje, co czyni takie przejścia coraz mniej prawdopodobnymi wraz z upływem czasu przeszukiwania. Można pokazać, że jeśli *T* zmniejsza się wystarczająco wolno to „symulowane wyżarzanie” znajduje globalne optimum z prawdopodobieństwem bliskim 1. Implementacja strategii „symulowanego wyżarzania” została zarysowana w tab. 5-2. Zachowano tradycyjne określenie „Energia” dla oceny stanu.

Tab. 5-2. Algorytm „symulowanego wyżarzania”.

```
function SymulowaneWyżarzanie( problem, temp[] ) returns stan będący rozwiązaniem;
{
  current ←Węzeł(PoczątkowyStan[problem]);
  for t←1 to rozmiar(temp) do
  {
      T← temp[t];  // T jest coraz mniejsze
      if (T == 0) then return current;      // Koniec przeszukiwania
      next ←losowo wybrany następca dla current;
      dE = Energia[next] – Energia[current];
      if (dE >0 ) then current ←next;
      else current ←next  tylko z prawdopodobieństwem exp(dE/T);
  }
  return current;
}
```

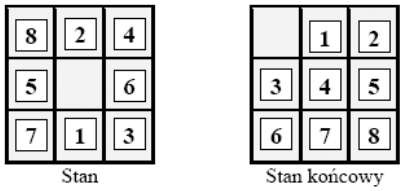
5.6. Pytania

1. Zdefiniować i zilustrować problem przeszukiwania.
2. Przedstawić strategie poinformowanego przeszukiwania? Kiedy strategia „najlepszy najpierw” jest poinformowana a kiedy nie?
3. Przedstawić strategie przeszukiwania: zachłanną i A*. Która z nich jest optymalna i w jakich warunkach?
4. Czy różni się przeszukiwania drzewa od przeszukiwania grafu?
5. Co oznaczają: „dominacja heurystyki” i „problem złagodzony”?
6. Omówić istotę strategii poinformowanego przeszukiwania lokalnego.
7. Na czym polega „symulowane wyżarzanie”?

5.7. Zadania

Zad. 5.1

W problemie „8 puzzli” znane są: stan początkowy i wymagany stan końcowy, jak na rysunku 5.20. Rozwiązać problem złagodzony wobec problemu 8 puzzli, w którym pojedyncza akcja polega na natychmiastowym przemieszczeniu (jedną akcją) jednego kafelka na wymagane miejsce docelowe (można kłaść jeden na drugi), stosując strategię **zachłanną przeszukiwania poinformowanego** przy zastosowaniu heurystyki odpowiadającej **metryce Manhattan**.



Rys. 5.20

Zad. 5.2

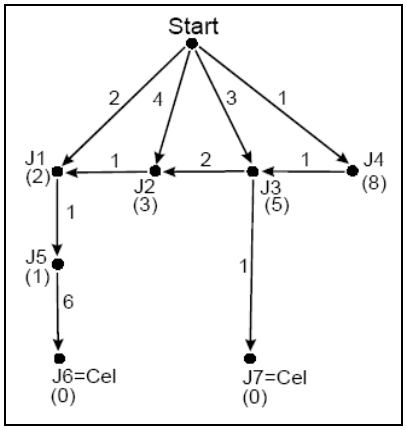
W podanym niżej grafie (rys. 5.21) pewnego problemu koszty operacji pomiędzy stanami podane są przez liczby przy łukach, a oszacowanie kosztów pozostałych jeszcze na drodze do celu (heurystyka) – przez liczby w nawiasach przy węzłach.

A) Zilustrować działanie strategii **przeszukiwanie A*** dla zadanego grafu problemu.

B) Czy heurystyka jest **dopuszczalna**?

C) Czy znaleziono **optymalne** rozwiązanie?

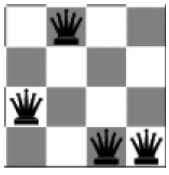
Uwaga: dwa stany spełniają warunek stopu (są docelowe), jednak tylko jeden z nich reprezentuje optymalne rozwiązanie.



Rys. 5.21: Graf problemu.

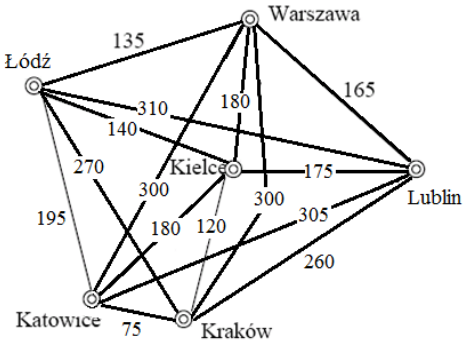
Zad. 5.3

Niech w problemie 4 królowych oszacowanie kosztów resztkowych (heurystyka) wyrażone jest liczbą zagrażających sobie wzajemnie par (tzn. dwie królowe są w tym samym wierszu lub kolumnie lub przekątnej). Możliwe do wykonania akcje polegają na przesuwaniu pionka w kolumnie (w każdej kolumnie jest jeden pionek). Dla podanego poniżej (rys. 5.22) stanu początkowego wykonać symulacje działania algorytmu lokalnego przeszukiwania „przez wspinanie”.



Rys. 5.22: Stan początkowy.

Zastosować przeszukiwanie A* dla rozwiązania konkretnego „problemu komiwojażera” zadanego poniższym rysunkiem (rys. 5.23):



Rys. 5.23

- Założyć, że ścieżka rozpoczyna i kończy się w Warszawie.
- Zdefiniować nietrywialną heurystykę (tzn. inną niż $h(n) = 0$).
- Pokazać drzewo decyzyjne rozwijane dla znalezienia rozwiązania, podać ścieżkę rozwiązania i jej koszt (w km, jako suma odległości częściowych podanych na rysunku).

6. Zaawansowane algorytmy przeszukiwania

IDA*

SMA*

Przeszukiwanie z wynikiem „o dowolnym czasie”

Real-time A*

Zadania

6.1. IDA* („Iterative deepening A*)

Algorytm A* wymaga list OPEN i CLOSE dla pamiętania wszystkich wygenerowanych węzłów. Może to prowadzić do dużych wymagań jego implementacji programowej wobec pamięci. Z pomocą dla zaprojektowania rozwiązania o mniejszej pamięciochłonności przychodzi nam zasada iteracyjnego przeszukiwania w głąb (IDS).

Załóżmy, że funkcja kosztu *f* jest monotoniczna (tzn. heurystyka jest „dopuszczalna” i „spójna”):

- $f(n) = g(n) + h(n) \leq f^*(n)$ (rzeczywisty koszt),
- $f(n) \leq f(n')$, gdzie n' jest następnikiem n

Idea algorytmu IDA* (tab. 6.1)

Wykonujemy iteracje (poinformowanej wersji) przeszukiwania „w głąb” ograniczone wartością maksymalnego progu dla wartości oceny ścieżki *f(n)*. Tzn. powstaje podprzestrzeń ograniczona aktualnie przyjętym progiem dla funkcji oceny $f(n)$, która przeszukujemy zgodnie ze strategią „w głąb” („depth-first”). Jeśli w danej iteracji nie zostaje znalezione rozwiązanie to w kolejnej iteracji próg jest odpowiednio zwiększany do najmniejszej wartości *f(n)* w poprzedniej podprzestrzeni przewyższającej jej próg.

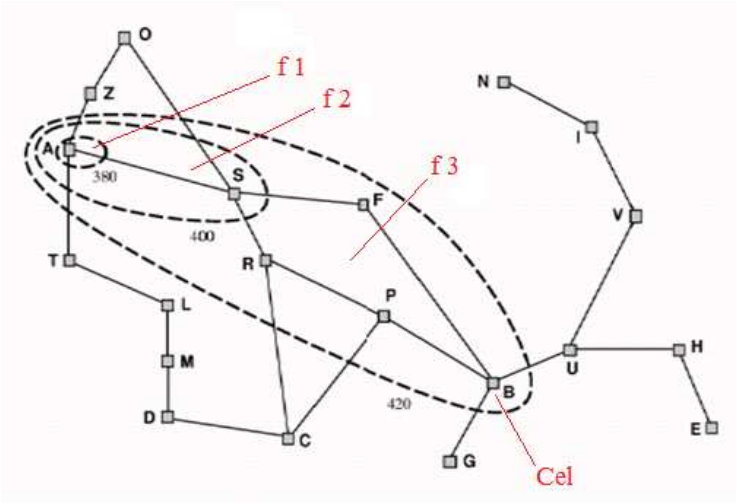
Własności IDA* wobec A*.

- Jeśli funkcja kosztu jest monotoniczna to IDA* znajduje optymalne rozwiązanie.
- IDA* wymaga znacznie mniej pamięci (liniowa złożoność pamięciowa w funkcji długości ścieżki) niż A*.
- Może wymagać więcej obliczeń – jeśli koszt jest znacząco niedoszacowany i wymaganych jest wiele kroków dla osiągnięcia $f = f^*$.
- W najgorszym przypadku dla IDA*: N obliczeń dla A* ale N^2 dla IDA*

Monotoniczna funkcja kosztu

Monotoniczność funkcji oceny wzdłuż każdej ścieżki umożliwia wyróżnienie kolejnych podprzestrzeni o coraz większych wartościach progu *fi* (rys. 6.1). Kolejne wartości *fi* (f1, f2, f3, …) odpowiadają coraz lepszemu „poinformowaniu” agenta i „zbliżaniu” się podprzestrzeni do optymalnego celu.

Przy „dobrej” heurystyce podprzestrzenie „rozciągają się” w kierunku celu, unikając nadmiernego przeszukiwania podprzestrzeni położonych dalej od ścieżki rozwiązania.



Rys. 6.1 Ilustracja podprzestrzeni przeszukiwanych w kolejnych iteracjach algorytmu IDA*.

Tab. 6.1 Schemat algorytmu IDA* („Iterative-Deepening-A*")

1) (Init) Ustaw f_B = oszacowanie wartości *f* dla węzła początkowego;
2) Wykonaj przeszukiwanie „w głąb" z węzła początkowego, przycinając gałęzie drzewa wtedy, gdy ich koszt $(g + h) > f_B$.
3) Jeśli znaleziono rozwiązanie to zakończ → **return**(optymalne rozwiązanie)
4) Zwiększ f_B do najmniejszej spośród wartości kosztu węzłów wizytowanych (ale nie rozszerzonych) w aktualnej iteracji i kontynuuj od kroku (2).

Efektywność algorytmu IDA*

- Złożoność czasowa: IDA* asymptotycznie zdąża do złożoności A*.
- Złożoność pamięciowa IDA* wynosi tylko O(d) , a A* tymczasem - $O(b^d)$.
- W IDA* unika się sortowania kolejki węzłów.
- IDA* posiada prostszą implementację – brak listy CLOSED (mniejsza lista OPEN).
- IDA* może wykonywać się szybciej mimo, iż generuje więcej węzłów niż A*.
- IDA* może rozwiązać problem, którego A* z powodu braku pamięci nie rozwiąże (lub rozwiąże po znacznie dłuższym czasie).

6.2. SMA* („Simplified memory bounded A*")

Algorytm SMA* zakłada istnienie jawnego ograniczenia na liczbę węzłów, które mogą być pamiętane podczas przeszukiwania A*. Występują tu zasadniczo dwa parametry:

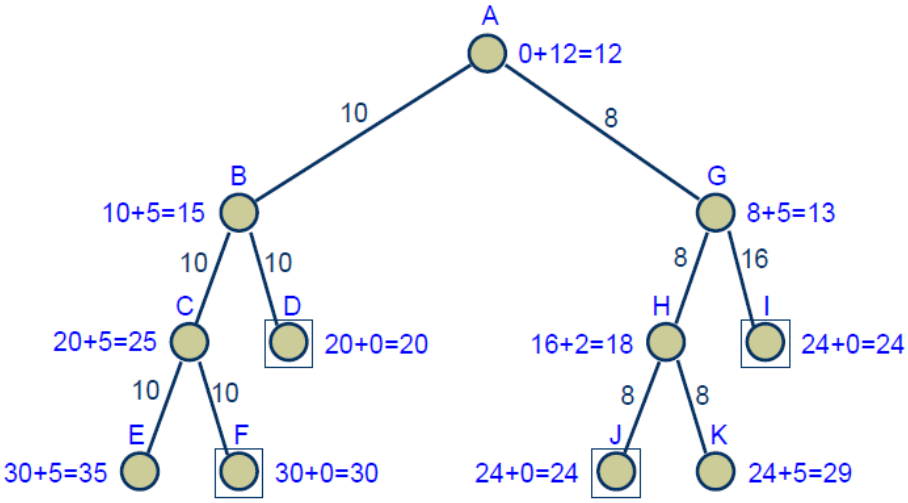
1. Ograniczenie określa maksymalną liczbę pamiętanych węzłów, tzn. znajdujących się sumarycznie na obu listach OPEN i CLOSE.
2. Ograniczenie określa maksymalną długość ścieżki.

Strategia SMA*:

- Rozwijany jest zawsze najgłębszy liść o aktualnie najmniejszej wartości kosztu *f*.
- Po rozwinięciu węzła (określeniu następników) modyfikuje się koszt *f* tego węzła (już wtedy niekońcowego), który przyjmuje wartość najmniejszej wartości kosztu spośród jego następników.
- Gdy wyczerpie się pamięć, należy obciąć płytsze węzły o najwyższej wartości *f* , ale zapamiętać koszt najlepszego takiego odrzuconego następnika w węźle-rodzicu.
- Ścieżki o długości większej niż „limit długości ścieżki" nie prowadzące do celu uzyskują koszt ∞ (nieskończenie duży).

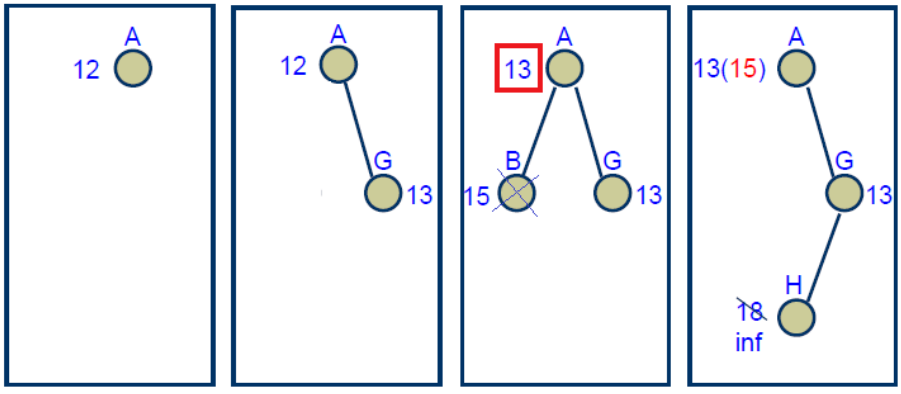
Przykład.

Załóżmy (jeszcze nieznaną algorytmowi) pełną przestrzeń stanów (o postaci drzewa), jak na rys. 6.2.



Rys. 6.2 Przykład przestrzeni stanów.

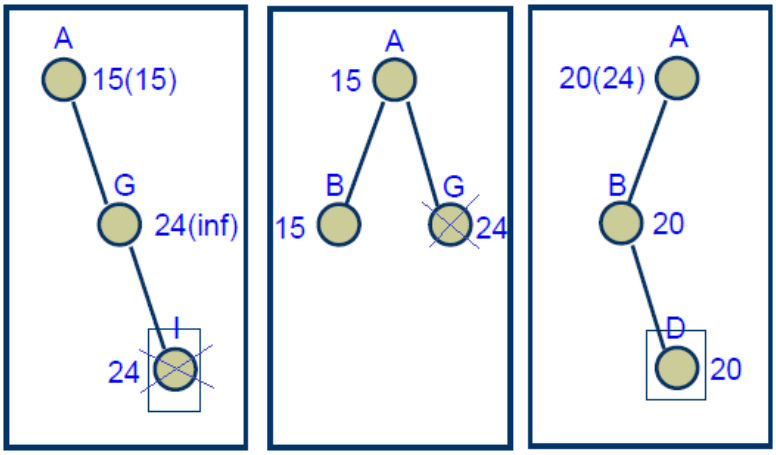
Wprowadzamy ograniczenia: „limit pamięci i „limit długości ścieżki" wynoszą 3. Pierwsze cztery kroki algorytmu SMA* dla problemu z rys. 6.2 przedstawiono na rys. 6.3



(1) (2) (3) (4)

Rys. 6.3 Cztery początkowe kroki w procesie przeszukiwania algorytmem SMA*.

Nowe w porównaniu do A* są tu kroki określone jako 2, 3 i 4. W kroku 2 następuje dodanie węzłów G i B (jeden po drugim) oraz modyfikacja kosztu *f*(A) najniższą wartością *f* jego następnika, czyli 13. W kroku 3 wybrany zostaje węzeł G ale limit pamięci (3) uniemożliwia jego rozwinięcie. Dlatego z pamięci usuwany jest („gorszy") węzeł B. Jego koszt (15) zostaje zapamiętany w węźle rodzica, czyli A. Efekt tych operacji widoczny jest w kroku 4. Teraz jest miejsce na dodanie najlepszego następnika węzła G – jest nim węzeł H o koszcie 18. Zostaje on wybrany. Węzeł H jest niekońcowy ale nie może zostać rozwinięty, gdyż osiągnięto limit długości ścieżki. W takiej sytuacji węzeł H uzyskuje koszt „nieskończony" i jest usuwany po to, aby zrobić miejsce w pamięci dla kolejnego następnika węzła B.



(5) (6) (7)

Rys. 6.4 Kolejne kroki w procesie przeszukiwania algorytmem SMA*.

Kolejne kroki (5) – (7) ilustruje rys. 6.4. W kroku (5) widzimy, że po usunięciu węzła H jego rodzic pamięta jeko koszt (nieskończoność) oraz przyjmuje koszt aktualnie najlepszego następnika, czyli 24. Węzeł I jest wprawdzie węzłem końcowym ale nie zostanie wybrany, gdyż rodzic węzła G, czyli A pamięta o innej ścieżce, której koszt wynosi aktualnie 15. Wybór pomiędzy węzłem I o koszcie 24 a inną ścieżką o koszcie 15, oczywiście kończy się ta druga ewentualnością. Co więcej węzeł I zostaje usunięty, gdyż osiągnięto limit pamięci. W kroku (6) ponownie dodany został węzeł B, a węzeł G w kolejnym kroku zostaje usunięty, aby móc rozwijać lepszy węzeł B. Koszt usuniętego G pamiętany w węźle jego rodzica A. Efekt ten widoczny jest w kroku (7), w którym też dodany zostaje najlepszy następnik węzła B. W następnym kroku ten węzeł D sam zostaje wybrany jako aktualnie najlepszy i okażę się, że jest to węzeł końcowy.

Tab. 6.1 Algorytm SMA*

```
function SMAstar(problem) returns solution
{ static: Queue; // kolejka węzłów uporządkowana kosztem f
    Queue ← MakeQueue(MakeNode(InitialState[problem]));
    do {
        if Queue jest puste then return ∅;
        n ← najgłębszy węzeł o najmniejszym koszcie w Queue
        if GoalTest(n) then return solution;
        s ← NextSuccessor(n);
        if s nie jest celem i jest na maksymalnej głębokości then   f(s) ← ∞
        else  f(s) ← Max(f(n), g(s)+h(s));
```

```
        if wszystkie następniki węzła n wygenerowane then
        ewentualnie modyfikuj koszt f węzła n i jego przodków
         …
        if wszystkie Successors(n) są w pamięci then
                usuń n z Queue ;
        if pamięć jest pełna then {
          usuń najpłytszy z węzłów o najwyższym koszcie f z Queue ;
          usuń go z listy następników swojego przodka;
          wstaw jego przodka do Queue, jeśli konieczne;
        }
        wstaw s do Queue ;
      } // koniec pętli do
}
```

Własności strategii SMA*:

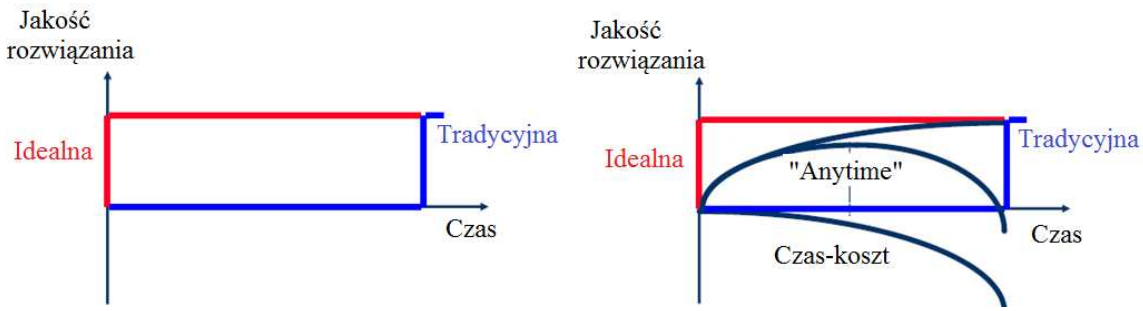
- Jest zupełna, jeśli w pamięci zmieści się ścieżka do najpłytszego węzła końcowego.
- Jest optymalna, jeśli w pamięci zmieści się optymalna ścieżka.
- Jeśli w pamięci mieści się całe drzewo, SMA* zachowuje się identycznie jak A*.

6.3. „Anytime A*" (przeszukiwanie z wynikiem „o dowolnym czasie")

Algorytmy przeszukiwania:

- Idealne wtedy, gdy maksymalna jakość uzyskana jest natychmiast po rozpoczęciu;
- Zazwyczaj maksymalizuje się jakość rozwiązania bez uwzględnienia nakładu czasu;
- Algorytmy „anytime" – uwzględniają zarówno kryterium jakości jak i czas uzyskania rozwiązania.

Algorytmy typu „anytime" posiadają tę zaletę, że dysponują rozwiązaniem niemal zaraz po rozpoczęciu. Rozwiązanie początkowo nie musi być optymalne (i zwykle nie jest). Jednak w dalszym czasie działania algorytmu rozwiązanie to jest poprawiane (rys. 6.5).



Rys. 6.5 Idea algorytmu przeszukiwania typu "anytime"

„Anytime A*"

Istotne różnice w porównaniu do A*:

1. Stosuje się **niedopuszczalną** heurystykę, co umożliwia szybkie znalezienie sub-optymalnych rozwiązań.
2. Kontynuuje przeszukiwanie po znalezieniu pierwszego rozwiązania redukując za jego pomocą listę OPEN (usuwając ścieżki od niego gorsze).
3. Jeśli lista OPEN staje się pusta znalezione rozwiązanie staje się optymalne.

Przykład niedopuszczalnej funkcji kosztu i heurystyki:

$$f'(n) = (1-w)\cdot g(n) + w\cdot h(n)$$

Powyższa funkcja jest tylko wtedy dopuszczalną funkcją kosztu, gdy $h(n)$ jest dopuszczalne i $w \leq 0.5$.

6.4. Real-time A*

A* najpierw znajduje pełną ścieżkę od węzła startowego do końcowego w trybie off-line, a dopiero potem jest ona wykonywana (jako sekwencja akcji) przez agenta. W trakcie przeszukiwania algorytm A* operuje na globalnym horyzoncie, co uniemożliwia jego wykonywanie przez agenta w trybie on-line (czyli w rzeczywistym czasie), bo agent nie może fizycznie przenosić się w jednym kroku z jednego końca przestrzeni do drugiego. Algorytm czasu rzeczywistego (np. RTA*) pozwala na natychmiastowe wykonywanie akcji pomimo, że brak jest jeszcze pełnego rozwiązania. Jest to możliwe dzięki operowaniu na lokalnym horyzoncie aktualnego stanu, jednak w odróżnieniu od algorytmów lokalnego przeszukiwania typu algorytmu „wspinaczkowego", RTA* jednocześnie gwarantuje osiągnięcie optymalnego rozwiązania.

Real Time Heuristic Search (RTA*):

- Wybiera najlepszy krok uwzględniając ograniczenia czasu i głębokości drzewa;
- Przy spełnieniu pewnych warunków gwarantuje osiągnięcie optymalnego stanu końcowego w skończonej liczbie kroków, ale długość ścieżki zwykle nie będzie optymalna.

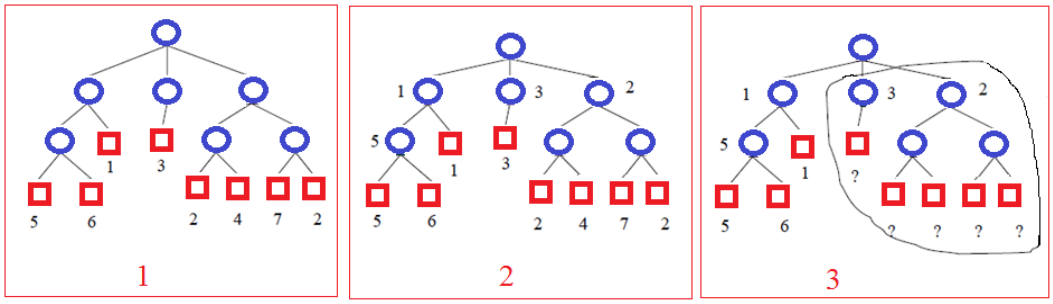
RTA* to A*, ale o ograniczonym (lokalnym) horyzoncie węzłów i wyborze dokonywanym w stałym czasie. Wymagane jest, aby heurystyka była dopuszczalna i funkcja kosztu była monotoniczna (czyli aby heurystyka była spójna).

RTA* zawiera następujące modyfikacje A*:

1. Podejmij decyzję o pojedynczej akcji:
   - Wykonaj podgląd w przód („*mini-min search*" z „*alpha pruning*");
   - Wybierz najlepszy węzeł w lokalnym sąsiedztwie (możliwy jest nawrót) (ocena ustalana jest względem aktualnego węzła).
2. Pamięć związana z sekwencją decyzji:
   - poprzednik: poprawienie błędnych akcji (nawrót)
   - unikanie pętli (pamiętaj „drugi najlepszy" węzeł-następnik).

Podgląd w przód („mini-min search" z „alpha-prunning") (rys. 6.6)

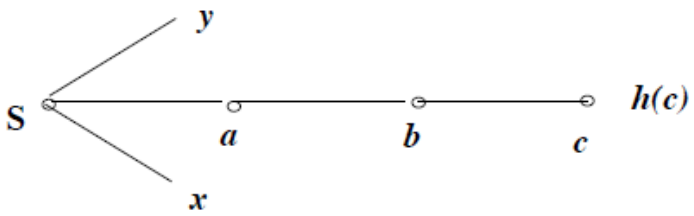
- 1: Przeszukuje liście drzewa i przekazuje zwrotnie do węzłów pośrednich najmniejszą wartość kosztu *f* dla jego następników.
- 2: Określa najmniejszą wartość „*alpha*" kosztu *f* dla liści aktualnego drzewa.
- 3: Usuwa korzenie, gałęzie i ich liście o koszcie większym niż *alpha*.



Rys. 6.6 Idea podglądu w przód (wyznaczanego dla korzenia drzewa) dla wyznaczenia najlepszego liścia w drzewie o zadanej "wysokości. Podczas przeszukiwania drzewa metodą "w głąb" (tu założono lewostronne obejście drzewa) uwzględnia się obcinanie poddrzewa wtedy, gdy ich korzeń ma większy koszt niż inne już znane poddrzewo.

Krok wyboru następnika

Podgląd w przód jest elementem wyznaczenia następnika danego węzła. Załóżmy, że agent jest w stanie S (na rys. 6.7).



Rys. 6.7 Ilustracja stosowania podglądu w przód" w kroku wyboru następnika węzła S.

Po „podejrzeniu ścieżki w przód" (od węzła „*a*" do węzła „c") możliwa jest zwrotna informacja do węzła-poprzednika „*a*" i zastąpienie dotychczasowego oszacowania jego kosztów resztkowych przez „lepiej poinformowany" – niemniejszy koszt resztkowy, ale nadal dopuszczalny:

$h(a) \le g(a \rightarrow c) + h(c) \le h^*(a)$;

Teraz możliwa jest lepiej poinformowana decyzja: „jaki krok wykonać ze stanu S?" : do „*y*", „*a*", lub „*x*".

Pamięć związana ze ścieżką

W procesie wyboru następnego węzła odróżnia się kroki wykonane „do przodu" i krok nawrotu, czyli powrotu do węzła-rodzica. Nie ma potrzeby pamiętania wszystkich wizytowanych węzłów, a jedynie wykonaną ścieżkę oraz to, które węzły były wizytowane podczas ruchu :"do przodu".

Idea nawrotu: należy wykonać **nawrót** do poprzedniego rzeczywistego stanu wtedy, gdy szacowany koszt osiągnięcia celu z tego stanu plus koszt powrotu do niego jest mniejszy niż koszt pójścia „do przodu" z aktualnego stanu.

Zauważmy - funkcja kosztu jest nadal postaci, $f(n) = g(n) + h(n)$,

ale teraz koszt $g(n)$ liczony jest względem aktualnego stanu a nie stanu początkowego.

W węźle z którego wykonany zostaje ruch do przodu pamiętany jest koszt „drugi najlepszy". Ma to na celu uniknięcie pętli – ponownego wyboru już wybranego węzła – wizytowany i „rozszerzany do przodu" węzeł przyjmuje „drugą najlepszą" wartość *f* spośród swoich następników.

Istnieje wykaz stanów - stany wizytowane podczas aktualnego ruchu „do przodu" są zapisywane w wykazie i przyjmują drugą najlepszą wartość. Stany znajdujące się w wykazie nie są ponownie modyfikowane w kroku „podglądu w przód".

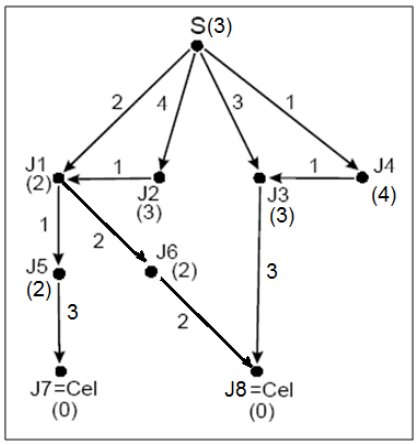
6.5. Zadania

Zad. 6.1

Rozwiązać podany problem znalezienia optymalnej ścieżki (od S do celu) stosując cztery poinformowane strategie przeszukiwania przestrzeni stanów: A*, IDA*, SMA* z limitem 3 i RTA* .

W nawiasach podano heurystykę (oszacowanie kosztów resztkowych) dla funkcji kosztu ścieżki.

W każdym z przypadków A), B), C) i D) podać: zasadę strategii, rozwijane drzewo decyzyjne i zwracaną ścieżkę. Porównać ze sobą wyniki i skomentować je.



7. Problemy z ograniczeniami

Problem CSP

Przeszukiwanie przyrostowe dla CSP

Lokalne przeszukiwanie dla CSP

Pytania

7.1. Problem CSP

Dotychczas rozpatrywaliśmy standardowy problem przeszukiwania, w którym z punktu widzenia strategii przeszukiwania stan stanowił „czarną skrzynką". Informacja o stanie zawierała jedynie: jego związki poprzez akcje z innymi stanami, wartość składowej heurystycznej i własności dla warunku zatrzymania. To zmienia się, gdy mówimy o specyficznym rodzaju problemów, jaki tworzą problemy wymagające spełnienia ograniczeń (ang. *constraint satisfaction problem*) (CSP):

- stan jest definiowany przez zmienne $\{X_i | i=1,2,..,n\}$, których wartości należą do odpowiednich dziedzin $\{D_i\}$;
- warunek zatrzymania to zbiór ograniczeń określający dopuszczalne kombinacje wartości dla zmiennych;
- rozwiązaniem jest dowolny stan spełniający te ograniczenia;
- rozwiązanie może być generowane przyrostowo.

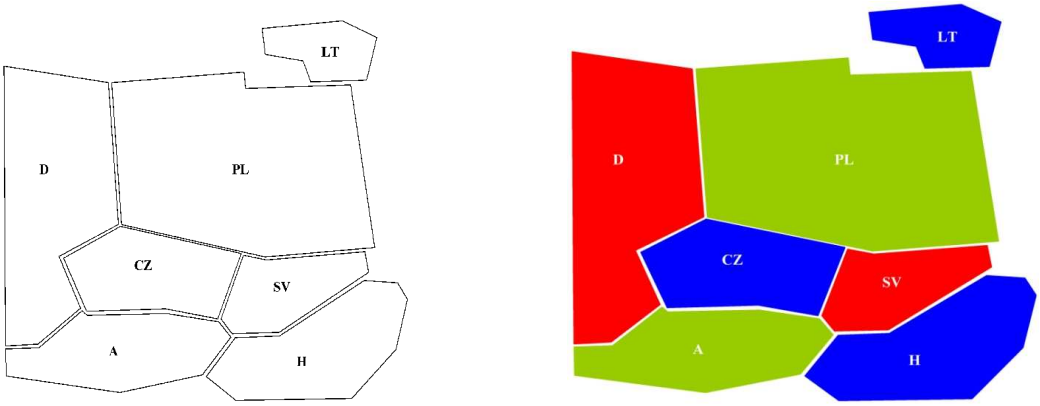
Dla zmiennych o skończonych dziedzinach możemy do rozwiązania problemu CSP zastosować strategie **przeszukiwania z ograniczeniami**.

7.1.1. Przykłady ograniczeń

Przykład. Kolorowanie mapy obejmującej kraje Europy Środkowo-Wschodniej (rys. 7.1). Wprowadzamy symbole zmiennych dla oznaczenia 7 krajów: $D, PL, CZ, SV, LT, A, H.$ Załóżmy, że dysponujemy jedynie trzema kolorami, tzn. wszystkie dziedziny zmiennych są takie same i wynoszą: $D_i = \{r, g, b\}$. Ograniczenia polegają na tym, że przylegające obszary muszą być pomalowane różnymi kolorami, co zapiszemy w postaci, np. $(\mathrm{D} \neq \mathrm{PL})$ lub $(\mathrm{D}, \mathrm{PL}) \in \{(r, g), (r, b), (g, r), (g, b), (b, r), (b, g)\}$.

Poszukiwane rozwiązania są zupełnymi (wszystkim zmiennym nadano wartości) i spójnymi (niesprzecznymi z ograniczeniami) podstawieniami wartości pod zmienne.

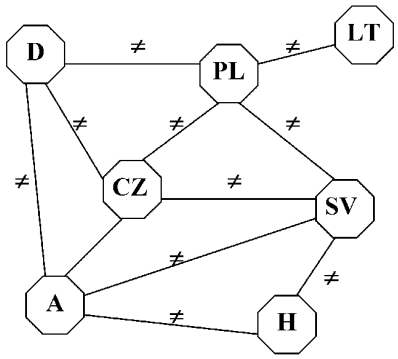
Np.: $\{D=r, PL=g, CZ=b, SV=r, LT=b, A=g, H=b\}$



(a) (b)

Rys. 7.1: Przykład problemu z ograniczeniami – kolorowanie mapy.

W tym przykładzie mamy do czynienia z ograniczeniami dwuargumentowymi (binarnymi) - każde ograniczenie stanowi związek pomiędzy dwiema zmiennymi. Przedstawiamy je w postaci **grafu ograniczeń**: zmienne są węzłami, łuki są ograniczeniami (etykieta łuku wyznacza rodzaj ograniczenia) (rys. 7.2).



Rys. 7.2: Przykład grafu ograniczeń dla problemu kolorowania mapy.

Ograniczenia mogą być zarówno jedno- i dwuargumentowe, jaki i wyższego rzędu. Jednoargumentowe ograniczenia dotyczą pojedynczej zmiennej. Np. $\mathrm{PL} \neq \mathrm{g}$. Binarne ograniczenia dotyczą par zmiennych. Np. $\mathrm{D} \neq \mathrm{PL}$. Ograniczenia wyższego rzędu dotyczą 3 lub większej liczby zmiennych, np. ograniczenia kolumnowe w krypto-arytmetyce.

Przykład. Krypto-arytmetyka jako problem CSP. Dana jest zaszyfrowana operacja matematyczna:

$$\begin{array}{r} T\ W\ O \\ +\,T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

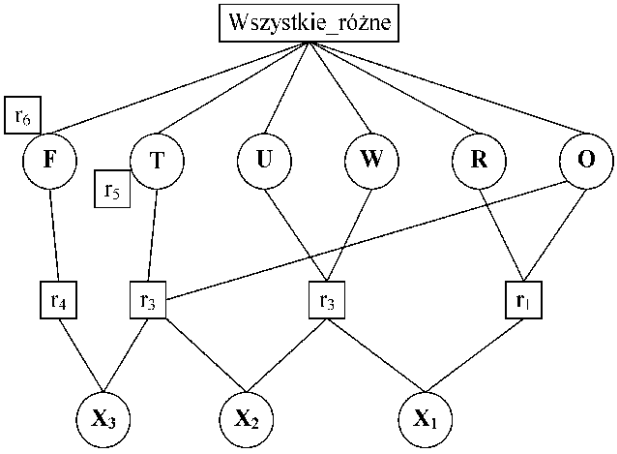
Należy znaleźć cyfry reprezentowane literami. Definiujemy zbiór zmiennych: F, T, U, W, R, O.

Dziedziny wartości są zawsze te same: $\mathrm{D} = \{0,1,2,3,4,5,6,7,8,9\}$. Ograniczenia, oprócz zmiennych wymagają też wprowadzenia stanów dla wartości zależnych od zmiennych: X_1 , X_2 , X_3 (*przeniesienia*) (rys. 7.3):

$Wszystkie\_różne(F,T,U,W,R,O)$,

$r1: O + O = R + 10 \cdot X_1$; `$r2: X_1 + W + W = U + 10 \cdot X_2$; $r3: X_2 + T + T = O + 10 \cdot X_3$

$r4: X_3 = F$, r5: $T \neq 0$, r6: $F \neq 0$



Rys. 7.3: Przykład grafu ograniczeń dla problemu krypto-arytmetyki.

7.1.2. Przeszukiwanie przyrostowe lub ze stanem kompletnym dla CSP

Wyróżnimy dwie zasadnicze strategie przeszukiwania dla problemów CSP: przeszukiwanie **przyrostowe** ze stanem częściowym i przeszukiwanie **ze stanem kompletnym**. Pierwsza z nich jest typową strategią dla tego typu problemów. Stan niekompletny oznacza, że tylko część zmiennych ma w nim przypisane wartości. Przeszukiwanie przyrostowe wyznacza ścieżkę rozwiązania, na której po kolei następują przyporządkowania wartości do kolejnych zmiennych. W takiej sytuacji rozwiązanie pojawia się na głębokości *n*, gdzie *n* jest liczbą wszystkich zmiennych. Typowa strategia przeszukiwania przyrostowego łączy niepoinformowane przeszukiwanie w głąb, ze sprawdzaniem ograniczeń i z możliwością wykonania nawrotu, czyli cofnięcia się na wyższy poziom drzewa przeszukiwania.

Złożoność tego przeszukiwania możemy szacować w następujący sposób: liczba następników węzła w drzewie zależy od głębi położenia l i liczby możliwych wartości zmiennych d:

$b_l = (n - l)\, d$ na głębokości l, a więc będzie $n!\, d^n$ liści.

Ponieważ sekwencja akcji prowadząca do stanu kompletnego, czyli do rozwiązania, jest nieistotna, więc alternatywnie do przeszukiwania przyrostowego można stosować przeszukiwanie ze stanem kompletnym. Należy zainicjalizować problem w dowolnym stanie kompletnym, niespełniającym ograniczeń, zdefiniować akcje wymiany wartości nadanych uprzednio zmiennym i tak długo zmieniać stan, jak długo nie spełnia on zadanych ograniczeń. Do tego celu nadają się ogólne strategie przeszukiwania poinformowanego, np. można zastosować przeszukiwanie lokalne z heurystyką.

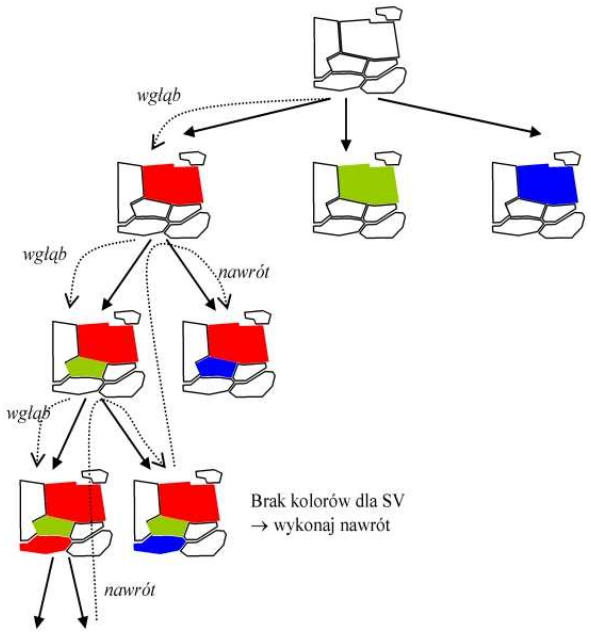
7.2. Przeszukiwanie przyrostowe dla CSP

Przeszukiwanie przyrostowe stanowi standardowy sposób przeszukiwania przestrzeni problemu CSP. Węzły drzewa reprezentują zwykle niekompletne stany, które wyznaczone są przez wartości dotychczas przypisane zmiennym. Węzeł początkowy reprezentuje stan pusty - przyporządkowanie puste { }. Funkcja generowania następnika (akcja) polega wybraniu zmiennej w dotychczasowym stanie, która nie ma wartości i na nadaniu jej wartości w taki sposób, aby nie wywołać konfliktu (nie naruszyć ograniczeń) z dotychczas przyporządkowanym zmiennym. W sytuacji, gdy wybrany stan nie spełnia ograniczeń należy zrezygnować z niego i wykonać „nawrót" w drzewie przeszukiwania. Warunek zatrzymania jest znany z góry: stan jest zupełny i spełnia ograniczenia.

7.2.1. Przeszukiwanie z nawrotami

Zauważmy, że przyporządkowania zmiennych są przemienne. Np. wykonamy najpierw $[D = r]$ a następnie $[PL = g]$ i uzyskamy to samo co wykonanie najpierw $[PL = g]$ a potem $[D = red]$. Dlatego wystarczy rozpatrywać przyporządkowania tylko jednej zmiennej na każdym poziomie drzewa decyzji. Pozwala to znacznie zmniejszyć rozmiar drzewa przeszukiwania, gdyż nie musimy juz rozpatrywać alternatyw dla zmiennych. Wtedy stopień rozgałęzienia drzewa wynosi: $b = d$ (niezależnie od poziomu l) i mamy najwyżej d^n liści.

Przeszukiwanie z nawrotami jest podstawową formą przyrostowego przeszukiwania dla CSP. Opiera się ono ślepym przeszukiwaniu w głąb, w którym pojedyncza akcja odpowiada przyporządkowaniu wartości pojedynczej zmiennej, sprawdzaniu ograniczeń dla zmiennych danego stanu i wykonywaniu nawrotów przy niespełnianiu ograniczeń przez aktualny stan. Ideę nawrotów ilustruje rys. 7.4.



Rys. 7.4: Idea przeszukiwania z nawrotami dla problemu CSP.

7.2.2. Usprawnienia przeszukiwania z nawrotami

Usprawnienia podstawowej strategii mają na celu wczesne wykorzystanie wszelkiej informacji, która wynika z grafu ograniczeń dla problemu. Pozwoli to w praktyce zmniejszyć rozmiar drzewa przeszukiwania i wcześnie wykrywać ścieżki nie prowadzące do rozwiązania. Omówimy kilka usprawnień, które mają ogólny charakter dla dyskretnych CSP, czyli mogą zawsze zostać zaimplementowane dla tego typu problemów.

Podczas przeszukiwania możemy kierować się wynikami następującej analizy:

(A) Której **zmiennej** jako następnej powinno się przyporządkować wartość? Decyduje to o poziomach drzewa przeszukiwań. Mamy tu dwa kryteria, które są badane po kolei:

(A1) najbardziej **ograniczona** zmienna, lub

(A2) najbardziej **ograniczająca** zmienna;

(B) W jakiej kolejności powinno się nadawać **wartości**? Decyduje to o kolejności następników – od lewej do prawej. Wiemy, że przy strategii przeszukiwania w głąb w tej właśnie kolejności będą wizytowane następniki aktualnego węzła. Mamy tu jedno kryterium:

(B1) najmniej ograniczająca wartość

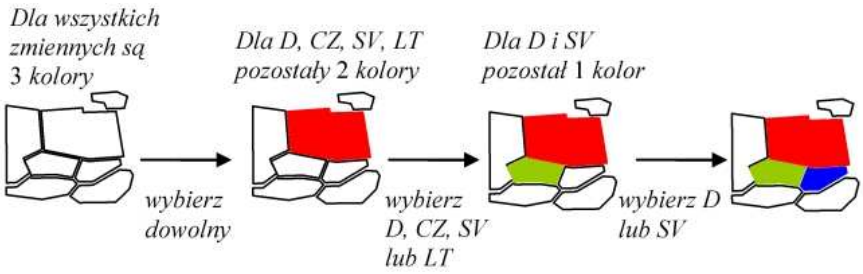
(C) Czy na podstawie **częściowego stanu** już możemy zdecydować o porażce aktualnej ścieżki?

(C1) sprawdzanie wprzód,

(C2) spójność łuków.

A1) Najbardziej ograniczona zmienna

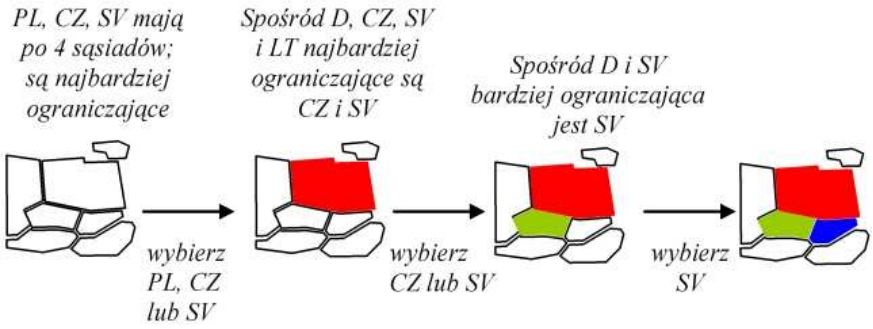
Należy wybrać zmienną o najmniejszej liczbie możliwych pozostałych jeszcze wartości (rys. 7.5), co minimalizuje liczbę następników aktualnego węzła w drzewie przeszukiwania.



Rys. 7.5: Przykład korzystania z kryterium A1) „najbardziej ograniczona zmienna”.

A2) Najbardziej ograniczająca zmienna

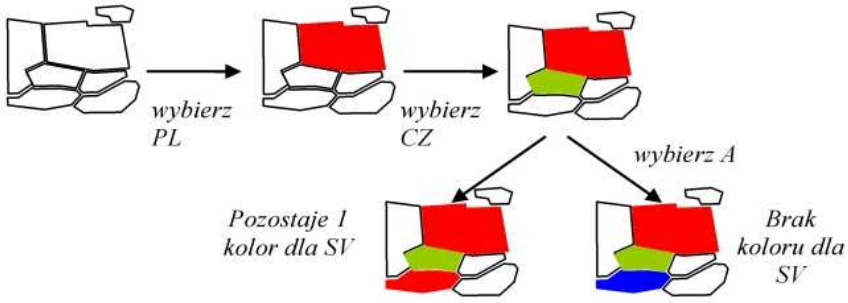
Jest to dodatkowe kryterium w stosunku do A1), które rozstrzyga o wyborze następnej zmiennej w sytuacji dwie lub więcej zmiennych są równo ograniczone. Spośród takich najbardziej ograniczonych zmiennych należy wybrać zmienną, która narzuca najwięcej ograniczeń na pozostałe zmienne (rys. 7.6).



Rys. 7.6: Przykład korzystania z kryterium A2) „najbardziej ograniczająca zmienna”.

B1) Najmniej ograniczająca wartość

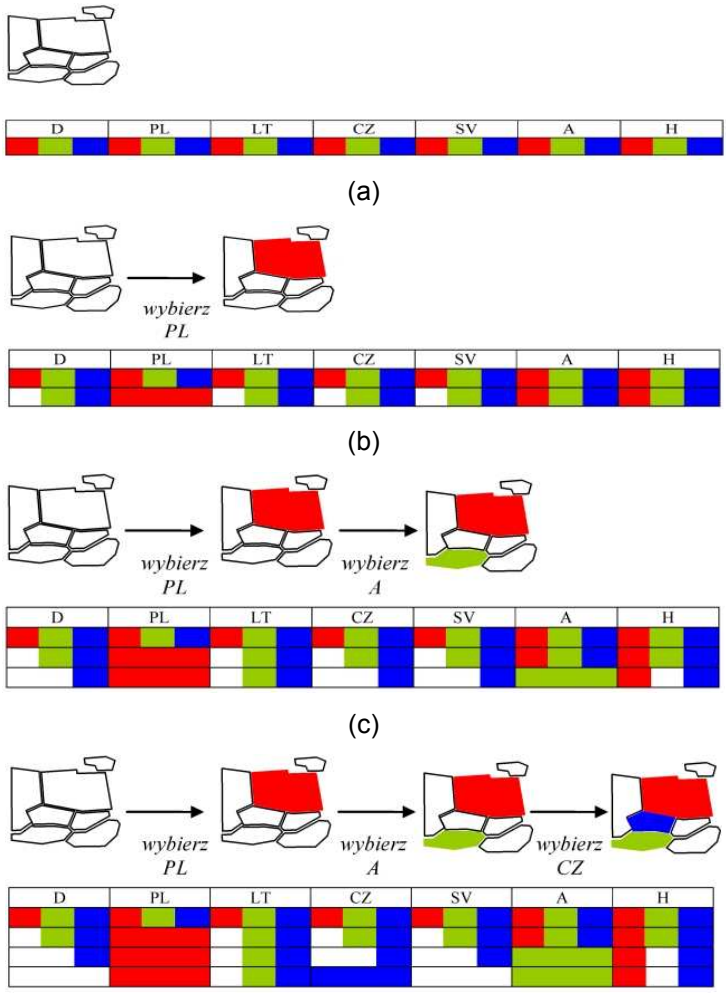
Po wybraniu zmiennej uporządkuj jej wartości zaczynając od wartości najmniej ograniczającej, czyli takiej, która wyklucza najmniej wartości pozostałych zmiennych (rys. 7.7).



Rys. 7.7: Przykład korzystania z kryterium B1) „najmniej ograniczająca wartość”.

C1) Sprawdzanie w przód

To kryterium wymaga wprowadzeniu dodatkowej pamięci - należy w ten sposób śledzić pozostałe, jeszcze legalne wartości dla pozostałych nieustalonych zmiennych. Pozwoli to przerwać poszukiwania na danej ścieżce wtedy, gdy jakaś zmienna nie ma już żadnych legalnych wartości.

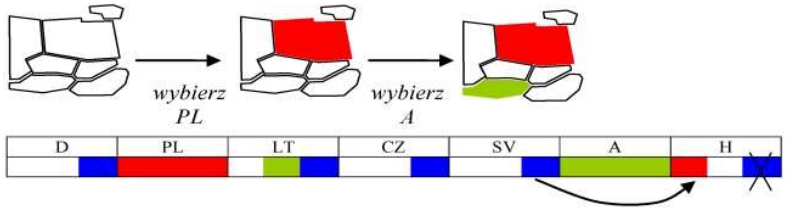


Rys. 7.8: Przykład korzystania z kryterium C1) „sprawdzanie wprzód". (a) w stanie początkowym wszystkie wartości są legalne dla każdej zmiennej; (b) przypisanie (PL=r) zmniejsza legalne wartości dla D,CZ,SV i LT; (c) przypisanie (A=g) sprawia, że D, CZ, SV i H tracą kolor g; sprawdzanie w przód jeszcze nie wykryje, że już w tym stanie D i CZ są w konflikcie; (d) przypisanie (CZ=b) kasuje już wszelkie możliwe wartości D i SV, więc należy wykonać nawrót.

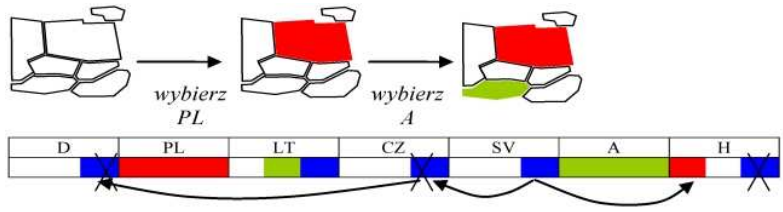
Sprawdzanie w przód dzięki istniejącym ograniczeniom propaguje informację o dopuszczalnych wartościach od związanych do niezwiązanych zmiennych (rys. 7.8). Jednak nie umożliwia to wcześniejszego wykrycia porażki ścieżki dopóki nie zabraknie wartości dla pewnej zmiennej. W przykładzie na rys. 7.8(c) D i CZ mają jeszcze wartości niebieskie, ale z warunków zadania (ograniczenia) wiemy, że nie mogą jednocześnie mieć tej samej wartości. Pomocna będzie analiza spójności ograniczeń dla całego stanu.

C2) Spójność łuków (ograniczeń)

W tym kryterium sprawdzamy czy dla każdego łuku w grafie ograniczeń istnieje jeszcze przynajmniej jedna kombinacja legalnych wartości jego zmiennych, spełniająca jego ograniczenie. Powiemy, że łuk $X \rightarrow Y$ jest spójny wtedy i tylko wtedy gdy dla każdej wartości x nadanej X istnieje jakaś dopuszczalna wartość y dla Y. Z kolei, jeśli dla wartości x lub y nie ma przynajmniej jednej wartości drugiej zmiennej, to ta wartość przestaje być legalna (rys. 7.9(a)). Jeżeli X straci jakąś wartość, sąsiedzi X (w sensie istnienia między nimi ograniczeń) muszą być sprawdzeni ponownie. Mamy więc do czynienia z propagacją badania spójności łuków dla legalnych wartości w danym stanie. W naszym przykładzie to, że zmienna H straci wartość b, nie jest jeszcze krytyczne. Ale CZ straci b, i łuk między SV a CZ przestaje być spójny (rys. 7.9(b)). Teraz już po dwóch akcjach wykryjemy, że w tym stanie SV i CZ są już w konflikcie (podobnie jest z parą CZ i D).



(a)



(b)

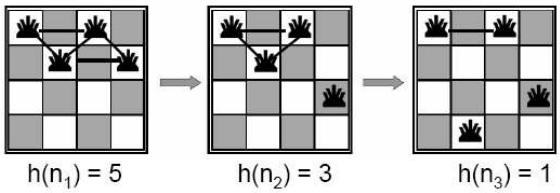
Rys. 7.9: Przykład korzystania z kryterium C2) „spójność łuków”. (a) Po sprawdzeniu spójności łuku $(\mathrm{SV}, \mathrm{H})$ zmienna H traci wartość b. (b) Łuki pomiędzy SV i CZ oraz CZ i D przestają być spójne.

7.3. Lokalne przeszukiwanie dla CSP

Pamiętamy, że strategie lokalnego przeszukiwania, takie jak „przez wspinanie” czy „symulowane wyżarzanie”, zazwyczaj operują na „kompletnych" stanach. W znaczeniu problemu CSP oznacza to, że w takim stanie wszystkie zmienne mają przypisane wartości. Aby móc zastosować te lokalne metody przeszukiwania dla rozwiązania problemu CSP należy ten problem wyrazić następująco:

1. Dopuszczamy stany kompletne z konfliktami wartości zmiennych (nie spełniające ograniczeń).
2. Operatory przeszukiwania (akcje) dokonują zmian wartości zmiennych, czyli przechodzimy z jednego stanu kompletnego w inny stan kompletny.
3. Selekcja zmiennych i wartości: zastosować oszacowanie heurystyczne podające liczbę konfliktów i wybrać wartość, która narusza najmniejszą liczbę ograniczeń.

Przykład. Heurystyka wyrażająca liczbę konfliktów w problemie 4-królowych. Liczba stanów wynosi: $4^4 = 256$. W terminach CSP mamy 4 zmienne, gdzie każdej z nich mogą być nadane 4 wartości. Pojedyncza akcja to przesunięcie królowej w kolumnie. Ocena stanu równa się liczbie par wzajemnych ataków (konfliktów). Warunkiem zatrzymania procesu przeszukiwania powinien być brak wzajemnych ataków (konfliktów), ale jak wiemy, w przeszukiwaniu lokalnym nie zawsze będzie to możliwe do osiągnięcia. Na rys. 7.10, trzy pokazane stany posiadają oceny: 5,3,1.



Rys. 7.10: Przykład strategii minimalizującej liczbę konfliktów w problemie „4 królowych”.

7.4. Pytania

1. Jak reprezentujemy problem z ograniczeniami (CSP)?
2. Wyjaśnić zasadę działania algorytmu przeszukiwania przyrostowego dla CSP ?

3. Na czym polegają usprawnienia algorytmu przeszukiwania z nawrotami?
4. Na czym polega przeszukiwanie ze stanem kompletnym dla CSP?

7.5. Zadania

Zad. 7.1

Dane jest zadanie krypto-arytmetyczne: podstaw cyfry dziesiętne pod zmienne, tak, aby operacja dodawania była poprawna. Każda zmienna reprezentuje inną cyfrę.

$$\begin{array}{r} \mathbf{TWO} \\ +\,\mathbf{TWO} \\ \hline \mathbf{FOUR} \end{array}$$

Należy zdefiniować graf ograniczeń dla tego problemu. Przeprowadzić symulację działania podstawowego algorytmu dla CSP – przeszukiwania przyrostowego z nawrotami – dla 3 pierwszych kroków.

Zad. 7.2

Dany jest problem krypto-arytmetyki:

$$\begin{array}{r} \mathbf{ABC} \\ +\,\mathbf{ADC} \\ \hline \mathbf{CEF} \end{array}$$

Możliwe rozwiązanie to:

$$\begin{array}{r} 173 \\ +153 \\ \hline 326 \end{array}$$

Wyznaczyć graf ograniczeń dla problemu. Zaproponować heurystyczną ocenę dla stanów zupełnych (rozwiązań). Zastosować przeszukiwanie lokalne „[przez wspinanie” korzystające z tej funkcji oceny. Stan początkowy dany jest jako: [A=0, B=1, C=2, D=3, E=4, F=5].

8. Klasyczne planowanie działań

- Agent realizujący cel – przeszukiwanie, planowanie
- Klasyczne planowanie
- Plany częściowe
- Przykład: tworzenie planu częściowo uporządkowanego
- Pytania

8.1. Agent realizujący cel

W rozdziale 1 wymieniliśmy trzy sposoby wyboru akcji (działania) logicznego agenta realizującego cel: przez **wnioskowanie**, **przeszukiwanie** lub **planowanie**. Pierwszy z tych sposobów (przez wnioskowanie) nie wymagał dodatkowych mechanizmów, wykraczających poza system logicznego wnioskowania (rozdz. 3 i 4). Wprawdzie w nieefektywny sposób, poprzez dużą liczbę reguł wyboru akcji zawartych w bazie wiedzy, ale możliwe było zrealizowanie takiego agenta przez system logicznego wnioskowania. W rozdz. 5 i 6 poznaliśmy dodatkową, uniwersalną metodykę przeszukiwania przestrzeni stanów, która może zostać zastosowana dla wyboru akcji agenta. Wymagamy wtedy jedynie, aby problem działania agenta został przedstawiony w postaci: przestrzeni stanów, możliwych akcji i ich kosztów, warunku celu i ewentualnych ocen heurystycznych dla stanów. W wyniku zastosowania strategii przeszukiwania przestrzeni stanów agent znajdzie sekwencję akcji wiodącą od stanu początkowego do stanu końcowego, w którym spełniony jest warunek celu.

W tym rozdziale przedstawimy metodykę realizacji celu za pomocą konstruowania planu działania (rozumianego jako sekwencja akcji w przestrzeni stanów). W szczególności sam proces tworzenia planu może być realizowany w postaci przeszukiwania przestrzeni (częściowych) planów.

8.1.1. Przeszukiwanie przestrzeni planów

Jak dotąd zakładaliśmy, że stan problemu reprezentowany jest zwykle prostą strukturą danych potrzebnych dla *generacji następnika, funkcji oceny* i sprawdzenia *warunku zatrzymania.* W istocie taki stan ma charakter „czarnej skrzynki" i nie daje agentowi żadnych wskazówek co do wyboru akcji. Dla problemów, w których występuje bardzo duża liczba potencjalnie możliwych akcji, powoduje to generowanie bardzo dużej liczby następników, z których większość nie jest istotna.

Przykład. Zwykłe przeszukiwanie stanów zawiedzie z powodu dużego stopnia rozgałęzienia drzewa potrzebnego dla realnych, życiowych problemów. Dany jest prosty cel dla człowieka: *„kup karton mleka, bochen chleba i kwiaty oraz wróć z nimi do domu".* Ponieważ człowiek może wykonywać wiele czynności, z których większość nie jest związana z zadanym celem, próba zastosowania przeszukiwania doprowadzi do drzewa o wysokim współczynniku rozgałęzienia (rys. 8.1). Funkcja oceny, w tym składowa heurystyczna nie pozwala eliminować zbędnych akcji a jedynie wybierać stany.

Powyższy przykład wskazuje, że w wielu rzeczywistych problemach potrzebny jest silniejszy mechanizm niż przeszukiwanie, który wyznaczałby sekwencję akcji dla agenta prowadzącą do celu, korzystając z następujących założeń:

- stan przestaje być czarną skrzynką i posiada własności warunkujące stosowanie dla nich akcji;
- akcje wyznaczane są po to aby spełnić własności stanu docelowego,

Ten mechanizm nazywamy planowaniem.

Rys. 8.1: Bardzo wysoki stopień rozgałęzienia drzewa przeszukiwania.

8.1.2. Planowanie

Planowanie ma na celu „usprawnienie" przeszukiwania poprzez „otwarcie" reprezentacji stanów, celów i akcji. Algorytmy planujące korzystają z formalnych języków dla opisu stanów, celów i akcji – zwykle jest to logika predykatów lub jej podzbiór. Stany i stan końcowy (cel) są reprezentowane poprzez zbiory formuł, które są spełnialne.

Akcje są charakteryzowane poprzez logiczne opisy warunków wykonania i efektów akcji. Pozwala to na określenie bezpośrednich związków pomiędzy stanami a akcjami i na pominięcie bezzasadnych akcji w danym stanie.

Akcje są dodawane do planu w dowolnej kolejności wtedy, gdy są potrzebne, a nie w wyniku sztywnej sekwencji prowadzącej od stanu początkowego do końcowego (wykonując *oczywiste* i *ważne* decyzje w pierwszej kolejności, prowadzimy najpierw do ograniczenia przestrzeni rozwiązania a następnie stopniowo uszczegóławiamy plan).

Problem jest często formułowany jako koniunkcja pod-problemów. Możemy wtedy wyróżnić podcele i wyznaczać osobne plany dla osiągnięcia kolejnych podcelów.

Reprezentacja planu – jest to sekwencja operatorów w przestrzeni planów, która (a) rozszerza plan początkowy (prosty, niepełny plan) do końcowego; lub (b) modyfikuje pełny, ale błędny plan do końcowego. W porównaniu do przeszukiwania przestrzeni stanów mamy tu znacznie mniejszą wariantowość i dowolność kolejności operatorów dla planów niż akcji dla stanów.

Funkcję prostego agenta korzystającego z podfunkcji tworzenia planu dla określania sekwencji akcji przedstawiono w tabeli 8-1.

8.2. Klasyczne planowanie

8.2.1. Planowanie w języku logiki

Omówimy tu planowanie wyrażone w rachunku sytuacyjnym logiki predykatów.

Stan początkowy reprezentowany jest jako koniunkcja predykatów, zwykle działających na stałych. Np. $W(\text{dom}) \wedge \neg \textit{Mieć}(\text{mleko}) \wedge \neg \textit{Mieć}(\text{chleb}) \ldots$

Cele (stany końcowe) reprezentowane są jako koniunkcje predykatów i mogą one zawierać zmienne. Np.

$\exists x\ At(x) \wedge Sprzedaje(x,$ mleko)

Zmienne w formule celu są kwantyfikowane egzystencjalnie i jest to formuła zapytania.

Tabl. 8-1. Funkcja prostego agenta planującego

```
funkcja ProstyAgentPlanujący(percepcja); wynik: akcja;
{
   Dane statyczne (trwałe): KB – baza wiedzy (zawiera opisy akcji);
        p – plan, początkowo = BrakPlanu;
        t – licznik czasu, początkowo = 0;
   Lokalne zmienne: cel – aktualny cel; stan – aktualny stan;
   TELL(KB, UtwórzFormułęPercepcji(percepcja, t));
   stan := STAN(KB, t);
   if ((p = = BrakPlanu)
        then {
        cel := ASK(KB, UtwórzFormułęZzapytania(t));
         p := IdealnyPlaner(stan, cel, KB);
        }
   if ((p = = BrakPlanu) or (p = = ∅))
        then akcja := BrakAkcji;
        else {
                akcja := PIERWSZY(p);
                p := RESZTA(p);
                }
   TELL(KB, UtwórzFormułęAkcji(akcja, t);
    t := t+1;
   return akcja;
}
```

Operatory są formułami o postaci *aksjomatu następnika stanu* i są powiązane z akcjami (np. predykatem *Rezultat* – patrz. punkt *2.3*). Np.

$\forall a,s\ Mieć(\text{mleko}, Rezultat(a,s)) \Leftrightarrow$

$[(a = Kupić(\text{mleko}) \wedge W(\text{Supermarket}, s) \vee (Mieć(\text{mleko}, s) \wedge a \neq Stracić(\text{mleko}))]$

8.2.2. STRIPS

„Klasyczna" budowa planu odwołuje się do języka STRIPS (*Stanford Research Institute Problem Solver*) będącego ograniczoną formą rachunku sytuacyjnego. W tym podejściu stany niekońcowe są reprezentowane jako koniunkcje literałów bez zmiennych, czyli predykatów zawierających jedynie stałe. Stany końcowe (cele) są również koniunkcjami predykatów, ale obok stałych mogą też zawierać zmienne, dla których zakłada się kwantyfikację egzystencjalną. Operator składa się z 3 części (tab. 8-2):

1. Warunek operatora (PRECOND) – koniunkcja pozytywnych predykatów atomowych warunkująca wykonalność operatora.
2. Akcja (ACTION) – jej wykonanie przez agenta przeprowadzi środowisko do nowej sytuacji.
3. Następnik – efekt (EFFECT) – efekt operatora opisany za pomocą koniunkcji predykatów atomowych (pozytywnych lub zanegowanych).

Operator przekształca sytuację opisaną w jej warunku w sytuację opisaną w jej następniku i podaje związaną akcję. Pełny opis stanu nie jest konieczny – dla poznania stanu w aktualnej sytuacji wystarczy śledzić zmiany zachodzące od sytuacji początkowej. Brak jest też jawnych indeksów sytuacji, gdyż ze struktury operatora jawnie wynika co jest sytuacją poprzednią a co następną, a poza tym operator ma charakter generyczny i nie zależy od indeksu sytuacji.

Tabl. 8-2. Przykład operatora w STRIPS jako symbol graficzny i w zapisie tekstowym.

W(*tu*), Droga(*tu*, *tam*)
Pójść(tam)
W(*tam*), ¬W(*tu*)

Op(ACTION*: Pójść*(*tam*),
PRECOND*: W*(*tu*) ∧ *Droga*(*tu*, *tam*),
EFFECT: *W*(*tam*) ∧ ¬*W*(*tu*))

Operatory w STRIPS

Operator jest **stosowalny** w stanie *S*, jeżeli warunek operatora jest spełniony w tym stanie.

Operator zawierający symbole zmiennych reprezentuje rodzinę akcji. Wykonany może być tylko w pełni wartościowany operator (po podstawieniu wartości za wszystkie zmienne). Procedura planująca musi zapewnić dostępność wartości dla zmiennych w danym stanie. W stanie powstałym dzięki zastosowaniu operatora spełnione są:

- wszystkie predykaty zawarte w następniku operatora,
- wszystkie predykaty zawarte w warunku operatora, które nie zostały zanegowane przez następnik operatora.

Stan docelowy jest reprezentowany w postaci operatora GOAL nie posiadającego części „następnika", a INIT jest operatorem dla stanu początkowego i z kolei nie posiada „warunku". Operatory GOAL i INIT nie posiadają zmiennych.

Przykład. Plan w STRIPS. Problem transportu przesyłek lotniczych (cargo) pomiędzy dwoma lotniskami może być następująco reprezentowany w STRIPS:

INIT(W(C_1,WAW) ∧ W(C_2, FRA) ∧ W(P_1, WAW) ∧ W(P_2, FRA) ∧ Cargo(C_1) ∧ Cargo(C_2)
∧ Samolot(P_1) ∧ Samolot(P_2) ∧ Lotnisko(WAW) ∧ Lotnisko(FRA))
GOAL(W(C_1,FRA) W(C_2, WAW))
ACTION(Załadunek(c, p, a),
PRECOND: W(c,a) ∧ W(p,a) ∧ Cargo(c) ∧ Samolot(p) ∧ Lotnisko(a)
EFFECT: ¬W(c,a) ∧ W(c,p))
ACTION(Rozładunek(c, p, a),
PRECOND: W(c,p) ∧ W(p,a) ∧ Cargo(c) ∧ Samolot(p) ∧ Lotnisko(a)
EFFECT: W(c,a) ∧ ¬W(c,p))
ACTION(Lot(p, od, do),
PRECOND: W(p,od) ∧ Samolot(p) ∧ Lotnisko(od) ∧ Lotnisko(do)
EFFECT: ¬W(p,od) ∧ W(p,do))

Rozwiązaniem podanego problemu może być następujący plan:

[Załadunek(C_1; P_1; WAW); Lot(P_1; WAW; FRA); Załadunek(C_2; P_2; FRA); Lot(P_2; FRA; WAW)] .

Uwaga. Wadą powyższej reprezentacji problemu w STRIPS jest to, że generowany może być plan, który zezwala na przelot samolotu z lotniska do tego samego lotniska docelowego.

ADL

Po pojawieniu się STRIPS w kolejnych latach okazało się, że jego siła wyrazu nie wystarczy dla opisu wielu praktycznych dziedzin. Dlatego też upowszechniła się odmiana tego języka nazwana „Action Description Language" (ADL) (Tabl. 8-3).

Tabl. 8-3 Porównanie języka STRIPS i ADL

STRIPS	ADL
Tylko pozytywne literały w warunkach, np. $W(\text{dom}) \wedge Ma(\text{mleko})$	Pozytywne i zanegowane literały w warunkach: np. $\neg W(\text{dom}) \wedge \neg Ma(\text{mleko})$
Założenie „zamkniętego świata": brakujące literały są niespełnione (False).	Założenie „otwartego świata": spełnianie brakujących literałów nie jest znane.
$(P \wedge \neg Q)$ oznacza: *dołącz P i usuń Q*	$(P \wedge \neg Q)$ oznacza: *dołącz P i* $\neg Q$*, oraz usuń* $\neg P$ *i Q*
Tylko bazowe literały (predykatowe) w operatorze „cel". Np. $W(\text{P1,WAW}) \wedge W(\text{P2,WAW})$	Kwantyfikowane zmienne w operatorze „cel". Np. $\exists x\, At(\text{P1},x) \wedge At(\text{P2},x)$
Cele są koniunkcyjnej postaci. Np. $W(\text{dom}) \wedge Ma(\text{mleko})$	Cele są koniunkcyjne i alternatywne. Np. $W(\text{dom}) \wedge (Ma(\text{mleko}) \vee Ma(\text{banany}))$
Brak wbudowanej semantyki dla równości (=). Brak typów.	Wbudowany predykat równości (np. x=y). Zmienne są określonych typów. Np. $(x: \text{Miasto})$

Przykład. W języku ADL akcję *Lot* z poprzedniego przykładu reprezentować możemy w następujący sposób:

ACTION(Lot(p:Samolot, od:Lotnisko, do:Lotnisko),
PRECOND: W(p,od) ∧ (od ≠ do)
EFFECT: ¬W(p,od) ∧ W(p,do))

Teraz w warunku operatora występuje dodatkowo wyrażenie (od ≠ do), które wyklucza planowanie wykonania lotu z dowolnego lotniska na to samo lotnisko. Tego nie można było wyrazić w STRIPS.

8.3. Przestrzeń planów częściowych

Problem wyznaczenia planu rozwiążemy poprzez przeszukiwanie przestrzeni możliwych planów dla zadanego problemu agenta. To przeszukiwanie będzie realizowane w postaci strategii specyficznych dla problemu planowania.

Jeszcze raz zwróćmy uwagę na zasadniczą różnicę dwóch zagadnień przeszukiwania:

1. dla rozwiązania problemu agenta wykonuje się przeszukiwanie przestrzeni stanów dla problemu,
2. dla znalezienia planu przeszukiwania będą prowadzone w przestrzeni planów.

Idea poszukiwania planu: rozpoczynamy z planem początkowym (niedużym) i generujemy kolejne (coraz dokładniejsze) plany częściowe, dopóki nie znajdziemy pełnego planu, wyznaczającego

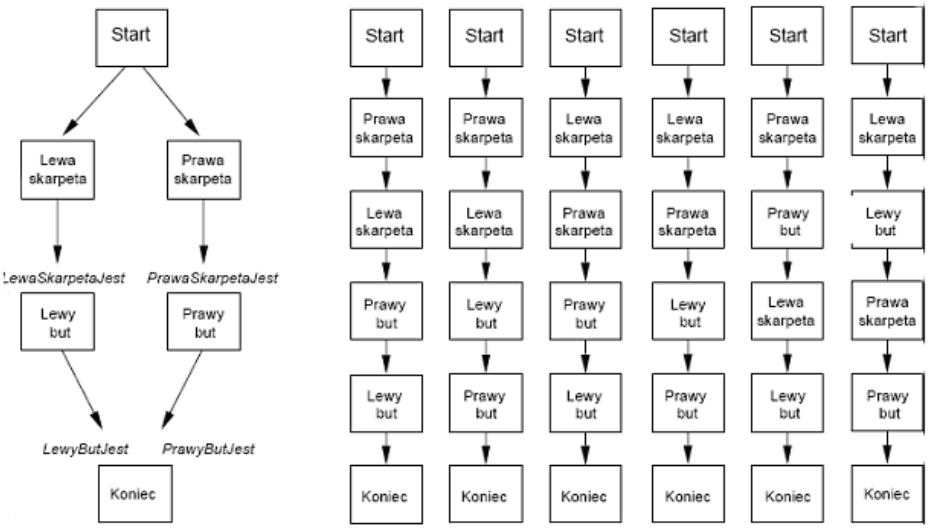
akcje stanowiące rozwiązanie problemu agenta. Plan składa się z instancji operatorów zdefiniowanych dla problemu. Kolejność kroków planu (zastosowanych operatorów) musi być taka, aby były spełnione warunki wstępne (poprzedniki) dla każdej z nich.

8.3.1. Plan częściowo uporządkowany

Plan częściowo uporządkowany (PczU) to plan w którym kolejność niektórych kroków względem siebie jest ustalona, ale niekoniecznie wszystkich kroków. Plan w pełni uporządkowany stanowi sekwencję kroków.

Zamiana planu częściowo uporządkowanego na w pełni uporządkowany jest to tzw. **linearyzacja** planu: plan w pełni uporządkowany uzyskany po dodaniu w nim ograniczeń porządkujących. W ogólności możemy otrzymać wiele alternatywnych wyników linearyzacji planu.

Przykład. Plan częściowo uporządkowany i w pełni uporządkowany (rys. 8.2).



Rys. 8.2: Plan częściowo uporządkowany i alternatywne sposoby jego linearyzacji (według [6]).

Reprezentacja planu PczU

Formalnie biorąc reprezentacja planu PczU obejmuje elementy:

- Zbiór kroków *S*, czyli zbiór zastosowanych operatorów.
- Zbiór ograniczeń porządkujących kroki planu, PORZĄDEK(*plan*), zapisanych w postaci $S_i < S_j$ (tzn. S_i poprzedza S_j, ale niekoniecznie bezpośrednio).
- Zbiór ograniczeń narzuconych na zmienne, o postaci *v*=*x*, gdzie *v* jest zmienną, a *x* - stałą lub inną zmienną, PRZYPISANIA(*plan*).
- Zbiór związków przyczynowo-skutkowych, ZWIĄZKI(*plan*), o postaci $S_i \rightarrow^c S_j$; krok S_i tworzy warunek *c* dla S_j ; taki związek określa powód stosowania kroku S_i.

Plan częściowo uporządkowany może posiadać wiele linearyzacji (każda jest równie dobra, gdyż interesuje nas dowolna sekwencja akcji prowadząca do celu). Ale taki plan może być tez wykonywany częściowo równolegle.

Rozbudowa planu jest sterowana przeszukiwaniem przestrzeni planów. Warunkiem zakończenia jest uzyskanie **planu kompletnego**, czyli planu, w którym warunek każdej jego akcji jest spełniony.

$$\forall(c, S_j) \quad S_i < S_j$$

$c \in \text{EFEKTY}(S_i)$

$\neg\, \exists S_k : \neg c \in \text{EFEKTY}(S_k) \wedge (S_i < S_k < S_j)$

Powyższy zapis mówi o tym, że dla każdego warunku wstępnego dowolnego operatora istnieje inny operator powodujący spełnienie tego warunku i nie istnieje jakikolwiek operator pomiędzy nimi, który kasuje ten warunek.

8.3.2. Tworzenie planu częściowo uporządkowanego

W strategii przeszukiwania przestrzeni planów wyróżnimy następujące kroki (patrz tab. 8-4, 8-5):

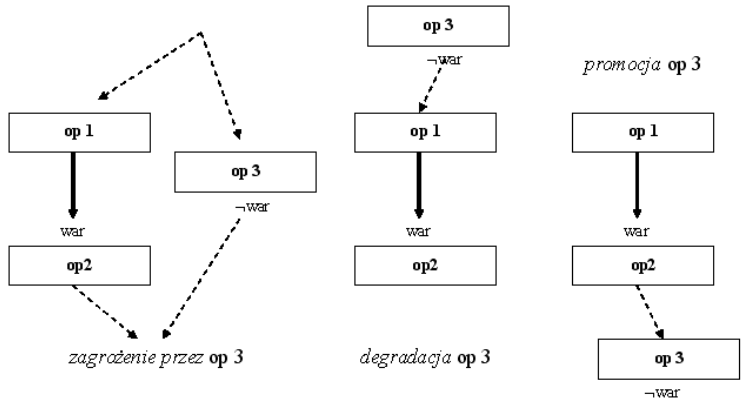
1. Tworzymy plan początkowy zawierający jedynie akcje INIT i GOAL
2. W każdym kolejnym kroku albo dodajemy nową operację albo wykorzystujemy już obecną (poprzez dodanie połączenia przyczynowo-skutkowego), aby zapewnić spełnienie jednego z warunków wstępnych którejś z operacji już utworzonego planu.

Powtarzamy krok 2, dopóki nie uzyskamy planu końcowego (wtedy → koniec) albo operację niewykonalną → wtedy :

3.a. próbujemy zmienić kolejność operacji (degradacja, promocja), lub

3.b. wykonujemy nawrót (w ramach decyzji podjętych w kroku 2) i zmieniamy warunki wykonania operacji.

Rozwiązywanie zagrożeń jest możliwe na dwa sposoby – przez **degradację** (cofnięcie kroku stanowiącego zagrożenie) lub **promocję** (przesunięcie kroku zagrożenia do przodu (rys. 8.3)



Rys. 8.3: Idea dwóch sposobów rozwiązywania zagrożeń.

Tabl. 8-4. Funkcja tworzenie planu częściowo uporządkowanego

```
function AgentPCzU(init, cel, operatory) : returns plan
{
        plan := UtwórzPlanMinimalny(init, cel);
        while(true)
        {
                if (KOMPLETNY(plan) == true) then return plan;
                [krok_wym, c] := WybierzPodcel(plan);
                WybierzOperator(plan, operatory, krok_wym, c);
                krok_zagrożenie := RozwiążZagrożenia(plan);
                if (krok_zagrożenie ≠ ∅)  NawrótNapraw(krok_zagrożenie);
        }
}
```

Tabl. 8-5. Podfunkcje PCzU.

```
function WybierzPodcel(plan) : returns [krok_wym, c]
{       wybierz krok planu (krok_wym) ze zbioru KROKI(plan) o warunku wstępnym c,
        który nie został jeszcze spełniony;
        return [krok_wym, c];
}

procedure WybierzOperator(plan, operatory, krok_wym, c)
{       wybierz krok_dod ze zbioru operatory lub KROKI(plan), taki, że jego efektem jest c;
        if (brak krok_dod ) then return;
        Dodaj związek przyczynowy (krok_dod →c krok_wym) do ZWIĄZKI(plan);
        Dodaj porządek ( krok_dod  <  krok_wym ) do PORZĄDEK(plan);
        if (krok_dod jest nowo dodanym krokiem ze zbiory operatory) then
        {  dodaj krok_dod do KROKI(plan);
           dodaj (Start < krok_dod < Finish ) do PORZĄDEK(plan);
        }
}

function RozwiążZagrożenia(plan) : returns krok_zagrożenie
{
        for (każdy krok_zagrożenie, który zagraża związkowi (krok_A →^c krok_B) ze zbioru ZWIĄZKI(plan)
        do
        {
            Wybierz {
                  Degradacja: dodaj krok_zagrożenie < krok_A do PORZĄDEK(plan);
                        lub
                  Promocja: dodaj krok_B < krok_zagrożenie do PORZĄDEK(plan);
                }
                if (ZGODNY(plan) == false) then return krok_zagrożenie ;
        }
        return ∅ ;
}
```

8.4. Przykład budowy planu

Podamy przykład budowy planu częściowo-uporządkowanego dla problemu agenta potrzebującego kilku produktów ze sklepów. Rozwiązaniem problemu (celem agenta) jest posiadanie: mleka, chleba i kwiatów oraz ponowne przebywanie w domu. Zdefiniujmy podstawowe (abstrakcyjne) akcje agenta:

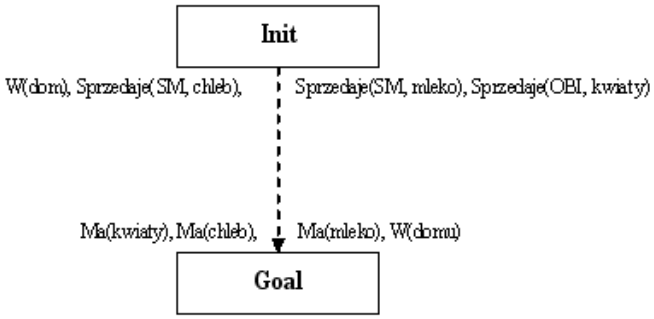
1. Zakładamy, że akcja PÓJŚĆ pozwala przemieścić agenta pomiędzy dowolnymi dwoma miejscami.
2. Akcja KUPUJE oznacza zakup produktów i dla uproszczenia opisu nie wymaga ona precyzowania kwot potrzebnych dla dokonania zakupu. Niech OBI oznacza *market* przemysłowo-budowlany a SM – *supermarket*.
3. INIT to akcja inicjalizacji – wyznacza ona stan początkowy, w którym agent jest w domu a produkty są w sklepach.

4. GOAL to akcja zakończenia – wyznacza ona stan docelowy, w którym agent posiada produkty i jest w domu.

Przykład. Zdefiniujmy operatory, których instancje będą krokami planu.

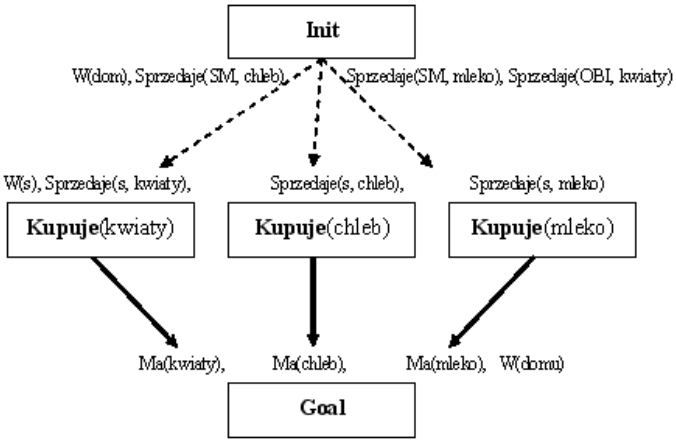
- Op(ACTION: Init,
  EFFECT: W(dom) ∧ Sprzedaje(OBI, kwiaty) ∧ Sprzedaje(SM, mleko) ∧ Sprzedaje(SM, chleb)).
- Op(ACTION: Goal,
  PRECOND: Ma(kwiaty) ∧ Ma(mleko) ∧ Ma(chleb) ∧ W(dom))
- Op(ACTION: Pójść(tam),
  PRECOND: W(tu),
  EFFECT: W(tam) ∧ ¬ W(tu)).
- Op(ACTION: Kupuje(x),
  PRECOND: W(sklep) ∧ Sprzedaje(sklep,x),
  EFFECT: Ma(x)).

Plan początkowy (rys. 8.4) zawiera instancje dwóch domyślnych operatorów INIT i GOAL.



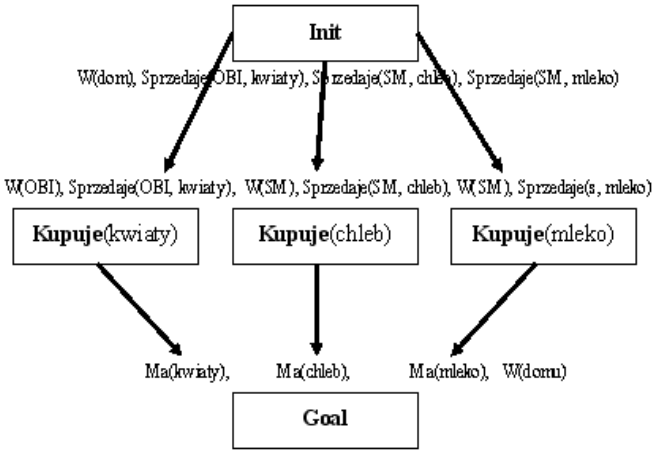
Rys. 8.4: Plan początkowy: PORZĄDEK(*plan_początkowy*) = {Init < Goal }

Rozbudowa planu początkowego – dodane zostają trzy instancje (kroki) operatora KUPUJE, które zapewniają zachodzenie trzech predykatów MA – warunków operatora końcowego (rys. 8.5). Pogrubione strzałki reprezentują związki przyczynowo-skutkowe a cienkie strzałki wskazują „porządek" czyli kolejność wykonywania operacji.



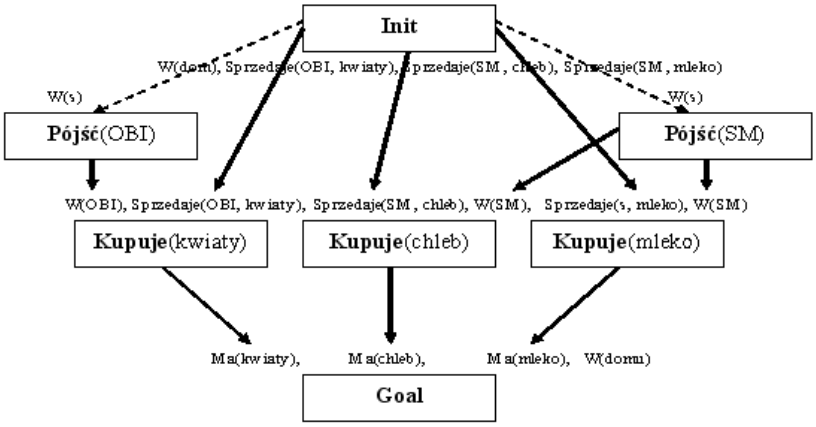
Rys. 8.5: Plan po dodaniu trzech kroków KUPUJE.

Kolejna rozbudowa planu polega na stwierdzeniu faktu, że już istniejący w planie operator INIT wyznacza odpowiednie warunki dla trzech kroków KUPUJE (rys. 8.6).



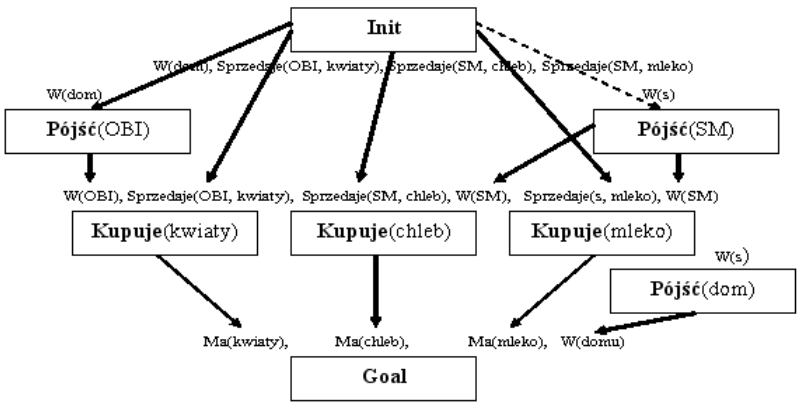
Rys. 8.6: Plan po dodaniu związków przyczynowo-skutkowych INIT - KUPUJ.

Pozostały jeszcze do spełnienia warunki zadane predykatem W dla 4 kroków planu. W tym celu dodamy najpierw dwa kroki – instancje operatora PÓJŚĆ, tworzące warunki dla 3 kroków KUPUJE.



Rys. 8.7: Plan po spełnieniu warunków dla kroków KUPUJ.

Warunki dla kroków PÓJŚĆ zostaną spełnione po dodaniu ich związków przyczynowych z krokiem INIT. Jednak wtedy powstanie kłopot (zagrożenie) spowodowany rozpoczynaniem z domu - agent nie może być następnie jednocześnie w 2 miejscach - OBI i SM. Zanim to jednak nastąpi dodanie już pierwszego związku INIT – PÓJŚĆ zmienia spełniony dotąd od początku warunek W(dom) dla operatora GOAL. Oczywiście zarówno próba degradacji (przed INIT) jak i promocji (po GOAL) zawiodą, więc wykonywany jest nawrót i dodanie nowego kroku PÓJŚĆ(dom) dla spełnienia warunku W(dom) kroku GOAL (rys. 8.7) Teraz ponownie można dodać związek przyczynowy dla INIT – PÓJŚĆ(OBI) (rys. 8.8) .

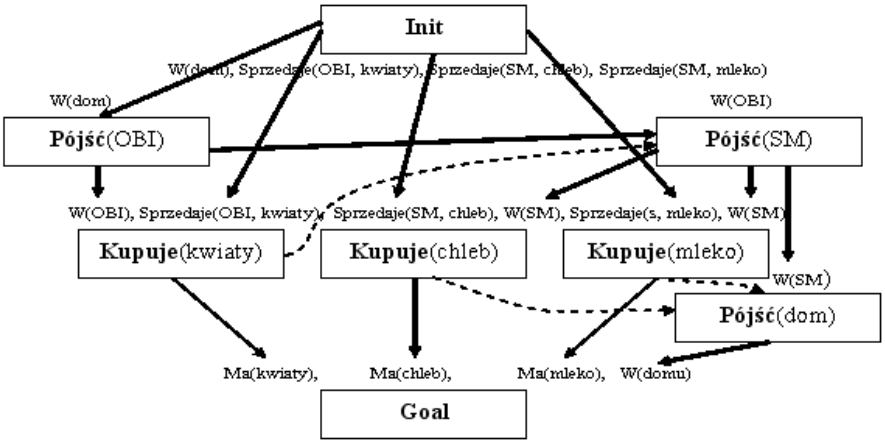


Rys. 8.8: Plan po wprowadzeniu kroku PÓJŚĆ(dom).

Teraz nastąpi próba spełnienia warunku dla operatora PÓJŚĆ(dom). Załóżmy, że dodany zostanie związek przyczynowy z operatorem PÓJŚĆ(SM). Należy jeszcze zapewnić, żeby akcja powrotu do domu wystąpiła po kupieniu mleka i chleba w supermarkecie (dodatkowe dwa kreskowane łuki dla relacji porządku), gdyż zmienia ona warunki dla obu kroków zakupu. Z tego samego powodu również akcja zakupu kwiatów musi odbyć się wcześniej. Ze względu na warunek W(OBI) kroku KUPUJE(kwiaty) niezgodny z warunkiem W(SM) kroku PÓJŚĆ(dom) ta operacja zakupu w OBI musi też odbyć się wcześniej niż akcja PÓJŚĆ(SM). Teraz pozostaje już tylko jeden możliwy związek przyczynowy dla kroku PÓJŚĆ(SM), konieczny po to, aby spełnić jego warunek W(s):

$$\text{PÓJŚĆ(OBI)} \quad \rightarrow^{W(\text{OBI})} \quad \text{PÓJŚĆ(SM)}.$$

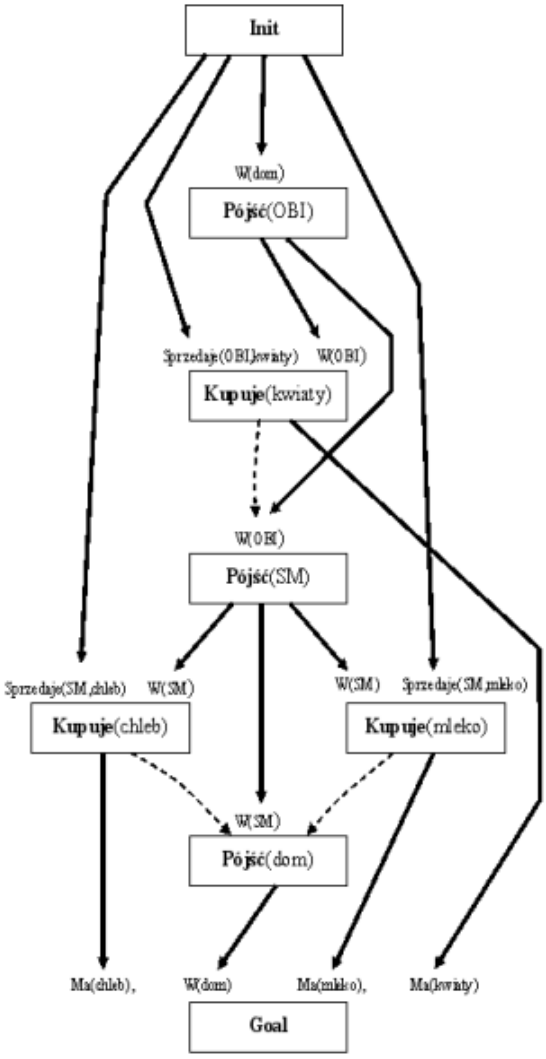
Ostateczny plan częściowo-uporządkowany dla tego problemu przedstawia rys. 8.9.



Rys. 8.9: Końcowy plan częściowo-uporządkowany dla omawianego problemu.

Pozostaje nam jeszcze wykonać linearyzację powyższego planu. W tym celu uporządkujemy kroki tak, aby plan był wykonywany „od góry do dołu". Pogrubione łuki wskazują zależności przyczynowo-skutkowe zapewniające spełnienie warunków dla poszczególnych akcji. Tym samym wyznaczają one także kolejność. Dodatkowo kolejność narzucana jest przez przerywane łuki porządku.

Po uwzględnieniu wszystkich wymagań co do kolejności kroków otrzymujemy postać planu łatwą do linearyzacji. Kroki wykonywane są od „góry do dołu", a kroki znajdujące się na tym samym poziomie mogą być wykonane w dowolnej kolejności lub równolegle (rys. 8.10).



Rys. 8.10: Uporządkowanie kroków w kolejności ich wykonywania w planie częściowo-uporządkowanym.

8.5. Pytania

1. Wyjaśnić problem planowania.
2. Czym są STRIPS i ADL?
3. Omówić podstawową strategię tworzenia planu częściowo-uporządkowanego.
4. Wyjaśnić istotę operacji „promocji" i „degradacji" oraz przyczyny ich stosowania podczas tworzenia planu.

8.6. Zadania

Zad. 8.1

Problem „anomalia Sussmana". Należy uporządkować trzy bloczki (o nazwach A, B i C) stojące na stole w taki sposób, aby A znalazło się na górze B, a ten z kolei – na górze C. Jednak jednocześnie można przemieszczać jedynie jeden blok. Stan początkowy polega na tym, że B znajduje się na stole, zaś C na górze bloku A, a A – na podstawie.

Zaprojektować plan uporządkowania tych 3 bloków: zdefiniować właściwe operatory w STRIPS , prześledzić krok po kroku generację planu częściowo uporządkowanego.

Zad. 8.2

Dany jest problem zmiany zepsutej opony w samochodzie. Mówiąc dokładniej: stan początkowy to: zepsuta opona na osi a dobra opona zapasowa w bagażniku, a stan docelowy to: dobra opona zapasowa właściwie założona na oś samochodu

Mamy do dyspozycji 4 akcje: 1) wyjąć zapasową oponę z bagażnika, 2) zdemontować zepsutą oponę z osi, 3) założyć zapasową oponę na oś, 4) pozostawić samochód na noc bez nadzoru.

Załóżmy, że samochód pozostawiono w wyjątkowo niebezpiecznej okolicy, skutkiem czego w nocy ukradzione zostaną opony. Zdefiniować powyższy problem w języku ADL.

9. Zaawansowane techniki planowania

Graf planujący.

Algorytm „Graphplan”

Planowanie hierarchiczne

Zadania

9.1. Graf planujący

Wprowadzimy teraz specjalną strukturę o nazwie **graf planujący,** która pozwoli przedstawić zależności pomiędzy operatorami i predykatami warunków i efektów akcji wzdłuż „osi czasu”.

Graf planujący obejmuje „stany” charakteryzowane predykatami, „akcje” i mutex-y:

- Sekwencja poziomów odpowiadająca przedziałom czasu – na przemian występują poziomy „stanów” (literały) (S_i) i „akcji” (A_i).
- Poziom S_0 zawiera literały spełnione w stanie początkowym.
- Akcja znajduje się w A_i wtedy, gdy jej warunek jest w S_i. Krawędzie prowadzą od warunków w S_i do efektów akcji w S_{i+1}.
- Krawędź pokazująca trwałość (niezmienność) literału, łączy literał w S_i z tym samym literałem w S_{i+1}.
- Wzajemna rozłączność - reprezentowana jest łukami typu „mutex” („mutual exclusion”):
    - Dwie akcje w A_i wzajemnie się wyłączają, jeśli jedna z nich (jej efekt) neguje warunek wykonania drugiej lub neguje efekt drugiej akcji.
    - Dwa literały w S_{i+1} połączone są łukiem „mutex” jeśli jeden jest negacją drugiego lub jeśli każda para generujących je akcji w A_i jest połączona łukiem „mutex”.
    - Akcja nie może wystąpić w A_i jeśli dowolna para jej warunków jest w S_i uznana za wykluczającą się (jest połączona łukiem „mutex”).
    - Dwie akcje w A_i wzajemnie się wyłączają, jeśli warunek jednej z nich wyklucza się z warunkiem drugiej.

Uwagi.

1. Liczba poziomów w grafie planującym, po których uzyskany zostaje określony literał jest dobrym oszacowaniem (optymistycznym) tego, jak trudno jest uzyskać ten literał wychodząc ze stanu początkowego.
2. Graf planujący wymaga literałów w postaci predykatowej – pozbawionych zmiennych. Ponieważ zarówno STRIPS jak i ADL dopuszczają jedynie literały podstawowe, tzn. występujące w nich termy nie zawierają symboli funkcji, obie reprezentacje mogą zostać przekształcone do postaci predykatowej.

Graf planujący może posłużyć do:

1. wyznaczenia oszacowania heurystyki dla „poinformowanej” strategii przeszukiwania przestrzeni planów, lub
2. do bezpośredniego wyznaczenia planu przez algorytm nazwany GRAPHPLAN-em.

Przykład

Aby „mieć ciastko, zjeść ciastko i nadal je mieć”, zdefiniujemy operatory „jeść” (eat) i „piec” (bake) a warunek sytuacji końcowej utworzą iloczyn predykatów „mieć” (have) i „zjedzone” (eaten) :

OP(Init,

EFFECT: (*have(Cake)* ∧ ¬ *eaten(Cake)*))

OP(Goal,

PRECOND: (*have(Cake)* ∧ *eaten(Cake)*))

OP(Action(*eat(Cake)*),

PRECOND: *have(Cake)*

EFFECT: *eaten(Cake)* ∧ ¬*have(Cake)*))

OP(Action(*bake(Cake)*),

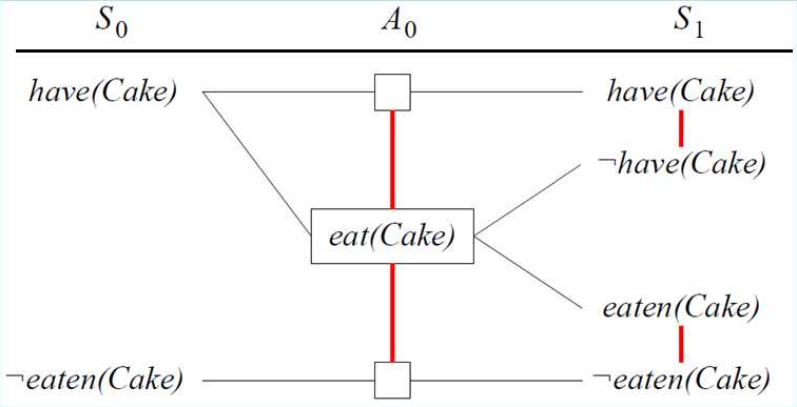
PRECOND: ¬*have(Cake)*

EFFECT: *have(Cake)*))

Początkowy poziom akcji (rys. 9.1).

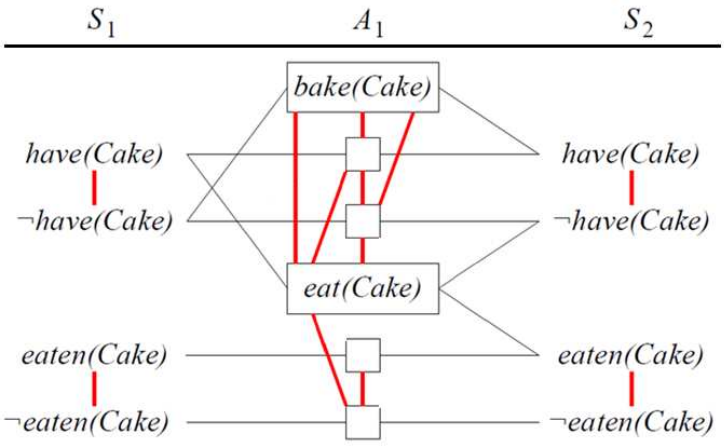
W S_1 literały *have*(*Cake*) i ¬*have*(*Cake*) są ze sobą sprzeczne i tworzą mutex; podobnie jest z *eaten*(*Cake*) i ¬*eaten*(*Cake*).

W A_0 operacja *eat*(*Cake*) tworzy mutex-y z operacjami „przeroczystości”, gdyż ich efekty w S_1 są ze sobą sprzeczne.



Rys. 9.1 Przykładowy problem “zjeść ciastko i mieć ciastko”. Pierwszy poziom akcji i stanu.

Drugi poziom akcji (rys. 9.2) obejmuje operacje „eat(Cake)” i „bake(Cake)”. Operacja *eat(Cake)* tworzy mutex z operacją *bake(Cake)* a obie operacje tworzą mutex-y z wiekszością operacji przezroczystości.



Rys. 9.2 Drugi poziom akcji w omawianym przykładzie.

Osiągnięcie sytuacji końcowej zapewnia sekwencja klastrów operacji:

1: { *eat(Cake)* }

2: { *bake(Cake)*, przezroczytość(*eaten(Cake*))}.

Ale także każda wielokrotność tej pary operacji również spełnia warunek sytuacji końcowej.

Aby wprowadzić jednoznaczność planu należy rozszerzyć warunek operatora *eat(Cake)* o predykat ¬*eaten(Cake).*

9.2. Algorytm „Graphplan”

Algortym **Graphplan** pobiera problem planowania zdefiniowany w STRIPS i wyznacza sekwencję akcji prowadzącą z początkowego stanu problemu do docelowego stanu problemu.

Graphplan operuje na grafie planującym, w którym:

- węzły odpowiadają akcjom i literałom uporządkowanym w naprzemienne poziomy,
- łuki są 3 rodzajów: od literału (warunku) do akcji, od akcji do literału (efektu), od literału do literału (gdy nie ulega zmianie),

Pamiętane są listy wykluczających się literałów, czyli takich, które nie mogą być jednocześnie prawdziwe, a także listy wykluczających się akcji – nie mogą być wspólnie wykonywane na danym poziomie.

Zasada działania (tab. 9-1)

Algorytm “Graphplan” iteracyjnie rozszerza graf planujący (wstecz - „od tyłu do przodu), sprawdzając, czy nie ma rozwiązań o długości *l*-1 zanim zajmie się planami o długości *l*.

Algorytm Graphplan wyznacza plan na podstawie grafu planującego analizując go od ostatniego poziomu wstecz. Iteracyjny sposób analizy odpowiada przeszukiwaniu wszerz pewnego drzewa decyzyjnego:

- Węzłem drzewa jest podzbiór zgodnych ze sobą akcji na jednym poziomie grafu, których efektem są wszystkie te literały, które uznane zostały za cele w poprzedniku węzła.
- Węzeł początkowy jest zbiorem celów na ostatnim poziomie S_n grafu planującego.
- Istnieje krawędź w drzewie decyzyjnym od węzła (=podzbiór w S_i) do węzła(=podzbiór w S_{i-1}), jeśli istnieje podzbiór zgodnych akcji w A_{i-1} , które łączą je jako efekty i warunki.
- Celem przeszukiwania jest osiągnięcie S_0.

Graphplan na przemian:

a) podejmuje wykrycie rozwiązania („czy osiągnięty został stan początkowy problemu”?) i

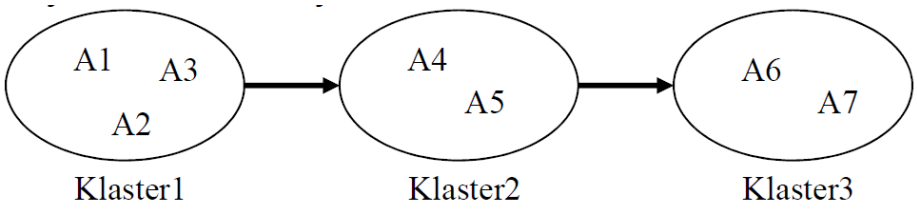
b) rozszerza graf o podzbiory akcji i warunki poprzedniego poziomu w grafie planującym (porusza się WSTECZ wzdłuż grafu planujacego).

Wynikiem pracy algorytmu GRAPHPLAN jest plan, w którym akcje nie są ani w pełni uporządkowane ani częściowo uporządkowane (PCzU). Powstaje pośrednie rozwiązanie (rys. 9.3):

- Plan składa się z sekwencji klastrów akcji,
- W ramach klastra kolejność akcji jest dowolna,
- Kolejność klastrów jest ustalona.

Tab. 9-1 Funkcja GRAPHPLAN

```
function GRAPHPLAN(problem) returns solution lub failure
{  graph := GenerujPlanPoczątkowy(problem);
   goals := Cele[problem];
   loop do {
         if goals niepuste then do {
            solution := WyznaczRozwiązanie(graph, goals,
                                                Długość(graph));
             if solution ≠ failure then return solution;
         }
         else if  BrakRozwiązania(graph)
               then return failure ;
         graph := RozszerzGraf(graph, problem);
         goals := {wszystkie nie-mutex-y na ostatnim poziomie grafu};
   }
}
```



Rys. 9.3 Struktura planu generowanego przez algorytm GRAPHPLAN.

Zalety GRAPHPLAN-u w porównaniu do planera PCzU:

- Znaczna poprawa efektywności w porównaniu z planerem PCzU, dzięki:
  - wzajemnemu wykluczaniu, równoległemu planowaniu, przycinaniu nieosiągalnych zbiorów celów, nie ma potrzeby podstawiania pod zmienne podczas przeszukiwania.
- Zaleta zakończenia pracy, gdy problem jest nierozwiązywalny:
  - wykrycie sytuacji, gdy dany poziom stanów w grafie planującym jest identyczny z kolejnym poziomem;
  - jeśli nie znaleziono planu przed powstaniem takiej sytuacji, to oznacza brak rozwiązania.

9.3. Planowanie hierarchiczne (*)

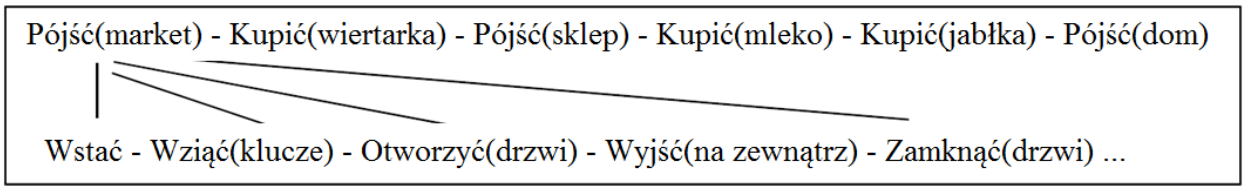
(*) Materiał informacyjny, nieegzaminacyjny

Planowanie hierarchiczne odpowiada konstruowaniu programu metodą **zstępującą** i stosuje zasadę **hierarchicznej dekompozycji:**

- rozpoczynamy od planu opisanego przy pomocy operatorów **abstrakcyjnych** (opis wysokiego poziomu);
- W kolejnych krokach każde wystąpienie abstrakcyjnego operatora zastępuje się jego **implementacją** – czyli **planem**, który go realizuje stosując przy tym operatory niższego poziomu;
- Proces dekompozycji jest kontynuowany tak długo, aż plan zawierać będzie jedynie operatory **podstawowe** (elementarne).

Przykład

Plan o postaci: [Pójść(sklep), Kupić(mleko), Pójść(dom)], jest dobry dla człowieka, ale nie dla robota (maszyny). Nawet pewna konkretyzacja tego planu nam nie wystarczy (rys. 9.4).



Rys. 9.4 Plan wyrażony w terminach abstrakcyjnych akcji,

Potrzebny jest plan konkretny, posługujący się akcjami wykonywalnymi przez robota. Np. NaWprost(2 m), Obrót(+90 stopni).

Wyróżnimy hierarchię warstw abstrakcji dla akcji:

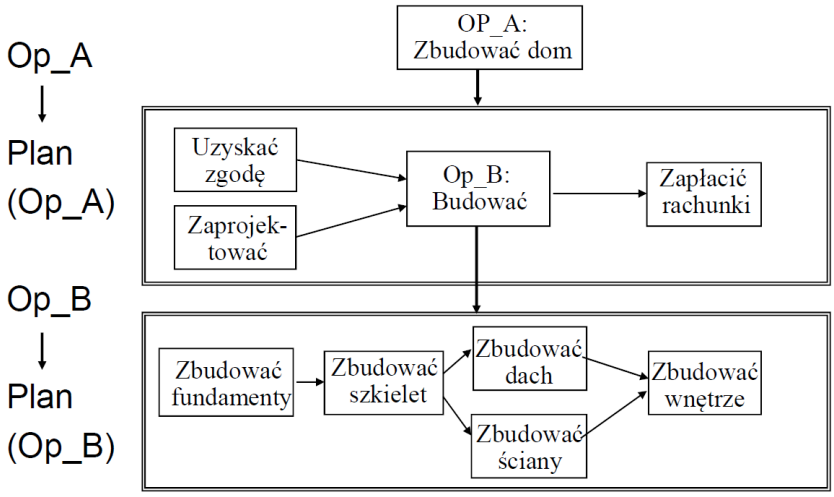
- Operatory najniższego poziomu to **operatory podstawowe**;
- **Operatory abstrakcyjne** występują na wyższych poziomach

Możliwość utworzenia planu dla rozwiązania danego zadania określimy mianem **rozwiązania**.

- **Rozwiązanie abstrakcyjne** to zupełny i poprawny plan zawierający przynajmniej jeden operator (krok) abstrakcyjny.
- **Rozwiązanie** to zupełny i poprawny plan zawierający wyłącznie operatory podstawowe (elementarne).
- Plan P nazywamy **abstrakcją** planu p, jeśli p można otrzymać z P stosując metodę dekompozycji.
- **Własność zstępujących rozwiązań**: każde rozwiązanie abstrakcyjne jest abstrakcją pewnego rozwiązania.
- **Własność wstępujących rozwiązań**: żaden sprzeczny plan abstrakcyjny nie jest abstrakcją rozwiązania.

Przykład dekompozycji zadania

Operator wyższego poziomu odpowiada planowi na niższym poziomie abstrakcji (rys. 9.5)



Rys. 9.5 Przykład planu hierarchicznego.

Do notacji stosowanej w STRIPS dodajemy zbiór metod dekompozycyjnych. Każda metoda jest wyrażeniem postaci Decompose(o, p), gdzie o jest operatorem abstrakcyjnym, a p jest planem. Np. zobacz tablicę 9-6.

Tab. 9.6 Przykład planu hierarchicznego

Decompose(Budować, Plan(KROKI: {S1:Zbudować(fundamenty), S2: Zbudować(szkielet), S3: Zbudować(dach), S4: Zbudować(ściany), S5:Zbudować(wnętrze)} PORZĄDEK: {S1<S2<S3<S5, S2<S4<S5}, PRZYPISANIA: {} ZWIĄZKI: {S1 $\rightarrow^{fundamenty}$ S2, S2 $\rightarrow^{szkielet}$ S3, S2 $\rightarrow^{szkielet}$ S4, S3 $\rightarrow^{dach}$ S5, S4 $\rightarrow^{ściany}$ S5 }))

Co należy uwzględnić podczas dekompozycji operatora?

- Efekty mogą zależeć od sytuacji; Np. Przekręć(kontakt) → światło zapala się, gdy było włączone, w przeciwnym razie gaśnie.
- Czas trwania: akcja mają czas trwania, podlegają ograniczeniom czasowym;
- Zasoby: akcje kosztują zasoby (np. zakupy wymagają pieniędzy), ale akcje mogą też generować nowe zasoby (np. pobrać pieniądze).

Efektywność tworzenia planu hierarchicznego

Przyjmijmy założenie: rozwiązanie wymaga 64 kroków, w każdym występuje 1 z 3 alternatywnych operatorów.

Standardowe rozwiązanie: w najgorszym przypadku wymaga sprawdzenia 3^{64} operacji; złożoność jest wykładniczą funkcją długości rozwiązania.

Planowanie hierarchiczne: jeśli każdy abstrakcyjny operator zostanie zdekomponowany na 4 kroki a liczba alternatywnych dekompozycji wynosi zawsze 3, z których dokładnie 1 jest elementem rozwiązania, to istnienie 64 kroków wymaga 3 warstw i w najgorszym przypadku należy sprawdzić:

3 * 4 + 3 * 4 * 4 + 3 * 4 * 4 *4 operacji; złożoność jest wykładnicza względem liczby warstw hierarchii, ale liczba warstw jest logarytmem z liczby kroków.

Dekompozycja implementacji operatora

Plan P jest **poprawną** implementacją (dekompozycją) operatora O wtedy, gdy zachodzi:

1. Plan P jest poprawny, tzn. zbiory ograniczeń i przypisań są niesprzeczne;
2. Każdy efekt w O jest zapewniony przez jakiś krok w planie P i nie jest ponownie usuwany w innym kroku;
3. Każdy warunek kroku w planie P musi być uzyskany w poprzednim kroku lub być warunkiem wstępnym dla O i przed wykonaniem kroku nie został usunięty.

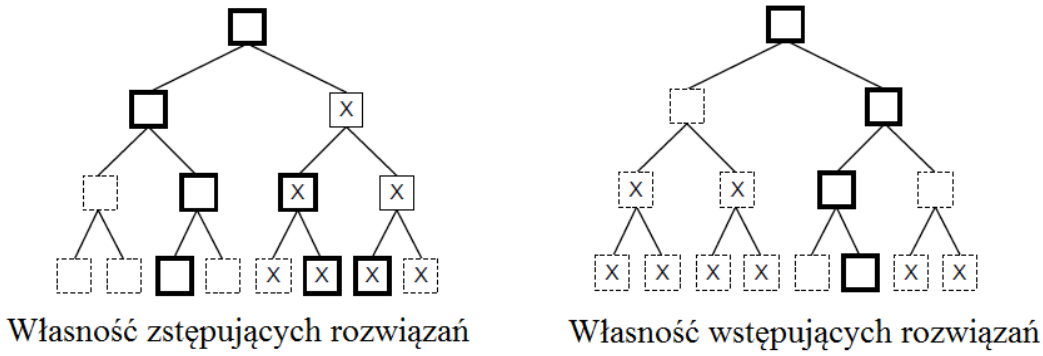
Dekompozycja operatora jest sensowna wtedy, gdy zachodzi ograniczona interakcja pomiędzy pojedynczymi warstwami:

→ ponowne użycie, biblioteki planów.

Dekompozycja musi prowadzić do akcji wykonywalnych przez agenta.

Planowanie hierarchiczne jest możliwe wtedy, gdy zachodzi jeden z warunków (rys. 9.6):

1. dla każdego abstrakcyjnego planu istnieje odpowiedni plan wykonywalny (własność **zstępujących rozwiązań**);
2. żaden niezgodny plan abstrakcyjny nie ma odpowiedniego planu wykonywalnego (własność **wstępujących rozwiązań.**

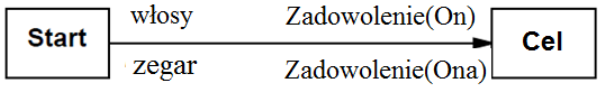


Rys. 9.6

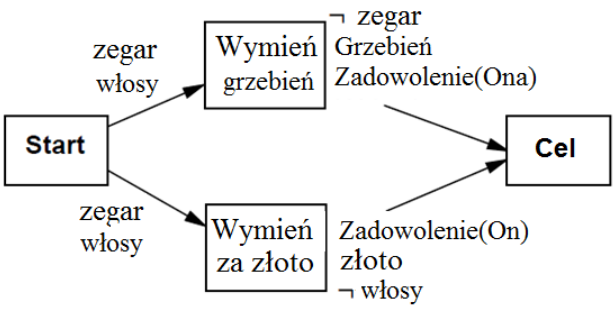
Własność wstępujących rozwiązań nie zawsze zachodzi (rys. 9.7). Należy dokonać modyfikacji niezgodnego planu abstrakcyjnego, aby uzyskać plan zgodny.

Niech, plan=M(o, plan_we), będzie metodą dekompozycyjną – modyfikującą plan „plan_we" do postaci „plan" w wyniku implementacji operatora „o". Operator „a" występujący w p nazywamy **operatorem głównym** metody M, jeśli każdy warunek operatora o jest warunkiem operatora a i każdy efekt operatora o jest efektem operatora a.

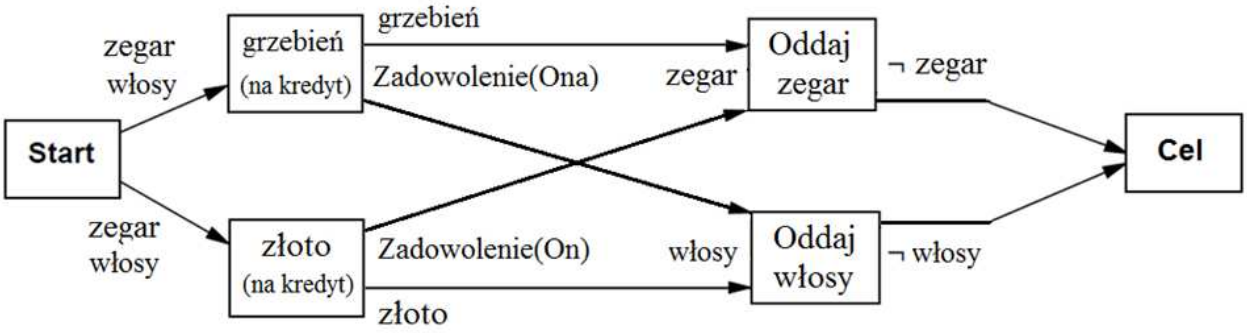
Można pokazać, że jeśli każda metoda dekompozycyjna, którą stosuje planer, posiada dokładnie jeden operator główny, to zagwarantowana jest własność wstępujących rozwiązań.



Plan początkowy



Niezgodny plan abstrakcyjny



Rys. 9.7 Modyfikacja niezgodnego planu w celu uzyskania poprawnego planu:

Tab. 9-1 Algorytm planowania przez hierarchiczną dekompozycję

```
function HD_PCzU(plan, operatory, metody) : returns plan
// plan : abstrakcyjny plan zawierający kroki Init i Goal (i ewentualnie
// dalsze)
{ do { // wykonuj pętlę
        if CzyRozwiązanie(plan) then return plan;
        [krok_wym, c] := WybierzPodcel(plan);
        WybierzOperator(plan, operator, krok_wym, c);
        krok_abstr := WybierzAbstrakcyjny(plan);
        WykonajDekompozycję(plan, metody, krok_abstr);
        krok_zagrożenie := RozwiążZagrożenia(plan);
        if (krok_zagrożenie ≠ ∅)  NawrótNapraw(krok_zagrożenie);
        }
}
```

Procedury WybierzPodcel, WybierzOperator i RozwiążZagrożenie są takie same jak w algorytmie planera PCzU (rozdział 8).

Funkcja CzyRozwiązanie dodatkowo sprawdza, czy każdy operator w planie jest operatorem podstawowym.

Funkcja WybierzAbstrakcyjny wybiera dowolny abstrakcyjny operator występujący w planie.

Procedura WykonajDekompozycję wybiera metodę dekompozycyjną dla instancji operatora *krok*$_{abstr}$ i odpowiednio modyfikuje aktualny plan.

Niech wybraną metodą dekompozycji jest

P=Dekompozycja(*krok*$_{abstr}$, *p*).

Modyfikacja aktualnego planu *P* polega wtedy na:

- Do zbioru kroków planu *P* dodajemy wszystkie operatory planu *p* i usuwamy *krok*$_{abstr}$.
- Do zbioru przypisań zmiennych planu *P* dodajemy przypisania zmiennych planu *p*.
- Zastępujemy każdy element porządku o postaci,

  krok$_a$ < *krok*$_{abstr}$,

  zbiorem porządków { krok$_a$ < krok$_{m1}$, …, krok$_a$ < krok$_{mi}$},

  gdzie *krok*$_{m1}$,…, *krok*$_{mi}$, to maksymalne (w relacji <) kroki w *p*.
- Każdy element porządku postaci

  krok$_{abstr}$ < *krok*$_a$,

  zastępujemy zbiorem { *krok*$_{m1}$ < *krok*$_a$, …, *krok*$_{mj}$ < *krok* $_a$},

  gdzie *krok*$_{m1}$, …, *krok*$_{mj}$ są minimalnymi krokami planu *p*.
- Każdy związek przyczynowy występujący w planie *P* o postaci

  krok$_i$ $\rightarrow^c$ krok$_{abstr}$,

  zastępujemy zbiorem związków o postaci krok$_i$ $\rightarrow^c$*krok*$_m$,

  gdzie *krok*$_m$ jest instancją operatora w *p* z warunkiem wstępnym c, i c nie jest warunkiem wstępnym żadnego wcześniejszego operatora w *p*.
- Podobnie, każdy związek postaci

  krok$_{abstr}$ $\rightarrow^c$ *krok*$_j$, występujący w *P*,

  zastępujemy zbiorem powiązań postaci *krok*$_m$ $\rightarrow^c$ *krok*$_j$,

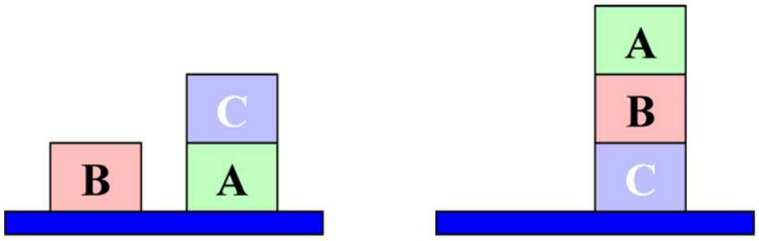
  gdzie *krok*$_m$ jest operatorem w *p* z efektem c i c nie jest efektem żadnego późniejszego operatora w *p*.

Jeśli operacja dekompozycji prowadzi do sprzeczności, wykonujemy nawrót.

9.4. Zadania

Zad. 9.1

Znaleźć plan rozwiązania „problemu Sussmana” stosując algorytm „Graphplan”.



Stan początkowy Stan docelowy

Zdefiniować potrzebne operatory w języku ADL,

Podać jeden wynik częściowy - rozwijany graf planujący, uzyskany po pierwszej iteracji algorytmu planowania,

Podać końcowy wynik.

Zad. 9.2

Planowanie hierarchiczne (TO DO!)

Część III: Niepewność i wnioskowanie bayesowskie

10. Wiedza niepewna i niedokładna

- Wiedza niedoskonała
- Reprezentacja probabilistyczna
- Wnioskowanie przez wyliczanie
- Warunkowa niezależność
- Sieci Bayesa
- Pytania

10.1. Wiedza niedoskonała

W dotychczasowych sposobach reprezentacji wiedzy wychodziliśmy z założenia istnienia wiedzy pewnej, zdanie lub formuła albo zachodziły albo nie. W rzeczywistych problemach realnego świata zwykle nie dysponujemy taką komfortową wiedzą o problemie.

Przykład. Załóżmy, że zamierzamy lecieć samolotem i zastanawiamy się na porą wyjazdu na lotnisko. Niech akcja: A_t oznacza: „*wyjazd na lotnisko na t minut przed odlotem*". Powstaje zasadnicze pytanie (niepewność): „*czy akcja A_t pozwoli mi dotrzeć na czas*?" Nie jest na nie łatwo odpowiedzieć, gdyż jej spełnienie zależy od wielu czynników. Np.

- Częściowa obserwowalność (sytuacja na drodze, plany innych kierowców, itd.)
- Niedokładny pomiar (raporty o ruchu drogowym)
- Niepewność wykonania akcji („złapanie gumy", itp..)
- Znaczna złożoność modelowania i przewidywania ruchu.

Możemy uznać, że rozwiązanie powyższego problemu wyłącznie na gruncie logiki:

- obarczone jest ryzykiem niepowodzenia: jest wysoce prawdopodobne, że przyjęcie "A_{25} pozwoli dotrzeć na czas" nie będzie spełnione; lub
- prowadzi do wniosków, które nie wystarczą do podjęcia decyzji, np. wyliczymy jawnie uwarunkowania: "A_{25} pozwoli dotrzeć na czas jeżeli nie będzie wypadku na moście i nie pada i moje opony będą sprawne, itd.", lub „A_{1440} z wystarczającą pewnością pozwoli dotrzeć mi na czas ale będę musiał nocować na lotnisku …"

Pojęcie „wiedza niedoskonała" jest dość ogólne i obejmuje ono takie przypadki, jak:

1. wiedza niepełna,
2. wiedza niepewna,
3. wiedza niedokładna.

Wiedza niepełna

Z takim rodzajem wiedzy spotykamy się dość często. Na gruncie logiki istnieje pojęcie logiki niemonotonicznej („domyślnej"), której celem jest przetwarzanie wiedzy niepełnej, tzn. takiej, w których dopuszczalne jest, że pojawienie się nowych stwierdzeń może anulować poprawność wcześniejszych formuł. W tym celu w reprezentacji logicznej wprowadza się „wartości domyślne" dla atrybutów obiektów zadanej kategorii. Np. $Rel(R,A,B)$ może oznaczać, że B jest wartością domyślną dla atrybutu R dla obiektu kategorii A.

Np. Zakładam, że mój samochód „nie złapie gumy": $Rel(Guma,$ Samochód, Nie). Zakładam poprawność A_{25}*" :* $Rel(Czas\_wyjazdu,$ Ja, 25)

Zdania wyznaczające te wartości domyślne dla konkretnych obiektów zawartych w bazie wiedzy są tak długo poprawne dopóki nie pojawi się świadectwo zaprzeczające". Główne pytania projektowe: „Jakie założenie jest poprawne? Jak radzić sobie w warunkach wystąpienia zaprzeczenia?"

Przetwarzanie niepewnej wiedzy

Jeśli nie jesteśmy pewni, czy dana formuła jest spełniona czy nie, to w miarę natychmiastowym rozwiązaniem jest rozszerzenie notacji logicznej o wartości wyrażające stopień pewności.

Np. zdefiniujemy reguły ze stopniem ich pewności:

$$,,A_{25} \mid\!\rightarrow_{0.3}$$ „zdążę na czas".

$$,,Zraszacz \mid\!\rightarrow_{0.99} MokraTrawa".$$

$$,,MokraTrawa \mid\!\rightarrow_{0.7} Deszcz".$$

Głównym problemem jest tu łączenie reguł. Np. z przechodniości implikacji wynikałoby, że *„Zraszacz"* z wysokim stopniem pewności powoduje *„Deszcz*"!

Najlepsze narzędzia dla przetwarzania wiedzy niepewnej daje nam rachunek prawdopodobieństwa. Poszczególnym stwierdzeniom przypisuje się prawdopodobieństwo ich prawdziwości a rachunek prawdopodobieństwa pozwala wyznaczać pewność zdarzeń łącznych lub warunkowych, czyli prowadzić **wnioskowanie probabilistyczne** przy niepewnej wiedzy:

Np. modelujemy stopień wiary agenta w powodzenie akcji: *„A_{25}* pozwoli dotrzeć na czas z prawdopodobieństwem 0.04". Stopień wiary w powodzenie akcji (lub ogólnie – danego stwierdzenia) zwykle zależy od obserwacji efektów zjawisk wpływających na powodzenie tej akcji. Modelujemy tę zależność w terminach prawdopodobieństwa warunkowego. W zapytaniach kierowanych do systemu wnioskowania pytamy o prawdopodobieństwa zajścia ukrytych zdarzeń warunkowanych obserwacjami.

Przetwarzanie niedokładnej wiedzy

Wiedza niedokładna to taka, w której przynależność obiektów do niektórych relacji, odpowiadających predykatom, nie jest znana dokładnie. Popularnym sposobem przetwarzania wiedzy niedokładnej jest Popularnym sposobem przetwarzania wiedzy niedokładnej jest **logika rozmyta**. W logice rozmytej mamy do czynienia z funkcją przynależności obiektu do zbioru (wzgl. spełniania relacji), wyrażającą stopień wiarygodności (pewności) przynależenia do zbioru a nie z zerojedynkową funkcją charakterystyczną dla zbioru (relacji). Prowadzi to do odpowiedniego rozszerzenia operacji logicznych z arytmetyki na wartościach {0,1} na arytmetykę dla wartości z przedziału [0, 1].

10.2. Reprezentacja probabilistyczna

Wiedza niepewna jest często efektem naszej niechęci do wyliczania wyjątków, kwalifikowania zjawisk, itp. Jest on także efektem naszej ignorancji – braku znajomości właściwych faktów, warunków początkowych, itp. Nasza wiedza, w tym przyjęte wartości prawdopodobieństwa mają subiektywny charakter. Prawdopodobieństwo twierdzenia odzwierciedla własną wiedzę i przekonania agenta.

Przykład

Podanie wartości: $P(A_{25} \mid \text{brak doniesień o wypadkach}) = 0.06;$ nie jest twierdzeniem o rzeczywistym świecie a subiektywną wiarą agenta w powodzenie akcji. Prawdopodobieństwo twierdzenia zmienia się wraz z nową obserwacją. Np. $P(A_{25} \mid \text{brak doniesień o wypadkach, godz. } 17) = 0.15$

Składnia reprezentacji niepewności

Podstawowym pojęciem w rachunku prawdopodobieństwa jest pojęcie zmiennej losowej. Będą one odpowiadały symbolom języka. Podobnie, jak w rachunku zdań, możliwe światy definiowane są poprzez przypisywanie wartości symbolowi języka (zmiennej losowej).

Ze względu na zakres wartość zmiennej losowej może ona być typu Boolean, np., *Dziura* („czy mam dziurę w zębie?"), być dyskretną zmienną losową, np. *Pogoda* $\in$ {*słoneczna, deszczowa, zachmurzenie, śnieg*} lub ciągłą zmienną o niepoliczalnym zbiorze możliwych wartości. Dla opisu świata za pomocą zbioru zmiennych losowych wymagamy, aby dziedziny wartości tych zmiennych były wyczerpujące i wzajemnie rozłączne.

Elementarne zdanie tworzone jest dzięki przypisaniu wartości do zmiennej losowej. Np.,

$Pogoda = słoneczna, Dziura = false$ (czyli $\neg Dziura$)

Zdania złożone tworzone są ze zdań prostych łączonych spójnikami logicznymi. Np.,

$Pogoda = słoneczna \vee Dziura = false$

Zdarzenie atomowe to zupełna specyfikacja stanu świata, co do którego agent nie ma pewności. Np., jeśli świat składa się jedynie z dwóch zmiennych typu Boolean - *Dziura* i *Ból-zęba*, to mamy 4 różne zdarzenia atomowe:

$Dziura = false \wedge Ból\text{-}zęba = false$
$Dziura = false \wedge Ból\text{-}zęba = true$
$Dziura = true \wedge Ból\text{-}zęba = false$
$Dziura = true \wedge Ból\text{-}zęba = true$

Zdarzenia atomowe są wzajemnie rozłączne i wyczerpujące, tzn. ich zbiór w pełni opisuje wszystkie stany modelowanego świata.

Prawdopodobieństwa a priori ma postać prawdopodobieństwa bezwarunkowego dla zdań. Np., P(*Dziura* = true) = 0.1 i P(*Pogoda* = słoneczna) = 0.72 ; odpowiada początkowej wierze zanim nie dotrą nowe obserwacje.

Rozkład prawdopodobieństwa podaje wartości dla wszystkich możliwych przypisań do zmiennej. Np. $P(Pogoda) = [0.72, 0.1, 0.08, 0.1]$ (rozkład jest znormalizowany, tzn. suma elementów wektora wynosi 1).

Rozkład **łącznego prawdopodobieństwa** dla zbioru zmiennych losowych podaje prawdopodobieństwa każdego zdarzenia atomowego nad tymi zmiennymi. Np. rozkład łączny dwóch uprzednio definiowanych zmiennych losowych wynosi, $P(Pogoda,Dziura) = [\ ]_{4 \times 2}$ (macierz o rozmiarze 4x2) (tabela 10-1).

Tab. 10-1 Rozkład łączny dwóch zmiennych dyskretnych.

Pogoda =	*słoneczna*	*deszcz*	*zachmurzenie*	*śnieg*
Dziura = true	0.144	0.02	0.016	0.02
Dziura = false	0.576	0.08	0.064	0.08

Prawdopodobieństwo a posteriori ma postać prawdopodobieństwa warunkowego, gdyż nasza pewność zdarzenia warunkowana jest teraz nowo wykonaną obserwacją. Np., P(*dziura* | *ból-zęba*) = 0.8 , tzn. określamy prawdopodobieństwo zajścia dziury przy założeniu, że zachodzi *ból-zęba* i jest to wszystko co wiemy o tym świecie. Aby zdefiniować cały rozkład warunkowy tych zmiennych, $P(Dziura \mid Ból\text{-}zęba)$, trzeba podać wartości dwóch wektorów dwu-elementowych.

Jeśli wiemy więcej, np. to że rzeczywiście istnieje dziura, to oczywiście mamy: P(*dziura* | *ból-zęba, dziura*) = 1. Z drugiej strony nowa obserwacja może nie mieć żadnego znaczenia dla oceny prawdopodobieństwa dziury w zębie. Np. P(*dziura* | *ból-zęba, słoneczna*) = P(*dziura* | *ból-zęba*) = 0.8.

Definicja **prawdopodobieństwa warunkowego** zdarzenia odwołuje się do zdarzenia łącznego zmiennej zależnej i niezależnej:

$$P(a \mid b) = P(a \wedge b) / P(b) \text{ , jeśli } P(b) > 0 .$$

Zauważmy, że $P(a|b)$ i $P(b|a)$ korzystają z tego samego zdarzenia łącznego $P(a \wedge b)$. Stąd powstaje **reguła dla iloczynu** dwóch zmiennych, która wiąże go z dwoma prawdopodobieństwami warunkowymi:

$$P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$$

Istnieje odpowiednia postać tej reguły dla całego rozkładu łącznego dwóch zmiennych:

$$P(A, B) = P(A \mid B)\, P(B) = P(B \mid A)\, P(A)$$

Np. rozkład łączny, $P(Pogoda,Dziura) = P(Pogoda \mid Dziura)\, P(Dziura)$, uzyskuje sie z wyliczenia 4×2 równań, a nie mnożenia macierzy.

Regułę dla sekwencji (łańcucha) większej liczby zmiennych otrzymamy po wielokrotnym zastosowaniu reguły dla iloczynu:

$$\begin{aligned} P(X_1, \ldots, X_n) &= P(X_1,\ldots,X_{n-1})\, P(X_n \mid X_1,\ldots,X_{n-1}) \\ &= P(X_1,\ldots,X_{n-2})\, P(X_{n-1} \mid X_1,\ldots,X_{n-2})\, P(X_n \mid X_1,\ldots,X_{n-1}) \\ &= \ldots \\ &= \Pi_{i=1}^{n} P(X_i \mid X_1, \cdots, X_{i-1}) \end{aligned}$$

10.3. Wnioskowanie przez wyliczanie

W systemach wnioskowania probabilistycznego zwykle wyróżniamy trzy grupy zmiennych: zmienne reprezentujące ukryte przyczyny Y, zmienne reprezentujące obserwacje E, zmienne ukryte H.

W procesie wnioskowania dysponującego zbiorem zmiennych X interesuje nas ustalenie rozkładu łącznego a posteriori dla zmiennych zapytania Y, przy zadanych specyficznych wartościach e dla zmiennych obserwowanych E. Pozostałe zmienne są to tzw. zmienne ukryte: $H = X - Y - E$

Dla uzyskania wyniku wnioskowania wymagane jest sumowanie prawdopodobieństw łącznych dla zmiennej zapytania, wykonywane po wszystkich ukrytych zmiennych, przy zadanych wartościach zmiennych obserwacji:

$$P(Y \mid E = e) = \alpha\, P(Y, E = e) = \alpha\, \Sigma_h P(Y, E= e, H= h)$$

Niedogodnością tego postępowania jest wysoka złożoność obliczeniowa wynosząca w najgorszym przypadku $O(d^n)$, *gdzie n jest liczbą zmiennych a d* jest największą liczbą realizacji zmiennej. Złożoność pamięciowa jest taka sama - $O(d^n)$.

Przykład. Wnioskowanie przez wyliczanie. Dany jest rozkład prawdopodobieństwa łącznego trzech zmiennych (tab. 10-2).

Tab. 10-2: Rozkład łączny trzech zmiennych binarnych.

Ból-zęba / Dziura	*ból-zęba*		¬ *ból-zęba*	
	wykryte	¬*wykryte*	*wykryte*	¬*wykryte*
dziura	.108	.012	.072	.008
¬*dziura*	.016	.064	.144	.576

Dla każdego zdania *s*, sumujemy zdarzenia atomowe, dla których zdanie to zachodzi:

$$P(s) = \Sigma_{\omega:\omega \models s} P(s)$$

Np. $P(\textit{ból-zęba}) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$ (tab. 10-.3).

Tab. 10-3: Prawdopodobieństwo zdarzenia jednej zmiennej obliczane na podstawie rozkładu łącznego trzech zmiennych binarnych.

Ból-zęba / Dziura	*ból-zęba*		¬ *ból-zęba*	
	wykryte	*¬wykryte*	*wykryte*	*¬wykryte*
dziura	.108	.012	.072	.008
¬ dziura	.016	.064	.144	.576

Na podstawie tabelki 10-3 możemy policzyć prawdopodobieństwo warunkowe zdarzenia. Np. tabelka 10-4 ilustruje sposób wyliczenia poniższego zdarzenia:

$$P(\neg dziura \mid ból - zeba) = \frac{P(\neg dziura \wedge ból - zeba)}{P(ból - zeba)}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

Tab. 10-4: Prawdopodobieństwo zdarzenia warunkowego obliczane na podstawie rozkładu łącznego trzech zmiennych binarnych.

Ból-zęba / Dziura	*ból-zęba*		¬ *ból-zęba*	
	wykryte	*¬wykryte*	*wykryte*	*¬wykryte*
dziura	.108	.012	.072	.008
¬ *dziura*	.016	.064	.144	.576

Alternatywnie możemy potraktować mianownik jako współczynnik normalizujący α, który da się policzyć na podstawie warunku takiego, że suma wszystkich realizacji rozkładu prawdopodobieństwa warunkowego wynosi 1. Np. policzmy rozkład zmiennej lodowej Dziura pod warunkiem zdarzenia (Ból-zęba = *ból-zęba*):

$P(\text{Dziura} \mid \textit{ból-zęba}) = \alpha\ P(\text{Dziura}, \textit{ból-zęba})$

$= \alpha\ [P(\text{Dziura}, \textit{ból-zęba}, \textit{wykryta}) + P(\text{Dziura}, \textit{ból-zęba}, \neg\textit{wykryta})]$

$= \alpha\ [< 0.108, 0.016 > + < 0.012, 0.064 >] = \alpha < 0.12, 0.08> = <0.6, 0.4>$

W powyższym α wynosi 5, gdyż zachodzi: $5 \cdot 0.12 + 5 \cdot 0.08 = 1$.

Ogólna zasada postępowania w metodzie wnioskowania przez wyliczanie: znajdź rozkład dla badanej zmiennej ustalając wartości zmiennych obserwowanych i sumując po wszystkich realizacjach zmiennych ukrytych.

Niezależność zmiennych

Dwie zmienne losowe A i B są niezależne wtw. $P(A|B) = P(A)$, $P(B|A) = P(B)$ i $P(A, B) = P(A)\ P(B)$

Np. dla zbioru zmiennych {Dziura, Ból-zęba, Wykryta, Pogoda} możemy uznać, że niezależna jest zmienna Pogoda od pozostałych zmiennych {Dziura, Ból-zęba, Wykryta}. Wtedy zachodzi:

P(Ból-zęba, Wykryta, Dziura, Pogoda) = P(Ból-zęba, Wykryta, Dziura) P(Pogoda).

Zamiast 32 zdarzeń łącznych, dla których należałoby podać wartości prawdopodobieństwa, musimy teraz podać jedynie (8 + 4 =) 12 wartości prawdopodobieństwa.

Pełna niezależność zmiennych losowych jest pożądana, ale w praktyce wystąpi bardzo rzadko. Co zrobić, jeśli nasz problem obejmuje setki zależnych zmiennych? W takiej sytuacji może pomóc *niezależność warunkowa*.

10.4. Warunkowa niezależność

Definicja. Zmienne X, Y są warunkowo niezależne pod warunkiem zmiennej Z jeżeli zachodzi:

$$\mathrm{P}(X, Y \mid Z) = \mathrm{P}(X|Z)\ \mathrm{P}(Y|Z).$$

Własności

1. Niezależność X i Y nie implikuje niezależności warunkowej.

2. Niezależność warunkowa $(X, Y \mid Z)$ nie implikuje niezależności X i Y.

3. Niezależność warunkowa $(X, Y \mid Z)$ nie implikuje niezależności dla negacji warunku - $(X, Y \mid \neg Z)$.

Przykład. Rozkład łączny trzech zmiennych $\mathrm{P}(\textit{Ból-zęba, Dziura, Wykryta})$ ma $(2^3-1=)\ 7$ niezależnych elementów.

„*Jeśli masz dziurę, to prawdopodobieństwo jej wykrycia podczas badania nie zależy od tego, czy odczuwasz ból-zęba*":

(1) $\mathrm{P}(\textit{wykryta} \mid \textit{ból-zęba} \wedge \textit{dziura}) = \mathrm{P}(\textit{wykryta} \mid \textit{dziura})$

„*Ta niezależność zachodzi również wtedy, gdy nie masz dziury*":

(2) $\mathrm{P}(\textit{wykryta} \mid \textit{ból-zęba} \wedge \neg\textit{dziura}) = \mathrm{P}(\textit{wykryta} \mid \neg\textit{dziura})$

Mówimy, że zmienna *Wykryta* ***jest*** warunkowo niezależna od *Ból-zęba* pod warunkiem zmiennej *Dziura*.

Dla takich warunkowo niezależnych zmiennych *Wykryta* i *Ból-zęba* mamy równoważne zdania:

$\mathrm{P}(\textit{Ból-zęba} \mid \textit{Wykryta, Dziura}) = \mathrm{P}(\textit{Ból-zęba} \mid \textit{Dziura})$

$\mathrm{P}(\textit{Ból-zęba, Wykryta} \mid \textit{Dziura}) = \mathrm{P}(\textit{Ból-zęba} \mid \textit{Dziura})\ \mathrm{P}(\textit{Wykryta} \mid \textit{Dziura})$

Wypiszemy pełny rozkład łączny w naszym przykładzie stosując regułę sekwencji (łańcucha):

$$\mathrm{P}(\textit{Ból-zęba, Wykryta, Dziura}) =$$

$$= \mathrm{P}(\textit{Ból-zęba} \mid \textit{Wykryta, Dziura})\ \mathrm{P}(\textit{Wykryta, Dziura})$$

$$= \mathrm{P}(\textit{Ból-zęba} \mid \textit{Wykryta, Dziura})\ \mathrm{P}(\textit{Wykryta} \mid \textit{Dziura})\ \mathrm{P}(\textit{Dziura})$$

Ale zmienne *Wykryta* i *Ból-zęba* są warunkowo niezależne pod warunkiem *Dziura*. Stąd wynika możliwość uproszczenia obliczeń:

$$\mathrm{P}(\textit{Ból-zęba, Wykryta, Dziura}) = \mathrm{P}(\textit{Ból-zęba} \mid \textit{Dziura})\ \mathrm{P}(\textit{Wykryta} \mid \textit{Dziura})\ \mathrm{P}(\textit{Dziura})$$

Zamiast 8 wartości prawdopodobieństwa (z czego 7 niezależnych) dla rozkładu łącznego 3 zmiennych losowych binarnych wystarczy nam podanie $(2 + 2 + 1 =)\ 5$ niezależnych wpisów (wartości prawdopodobieństw. To co wydaje się być jeszcze małą korzyścią w tym przykładzie, bardzo szybko zyskuje na znaczeniu wtedy, gdy rośnie liczba zmiennych przyjmujących wiele możliwych wartości.

W większości przypadków stosowanie niezależności warunkowej zmniejsza rozmiar reprezentacji rozkładu łącznego dla n zmiennych z wykładniczego względem n na liniowy względem n. Dlatego

też niezależność warunkowa jest podstawowym i efektywnym sposobem reprezentacji niepewnej wiedzy w modelu probabilistycznym.

Reguła Bayesa

Z reguły dla iloczynu zdarzeń zmiennych losowych:

$$P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$$

wynika **reguła Bayesa**:

$$P(a \mid b) = P(b \mid a)\, P(a) / P(b)$$

W ogólności dla całego rozkładu prawdopodobieństwa zmiennej reguła Bayesa przyjmuje postać:

$$P(Y \mid X) = P(X \mid Y)\, P(Y) / P(X) = \alpha\, P(X \mid Y)\, P(Y)$$

Pożyteczność reguły Bayesa polega na tym, pozwala wyrazić nieznane prawdopodobieństwo $P(Y \mid X)$ ukrytej przyczyny Y warunkowane znanymi obserwacjami X na podstawie znanego i łatwiej definiowalnego prawdopodobieństwa „przyczyna – skutek", tzn. $P(X \mid Y)$.

$$P(przyczyna \mid skutek) = P(skutek \mid przyczyna)\, P(przyczyna) / P(skutek)$$

Jeżeli jest wiele zmiennych reprezentujących *skutek* to wprowadzone wcześniej pojęcie warunkowej niezależności pozwala nam zmniejszyć wymiar rozpatrywanych rozkładów. Np. korzystając z warunkowej niezależności dwóch skutków *Ból-zęba* i *Wykryta* względem ukrytej przyczyny *Dziura* otrzymamy:

$$P(Dziura \mid ból\text{-}zęba \wedge wykryta) = \alpha\, P(ból\text{-}zęba \wedge wykryta \mid Dziura)\, P(Dziura)$$

$$= \alpha\, P(ból\text{-}zęba \mid Dziura)\, P(wykryta \mid Dziura)\, P(Dziura)$$

Zależności przyczynowo skutkowe i warunkową niezależność możemy wyrazić w postaci graficznej (tzw. sieć Bayesa). Dla sytuacji podanej na rys. 10.1 prawdopodobieństwo zdarzenia łącznego wyznaczymy jako iloczyn prawdopodobieństwa zmiennej *Przyczyna* i warunkowych prawdopodobieństw zmiennych zależnych ($Efekt_i$):

$$P(Przyczyna, Efekt_1, \ldots, Efekt_n) = P(Przyczyna)\, \Pi_i\, P(Efekt_i \mid Przyczyna)$$

To oznacza, że dla n zmiennych liczba czynników wynosi n i złożoność obliczeniowa jest liniowa względem n.

Rys. 10.1: Graficzna reprezentacja niezależności warunkowej zmiennych i wiedzy przyczynowo- skutkowej.

10.5. Sieć Bayesa

Sieć Bayesa to prosta notacja graficzna przeznaczona dla wyrażenia warunkowej niezależności i tym samym dla zwartego reprezentowania pełnych rozkładów łącznych wielu zmiennych losowych.

Składnia:

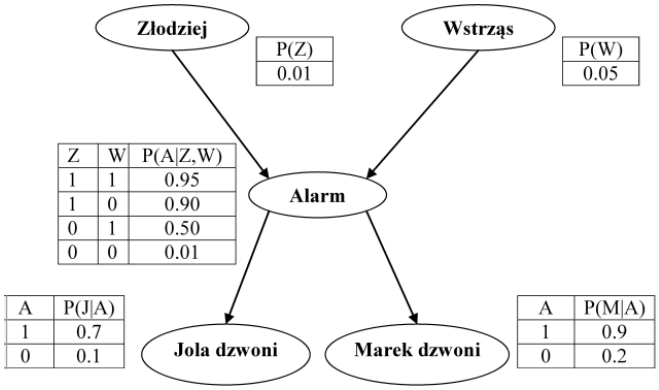
- zbiór węzłów, jeden węzeł dla jednej zmiennej
- acykliczny graf skierowany

- zawiera rozkład warunkowy dla każdego węzła względem zadanych węzłów „rodziców”:

  $P(X_i \mid \text{Rodzice}(X_i))$

- W najprostszej postaci rozkład warunkowy dla węzła podany jest w postaci tablicy, dając prawdopodobieństwa dla wartości X_i przy zadanych kombinacjach wartości zmiennych - rodziców.

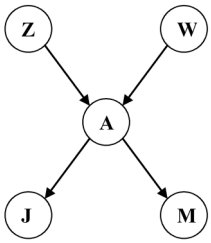
Przykład. Jestem w pracy. Dzwoni sąsiadka Jola i mówi, że dzwoni alarm w moim domu, lecz drugi sąsiad Marek nie dzwoni. Czasem alarm wywalany jest przez wstrząsy i wibracje. A może jest tam złodziej? Wprowadzamy zmienne: „Złodziej, Wstrząs, Alarm, JolaDzwoni, MarekDzwoni”. Sieć reprezentuje wiedzę „przyczynowo-skutkową” (rys. 10.2): złodziej może wyzwolić alarm; wstrząsy mogą wyzwolić alarm; alarm może spowodować telefon od Marka; alarm może spowodować telefon od Joli.



Rys. 10.2: Ilustracja sieci Bayesa dla problemu „przyczyny i skutki alarmu”.

Zaletą sieci Bayesa jest to, że pełny (globalny) rozkład łączny wszystkich zmiennych opisujących dany problem jest iloczynem jedynie lokalnych rozkładów warunkowych (rys. 10.3):

$$P(X_1, \ldots, X_n) = \Pi_i P(X_i \mid Rodzice(X_i))$$



Rys. 10.3: Struktura sieci Bayesa wyraża zmienne i ich lokalne zależności, których rozkłady prawdopodobieństwa należy zdefiniować.

Np. dla powyższego przykładu zdarzenie łączne 5 zmiennych jest iloczynem 5 zdarzeń warunkowych (z wyjątkiem zmiennych niezależnych T, Z).

$$P(J \wedge M \wedge A \wedge \neg Z \wedge \neg W) = P(J \mid A)\, P(M \mid A)\, P(A \mid \neg Z, \neg W)\, P(\neg Z)\, P(\neg W)$$
$$= 0.7 \cdot 0.9 \cdot 0.01 \cdot 0.99 \cdot 0.95 \approx 0.00059 .$$

Zwarta reprezentacja

Tablica wartości rozkładu warunkowego dla zmiennej X_i o typie Boolean i zależnej od k rodziców typu Boolean posiada 2^k wierszy, wynikających z kombinacji wszystkich wartości rodziców. Każdy wiersz wymaga jedynie reprezentacji liczby p dla zdania ($X_i = true$), gdyż w tym przypadku prawdopodobieństwo zdania ($X_i = false$) to po prostu 1-p. Jeśli żadna zmienna nie posiada więcej niż k rodziców to cała sieć wymaga zdefiniowania rzędu $O(n \cdot 2^k)$ liczb. To oznacza, że rozmiar danych

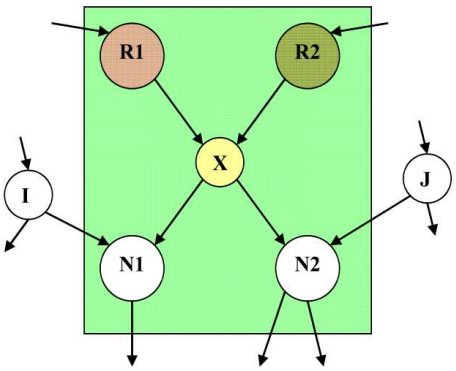
dla sieci rośnie liniowo względem n, podczas gdy dla pełnego rozkładu łącznego ta złożoność wynosi $O(2^n)$.

Przykład. Dla poprzedniej sieci, opisującej przyczyny i skutki alarmu, liczba elementów w tabelach prawdopodobieństwa wynosi ($1 + 1 + 4 + 2 + 2 =$) 10, podczas gdy pełen rozkład łączny wymagałby zdefiniowania (2^5-$1 =$) 31 liczb.

Lokalna semantyka i koc Markowa

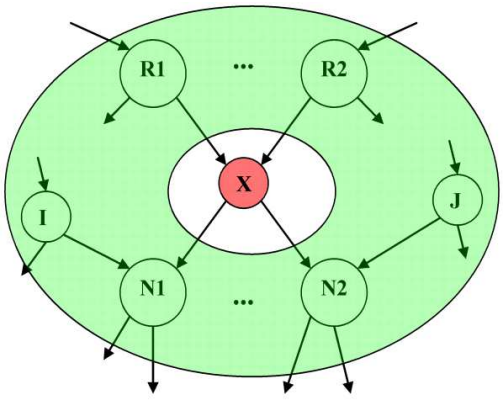
Teraz zastanówmy się jaki jest zakres lokalnych zależności, poprzez które wyraża się znaczenie (semantyka) danej zmiennej dla opisu problemu (wyznaczenia prawdopodobieństwa zdarzenia łącznego). Rozpatrzymy najogólniejszy przypadek zmiennej ukrytej, tzn. posiadającej zarówno rodziców jak i następców w sieci Bayesa.

Lokalna semantyka: każdy węzeł (zmienna) jest warunkowo niezależny od węzłów, nie będących jego następcami, jeśli zadani są jego (jej) rodzice. Będą to jedyne rozkłady warunkowe, poprzez które wyraża się znaczenie danego węzła dla opisu problemu. Wystarczy rozpatrywać dany węzeł jako zależny jedynie od swoich rodziców i swoich następców (rys. 10.4).



Rys. 10.4: Lokalna semantyka – zależności zmiennej wtedy, gdy zadani są jej rodzice.

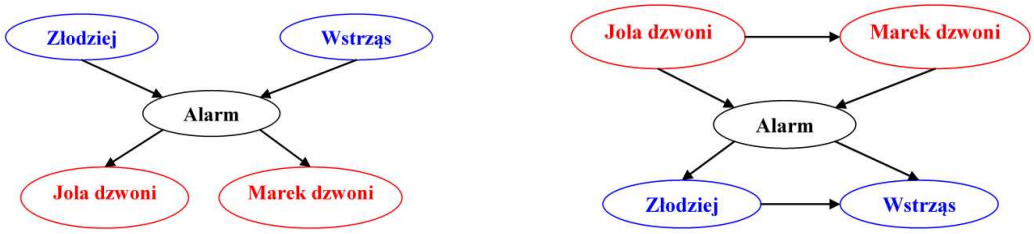
Pojęcie „koc Markowa" jest bardziej ogólne od „lokalnej semantyki". Przedstawia ono wszystkie potencjalne węzły nadające znaczenie danej zmiennej. Koc Markowa dla danego węzła tworzą łącznie: jego *rodzice* + *jego dzieci* + *rodzice dzieci* (rys. 10.5). Każdy węzeł jest warunkowo niezależny od innych nie należących do jego koca Markowa.



Rys. 10.5: Koc Markowa.

Koc Markowa wyraża to, że zasadniczo możliwe są dwa rodzaje lokalnych struktur, w których zależności są niejako odwrócone i różniące się lokalną semantyką. Są to struktury: przyczynowo-skutkowa i diagnostyczna (rys. 10.6). Struktura przyczynowo-skutkowa jest naturalnym sposobem definiowania wiedzy przez człowieka. Sieciami o takiej strukturze posługujemy się najczęściej.

Alternatywnie struktura diagnostyczna określa związki prowadzące od obserwacji do ukrytej przyczyny. W większości problemów trudno jest zdefiniować rozkłady warunkowe wyrażające taką lokalną semantykę.



(a) (b)

Rys. 10.6: Dwie struktury sieci o różnych lokalnych semantykach dla opisu tego samego problemu: (a) przyczynowo-skutkowa i (b) diagnostyczna.

Konstrukcja sieci Bayesa

Typowy algorytm konstrukcji sieci Bayesa (tab. 10-5) wykorzystuje lokalne struktury. Nie przesądza on o semantyce tych struktur – mogą one być zarówno przyczynowo-skutkowe jaki i diagnostyczne. Po ich zidentyfikowaniu i zdefiniowaniu odpowiadających im rozkładów prawdopodobieństwa łatwo jest już stworzyć sieć o globalnej semantyce.

Tabl. 10-5. Algorytm konstrukcji sieci Bayesa.

1. Wybierz porządek zmiennych $X_1, \ldots, X_n$ - zaczynamy od zmiennych niezależnych (rodziców), potem dodajemy ich „dzieci" a na końcu zmienne „liście" nie mające „dzieci" - wierzchołki reprezentują funkcje zmiennych losowych.
2. **for** $i = 1$ **to** n
- dodaj X_i do sieci
- wybierz rodziców spośród $X_1, \ldots, X_{i-1}$ tak, aby $P(X_i \mid rodzice(X_i)) = P(X_i \mid X_1, \ldots X_{i-1})$

  Wybór rodziców gwarantuje to, że:

  $P(X_1, \ldots, X_n) = \pi_{i=1} P(X_i \mid X_1, \ldots, X_{i-1})$ (reguła łańcucha) =

  = $\pi_{i=1} P(X_i \mid rodzice(X_i))$ (zapewnione przez konstrukcję sieci)

Przykład. Załóżmy, że w problemie „złodziej czy trzęsienie" wybraliśmy kolejność: *J, M, A, Z, T.* Taka kolejność odpowiada strukturze diagnostycznej ani przyczynowo-skutkowej w tym problemie. Powstaje sieć o strukturze podanej na rys. 10.6(b). Sieć jest mniej zwarta niż poprzednio: wymaga teraz podania $(1 + 2 + 4 + 2 + 4 =)\ 13$ liczb wyrażających wartości zdarzeń rozkładów warunkowych.

Podsumowanie podejścia probabilistycznego

1. Rachunek prawdopodobieństwa jest formalizmem zapisu niepewnej wiedzy.
2. Rozkład prawdopodobieństwa łącznego wyznacza prawdopodobieństwa zajścia każdego zdarzenia atomowego.
3. Wnioskowanie może polegać na sumowaniu nad wszystkimi zdarzeniami atomowymi spełniającymi zapytanie i obserwacje.
4. Dla złożonych dziedzin zmiennych losowych należy znaleźć sposób redukcji rozmiaru prawdopodobieństwa łącznego. Niezależność i warunkowa niezależność umożliwiają redukcję powyższego rozmiaru.
5. Sieci Bayesa są naturalną reprezentacją dla (indukowanej przyczynowo) niezależności warunkowej. Topologia sieci i rozkłady warunkowe dla zmiennych-węzłów sieci tworzą zwartą reprezentację rozkładu łącznego.

10.6. Wnioskowanie w logice rozmytej

Logika rozmyta to metodyka przetwarzania **wiedzy niedokładnej.** Za jej pomocą pojęcia **nieprecyzyjne**, takie jak „*być może*", „*prawie*", „*dużo*", „*mało*", mogą być stosowane we wnioskowaniu, które prowadzi do **nieprecyzyjnych** konkluzji na podstawie nieprecyzyjnych przesłanek.

- Klasyczna teoria mnogości bazuje również na klasycznej logice. Rozważmy zbiór wartości *D*. Każdy podzbiór $A \subseteq D$ może być opisany za pomocą funkcji charakterystycznej:

$$\chi_A(x) = \begin{cases} 0, & x \notin A \\ 1, & x \in A \end{cases}$$

  Wyrażenie, $x \in A$, może być albo prawdziwe albo fałszywe.

- W teorii **zbiorów rozmytych** zbiór *A* ma tę cechę, że elementy mogą do niego należeć tylko częściowo (w pewnym stopniu).

Funkcja przynależności do zbioru rozmytego

Odpowiednikiem funkcji charakterystycznej χ_A w teorii zbiorów rozmytych jest **funkcja przynależności**

$$\mu_A : 0 \leq \mu_A(x) \leq 1.$$

Element *x* może być elementem zbioru rozmytego *A* w większym lub mniejszym stopniu – od zerowej do całkowitej przynależności. Wartość funkcji przynależności większa od zera oznacza przynależność *x* do *A* w określonym stopniu

Funkcja przynależności zależy od subiektywnego odczucia osoby definiującej zbiór rozmyty.

Wiedzę dziedzinową zapisuje się w logice rozmytej za pomocą zmiennych lingwistycznych i reguł rozmytych.

Podstawowe **własności** funkcji przynależności do zbioru:

$$\overline{A} = D - A$$
$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$$
$$\mu_{\overline{A}}(x) = 1 - \mu_A(x)$$
$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$$

Różnicę pomiędzy logiką rozmytą a prawdopodobieństwem wyjaśnimy na przykładzie. Dane jest zdarzenie = „*Jestem średniego wzrostu*".

- Wyrażenie probabilistyczne, *P*(*zdarzenie*) = *a*, oznacza, że:
  - nie mamy pełnej wiedzy o precyzyjnie określonym zdarzeniu,
  - wartość *a* może ulec zmianie po uzyskaniu dodatkowych danych.
- W logice rozmytej określany jest stopień przynależności elementu „*mój wzrost*" do zbioru „*osób średniego wzrostu*". $\mu_{\text{„średni\_wzrost"}}$ (*mój_wzrost*) = *b*.
  - Zdarzenie nie jest precyzyjnie określone
  - Funkcja przynależności nie zależy od dodatkowych danych i nie ulegnie zmianie w miarę zdobywania dodatkowych danych.

Semantyka logiki rozmytej

Niech D jest zbiorem wartości zdania (literału) x, a $f(x)$ jest funkcją wartościowania zdania. Możemy zdefiniować podzbiór A elementów x, dla których $f(x)$ przyjmuje wartość „prawda”. Wtedy funkcja charakterystyczna $\chi_A(x)$ jednoznacznie wyznacza funkcję wartościowania zdania: $f(x)$ = True $\Leftrightarrow \chi_A(x) = 1$.

Tak więc pojęcia logiczne prawdy i fałszu mogą zostać zdefiniowane za pomocą funkcji charakterystycznej zbioru.

Dla zbiorów rozmytych, uogólnienie funkcji charakterystycznej do funkcji przynależności pociąga za sobą uogólnienie pojęcia prawdy: literały mogą przyjmować wartości z zakresu [0,1].

Semantyka podstawowych operacji logicznych w logice rozmytej:

$$\neg f(x) = 1 - f(x)$$
$$f_1(x) \vee f_2(x) = \max\{f_1(x), f_2(x)\}$$
$$f_1(x) \wedge f_2(x) = \min\{f_1(x), f_2(x)\}$$

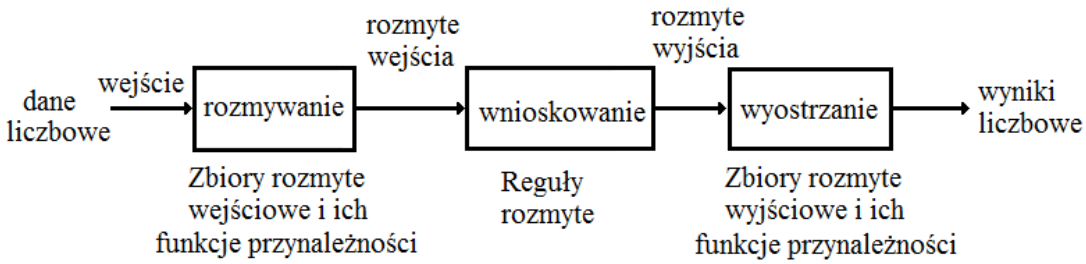
Semantyka implikacji rozmytej:

$$f_1(x) \Rightarrow f_2(y) \quad \equiv \quad \neg(f_1(x) \vee f_2(y)) \vee f_2(y)$$
$$\equiv \max\{1 - \max[f_1(x), f_2(y)], f_2(y)\}$$

Uwaga: podczas wnioskowania rozmytego $f_2(y)$ nie jest znane - przyjmuje się wtedy, że stopnie prawdziwości przesłanki i konkluzji są równe, tzn. $f_1(x) = f_2(y)$.

Przykład wnioskowania rozmytego

Strukturę typowego systemu sterowania będącego zastosowaniem logiki rozmytej przedstawiono na rys. 10.7.



Rys. 10.7 Zastosowanie logiki rozmytej w sterowaniu procesem (obiektem).

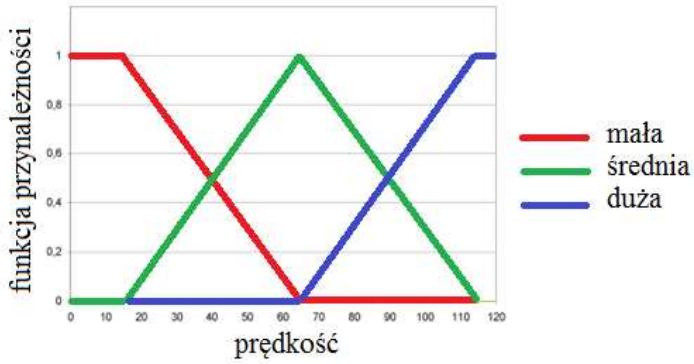
Poszczególne kroki algorytmu omawiane są poniżej.

1) Rozmywanie”

Wielkości liczbowe (np. pomiar prędkości, $x \in D$) są zamienianie na wielkości „jakościowe” – lingwistyczne (np. A_1 = „mała”, A_2 = „średnia”, A_3 = „duża”) i określane są ich stopnie przynależności (np. $f_{A1}(x)$, $f_{A2}(x)$, $f_{A3}(x)$) . Tzn. dla każdej wielkości wejściowej i każdej wartości lingwistycznej tworzymy **zdanie** i określamy ich stopnie przynależności kierując się zadaną wartością zmiennej (np. oznaczmy to jako $f_{A1}(x)$, $f_{A2}(x)$, $f_{A3}(x)$).

Przykład.

Dane są trzy funkcje przynależności zmiennej x („prędkość pojazdu”) do zmiennych lingwistycznych „mała”, „średnia”, „duża” (rys. 10.8). Załóżmy, że zmienna wejściowa przyjmuje wartość 40 ($x = 40$). Daje to następujące wartości przynależności zmiennej do zbiorów „zmiennych lingwistycznych”: $f_{A1}(40) = 0.5, f_{A2}(40) = 0.5, f_{A3}(40) = 0.$



Rys. 10.8 Przykład trzech funkcji przynależności dla zmiennej “prędkość”.

2) Wnioskowanie

Dany jest zestaw wyrażeń logicznych (tzw. „*reguł*”), których przesłanki są koniunkcjami stwierdzeń dla zmiennych wejściowych, zaś konkluzjami są stwierdzenia (rozmyte) o wielkości wyjściowej.

Przykład.

Definiujemy dwie wielkości wejściowe x_1, x_2 (prędkość i odległość od przeszkody) i jedną wielkość wyjściową y (wymagane hamowanie). Aksjomat (własność) dziedziny (tzw. „*reguła*”) może być postaci:

jeśli „*prędkość jest mała* (A_1)” i „*odległość jest duża* (B_3)” to wystarczy „*słabe hamowanie* (Y_2)”. Zapisując tę własność w logice rozmytej otrzymamy wyrażenie:

$$f_{A1}(x_1) \wedge f_{B3}(x_2) \Rightarrow f_{Y2}(y)$$

Jednocześnie określa się stopień prawdziwości przesłanki (na podstawie stwierdzonego w fazie rozmywania stopnia prawdziwości każdego ze zdań).

Np. $\min(0.5, 0.3) = 0.3$; dla iloczynu logicznego

Przyjmuje się, że stopień prawdziwości konkluzji i przesłanki są sobie równe dla każdej reguły.

Np. $0.3 \Rightarrow 0.3$

3) Wyostrzanie: agregacja i wartość średnia

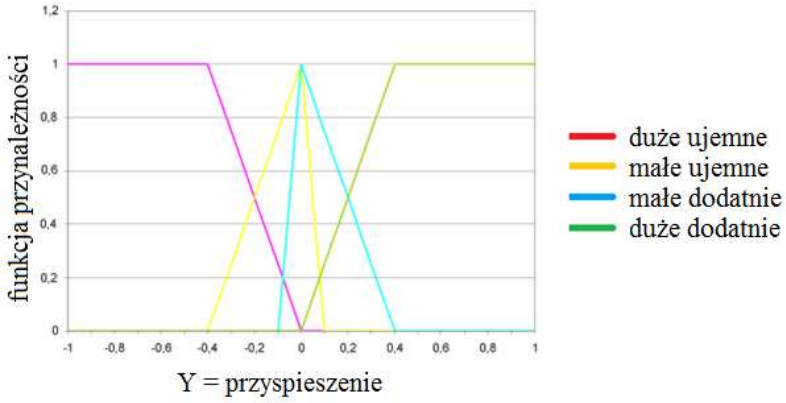
Krok agregacji to łączenie poszczególnych wyników kroków wnioskowania. Niech $f_{Y1}(y), f_{Y2}(y), \ldots, f_{Y4}(y)$, są wartościami funkcji przynależności do zbioru dla zmiennej wynikowej określonymi podczas etapu wnioskowania. Wartości te są łączone ze sobą w celu znalezienia najlepszej wartości dla zmiennej „y”, uwzględniając przebieg funkcji przynależności tej zmiennej wyjściowej.

Przykład

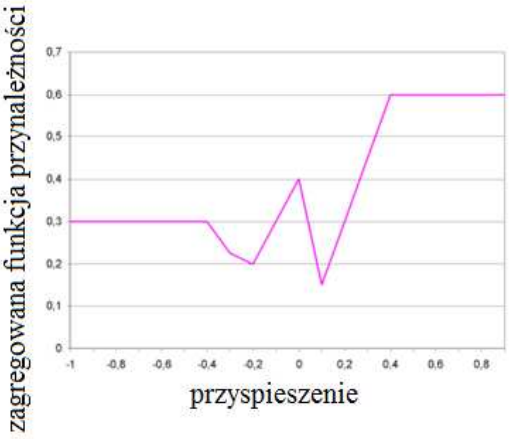
Niech y oznacza „przyspieszenie” a jej wartości lingwistyczne to: Y_1=„duże ujemne” (hamowanie), Y_2=„małe ujemne”, Y_3=„małe dodatnie”, Y_4=„duże dodatnie”. Przebieg tych funkcji ilustruje rys. 10.9. Połączona funkcja przynależności $\mu_y(y|\boldsymbol{x})$ jest ważoną sumą 4 funkcji przynależności:

$$\mu_y(y \mid \boldsymbol{x}) = \sum_{i=1}^{4} f_{Yi}(y \mid \boldsymbol{x}) \cdot \mu_{Yi}(y)$$

Np. dla zbioru wartości $f_{Yi}(\mathbf{x})$= {0.3, 0.3, 0.0, 0.6} i funkcji przynależności z rys. 10.9 powstanie ważona funkcja przynależności o postaci podanej na rys. 10.10.



Rys. 10.9 Przykładowe funkcje przynależności dla zmiennej wynikowej “przyspieszenie”



Rys. 10.10 Przykład ważonej funkcji przynależności dla zmiennej wynikowej “przyspieszenie”

Na koniec określana jest **wartość średnia** ważonej funkcji przynależności zmiennej Y, czyli konkretna liczba reprezentująca w systemie sterowania wymagana do zadania wartość (np. przyspieszenia lub hamowania).

Wartość y_{CoA} (ang. ***CoA*** - „center of area”) odpowiada „wartości oczekiwanej” zmiennej y wtedy, gdy znormalizowana funkcja przynależności pełni rolę „gęstości prawdopodobieństwa” rozkładu zmiennej y:

$$y_{CoA}(\boldsymbol{x}) = \frac{\int_{y \in D} y \cdot \mu_y(y \mid \boldsymbol{x})\, dy}{\int_{y \in D} \mu_y(y \mid \boldsymbol{x})\, dy}$$

10.7. Pytania

1. Na czym polega reprezentacja niepewności w modelu probabilistycznym – zdarzenia atomowe, odpowiedzi na zapytania?
2. Omówić pojęcie niezależności i warunkowej niezależności. Dlaczego je stosujemy?
3. Co to jest sieć Bayesa i do czego służy?
4. Przedstawić schemat wnioskowania w logice rozmytej stosowany w systemach sterowania.

10.8. Zadania

Zad. 10.1

Inwestor giełdowy rozważa dwie hipotezy: Z (spółka giełdowa A przyniesie zysk w tym roku), ⌐Z (spółka A poniesie stratę w tym roku); starając się łączyć z nimi obserwacje (fakty):

Y1 : indeks giełdowy spółki A wzrósł od początku roku więcej niż średni wzrost dla tego typu spółek,

Y2: spółka ogłosiła na początku roku prognozę zysku,

Y3: istnieje korzystny kurs walutowy dla działalności spółki A,

Y4: spółka prowadzi rozmowy o przejęciu innej firmy,

Y5: ogólne perspektywy wzrostu gospodarczego w kraju i na głównych rynkach spółki nie są korzystne.

Zależności prawdopodobieństwa pomiędzy takimi faktami a hipotezami inwestor szacuje jako:

$P(Y1|Z) = 0.7$; $P(Y1|\neg Z) = 0.3$; $P(Y2|Z) = 0.8$; $P(Y2|\neg Z) = 0.4$;

$P(Y3|Z) = 0.4$; $P(Y3|\neg Z) = 0.2$; $P(Y4|Z) = 0.9$; $P(Y4|\neg Z) = 0.5$;

$P(Y5|Z) = 0.2$; $P(Y5|\neg Z) = 0.4$;

Wyznaczyć prawdopodobieństwa hipotezy Z warunkowane różnymi obserwacjami:

(1) Y1 = true ,

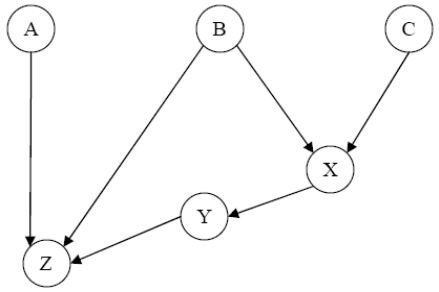
(2) (Y2 = true) i (Y3 = true),

(3) (Y1 = true) i (Y4 = true) i (Y5 =true).

Zad. 10.2

Zmienne losowe będące węzłami sieci Bayesa na poniższym rysunku 10.11 mogą przyjmować wartości 0 lub 1. Dla tej sieci należy:

a) zaproponować tabele rozkładów zmiennych losowych wystarczających do określenia dowolnego prawdopodobieństwa,
b) dla zaproponowanych rozkładów obliczyć P(X=1| B=1),
a) zbadać czy zachodzi warunkowa niezależność zmiennych A i C pod warunkiem (*dla znanych*) (X, Z),
a) określić koc Markowa dla zmiennej Y.



Rys. 10.11

Zad. 10.3

System wnioskowania rozmytego ma wspomagać decyzję o udzieleniu kredytu w oparciu o spełnianie dwóch zmiennych:

- h1 = „przyznać kredyt o limicie 100.000 zł (lub 200.000 zł, lub 400.000 zł)”,
- h2 = „zakładając czas kredytowania 30 miesięcy, stopę procentową 5%, przyznać kredyt wymagający miesięcznych rat 2.000 zł (lub 4.000 zł, lub 6.000 zł)”.
- Zmienne wejściowe (kryteria):

- A) Średni przychód w ostatnich 6 miesiącach: < 2.000 zł; 10.000 zł>;
- B) Łączna kwota inwestycji: < 100.000 zł; 1.000.000 zł >
- C) Współczynnik kapitału własnego dla inwestycji: < 0.0, 0.5> ;
- D) Liczba członków rodziny: < 1, 5>.

• Zdefiniować realistyczne funkcje przynależności i reguły wnioskowania rozmytego.

Pokazać przebieg i wyniki wnioskowania dla następujących danych:

1. A = 4.000, B = 100.000 zł, C = 0.5, D = 2, lub
2. A = 6.000, B = 200.000 zł, C = 0.4, D = 3.

11. Wnioskowanie w sieci Bayesa

- Dokładne wnioskowanie „przez przeliczanie”
- Dokładne wnioskowanie „z eliminacją zmiennych”
- Przybliżone wnioskowanie metodami „symulacji stochastycznej”

11.1. Dokładne wnioskowanie „przez przeliczanie”

W procesie wnioskowania dla zadanego zapytania należy obliczyć prawdopodobieństwo a posteriori zdarzenia $P(X_{pyt} \mid E_{obs}=e)$, dla zbioru zmiennych pytania, $X_{pyt} = \{X_1, ..., X_n\}$, przy zadanym obserwowanym zdarzeniu e, tzn. dysponując wartościami dla zmiennych opisujących zdarzenie, $E_{obs} = \{E_1, ..., E_m\}$. Poza tym istnieją zmienne ukryte $Y_{ukr} = \{Y_1, ..., Y_l\}$.

Zbiór zmiennych dla problemu: $X = X_{pyt} \cup E_{obs} \cup Y_{ukr}$.

Zapytanie może mieć prosty charakter i dotyczyć tylko jednej zmiennej, np.: $P(X_i \mid E=e)$. Zapytanie może dotyczyć rozkładu łącznego wielu zmiennych, np. $P(X_i, X_j \mid E=e) = P(X_i \mid E=e)\, P(X_j \mid X_i, E=e)$.

11.1.1. Dokładne wnioskowanie probabilistyczne

Rozważmy pytanie o jednej zmiennej. Zakładamy, że zmienna w pytaniu nie należy do zbioru obserwowanych zmiennych (w przeciwnym razie jej wartościowanie byłoby natychmiast znane). Wiemy, że każdy rozkład warunkowy dla zmiennych może być obliczony w wyniku sumowania elementów rozkładu łącznego zmiennych:

$$P(X \mid e) = \alpha\, P(X, e) = \alpha\, \Sigma_Y P(X, e, y).$$

Wiemy też, że elementy rozkładu łącznego obliczymy na podstawie iloczynu rozkładów warunkowych.

Przykład. Pytanie kierowane jest do znanej nam z rozdz. 10 sieci Bayesa opisującej „włamanie czy trzęsienie”: Niech konkretne pytanie brzmi: „jakie jest prawdopodobieństwo włamania, gdy jednocześnie dzwonią Jola i Marek”. Ukrytymi zmiennymi są tu „Wstrząs” i „Alarm”.

$$P(Z \mid j, m) = P(Z, j, m) / P(j, m) = \alpha\, P(Z, j, m) = \alpha\, \Sigma_w \Sigma_a P(Z, w, a, j, m)$$

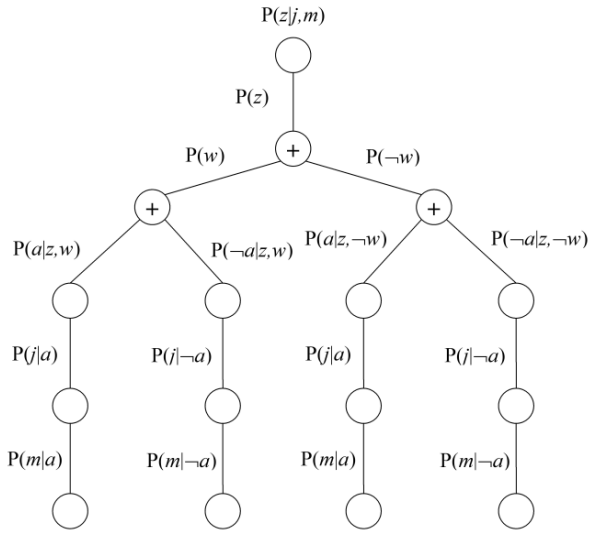
Semantyka sieci Bayesa pozwala nam wyznaczyć elementy rozkładu łącznego jako iloczyny wartości zdarzeń podanych w tablicach prawdopodobieństw warunkowych.

$$P(Z \mid j, m) = \alpha\, \Sigma_w \Sigma_a P(Z)\, P(w)\, P(a \mid Z, w)\, P(j \mid a)\, P(m \mid a) = \alpha\, P(Z) \Sigma_w P(w) \Sigma_a P(a \mid Z, w)\, P(j|a)\, P(m|a)$$

Procedura obliczania powyższego prawdopodobieństwa dla zmiennej zapytania może być wyrażona w postaci przeszukiwania drzewa wartościowań zmiennych (rys. 11.1) metodą *„przeszukiwania w głąb*”.

11.1.2. Algorytm dokładnego wnioskowania „przez przeliczanie” zmiennych

Sposób obliczeń zilustrowany w powyższym przykładzie może zostać w ogólności przedstawiony w postaci algorytmu wnioskowania „przez przeliczanie” zmiennych (tab. 11.1).



Rys. 11.1: Drzewo obliczeń dla znalezienia wartości $P(z|j, m)$.

Tabl. 11-1. Algorytm wnioskowania „przez przeliczanie" zmiennych.

```
function WNIOSKUJPRZEZPRZELICZANIE(X, e, Sb) returns rozkład zmiennej X
Parametry:   X, // zmienna w pytaniu
             e, // obserwacje – wartości zmiennych E_obs
             Sb; // sieć Bayesa o zmiennych {X}∪ E_obs ∪ Y_ukr
{
    Q(X) ← rozkład zmiennej X, początkowo pusty
    foreach wartość x_i zmiennej X do
    {
        rozszerz e o wartość x_i dla X;
        Q(x_i) ← PRZELICZWSZYSTKO(ZMIENNE(Sb), e);
    }
    return NORMALIZUJ(Q(X));
}
```

```
function PRZELICZWSZYSTKO(zmienne, e) returns wartość rzeczywista
{
    if (zmienne == ∅) then return 1.0;
    Y ← PIERWSZA(zmienne);
    if  wartość y zmiennej Y występuje w e
    then return P(y | rodzice(Y)) · PRZELICZWSZYSTKO(RESZTA(zmienne), e);
    else return Σ_y P(y | rodzice(Y)) · PRZELICZWSZYSTKO(RESZTA(zmienne), e_y);
        // gdzie e_y to e rozszerzone o Y= y
}
```

Złożoność obliczeniowa powyższego algorytmu probabilistycznego wnioskowania wynosi $O(2^n)$, dla n zmiennych typu logicznego (*Boolean*), czyli jest to wysoka złożoność. Taka metoda wnioskowania może być realizowana w praktyce tylko dla niezbyt dużej liczby zmiennych.

Na rysunku 11.1 zauważymy, że niektóre obliczenia są wykonywane wielokrotnie. Np. iloczyny: $P(j|a)\ P(m|a)$ i $P(j|\neg a)\ P(m|\neg a)$ są wyliczane dwukrotnie. Dla poprawy efektywności wnioskowania

powinniśmy móc uniknąć takich nadmiarowych obliczeń. Jest to celem kolejnego sposobu dokładnego wnioskowania zwanego wnioskowaniem „z eliminacją zmiennych”.

11.2. Dokładne wnioskowanie „z eliminacją zmiennych”

Modyfikacja poprzedniego algorytmu wnioskowania przez przeliczanie polegać tu na wyeliminowaniu zbędnych i powtarzanych obliczeń. Idea postępowania jest prosta: wykonać obliczenia (tylko dla niezbędnych zmiennych) i zapamiętać wynik takich obliczeń na potrzeby dalszych obliczeń. Najprostsza postać tak zmodyfikowanego dokładnego wnioskowania probabilistycznego nosi nazwę wnioskowania „z eliminacją zmiennych”.

11.2.1. Idea algorytmu

Algorytm wnioskowania „z eliminacją zmiennych” oblicza wyrażenie, będące iloczynem wyrażeń dla zmiennych, sposobem od *od-prawej-do-lewej* strony (co odpowiada na rys. 11.1 kierunkowi *od dołu-na górę*). W trakcie obliczeń zapamiętywane są wyniki pośrednie dla zmiennych (tzw. czynniki) obliczane na podstawie podwyrażeń od których dana zmienna zależy.

Przykład. Dla sieci Bayesa opisującej przyczyny i skutki alarmu oraz zapytania rozpatrywanego w poprzednim przykładzie należy obliczyć:

$$\mathrm{P}(Z \mid j, m) = \alpha\, \mathrm{P}(Z) \sum_w \mathrm{P}(w) \sum_a \mathrm{P}(a \mid Z; w)\, \mathrm{P}(j|a)\, \mathrm{P}(m|a)$$

| Z | | W | | A | | J | | M |

Poszczególne wyrażenia częściowe oznaczyliśmy symbolem odpowiedniej zmiennej. Należy przechowywać wyniki obliczania tych wyrażeń w postaci tzw. czynników.

Czynnik M, to po prostu $\mathrm{P}(m|a)$, co nie wymaga sumowania po M Ta wartość ustalona jest jako:

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix}.$$

Po wyliczeniu pierwszego czynnika z prawej strony mamy:

$$\mathrm{P}(Z \mid j, m) = \alpha\, \mathrm{P}(Z) \sum_w \mathrm{P}(w) \sum_a \mathrm{P}(a \mid Z; w)\, \mathrm{P}(j|a)\, \mathrm{f_M}(\mathrm{A})$$

W podobny sposób zapisujemy czynnik dla J, $\mathrm{P}(j|a)$, jako dwu-elementowy wektor $\mathrm{f_J}(\mathrm{A})$.

$$\mathrm{P}(Z \mid j, m) = \alpha\, \mathrm{P}(Z) \sum_w \mathrm{P}(w) \sum_a \mathrm{P}(a \mid Z; w)\, \mathrm{f_J}(\mathrm{A}) \times \mathrm{f_M}(\mathrm{A})$$

Symbol „×” oznacza *produkt punktowy* dwóch czynników i stosowany jest dla odróżnienia, że nie chodzi tu to iloczyn dwóch wektorów ani też nawet mnożenie *punkt-po-punkcie* dwóch wektorów. Wynikiem tej operacji jest czynnik o rozszerzonym wymiarze, uwzględniający oba czynniki składowe (dokładne wyjaśnienie podamy wkrótce).

Czynnik dla A, $\mathrm{P}(a|\mathrm{Z}, w)$, to macierz o rozmiarze $2\times2\times2$, oznaczona jako $\mathrm{f_A}(\mathrm{A}, \mathrm{Z}, \mathrm{W})$.

$$\mathrm{P}(Z \mid j, m) = \alpha\, \mathrm{P}(Z) \sum_w \mathrm{P}(w) \sum_a \mathrm{f_A}(\mathrm{A}, \mathrm{Z}, \mathrm{W}) \times \mathrm{f_J}(\mathrm{A}) \times \mathrm{f_M}(\mathrm{A})$$

Teraz obliczamy sumę dla A jako iloczyn powyższych trzech czynników. Wynikiem będzie macierz 2×2, o indeksach z dziedziny Z i W. Oznaczmy:

$$\mathrm{f_{\bar{A}JM}}(\mathrm{Z}, \mathrm{W}) = \sum_a \mathrm{f_A}(\mathrm{A}, \mathrm{Z}, \mathrm{W}) \times \mathrm{f_J}(\mathrm{A}) \times \mathrm{f_M}(\mathrm{A}) = \mathrm{f_A}(\mathrm{a}, \mathrm{Z}, \mathrm{W}) \times \mathrm{f_J}(\mathrm{a}) \times \mathrm{f_M}(\mathrm{a}) + \mathrm{f_A}(\neg\mathrm{a}, \mathrm{Z}, \mathrm{W}) \times \mathrm{f_J}(\neg\mathrm{a}) \times \mathrm{f_M}(\neg\mathrm{a}).$$

Podobnie obliczamy sumę dla W jako iloczyn czynników $\mathrm{f_W}(\mathrm{W})$ i $\mathrm{f_{\bar{A}JM}}(\mathrm{Z}, \mathrm{W})$:

$$\mathrm{f_{\hat{W}\bar{A}JM}}(\mathrm{Z}) = \mathrm{f_W}(w) \times \mathrm{f_{\bar{A}JM}}(\mathrm{Z}, w) + \mathrm{f_W}(\neg w) \times \mathrm{f_{\bar{A}JM}}(\mathrm{Z}, \neg w)$$

Teraz możemy już obliczyć odpowiedź procedury wnioskowania na zadane pytanie jako iloczyn czynnika dla Z (tzn. $\mathrm{f_Z}(\mathrm{Z}) = \mathrm{P}(\mathrm{Z})$) i macierzy $\mathrm{f_{\hat{W}\bar{A}JM}}(\mathrm{Z})$:

$$\mathrm{P}(Z \mid j, m) = \alpha\, \mathrm{f_Z}(\mathrm{Z}) \times \mathrm{f_{\hat{W}\bar{A}JM}}(\mathrm{Z})$$

Produkt dwóch czynników

Podsumowując powyższy przykład zauważmy, że występują w nim obliczenia dwóch rodzajów: *produkt punktowy* dwóch czynników (produkt czynników) i *sumowanie* produktu czynników po wartościach zadanej zmiennej.

Produkt dwóch czynników f_1 i f_2 daje w wyniku nowy czynnik, którego zmienne stanowią sumę zbiorów zmiennych obu czynników składowych. Np.

$$f(X_1, ...,X_n, Y_1, ...,Y_k, Z_1, ...,Z_l) = f_1(X_1, ...,X_n, Y_1, ...,Y_k) \times f_2(Y_1, ...,Y_k, Z_1, ...,Z_l)$$

Liczba elementów w macierzy dla czynnika f jest funkcją liczby zmiennych. Jeśli wszystkie zmienne są binarne to liczba elementów czynnika f_1 wynosi 2^{n+k}, liczba elementów czynnika f_2 wynosi 2^{k+l}, a liczba elementów ich produktu f wynosi 2^{n+k+l}.

Wartości jakie przyjmują elementy produktu f są wynikiem mnożenia dwóch odpowiadających sobie liczb – elementów czynników f_1 i f_2 :

$$f(x_1, ...,x_n, y_1, ...,y_k, z_1, ...,z_l) = f_1(x_1, ...,x_n, y_1, ...,y_k) \cdot f_2(y_1, ...,y_k, z_1, ...,z_l)$$

11.2.2. Algorytm dokładnego wnioskowania „z eliminacją zmiennych”

Algorytm wnioskowania przedstawiono w tab. 11.2. Wyjaśnienia wymagają ponadto zasady postępowania z czynnikami i zmiennymi w celu dalszego upraszczania obliczeń.

Tabl. 11-2. Algorytm wnioskowania „przez przeliczanie” zmiennych.

```
function EliminacjaZmiennych(X, e, Sb) returns rozkład dla X
Parametry:  X, // zmienna w zapytaniu
            e, // obserwacja, wyrażona jako zdarzenie losowe dla zmiennych obserwowanych
            Sb; // sieć Bayesa wyrażająca rozkład łączny zmiennych losowych
{
    czynniki ← [] ;
    zmienne ← OdwróćKolejność(Zmienne(Sb));
    foreach zm in zmienne do
    {
        czynniki ← [UtwórzCzynnik(zm,e) ∪ czynniki];
        if zm jest zmienną ukrytą then czynniki ← Suma(zm, czynniki);
    }
    return Normalizuj(ProduktPunktowy(czynniki));
}
```

Zasady postępowania z czynnikami

Jeśli dany czynnik nie zależy od zmiennej, po której odbywa się sumowanie to należy go przesunąć przed znak sumy. Produkty czynników obliczane są dopiero wtedy, gdy musimy wykonać ich sumowanie po elementach zadanej zmiennej. Sprowadza się to do wymnożenia tych macierzy, które zawierają zmienną dla sumowania.

Eliminacja zmiennych

Podczas wnioskowania można wyeliminować obliczanie czynników dla:

1. zmiennych reprezentowanych przez takie węzły sieci Bayesa, które nie są rodzicami zmiennych zapytania lub obserwacji. Sumowanie po takich zmiennych daje zawsze 1.
2. takich rodziców, którzy są w **m-separacji** ze zmienną zapytania.

Pierwsze kryterium zilustrujemy na poniższym przykładzie.

Przykład. Dla naszej przykładowej sieci Bayesa (przyczyny i skutki alarmu) dla zapytania, $\mathrm{P}(j \mid z)$, wykonamy obliczenia:

$$\mathrm{P}(\mathrm{J} \mid z) = \alpha\, \mathrm{P}(z)\, \Sigma_w\, \mathrm{P}(w)\, \Sigma_a\, \mathrm{P}(a \mid z, w)\, \mathrm{P}(\mathrm{J} \mid a)\, \Sigma_m\, \mathrm{P}(m \mid a)$$

Ale suma po m, $\Sigma_m\, \mathrm{P}(m \mid a)$, wynosi z definicji 1. Zmienna m nie ma żadnego znaczenia dla naszego pytania. W tym przykładzie:

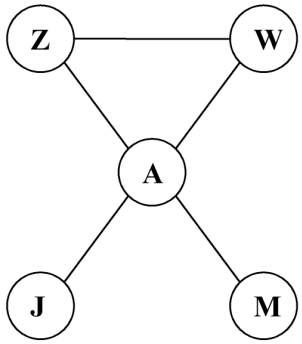
$$X_{\mathrm{pyt}} = \{Jola\ dzwoni\},\ E_{\mathrm{obs}} = \{Złodziej\}.$$
$$\mathrm{Rodzice}(X_{\mathrm{pyt}} \cup E_{\mathrm{obs}}) = \{Alarm,\ Wstrząs\}.$$

Czyli zmienna *Marek dzwoni* nie ma tu znaczenia.

Wyjaśnimy teraz pojęcie m-separacji.

Definicja. **Moralny graf** dla sieci Bayesa powstaje po połączeniu ze sobą wszystkich rodziców i usunięciu kierunku w łukach połączeń.

Np. Moralny graf dla naszej przykładowej sieci Bayesa podano na rys. 11.2:



Rys. 11.2: Moralny graf dla sieci Bayesa „przyczyny i skutki alarmu”.

Definicja. Zmienna *Y* jest w ***m*-separacji** z *X* poprzez *E* wtw. gdy *E* oddziela zmienne *Y* i *X* w moralnym grafie.

Można pokazać, że zmienna *Y* jest zbędna dla procesu wnioskowania o pytaniu *X* i obserwacji *E* gdy jest ona w *m*-separacji z *X* poprzez *E*.

Przykład. W naszej przykładowej sieci dla pytania, $\mathrm{P}(JolaDzwoni{=}true \mid Alarm{=}\mathrm{true})$, zarówno $\mathrm{Z}(Złodziej)$ jak i $\mathrm{W}(Wstrząs)$ nie grają żadnej roli. Są one w *m*-separacji ze zmienną pytania, $\mathrm{J}(JolaDzwoni)$, poprzez zmienną zdarzenia (obserwacji) $\mathrm{A}(Alarm)$.

11.3. Przybliżone wnioskowanie metodami „symulacji stochastycznej”

Ta grupa podejść zamiast wyliczać złożone funkcje prawdopodobieństwa a posteriori stosuje symulację procesu stochastycznego, generując próbki zdarzeń (wartości dla wszystkich zmiennych w sieci Bayesa) i aproksymując określone prawdopodobieństwa warunkowe na podstawie częstości wystąpień określonych wartości w generowanym zbiorze próbek.

Idea symulacji stochastycznej:

1) Generuj N próbek dla zadanego rozkładu prawdopodobieństwa

2) Oblicz na tej podstawie przybliżenie szukanego prawdopodobieństwa a posteriori P_{est}

3) Pokaż, że przybliżenie zbiega do rzeczywistego prawdopodobieństwa P.

Zasadnicze metody przybliżonego wnioskowania realizujące tę ideę:

- Próbkowanie z odrzucaniem (ang. *rejection sampling*): próbkujemy zdarzenia dla sieci bez obserwacji i odrzucamy próbki niezgodne z aktualną obserwacją.
- Wagi wiarygodności (ang. *likelihood weighting*): próbkujemy tylko zmienne aktualnie nie-obserwowane, gdyż zmienne obserwowane są już ustalone; próbka uzyskuje wagę wynikającą z prawdopodobieństwa obserwacji warunkowanych wartościami zmiennych w tej próbce.
- Łańcuch Markowa Monte Carlo (MCMC): próbka generowana jest w dynamicznym procesie losowej zmiany poprzedniej próbki w czasie, gdzie rozkład stacjonarny tego procesu przejść jest zgodny z rzeczywistym rozkładem a posteriori.

11.4. Pytania

Wyjaśnić metodę wnioskowania w sieci Bayesa “przez wyliczanie” zmiennych.

Wyjaśnić metodę wnioskowania w sieci Bayesa przez “eliminację zmiennych”.

Podać główne podejścia do wnioskowania zaliczane do “symulacji stochastycznej”.

11.5. Zadania

Zad. 8.1

Dla sieci Bayesa reprezentującej problem „przyczyny i skutki alarmu” (rys. 11.2) należy przeprowadzić dokładne wnioskowanie “przez przeliczanie” dla zapytania - „jakie jest prawdopodobieństwo zdarzenia „złodziej”, gdy obserwujemy fakty: „¬JolaDzwoni” i „MarekDzwoni”. Podać końcowy wynik.

Zad. 8.2

Dla sieci Bayesa reprezentującej problem „przyczyny i skutki alarmu” (rys. 11.2) należy wykonać dokładne wnioskowanie “z eliminacją zmiennych” dla pytania: „jakie jest prawdopodobieństwo zdarzenia „złodziej”, gdy obserwujemy fakty „JolaDzwoni” i „¬MarekDzwoni”. W szczególności:

(A) Sformułować początkowe równanie dla problemu; (B) Wyznaczyć i wyliczyć “czynniki” dla zmiennych; (C) Podać końcowy wynik.

12. „Dynamiczna Sieć Bayesa”

- Czas i niepewność. Modele Markowa. DBN.
- Wnioskowanie bayesowskie dla systemów dynamicznych
- Szczególne postacie DBN
- Zadania

12.1. Czas i niepewność

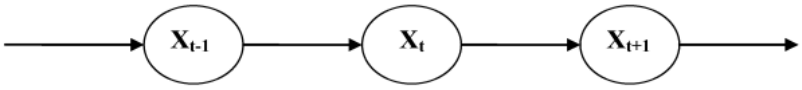
Jeśli środowisko agenta zmienia się w czasie to agent powinien śledzić i przewidywać zmiany. Zasada postępowania w modelu probabilistycznym: obliczanie rozkładów zmiennych (niewidocznego) stanu i (widocznej) obserwacji dla każdej sytuacji (chwili czasu). Niech: X_t = zbiór (niewidocznych) zmiennych stanu w chwili t , E_t = zbiór obserwowanych zmiennych w chwili t. Dalej zakładamy tu czas dyskretny; rozmiary odstępów pomiędzy kolejnymi chwilami czasu zależą od problemu. Wprowadźmy też notację sekwencji stanów w czasie: $X_{a:b} = X_a, X_{a+1}, \ldots, X_{b-1}, X_b$

12.1.1. Modele Markowa

Model Markowa wyznaczają odpowiedzi na pytania: „jak skonstruować sieć Bayesa dla zmiennych stanu i obserwacji”, oraz „jacy są rodzice zmiennych obserwacji”?

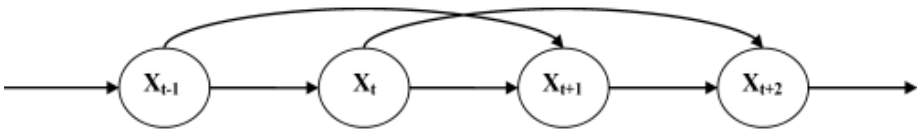
Fundamentalne znaczenie w tym modelu ma założenie Markowa: X_t zależy od ograniczonego podzbioru zmiennych „wcześniejszych” $X_{0:t-1}$.

Proces Markowa pierwszego rzędu: $P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$ (rys. 12.1).



Rys. 12.1: Ilustracja procesu Markowa pierwszego rzędu.

Proces Markowa drugiego rzędu: $P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-2}, X_{t-1})$ (rys. 12.2).



Rys. 12.2: Ilustracja procesu Markowa drugiego rzędu.

Istotne znaczenie ma także założenie Markowa odnośnie obserwacji: $P(E_t \mid X_{0:t}; E_{0:t-1}) = P(E_t \mid X_t)$. Czyli obserwacje w zadanej chwili czasu zależą jedynie od zmiennych stanu w tej samej chwili czasu. W ten sposób ustaliliśmy, że stany są rodzicami obserwacji.

12.1.2. Dynamiczna sieć Bayesa

Połączenie dwóch procesów Markowa pierwszego rzędu: jednego dla (ukrytych) zmiennych stanu a drugiego dla zmiennych obserwacji nazwiemy **dynamiczną siecią Bayesa** (ang. *Dynamic Baye-*

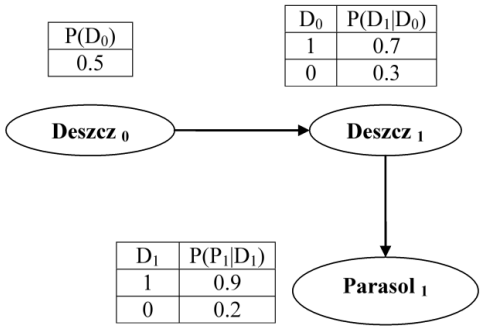
sian Network, DBN). Jest to wielokrotne powielenie pewnej sieci Bayesa, reprezentującej rozkłady obserwacji i stanu w kolejnych chwilach czasu, w sytuacji, gdy stan X_t i obserwacje E_t mogą zawierać jednocześnie wiele zmiennych.

Mówimy, że model DBN opisuje stacjonarny system, gdy:

1. model stanu (przejść), $P(X_t \mid X_{t-1})$, i
2. model obserwacji (wyjść), $P(E_t \mid X_t)$,

są ustalone (czyli niezmienne) dla wszystkich chwil t .

Przykład. Sieć Bayesa jako podstawa dynamicznej sieci Bayesa opisującej "świat parasola" (rys. 12.3). Model przejść: $P(Deszcz_t|Deszcz_{t-1})$. Model obserwacji: $P(Parasol_t \mid Deszcz_t)$.



Rys. 12.3: Sieć Bayesa opisująca „świat parasola".

Założenie Markowa pierwszego rzędu często nie jest spełnione w realnym świecie. Zauważmy, że w powyższym modelu prawdopodobieństwo deszczu zależy jedynie od tego czy padał on poprzedniego dnia czy nie. Jest to oczywiście mało realistyczne założenie. Możliwa poprawa modelu Markowa pierwszego rzędu tak, aby lepiej modelował on rzeczywisty świat może polegać na:

1. zwiększeniu rzędu procesu Markowa,

2. niekoniecznie zwiększanie rzędu modelu ale rozszerzenie stanu.

Np. poprzez dodanie takich zmiennych, jak Temperatura_t i Ciśnienie_t , możliwe jest lepsze (dokładniejsze) uwarunkowanie deszczu niż jedynie za pomocą zmiennej stanu Deszcz. Dlatego też w dalszej części rozdziału ograniczymy się do modelu Markowa pierwszego rzędu.

Pozostaniemy przy drugim sposobie poprawiania modelu Markowa i zajmiemy się modelami o postaci dynamicznej sieci Bayesa, w której możliwe jest wprowadzenie dowolnej liczby zmiennych stanu.

12.2. Wnioskowanie dla systemów dynamicznych

Załóżmy, że udało nam się już zdefiniować nasz świat w postaci dynamicznego modelu Bayesa. Powstaje pytanie do czego może on nam być pomocny? Oczywiście - do udzielania odpowiedzi na zadawane pytania. Ale nie tylko. Przedstawimy teraz cztery typowe zadania wnioskowania w bayesowskim modelu dynamicznym.

1. Filtracja: $P(X_t \mid e_{1:t})$

Filtracja to wyznaczanie wiarygodności aktualnego stanu ze względu na przeszłe obserwacje. Wyniki filtracji stanowią dane wejściowe dla procesu decyzyjnego racjonalnego agenta.

2. Predykcja: $P(X_{t+k} \mid e_{1:t})$, dla $k > 0$

Predykacja to proces przewidywania prawdopodobieństwa przyszłych stanów, jako efektów możliwych do wykonania sekwencji akcji. Jest to tak, jakby filtracja była pozbawiona kolejnych obserwacji.

3. Wygładzanie: $P(X_k \mid e_{1:t})$, dla $0 \le k < t$

Wygładzanie pozwala na lepsze oszacowanie (bliższe rzeczywistości) minionych rozkładów stanu, tzn. dla przeszłych chwil. Uzyskujemy lepsze oszacowanie niż wyniki podawane „na bieżąco" podczas filtracji. Jest to istotne zadanie realizowane w trakcie procesu uczenia.

4. Estymacja trajektorii – poszukiwanie najbardziej prawdopodobnej sekwencji stanów (wyjaśnianie obserwacji): $\arg\max_{X_{1:t}} P(x_{1:t} \mid e_{1:t})$.

Poszukiwanie najlepszej sekwencji prawdopodobieństw stanów jest istotnym zadaniem realizowanym na potrzeby procesu rozpoznawania środowiska i wyjaśniania obserwacji.

12.2.1. Filtracja

Przyjmuje ona postać rekursywnego algorytmu estymacji stanu:

$$P(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, P(\mathbf{X}_t \mid \mathbf{e}_{1:t}))$$

Aby wyznaczyć f w terminach modeli obserwacji i przejścia zauważamy, że:

$$\begin{aligned} P(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) &= P(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) = \alpha P(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \\ &= \alpha' P(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t}) \end{aligned}$$

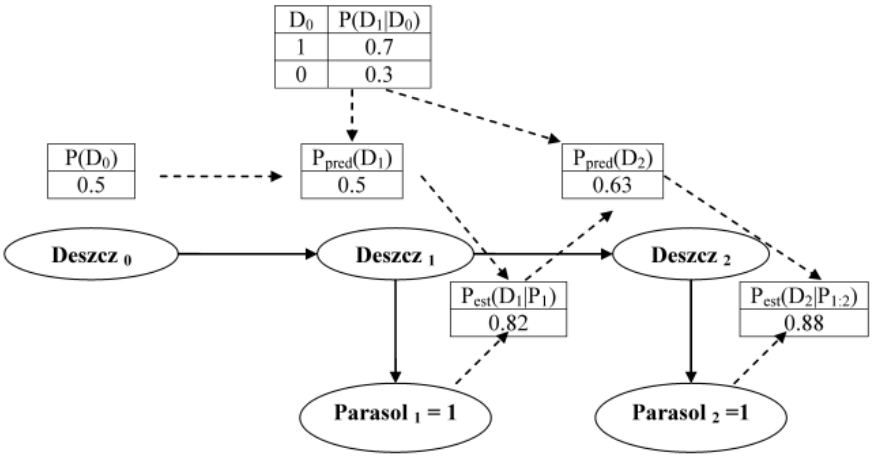
Powyższe równanie zawiera model obserwacji i predykcję o pojedynczym kroku. Ta predykcja może zostać obliczona poprzez sumowanie po wartościach X_t, tzn.

$$\begin{aligned} P(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) &= \alpha P(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t) P(x_t \mid \mathbf{e}_{1:t}) \\ &= \alpha' P(e_{t+1} \mid X_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t) P(x_t \mid e_{1:t}) \end{aligned}$$

Uzyskaliśmy szukaną postać rekursywną, gdyż ostatni człon jest sumowany po wszystkich wartościach zmiennej X_t, a to oznacza, że w chwili $t+1$ korzystamy z pełnego rozkładu $P(X_t \mid e_{1:t})$. Możemy potraktować wynik filtracji w chwili t (oznaczony jako $f_{1:t} = P(X_t \mid e_{1:t})$) jako „**komunikat**" przesyłany **w przód** wzdłuż sekwencji, modyfikowany każdorazowo przez funkcję przejścia i nową obserwację. W tych warunkach rekursywną estymację stanu zapiszemy jako:

$$f_{1:t+1} = \text{WPRZÓD}(f_{1:t}; e_{t+1}) .$$

Przykład. Dla sieci Bayesa opisującej deszcz (rys. 12.4) pokażemy proces filtracji stanu.



Rys. 12.4: Ilustracja filtracji stanu dla sieci opisującej „świat parasola".

Załóżmy, że nasz początkowy stopień przekonania o tym, czy padało w dniu 0 wynosi 0.5. Czyli zachodzi rozkład: $P(D_0)=[0.5 , 0.5]$. Pierwszego dnia pojawia się obserwacja „parasol", czyli $P_1 = 1$. Predykcja z chwili $t=0$ do $t=1$ dla zmiennej stanu wynosi:

$$P(D_1)= \Sigma_{d0} (D_1 | d_0) P(d_0) = [0.7 , 0.3] \text{ x } 0.5 + [0.3, 0.7] \text{ x } 0.5 = [0.5, 0.5]$$

Modyfikacja zmiennej stanu w oparciu o obserwację w chwili $t=1$ wynosi:

$$P(D_1| P_1=1)= \alpha P(P_1=1 | D_1) P(D_1) = \alpha [0.9, 0.2] [0.5, 0.5] = \alpha [0.45, 0.1] \cong [0.82, 0.18].$$

Drugiego dnia również pojawia się obserwacja „parasol", czyli $P_2 = 1$. Predykcja z $t=1$ do $t=2$ zmiennej stanu wynosi:

$$P(D_2 | P_1)= \Sigma_{d1} (D_2 | d_1) P(d_1 | P_1) = [0.7 , 0.3] \text{ x } 0.82 + [0.3, 0.7] \text{ x } 0.18 = [0.63, 0.37]$$

Modyfikacja zmiennej stanu w oparciu o obserwację w chwili $t=2$ wynosi:

$$P(D_2|P_1=1, P_2=1)= \alpha P(P_2=1 | D_2) P(D_2| P_1=1) = \alpha [0.9, 0.2] [0.63, 0.37] = \alpha [0.567, 0.074] \cong [0.88, 0.12].$$

Wynik jest zgodny z intuicją. Oczekiwanie deszczu wzrasta z dnia 1 na 2, gdy przez dwa dni obserwujemy parasole.

12.2.2. Predykcja

Zadanie predykcji może być postrzegane jako filtracja bez obserwacji. Możemy to wyrazić w postaci:

$$P(X_{t+k} | e_{1:t}) =\Sigma_{xt+k} P(X_{t+k+1} | x_{t+k}) P(x_{t+k} | e_{1:t}) \text{ , dla } k > 0$$

Predykcja korzysta jedynie z modelu przejścia a nie korzysta z modelu obserwacji.

Ciekawym pytaniem jest to co stanie się, gdy będziemy powtarzać predykcję coraz dłużej w czasie. Przewidywany rozkład będzie zdążał do stacjonarnego rozkładu dla procesu Markowa wyznaczonego przez model przejścia. Dla przykładu „deszczu i parasola" będzie nim rozkład jednorodny [0.5, 0.5].

12.2.3. Wygładzanie

Jest to proces obliczania rozkładu zmiennych stanu w poprzednich chwilach czasu, w całym okresie od pierwszej obserwacji do obserwacji aktualnej, z uwzględnieniem wszystkich obserwacji dostępnych w całym okresie:

$$P(X_k | e_{1:t}), \text{ dla } 0 \le k < t .$$

Podzielmy sekwencję obserwacji $e_{1:t}$ na („dotychczasową" i „przyszłą"): $e_{1:k}$ i $e_{k+1:t}$. Wtedy:

$$P(X_k | e_{1:t}) = P(X_k | e_{1:k;} e_{k+1:t}) = \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_{k;} e_{1:k})$$

$$= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k) = \alpha f_{1:k} b_{k+1:t}$$

Uwzględniliśmy tu tzw. „komunikat zwrotny", $b_{k+1:t}$, w analogii do wcześniej omawianego komunikatu „wprzód" $f_{1:k}$. Okazuje się, że tak jak komunikat wprzód może być obliczany rekursywnie poprzez filtrację, tak komunikat zwrotny może być obliczony poprzez rekursywny proces, który przebiega wstecz począwszy od t:

$$b_{k+1:t} = P(e_{k+1:t} | X_k) = \sum_{xk+1} P(e_{k+1:t} | X_k; x_{k+1}) P(x_{k+1} | X_k)$$

$$= \sum_{xk+1} P(e_{k+1:t} | x_{k+1}) P(x_{k+1} | X_k)$$

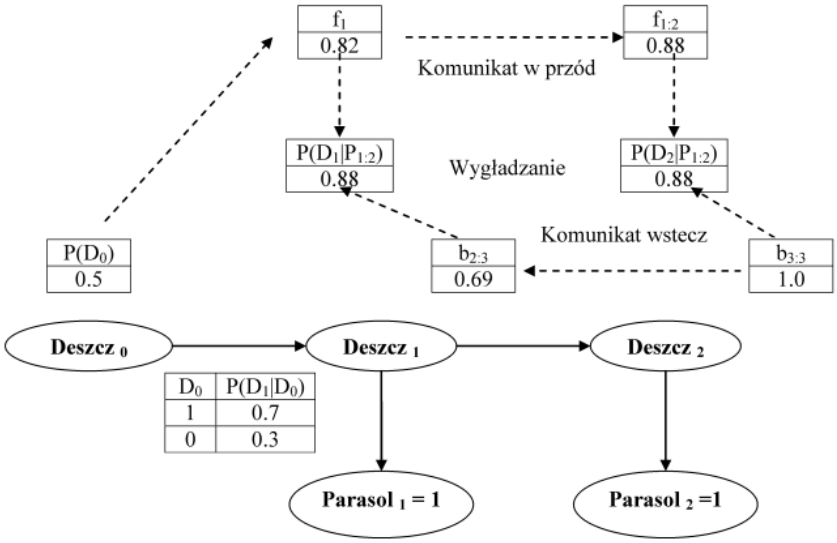
$$= \sum_{xk+1} P(e_{k+1} | x_{k+1}) P(e_{k+2:t} | x_{k+1}) P(x_{k+1} | X_k)$$

Zapiszemy to w skrócie jako: $b_{k+1:t} = \text{WSTECZ}(b_{k+2:t}; e_{k+1:t})$.

Podsumowując, proces wygładzania może być zrealizowany przez dwie rekursje: jedną wprzód a drugą wstecz. Proces rekursji wstecz inicjalizowany jest wartościami 1, tzn.:

$$b_{t+1:t} = P(e_{t+1:t} \mid X_t) = 1.$$

Przykład. Zastosujemy wygładzanie dla obliczenia rozkładu stanu w dniu t=1 na podstawie obserwacji w dniach t=1 i t=2.



Rys. 9.5: Ilustracja wygładzania rozkładu stanu dla sieci opisującej deszcz.

Zobaczmy, że prawdopodobieństwo deszczu dla wygładzonego rozkładu w chwili t=1 jest wyższe niż było uzyskane uprzednio w wyniku filtracji. Jest to wpływ obserwacji „parasol" w chwil t=2.

Algorytm „w przód-wstecz"

Złożoność obliczeniowa algorytmu wygładzania stanu dla jednej konkretnej chwili czasu k, uwzględniającego obserwacje $e_{1:t}$, jest liniowa względem czasu obserwacji: $O(t)$. Wynika to z faktu stałej liczby operacji wykonywanych w jednym kroku rekursji przez obie części obliczeń: „w przód" i „wstecz"

Jeśli interesują nas wygładzanie stanu dla wszystkich chwil obserwacji, to złożoność będzie kwadratowa względem czasu obserwacji: $O(t^2)$. Lepszym rozwiązanie, o liniowej złożoności obliczeniowej, stanowi algorytm „w przód-wstecz", w którym najpierw wykonuje się filtrację (komunikaty wprzód) i zapamiętuje estymowane rozkłady stanu w każdej chwili czasu, a następnie wykonuje rekursję wstecz korzystając z wcześniej obliczonych komunikatów wprzód. Algorytm podany został w tab. 12-1.

Tabl. 12-1. Algorytm „w przód-wstecz".

```
function WPRZÓDWSTECZ(ev, prior) returns wektor rozkładów prawdopodobieństw
PARAMETRY:   ev, // wektor obserwacji w krokach k= 1, ... ,t
             prior; // rozkład a priori początkowego stanu,
LOKALNE ZMIENNE: fv, // wektor komunikatów wprzód dla k=0,...,t
             b, // reprezentacja komunikatów wstecz , początkowo = 1
             sv; // wektor wygładzonych wartości stanu dla k =0, ..., t
{
     fv[0] ← prior;
     for i = 1 to t do
     fv[i] ← WPRZÓD(fv[i-1], ev[i]);
     for j= t step -1 to 1 do
```

```
        {
                sv[i] ← NORMALIZUJ(fv[i] × b);
                b[i] ← WSTECZ(b, ev[i]);
        }
        return sv;
}
```

12.2.4. Najbardziej prawdopodobna sekwencja stanów

Najbardziej prawdopodobna sekwencja wartości stanów wcale nie musi składać się wyłącznie z sekwencji najbardziej prawdopodobnych stanów, ustalanych w każdej chwili czasowej. Poszukiwanie takiej sekwencji możemy przedstawić jako problem poszukiwania „najlepszej" ścieżki w grafie, którego węzły odpowiadają możliwym wartościom wektora stanu w kolejnych chwilach czasu. Z uwagi na własność procesu Markowa pierwszego rzędu zachodzi rekursywnie zależność: najbardziej prawdopodobna („najlepsza") ścieżka prowadząca do każdego stanu x_{t+1} to najlepsze połączenie najbardziej prawdopodobnej ścieżki do pewnego stanu x_t z prawdopodobieństwem przejścia pomiędzy tymi dwoma stanami. Zapiszemy ten fakt jako:

$$\max_{x1\ldots xt} P(x_1, \ldots, x_t; X_{t+1} \mid e_{1:t+1}) = P(e_{t+1} \mid X_{t+1}) \cdot \max_{xt} [\, P(X_{t+1} \mid x_t) \cdot \max_{x1\ldots xt-1} P(x_1, \ldots, x_{t-1}; x_t \mid e_{1:t}) \,]$$

Powyższe równanie jest zbliżone do równania filtracji stanu poza 2 nowymi elementami:

- zamiast komunikatu wprzód $f_{1:t}$ mamy komunikat dotyczący prawdopodobieństwa najbardziej prawdopodobnej ścieżki dojścia do zadanego rozkładu stanu x_t, zebrany w wektor po wszystkich rozkładach:

  $m_{1:t} = \max_{x1\ldots xt-1} P(x_1, \ldots, x_{t-1}; X_t \mid e_{1:t})$;

- sumowanie po wartościach x_t zastąpione jest operacją maksymalizacji po x_t

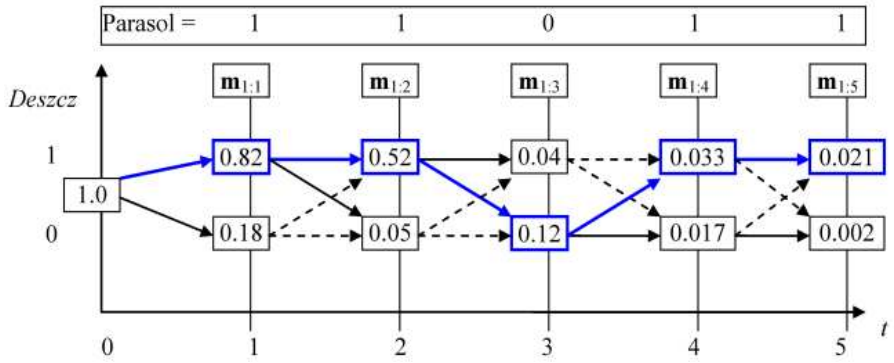
Podobnie jak proces filtracji, proces poszukiwania najbardziej prawdopodobnej sekwencji przebiega wprzód wzdłuż sekwencji obserwacji i oblicza komunikat m w każdej chwili czasu. Na koniec wektor m zawiera prawdopodobieństwa najbardziej prawdopodobnych ścieżek prowadzących do wszystkich możliwych wartości końcowego stanu.

Iteracyjna postać algorytmu poszukiwania najbardziej prawdopodobnej sekwencji stanów dla zadanej sekwencji obserwacji nosi nazwę algorytmu **przeszukiwania Viterbiego**:

$$\mathbf{m}_{1:t+1} = P(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \cdot \max_{xt} [\, P(\mathbf{X}_{t+1} \mid \mathbf{x}_t) \cdot \mathbf{m}_{1:t} \,]$$

Podczas przebiegu wprzód należy zapamiętać wszystkie wykonane przejścia pomiędzy wartościami stanu, tak, aby odtworzyć najlepszą ścieżkę, poruszając się od najlepszego końcowego stanu wstecz.

Przykład. Sekwencje rozkładów zmiennej stanu "deszcz" i obserwacji "parasol" reprezentujemy w postaci ścieżek w grafie decyzyjnym. Na rys. 12.6 pokazano graf możliwych sekwencji stanu i graf tworzony dla sekwencji obserwacji – zmiennej „Parasol" = $\{1, 1, 0, 1, 1\}$, podczas „przeszukiwania Viterbiego".



Rys. 12.6: Ilustracja „przeszukiwania Viterbiego” dla sieci „świat parasola”.

12.3. Szczególne postacie DBN

Dynamiczna sieć Bayesa (DBN) jest uogólnieniem m.in. dwóch często spotykanych modeli stochastycznych: Ukrytych Modeli Markowa (HMM) i Filtru Kalmana;

Ukryte Modele Markowa mają jedną dyskretną zmienną stanu, gdy tymczasem DBN może mieć wiele zmiennych stanu. Filtr Kalmana dopuszcza n zmiennych stanu, ale model przejść i wyjść musi być liniowy a rozkłady prawdopodobieństwa są rozkładami Gaussa.

Zaletą obu uproszczonych modeli jest istnienie efektywnych metod dokładnego wnioskowania (o analitycznej postaci). Tymczasem dla DBN w ogólności metody dokładnego wnioskowania mogą być bardzo złożone obliczeniowo. Dlatego dla nich najczęściej stosuje się przybliżone metody wnioskowania.

12.3.1. HMM (Ukryty Model Markowa)

W tym modelu istnieje pojedyncza zmienna stanu: X_t zmienna stanu w chwili t jest pojedynczą, dyskretną zmienną (zwykle zachodzi to też dla zmiennej obserwacji E_t).

Dziedziną X_t jest $\{ f_1, \ldots , S \}$ – skończony zbiór wartości dla stanu. Funkcja przejść jest liniowa i zadana jest w postaci macierzy: $T = \{ T_{ij} = P(X_{t=j} \mid X_{t-1=i}) \}$.

Również funkcja wyjść jest dana macierzą, np. O_t dla każdej chwili t, której elementy na przekątnej wynoszą $P(e_t \mid X_{t=i})$.

W takich warunkach komunikaty tworzone w funkcjach „w przód i wstecz” są wektorami :

$$f_{1:t+1} = \alpha \, O_{t+1}{}^{T} \, f_{1:t}$$

$$b_{k+1:t} = T \, O_{k+1} \, b_{k+2:t}$$

Zadania dokładnego wnioskowania w HMM są realizowane znacznie efektywniej niż w ogólnym przypadku. Np. algorytm „w przód – wstecz” posiada wtedy złożoność czasową $O(S^2 t)$ i pamięciową $O(S t)$.

12.3.2. Filtr Kalmana

Filtr Kalmana zostanie przedstawiony w następnym rozdziale, gdyż jego głównym przeznaczeniem jest efektywna filtracja w warunkach dobrze zdefiniowanych parametrycznych rozkładów prawdopodobieństwa.

Podsumowanie

Modele dynamiczne korzystają ze zmiennych stanu i obserwacji powtarzanych w czasie.

Przy spełnieniu założeń procesu Markowa i stacjonarności procesu, model dynamiczny składa się z: z modelu przejścia (stanu) $P(X_t | X_{t-1})$ i modelu obserwacji $P(E_t | X_t)$.

Specyficznym modelem dynamicznym są ukryte Modele Markowa (HMM) – posiadają one stany o pojedynczej zmiennej. Dynamiczne sieci Bayesa (DBN) stanowią uogólnienie zarówno modelu HMM jak i filtru Kalmana.

Typowe zadania wnioskowania w modelu dynamicznym to: filtracja, predykcja, wygładzanie, poszukiwanie najbardziej prawdopodobnej trajektorii (sekwencji). Wszystkie zadania mogą być wykonane rekursywnie ze stałym kosztem w pojedynczej iteracji.

12.4. Pytania

1. Wyjaśnić pojęcie modelu Markowa stosowanego dla opisu procesów stochastycznych.
2. Przedstawić podstawowe zadania dynamicznego wnioskowania stochastycznego w sieci DBN.
3. Omówić algorytm „w przód – wstecz”.
4. Omówić algorytm „przeszukiwanie Viterbiego”.

12.5. Zadania

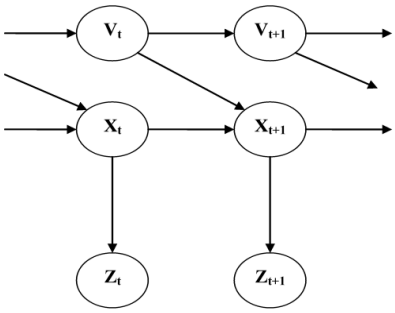
13. Wnioskowanie w DBN

- Filtr Kalmana
- Wnioskowanie dokładne w DBN
- Wnioskowanie przybliżone dla DBN
- Zadania

13.1. Filtr Kalmana

Jest to stochastyczny model systemu dynamicznego w postaci stanu obejmującego zmienne o ciągłych wartościach i także ciągłych wartościach obserwacji.

Np. model „*lotu pocisku"* zazwyczaj wymaga reprezentacji położenia i prędkości (rys. 13.1): $\mathbf{X}_t = [x, y, z], \quad V_t = [v_x, v_y, v_z]$.



Rys. 13.1: Ilustracja modelu przejść i wyjść – stan to pozycja X_t i prędkość V_t , a obserwacja – Z_t .

Założenia odnośnie modelu przyjęte w filtrze Kalmana:

1. początkowe rozkłady (*a priori*) są rozkładami Gaussa,
2. model przejść (stanu) ma postać liniową z zakłóceniami o charakterze szumu Gaussa,
3. model obserwacji (wyjść) także ma postać liniowej funkcji z zakłóceniami w postaci szumu Gaussa.

Filtracja dla sieci DBN o zmiennych losowych o rozkładach Gaussa dana jest równaniami filtru Kalmana. Występują w nim dwa zasadnicze kroki: krok predykcji i modyfikacji. Jeśli $\mathrm{P}(\mathrm{X}_t \mid \mathrm{e}_{1:t})$ ma postać rozkładu Gaussa to predykcja rozkładu stanu wynosi:

$$P(X_{t+1} \mid e_{1:t}) = \int_{x_t} P(X_{t+1} \mid x_t) \cdot P(x_t \mid e_{1:t})\, dx_t$$

i również ma rozkład Gaussa. Jeśli $\mathrm{P}(\mathrm{X}_{t+1}|\mathrm{e}_{1:t})$ ma postać rozkładu Gaussa to zmodyfikowany rozkład stanu wynosi:

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha \cdot P(e_{t+1} \mid X_{t+1}) \cdot P(X_{t+1} \mid e_{1:t})$$

i także ma rozkład Gaussa. Z tego wynika wniosek, że przy założeniach właściwych dla filtru Kalmana każdy rozkład stanu $\mathrm{P}(\mathrm{X}_t \mid \mathrm{e}_{1:t})$ (dla wszystkich t) jest wielowymiarowym rozkładem Gaussa. Oznaczmy ten rozkład jako $N(\mu_t , \Sigma_t)$, zależny od parametrów: wektora średnich i macierzy kowariancji.

13.1.1. Równania filtru Kalmana

Modele przejść i obserwacji:

$$P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) = N(\mathbf{F}\mathbf{x}_t, \Sigma_x)(\mathbf{x}_{t+1})$$
$$P(\mathbf{z}_t \mid \mathbf{x}_t) = N(\mathbf{H}\mathbf{x}_t, \Sigma_z)(\mathbf{z}_t)$$

gdzie $\mathbf{F}$ oznacza macierz funkcji przejść a $\mathbf{H}$ macierz funkcji wyjść, zaś $\mathbf{\Sigma}_x$ i $\mathbf{\Sigma}_z$ są macierzami kowariancji rozkładów Gaussa, odpowiednio zmiennej stanu $\mathbf{x}$ i obserwacji $\mathbf{z}$.

Równanie modyfikacji stanu (uwzględnia predykcję i obserwację):

$$\boldsymbol{\mu}_{t+1} = \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t)$$
$$\mathbf{\Sigma}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_x)$$

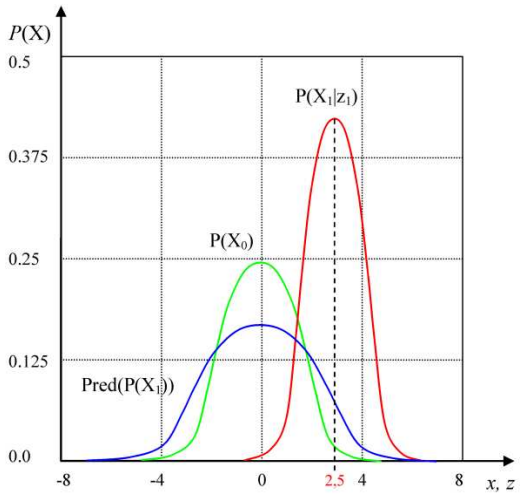
gdzie

$$\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_x)\mathbf{H}^T[\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_x)\mathbf{H}^T + \mathbf{\Sigma}_z]^{-1}$$

jest tzw. macierzą wzmocnienia filtru Kalmana. Zauważamy, że Σ_t i K_t nie zależą od wartości obserwacji (a jedynie od szumu obserwacji) i dlatego mogą być wyznaczone przed procesem filtracji.

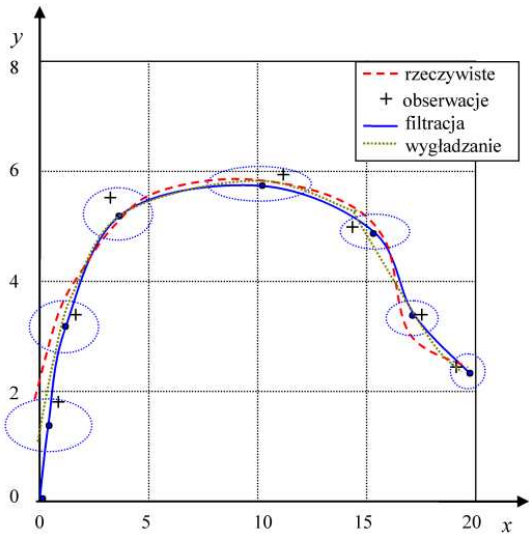
Przykład. Modelujemy przypadkowy spacer wzdłuż osi X jako system o jednej zmiennej stanu mającej rozkład Gaussa. Niech σ_t oznacza odchylenie standardowe (błąd) estymowanej zmiennej stanu w chwili *t*, σ_x – odchylenie standardowe (błąd) funkcji przejść, a σ_z – odchylenie standardowe (błąd) obserwacji. W tych warunkach parametry modyfikowanego rozkładu zmiennej stanu wyniosą (rys. 13.2):

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2)\, z_{t+1} + \sigma_z^2 \mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2} \quad ; \qquad \sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\, \sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$



Rys. 13.2: Rozkłady Gaussa: a priori ($P(x_0)$ o $\mu_0 = 0.0$ i $\sigma_0 = 1.0$) i po pierwszej obserwacji, $z_1 = 2.5$ ($P(x_1)$), przy parametrach błędu przejścia, $\sigma_x = 2$, i błędu obserwacji, $\sigma_x = 1$.

Przykład. Filtracja a wygładzanie. Ilustrację wyników śledzenia (wartości) zmiennej stanu filtrem Kalmana i wyników wygładzania pokazano na rys. 13.3.



Rys. 13.3: Ilustracja śledzenie położenia obiektu w płaszczyźnie X-Y – owale reprezentują odchylenia standardowe dla estymacji zmiennych stanu.

13.1.2. Odmiany filtru Kalmana

Rozszerzony filtr Kalmana („*extended KF*”)

W przypadku nieliniowego modelu przejść filtr Kalman nie może być stosowany. Istnieje natomiast tzw. rozszerzony filtr Kalmana, w którym aproksymuje się każdorazowo aktualne przejście funkcją liniową zadaną tzw. macierzą jakobianu.

Przełączany filtr Kalmana („*switched KF*”)

W wielu przypadkach obserwowane obiekty (lub systemy) nie zachowują się stacjonarnie. Np. jadący dotąd z ustaloną prędkością samochód nagle przyspiesza lub zmienia kierunek jazdy. Podobnie do przypadku zachowania sie pojazdu napotykającego przeszkodę, wszystkie te sytuacje nie da się modelować przy pomocy filtru Kalmana. Filtr może jedynie przewidzieć położenie odpowiadające środkowej wartości rozkładu a nie może reprezentować alternatyw. Dla opisu takich procesów na gruncie Filtru Kalman musimy zastosować wiele równolegle pracujących filtrów Kalmana, z których każdy „śledzi” odrębną hipotezę. Nazywamy to rozwiązanie przełączanym filtrem Kalmana.

13.2. Wnioskowanie dokładne dla DBN

Każdy Filtr Kalmana jest specyficzną dynamiczną siecią Bayesa wtedy, gdy zmienne posiadają rozkłady Gaussa. Oczywiście w rzeczywistym świecie nie zawsze niepewność może być modelowana rozkładem Gaussa. Np. w rzeczywistym świecie urządzenie pomiarowe może uleć zniszczeniu i jego błąd pomiarowy nie da się wtedy modelować rozkładem Gaussa. Potrzebne jest nam modelowanie błędów urządzenia o dowolnym rozkładzie prawdopodobieństwa. Za pomocą DBN możemy to osiągnąć, np. dodając zmienną stanu odpowiedzialną za zniszczenie urządzenia mającą odpowiedni rozkład prawdopodobieństwa. Dlatego też sieć DBN jest ogólniejsza od Filtru Kalmana.

13.2.1. „Naiwne" wnioskowanie dla DBN

Nasuwa się oczywisty sposób wnioskowania dla sieci DBN, który nazwiemy „naiwną" metodą „wnioskowania. Jego idea jest następująca:

1. „rozwiń" sieć Bayesa, czyli powielaj jej model przejść i wyjść tyle razy, ile potrzeba dla uwzględnienia zadanych dla niej obserwacji;
2. obliczaj dokładnie rozkłady zmiennych stanu dla kolejnych chwil obserwacji; tzn. w kolejnym kroku filtracji dla $t+1$: powiel sieć Bayesa dla $t+1$, „podsumuj" część sieci dla t stosując metodę wnioskowania z eliminacją zmiennych.

Wiemy, że metoda eliminacji zmiennych jako sposób wnioskowania w sieci Bayesa (dla jednej zmiennej stanu) ma stały wymagany czas obliczeń w każdym kroku iteracji. Niestety przy większej liczbie zmiennych stanu prowadzi to do złożoności obliczeniowej, która jest wykładniczą funkcją liczby zmiennych. Obliczane są bowiem czynniki o coraz większym rozmiarze, w których występują wszystkie zmienne stanu posiadające rodziców w poprzedniej chwili czasu. Tym samym wraz z długością sekwencji, najdłuższy czynnik dany jest zależnością $\mathrm{O}(d^{n+1})$ dla chwili t=d i przy n zmiennych stanu. Złożoność modyfikacji wynosi $\mathrm{O}(d^{n+2})$. Dlatego też metoda naiwna może co najwyżej znaleźć zastosowanie przy krótkich sekwencjach obserwacji i niezbyt dużej liczbie zmiennych.

13.3. Wnioskowanie przybliżone dla DBN

13.3.1. Metoda symulacji stochastycznej według „wag wiarygodności"

W tej metodzie aproksymujemy rzeczywisty rozkład zmiennych stanu za pomocą wag (wyrażających wiarygodność) dla N wybranych próbek wartości zmiennych stanu. Wiarygodności próbek wynikają z dotychczasowych obserwacji.

Zasadniczym nową ideą w porównaniu z dokładnym wnioskowaniem „naiwnym" jest to, że w kolejnej iteracji wcześniej obliczony rozkład wiarygodności próbek pełni rolę (przybliżonej) reprezentacji rozkładu prawdopodobieństwa stanu. To założenie pozwala uniknąć wstępnego „rozwijania" całej sieci DBN dla obserwacji i skupić sie jedynie na dwóch modelach, dla poprzedniej i aktualnej chwili obserwacji. Przypomina to komunikat wprzód stosowany podczas filtracji, ale tym razem przekazujący wiarygodności uzyskane dla N próbek wartości jednocześnie

Jednak obok poprawy efektywności obliczeń to rozwiązanie ma też zasadnicze wady:

1. obserwacje nie mają wpływu na wybór próbek i nie ma gwarancji, że będą one zgodne z obserwacjami; to oznacza, że w miarę wzrostu liczby obserwacji rozkład wiarygodności zbioru próbek będzie coraz bardziej niezgodny z rzeczywistym rozkładem, wynikającym z tych obserwacji;
2. dla uniknięcia problemu z pkt. 1 należałoby dla każdej obserwacji zwiększać liczbę próbek tak, aby zachować ich związek z kolejnymi obserwacjami; jednak wtedy liczba wymaganych próbek rośnie wykładniczo wraz z t.

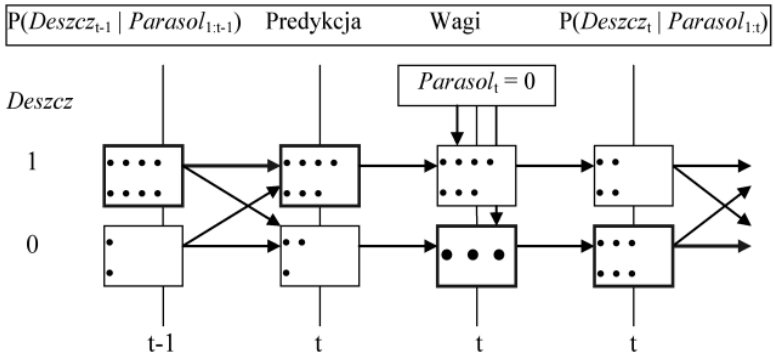
Efektywny sposób przybliżonego wnioskowania w DBN to tzw. **filtr cząstek,** omawiany w następnym punkcie.

13.3.2. Filtr cząstek

Ta metoda wnioskowania przybliżonego wyrasta z idei metody stochastycznej według „wag wiarygodności", ale poprawia jej wady dzięki pomysłowi każdorazowego przepróbkowywania rozkładu stanu dla zachowania jedynie wiarygodnych wartości wektora stanu. Zasadnicza idea **filtracji cząstek** (ang. *particle filter*) to zapewnienie, aby wszystkie „obszary" wartości zmiennych stanu o aktualnie znaczącej wiarygodności były efektywnie aproksymowane za pomocą skończonego zbioru (*populacji*) próbek wartości stanu, tzw. *cząstek* (ang. *particle*). Uzyskuje się to dzięki takie-

mu powielaniu cząstek, które jest proporcjonalnie do ich wiarygodności ze względu na aktualną obserwację.

Przykład. Ilustracja działania filtra cząstek na przykładzie sieci „świata parasola" (rys. 13.4). Najpierw w chwili t: 8 próbek wskazuje na *deszcz* a 2 na $\neg$*deszcz* . Każda próbka przekazywana jest dalej, do predykcji rozkładu w chwili t+1, uzyskanej dzięki zastosowaniu modelu przejścia - prowadzi to do nowej reprezentacji dziedziny zmiennej stanu za pomocą 10 próbek. Przewidywany rozkład w chwili t+1: 7 próbek ma reprezentować deszcz a 3 $\neg$*deszcz* . Kolejny krok to uwzględnienie efektu obserwacji w chwili t+1. Obserwacja ta wynosi $\neg$*parasol* . Prowadzi to do nadania istniejącym próbkom wag zgodnych z tą obserwacją – w efekcie duże wartości wag uzyskują 3 próbki reprezentujące $\neg$*deszcz* . Na koniec iteracji dla t+1 następuje modyfikacja rozkładu wiarygodności próbek, poprzez przepróbkowanie: wybierany jest nowy zbiór próbek reprezentujący nowy rozkład wiarygodności.



Rys. 13.4: Ilustracja filtra cząstek.

Metoda filtra cząstek znalazła szerokie zastosowanie, m.in. do śledzenia ruchu obiektów w obrazach, do automatycznej lokalizacji i tworzenia map w robotyce, itd.

Algorytm

Schemat algorytmu „filtra cząstek" podano w tab. 13-1.

Tabl. 13-1. Algorytm „filtr cząstek".

function FILTRCZĄSTEK(e, *N*, *dbn*) **returns** zbiór próbek wektora stanu
PARAMETRY: *e*, // nowa obserwacja
N, // wymagana liczba próbek
dbn; // dynamiczna sieć Bayesa (rozkład a priori $P(X_0)$),
// model przejść $P(X_1|X_0)$ i model obserwacji $P(E_1|X_1)$)
{
static: S, // pamięć próbek o rozmiarze *N*, zainicjowana zgodnie z $P(X_0)$
var: W; // wektor wag o rozmiarze *N*
for *i* = 1 to *N* do
{
S[*i*] ← próbka przewidywana w modelu $P(X_1 \mid X_0 = S[i])$;
W[*i*] ← $P(e \mid X_1 = S[i])$;
}
S←PRZEPRÓBKUJWEDŁUGWAG(*N*, S, W);
return S;
}

Filtr cząstek – równania

Pokażemy zgodność wiarygodności aktualnej populacji N próbek ze względu na obserwacje e (oznaczmy go przez $N(\mathrm{x}|\mathrm{e})/N$) z rozkładem prawdopodobieństwa stanu w DBN warunkowanego obserwacjami, dla pewnej chwili t. Przez **zgodność** rozumiemy tu, że wraz ze wzrostem liczby próbek N do nieskończoności obliczany w filtrze cząstek rozkład wiarygodności zdąża do rzeczywistego rozkładu prawdopodobieństwa dla sieci DBN.

Załóżmy, że ta zgodność ma miejsce w chwili t, tzn.: $\frac{N(x_t \mid e_{1:t})}{N} = P(x_t \mid e_{1:t})$

i prześledźmy kroki wykonywane w kolejnej iteracji.

1) Krok predykcji

Jest to predykcja rozkładu w następnej chwili czasu, aproksymowanego przez populację próbek

$$N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t})$$

2) Wyznaczenie nowych wag dla próbek

Obliczamy nowe wagi próbek wyrażające ich zgodność z nową obserwacją $\mathbf{e}_{t+1}$:

$$W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t})$$

3) Przepróbkowanie i modyfikacja rozkładu

Nowe próbkowanie zbioru według aktualnych wag: losowanych jest N nowych próbek ze zbioru istniejących próbek, w taki sposób że częstość wyboru odpowiada jej aktualnej wadze.

Modyfikacja wiarygodności nowo wybranych próbek:

$$\begin{aligned}\frac{N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1})}{N} &= \alpha \cdot W(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1}) = \alpha \cdot P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t}) = \\ &= \alpha \cdot P(\mathbf{e}_{t+1} \mid \mathbf{x}_{t+1}) \cdot \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) N(\mathbf{x}_t \mid \mathbf{e}_{1:t}) = \\ &= P(\mathbf{x}_{t+1} \mid \mathbf{e}_{1:t+1})\end{aligned}$$

Tym samym pokazaliśmy, że filtr cząstek poprawnie aproksymuje rzeczywisty rozkład stanu warunkowany obserwacjami.

Czy filtr cząstek jest efektywnym rozwiązaniem? Tak. W praktyce możemy zaobserwować, że nawet przy stosunkowo niedużej liczbie cząstek (co zapewnia efektywność obliczeniową) błąd aproksymacji rozkładu stanu uzyskiwany w metodzie „filtra cząstek" pozostaje zwykle ograniczony w czasie. Pokazują to wszelkie wyniki empiryczne, ale teoretycznego dowodu na to nie ma.

Podsumowanie

Filtr Kalmana modeluje filtrację dla stanu o n zmiennych, przy ograniczeniu modelu do postaci liniowej i rozkładów Gaussa.

Wnioskowanie dokładne w DBN nie jest w praktyce efektywne.

Filtr cząstek stanowi dobrą metodę przybliżonej filtracji w DBN.

13.4. Pytania

1. Omówić założenia i etapy obliczeń filtru Kalmana.
2. Przedstawić ideę dokładnego wnioskowania dla DBN.
3. Do czego służy i na czym polega filtr cząstek?

13.5. Zadania

Zad. 13.1

Modelujemy przypadkową jazdę pojazdu wzdłuż prostej poprzez aktualne odchylenie kątowe X_t, czyli jako system o jednej zmiennej stanu X mającej rozkład Gaussa. Obserwację reprezentuje kąt obrotu kierownicy od położenia „na wprost", w postaci zmiennej Z_t. Załóżmy, że dane są trzy kolejne pomiary (tab. 13-2).

Tab. 13-2

t :	1	2	3
Z_t :	-0.2	-0.1	0.1

Przeprowadzić symulację filtracji zmiennej X z użyciem filtru Kalmana przy następujących danych:

Prawdopodobieństwo a priori dla stanu $P(X_0)$: $N(\mu_0 = 0.0\ ,\ \sigma_0 = 1.0)$.

Odchylenia standardowe: $\sigma_x = 0.1$, $\sigma_z = 0.2$; gdzie σ_x – odchylenie standardowe (błąd) funkcji przejść, a σ_z – odchylenie standardowe (błąd) obserwacji.

Zad. 13.2

Zaproksymować rozkłady Gaussa opisujące model przejść i wyjść, podane w zadaniu 13.1, za pomocą rozkładów dyskretnych zmiennych, przyjmując po 5 wartości w przedziałach:

$$x = [-0.2, 0.2], \quad z = [-0.2, 0.2].$$

Dla takiego modelu i pomiarów podanych w warunkach zadania 13.1 rozwiązać zagadnienia filtracji i wygładzania prawdopodobieństwa a posteriori rozkładu stanu.

Zad. 13.3

Modelujemy dyskretną samo-lokalizację autonomicznego robota mobilnego w pewnym pomieszczeniu, w oparciu o pomiary otoczenia uzyskiwane przez podsystem analizy obrazu. Zakładamy istnienie dwóch stopni swobody pojazdu – ma on możliwość przesunięć wykonywanych wzdłuż osi X i Y. Pojedyncze akcje $\boldsymbol{a}_t$, które może wykonywać robot to*:*

$\boldsymbol{a}^1 = [-1, 0]$, $\boldsymbol{a}^2 = [1, 0]$, $\boldsymbol{a}^3 = [0, -1]$, $\boldsymbol{a}^4 = [0, 1]$.

Zakładamy, że zbiór możliwych położeń, rozpatrywanych podczas uczenia robota obejmuje 9 miejsc (tab. 13-3)

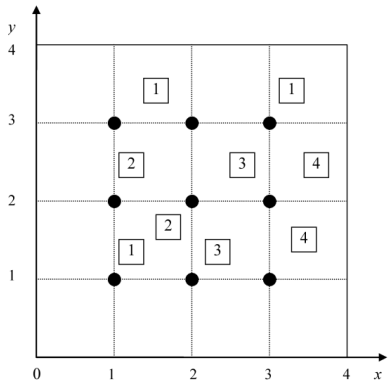
Tab. 13-3

$\mathbf{s}^1$	$\mathbf{s}^2$	$\mathbf{s}^3$	$\mathbf{s}^4$	$\mathbf{s}^5$	$\mathbf{s}^6$	$\mathbf{s}^7$	$\mathbf{s}^8$	$\mathbf{s}^9$
(1, 1)	(2, 1)	(3, 1)	(1, 2)	(2, 2)	(3, 2)	(1, 3)	(2, 3)	(3, 3)

Obraz z kamery (pomiar) obejmuje zawsze wycinek podłogi o rozmiarze 2x2 i o środku w aktualnym położeniu robota. Układ pomiarowy dostarcza dla każdej chwil *t* „zero-jedynkową" informację $\mathbf{z}_t$ o występowaniu w obrazie cyfr od 1 do 4 (informacja o wielokrotnym wystąpieniu danej cechy i jej położeniu nie jest dostępna):

$\mathbf{z}$ = (czy 4?, czy 3?. czy 2?, czy 1?).

Załóżmy, że w pomieszczeniu rozmieszczono poniższe cyfry (widok „z góry") (rys. 13.5).



Rys. 13.5:

Próba „wyjścia” przez robota poza ograniczenie skutkuje „odbiciem się” od ściany i powrotem w poprzednie miejsce.

Zdefiniować modele przejść i wyjść: $P(x_t \mid x_{t-1}, a_t), P(z_t \mid x_t).$

Przeprowadzić symulację wnioskowania metodą „filtra cząstek”, $P(x_t \mid z_t, a_t)$, gdy robot rozpoczyna w (nieznanym sobie) miejscu $\mathbf{s}^4$ a wybór jego akcji podyktowany jest potrzebą dojścia do celu, czyli miejsca $\mathbf{s}^9$.

Część IV: Uczenie

14. Uczenie się ze wzmacnianiem

Zadanie uczenia ze wzmacnianiem

Uczenie się strategii

Proces decyzyjny Markowa a funkcja użyteczności

Uczenie się funkcji użyteczności

Uczenie się strategii (Q-learning)

Pytania

Zadania

14.1. Zadanie uczenia ze wzmacnianiem

Uczenie się *ze wzmacnianiem* jest pewną odmianą uczenia z nadzorem, w którym jednak nie występują wzorcowe odpowiedzi (dostarczane przez nauczyciela) a jedynie występuje **krytyk**, który okazjonalnie nagradza lub karze decyzje podejmowane przez agenta.

Agent ma za zadanie **znaleźć strategię** wyboru akcji, która prowadzi do długoterminowej maksymalizacji nagród. Nagrody są też zwane wartościami wzmocnienia.

Nie ma tu jawnego podziału na fazę uczenia się i fazę stosowania zdobytej wiedzy. Np. każde dotarcie do celu podczas „normalnej" pracy agenta może posłużyć do modyfikacji aktualnej strategii działania agenta.

Scenariusz uczenia się ze wzmacnianiem

- Agent i środowisko wymieniają pomiędzy sobą informacje. Mechanizm tej wymiany jest określony przez scenariusz uczenia się ze wzmacnianiem.
- Wymiana informacji (interakcje) odbywa się w dyskretnych jednostkach czasu, zwanych krokami.
- Krok polega na obserwacji przez agenta stanów środowiska i wykonaniu akcji.
- Akcje te są oceniane i uczeń otrzymuje wartość wzmocnienia (nagrody).
- Akcje mogą również zmienić stan środowiska.

Ogólny schemat uczenia się ze wzmacnianiem zawiera tabela 14-1. Konkretne algorytmy uczenia się ze wzmacnianiem realizują podany scenariusz, a różnią się pomiędzy sobą konkretyzacją kroku 5 - uczenia się.

Tab. 14-1 Schemat uczenia się ze wzmacnianiem

FOR wszystkie kroki czasu t DO:
1. obserwuj aktualny stan s_t;
2. wybierz akcję a_t do wykonania w stanie s_t;
3. wykonaj akcję a_t;
4. obserwuj wzmocnienie r_t i następny stan s_{t+1};
5. ucz się na podstawie doświadczenia $\{s_t, a_t, r_t, s_{t+1}\}$

Agent uczący się ze wzmacnianiem

Wyróżnimy tu trzy zasadnicze formy uczenia ze wzmacnianiem wykorzystywane przez agentów uczących się:

1) Racjonalny agent uczy się funkcji użyteczności, U(*s*), określonej dla stanów *s* i korzysta z niej dla wyboru akcji maksymalizujących spodziewaną ocenę wyniku wykonania.

W takiej sytuacji mówimy o pasywnym uczeniu, gdyż strategia agenta - wybór akcji $\pi(s)$ w stanie *s* - jest ustalona z góry a zadaniem uczenia się jest jedynie określenie użyteczności tej strategii dla stanów (lub par: stan-akcja).

Agent racjonalny musi posiadać model środowiska T(*s*, *a*, *s*'), aby określać użyteczność stanu *s*, gdyż musi znać stany *s*' do których prowadzą go określone akcje. Zadanie uczenia może zostać rozszerzone na nauczenie się modelu środowiska.

2) Agent stosujący ***Q*-learning** uczy się funkcji stan-akcja-**wartość** (tzw. *Q*-funkcji), dającej spodziewaną wartość, *Q*(*a*, *s*), dla wyboru akcji *a* w danym stanie *s*. Jej związek z funkcją użyteczności to:

$$U(s) = \max_a Q(a,s)$$

Jednak zasadniczą różnicą jest to, że agent stosujący Q-learning **nie** musi posiadać **modelu** środowiska, gdyż nie analizuje on spodziewanych wyników dla określenia wartości akcji. Mówimy tu o **aktywnym uczeniu**, wymagającym eksploracji środowiska po to, aby nauczyć się **strategii decyzyjnej**, tzn. tego jaką akcję w danym stanie należy wybierać.

3) Agent **reaktywny** uczy się *polityki* (inaczej: **strategii decyzyjnej**), która odwzorowuje bezpośrednio obserwacje na akcje.

14.2. Uczenie się strategii

Kryterium maksymalizacji nagród

Na agenta wpływać można tylko za pomocą nagród (w nich jest zawarte zadanie stawiane przed agentem) i wymaga się od niego maksymalizacji nagród według pewnego kryterium.

Najlepsza strategia nie zawsze przynosi natychmiast wysokie nagrody, dużo skuteczniejsze są strategie przynoszące jak najwyższe skumulowane nagrody w długim okresie czasu. Strategie te muszą uwzględniać skutki wykonywanych akcji z pewnym opóźnieniem i są nazywane uczeniem się z opóźnionym wzmacnianiem. Dalej będziemy rozpatrywać wyłącznie ten typ uczenia się ze wzmacnianiem.

Skoro zależy nam na maksymalizacji nagród w długim okresie czasu, to kryterium może wyglądać następująco:

$$\max_r E\left(\sum_{t=0}^{\infty} r_t\right)$$

gdzie r_t to wartości nagród w kolejnych chwilach czasu. Podana zależność oznacza maksymalizację wartości oczekiwanej sumy nagród w przyszłości.

Według tego kryterium wszystkie nagrody są tak samo ważne. Warto jednak dobierać wagi zależnie od czasu, np. nagrody oddalone w czasie są mniej atrakcyjne od tych otrzymywanych natychmiast. Wprowadźmy **współczynnik dyskontowania**, $w \in [0,1]$, regulujący względną ważność nagród natychmiastowych i otrzymywanych z opóźnieniem. W ten sposób otrzymujemy kryterium z dyskontem:

$$\max_r E\left(\sum_{t=0}^{\infty} w^t \cdot r_t\right)$$

Przyjęcie $w = 0$, oznacza, że istotne dla nas stają się wyłącznie nagrody natychmiastowe, zaś dla $w_t = 1$ otrzymujemy wzór poprzedni, bez dyskonta. Przyjęcie pośredniej wartości z przedziału <0, 1> umożliwia regulowanie stopnia ważności nagród z różnymi czasami opóźnienia.

Przykład – giełda

Wyobraźmy sobie system będący graczem giełdowym, mający nauczyć się tak inwestować, aby zmaksymalizować przyszłe zyski. Nagrodą jest zysk/strata na danej inwestycji. Agent dokonuje transakcji, czyli wybiera akcję i obserwuje wzmocnienie, czyli zysk/stratę z inwestycji. Jego zadaniem jest maksymalizacja zdyskontowanego zysku z różnych inwestycji w określonym czasie.

14.3. Problem decyzyjny Markowa a funkcja użyteczności

Modelem matematycznym dla uczenia się ze wzmacnianiem jest **problem decyzyjny Markowa**, czyli problem znalezienia optymalnej strategii decyzyjnej dla środowiska, którego modelem jest proces decyzyjny Markowa.

Definicja. Proces decyzyjny Markowa jest definiowany jako czwórka {S, A, ρ, δ}, gdzie: S jest skończonym zbiorem stanów, A jest skończonym zbiorem akcji, ρ jest funkcją wzmocnienia, δ jest funkcją przejść stanów.

Definicja. W procesie decyzyjnym Markowa {S, A, ρ, δ} dla każdej pary {*s*, *a*} ∈ S × A, wartość funkcji wzmocnienia ρ(*s*, *a*) jest zmienną losową o wartościach rzeczywistych i oznacza nagrodę otrzymywaną po wykonaniu akcji *a* w stanie *s*.

Definicja. W procesie decyzyjnym Markowa {S, A, ρ, δ} dla każdej pary {*s*, *a*} ∈ S × A, wartość funkcji przejść δ(*s*, *a*) jest zmienną losową o wartościach ze zbioru S, oznaczającą następny stan po wykonaniu akcji *a* w stanie *s*.

Własność Markowa

Przyjrzyjmy się definicjom funkcji ρ i δ. Zauważmy, że ich jedynymi argumentami są *s*, aktualny stan, oraz *a* - aktualna akcja. Oznacza to, że w każdym kroku nagroda i następny stan zależą (probabilistycznie) tylko od **aktualnego** stanu i akcji, nie zależą w żaden sposób od stanów i akcji wcześniejszych.

Definicja strategii

Strategią dla procesu decyzyjnego Markowa {S, A, ρ, δ } jest dowolna funkcja π: S→ A.

Strategia oznacza więc sposób wyboru akcji w każdym stanie problemu opisywanego procesem decyzyjnym Markowa. Jakość strategii zależy oczywiście od tego, jakie nagrody przynosi posługiwanie się nią. Aby móc oceniać strategie (i następnie wybierać lepszą), należy przyporządkować poszczególnym strategiom i stanom oczekiwane wartości zdyskontowanych sum przyszłych nagród.

Prowadzi to do zdefiniowania **funkcji użyteczności** służącej do oceniania strategii.

Definicja funkcji użyteczności.

Dla procesu decyzyjnego Markowa {S, A, ρ, δ} **funkcja użyteczności** ze względu na strategię π jest określona dla każdego stanu *s* ∈ S następująco:

$$U^{\pi}(s) = E\left(\sum_{s=0}^{\infty} r_t \mid \pi, s_0 = s\right)$$

czyli jest to oczekiwana wartość sumy (dyskontowanych) nagród uzyskanych przy stosowaniu strategii π i dla sekwencji stanów rozpoczynanych od zadanego stanu.

Ustaliliśmy już oczywisty fakt, że lepsza jest ta strategia, która przyniesie wyższą wartość oczekiwaną nagród w przyszłości. Zatem wprost z definicji funkcji użyteczności wynika poniższa definicja.

Definicja

Strategia π^* jest **lepsza** od strategii π, jeśli $U^{\pi^*}(s) \geq U^{\pi}(s)$, dla każdego $s \in S$, i istnieje $s \in S$, dla którego $U^{\pi^*}(s) > U^{\pi}(s)$.

Definicja

Strategią optymalną jest każda strategia, dla której nie istnieje strategia od niej lepsza.

Przykład

Dany jest świat 3x4 na poniższym rysunku 14.1. Na rys. (a) za pomocą strzałek zaznaczono przyjętą strategię π w "świecie 4 x 3". Jest ona optymalna przy nagrodach, r(s)=-0.04, dla stanów niekońcowych, przy kryterium „bez dyskontowania". Na rys. (b) zaznaczono nauczone użyteczności stanów U^{π}(s) w "świecie 4x3" prowadzące do wyboru strategii π.

Agent wykonuje szereg prób w celu wyznaczenia optymalnej strategii π. Każdą próbę agent rozpoczyna od stanu (1,1) i zbiera doświadczenie na podstawie sekwencji przejść pomiędzy stanami do momentu, kiedy nie osiągnie jednego ze stanów końcowych (4,2) („sukces" +1) lub (4,3) („porażka" -1). Jego obserwacje polegają zarówno na rozpoznaniu stanu środowiska, jak i na odebraniu nagrody w danym stanie. Dla stanów niekońcowych przyjęto: r(s)=-0.04. Przykłady sekwencji próbnych i nagród uzyskiwanych w każdym stanie:

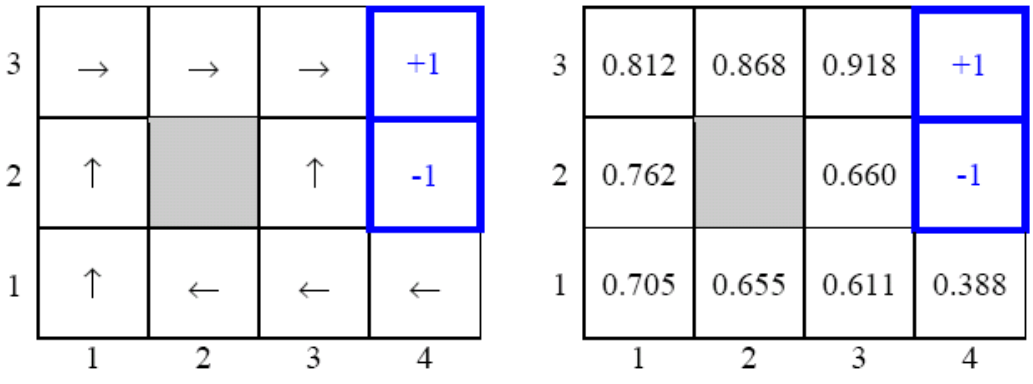
$$(1,1)/-0.04 \rightarrow (1,2)/-0.04 \rightarrow (1,3)/-0.04 \rightarrow (1,2)/-0.04 \rightarrow (1,3)/-0.04$$
$$(2,3)/-0.04 \rightarrow (3,3)/-0.04 \rightarrow (4,3)/+1$$

$$(1,1)/-0.04 \rightarrow (1,2)/-0.04 \rightarrow (1,3)/-0.04 \rightarrow (2,3)/-0.04 \rightarrow (3,3)/-0.04$$
$$(3,2)/-0.04 \rightarrow (3,3)/-0.04 \rightarrow (4,3)/+1$$

$$(1,1)/-0.04 \rightarrow (2,1)/-0.04 \rightarrow (3,1)/-0.04 \rightarrow (3,2)/-0.04 \rightarrow (4,2)/-1$$

Dla każdego stanu podczas uczenia przechowuje się średnią wartość funkcji użyteczności stanu, wynikającą z minionych prób. Każdorazowo na koniec próby, po osiągnięciu stanu końcowego, obliczane są zwrotnie aktualne wartości użyteczności stanów wizytowanych w danej próbie. Modyfikują one dotychczasowe średnie przechowywane dla każdego ze stanów.

Efektem uczenia są wartości użyteczności stanów i wynikająca z nich strategia wyboru stanów π.



Rys. 14.1 Przykład uczenia się funkcji użyteczności stanów

14.4. Uczenie się funkcji użyteczności

Definicja

Funkcja wartości akcji $Q^{\pi}(s,a,\pi(s))$ ze względu na strategię π przyporządkowuje każdej parze stan-akcja (*s, a*) oczekiwaną wartość zdyskontowanej sumy przyszłych nagród, jakie będą otrzymane przez system rozpoczynający działanie w stanie s i wykonujący akcję a zgodnie ze strategią π.

Jest to definicja analogiczna do definicji funkcji użyteczności, ale funkcja wartości akcji jest wygodniejsza ze względów obliczeniowych, gdyż jest reprezentacją jednocześnie funkcji użyteczności i strategii.

Zachodzi oczywista równość: $U^{\pi}(s) = Q^{\pi}(s,a,\pi(s))$

Algorytm TD (ang. *temporal difference*)

Większość algorytmów uczenia się ze wzmacnianiem opiera się na wartościowaniu strategii i wykorzystuje w tym celu metody różnic czasowych, zwane metodami **TD** (ang. *temporal difference*).

Metody **TD** polegają na wykorzystaniu obserwowanych przejść stanów do przewidywania nieznanej końcowej wartości. W przypadku wartościowania strategii wartością przewidywaną jest zdyskontowana suma przyszłych nagród. Jest to *dochód TD*, dla kroku *t*, zdefiniowany jako:

$$z_t = \sum_{k=0}^{\infty} w^k \cdot r_{r+k}$$

Co ważne zachodzi: $\mathrm{E}_{\pi}\{z_t\} = U^{\pi}(s_t)$,

czyli dochody **TD** zbiegają do funkcji użyteczności dla ustalonej strategii.

Niech system uczący się ze wzmacnianiem zaobserwował zmianę stanu: $\mathrm{U}(s_t) = -0{,}5$ i $\mathrm{U}(s_{t+1}) = 0{,}5$. Ponieważ zakłada się, że w kolejnych krokach prognoza wartości jest coraz dokładniejsza, należy zmienić (uaktualnić) $\mathrm{U}(s_t)$ tak, aby zbliżyć jej wartość do wartości dla następnego kroku.

Reguła modyfikacji TD, stosowana w celu aktualizacji funkcji użyteczności, ma postać:

$$\mathrm{U}'(s_t) = \mathrm{U}(s_t) + \alpha(r_t + w\mathrm{U}_t(s_{t+1}) - \mathrm{U}(s_t))$$

gdzie r_t to nagroda za wykonanie akcji $a = \pi(s_t)$ (zgodnej ze strategią π) w stanie s_t.

Nazwa metody – metoda różnic czasowych – bierze się ze stosowania w powyższym wzorze różnicy funkcji użyteczności dla dwóch kolejnych stanów. Współczynnik uczenia α określa „wpływ" tej różnicy na zmianę funkcji użyteczności.

14.5. Uczenie się strategii

Przy pomocy funkcji użyteczności oceniamy strategie decyzyjne. Teraz system nauczy się strategii optymalnej.

W algorytmie *Q-learning* (tab. 14-2) stosujemy aktualizację funkcji wartości akcji mającą postać:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha[\, r_t(s) + w \max_{a'} Q_t(s_{t+1}, a') - Q(s_t, a_t)\,]$$

Tab. 14-2 Algorytm *Q-learning*

FOR wszystkie kroki czasu t DO:

1) ObserwujAktualnyStan(s_t);
2) a_t = WybierzAkcję(s_t, Q_t);
3) WykonajAkcję(a_t);
4) Obserwuj(wzmocnienie r_t i następny stan s_{t+1});
5) $\Delta = r_t + w \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$;
6) $Q_t(s_t, a_t)$ = Uaktualnij($Q_t(s_t, a_t)$, Δ));

Zauważmy, że algorytm Q-learning jest niezależny od strategii, może posługiwać się strategią inną niż ta, której się uczy. Widać to w punkcie drugim algorytmu, gdzie nie ma żadnego warunku wyboru akcji. Zauważmy też, że aby posługiwać się algorytmem Q-learning nie musimy nic wiedzieć o środowisku.

Te obserwacje rodzą podstawowe pytanie dla procesu uczenia: czy lepiej jest nauczyć się modelu świata, przyjąć strategię decyzyjną i oceniać strategię poprzez funkcję użyteczności $U(s_t)$, czy może lepiej jest nauczyć się strategii decyzyjnej poprzez określanie funkcji wartości akcji, nie znając modelu świata?

W miarę jak środowisko staje się coraz bardziej złożone, zastosowanie uczenia zakładającego znajomość modelu środowiska jest coraz bardziej uzasadnione.

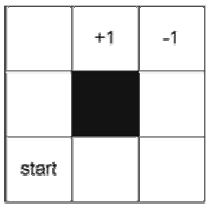
14.6. Pytania

1. Na czym polega zadanie uczenia się ze wzmocnieniem? Przedstawić ogólny algorytm uczenia się ze wzmocnieniem.
2. Jakie są główne formy uczenia ze wzmocnieniem?
3. Na czym polega dyskontowanie w kryterium nagród?
4. Zdefiniować funkcję użyteczności i funkcję wartości akcji.
5. Omówić przeznaczenie i budowę algorytmu uczenia TD.
6. Omówić przeznaczenie i budowę algorytmu Q-learning.

14.7. Zadania

Zad. 14.1

Przyjrzyjmy się modelowi przedstawionemu na rysunku 14.2. Przejścia pomiędzy sąsiadującymi stanami są dozwolone wyłącznie w pionie lub w poziomie (nie po przekątnej) i są jednakowe dla każdego stanu sąsiedniego. Stany oznaczone +1 oraz -1 to stany końcowe z wartościami wzmocnienia odpowiednio +1 i -1. Narysować tablicę przejść (z prawdopodobieństwami) i policzyć wartości funkcji użyteczności dla każdego stanu.



Rys. 14.1

15. Racjonalne decyzje

- Decyzje pojedyncze
- Podstawowe rodzaje klasyfikatorów numerycznych
- Sekwencja decyzji
- Programowanie dynamiczne

15.1. Decyzje pojedyncze

W niniejszym rozdziale zakładamy, że dysponujemy wieloma alternatywnymi modelami pewnego problemu (obiektu). Jesteśmy zainteresowani wyborem jednego z nich, najlepiej pasującego do aktualnych obserwacji problemu (obiektu). Zakładamy, że obserwacje dane są w postaci wektora liczb wyrażających cechy bądź atrybuty obiektu.

15.1.1. Klasyfikator numeryczny

Klasyfikacja wektora cech (wartości zmiennych obserwowanych) polega na podjęciu decyzji zgodnie z **funkcją decyzyjną** dla tego klasyfikatora i wiedzy nabytej uprzednio na podstawie zbioru uczącego (w procesie **uczenia**). Klasyfikator **numeryczny** polega na zastosowaniu funkcji decyzyjnej, która przypisuje wektor cech (obserwacji) $\mathbf{c} \in \Re^d$ do dyskretnej klasy (modelu) Ω_κ:

$$\zeta(\mathbf{c}) = \begin{cases} \Re^d \rightarrow \{1,2,\cdots,K\} \\ \mathbf{c} \rightarrow \kappa \end{cases} \tag{15.1}$$

Dokładna postać funkcji decyzyjnej klasyfikatora jest zwykle wynikiem procesu uczenia klasyfikatora. Ocena jakości i porównywanie różnych klasyfikatorów może uwzględniać szereg aspektów: wartość prawdopodobieństwa błędnej klasyfikacji, względnie koszt ryzyka, złożoność obliczeniowa funkcji decyzyjnej, zdolność do uogólnienia dla nieznanych próbek, zdolność do uczenia funkcji decyzyjnej, itd. Bardzo trudnym teoretycznym problemem jest określenie funkcji decyzyjnej, która optymalizuje wszystkie możliwe kryteria jakości klasyfikatora. Dlatego też w dalszej analizie problemu skupimy się na pierwszym kryterium – na stworzeniu optymalnego klasyfikatora w sensie minimalizacji ryzyka błędnej klasyfikacji.

Sensowne jest przyjęcie podstawowych założenia przy konstrukcji klasyfikatora:

1. jeśli 2 wektory cech należą do tej samej klasy to występuje pomiędzy nimi mała odległość,
2. jeśli wektory cech należą do różnych klas to pomiędzy nimi jest duża odległość.

Miarami odległości mogą być: funkcje gęstości prawdopodobieństwa, odległość *Euklidesowa*, metryka modułowa, odległość *Mahalanobisa*, itd.

Optymalny klasyfikator

Klasyfikator stochastyczny (teorio-decyzyjny) tworzony jest według następującej zasady: zgromadź założenia (wiedzę) o analizowanej dziedzinie; wyznacz ryzyko błędnej klasyfikacji (lub decyzji), wyznacz funkcję decyzyjną, która minimalizuje podane ryzyko.

Każda decyzja generuje pewne koszty (ryzyko). Po wielu decyzjach możemy określić średni koszt lub ryzyko. Obliczenie średnich wartości ryzyka wymaga istnienia pełnej informacji statystycznej (w postaci zadanego rozkładu zmiennej losowej) lub uprzedniej obserwacji dającej reprezentatywny zbiór próbek.

Założenia o analizowanej dziedzinie

Statystyczne własności klas reprezentowane są przez rodzinę n-wymiarowych rozkładów prawdopodobieństwa cech pod warunkiem klas: $p(\mathrm{c} \mid \Omega_k)$, k=1,...,K. Rozkład ten może mieć zadaną postać (parametryczny rozkład) lub nie (nieparametryczny). W procesie *uczenia* szacowane są parametry rozkładów względnie aproksymowane są całościowe rozkłady prawdopodobieństw w przestrzeni cech. Znany jest też rozkład prawdopodobieństwa klas ($p(\Omega_k)$, k=1,...,K).

Obliczenie ryzyka V

Oznaczamy koszty $\{r_{jk} = r\ (j \mid k)\ (j, k= 1,2, \ldots ;K)\}$ błędnej klasyfikacji, w wyniku której obiekt klasy o indeksie k jest błędnie klasyfikowany jako należący do klasy o indeksie j. Zakładamy, że koszty te są określane przez eksperta w zależności od zastosowania. Należy też oszacować prawdopodobieństwa błędnych decyzji, $p(j \mid k)$, j,k=1,...,K.

Teraz można już obliczyć sumaryczne ryzyko według wzoru:

$$\mathrm{V} = \sum_k \sum_j p_k\ p(j \mid k)\ r_{\mathrm{jk}}\ . \tag{15.2}$$

Reguła decyzyjna

Celem reguły decyzyjnej jest minimalizacja kosztu ryzyka V. Dla wyznaczenia ogólnej reguły należy przyjąć pewne założenia odnośnie funkcji kosztu.

Np. niech funkcja kosztu ma binarny charakter:

$$r_{kk} = 0, \qquad r_{ik} = 1, \text{ dla } \ i \neq k, \ \text{i} \ \ i,k = 1, \ldots , K.$$

Taka postać funkcji kosztu wyznacza główną rolę w minimalizowaniu kosztu ryzyka V funkcji prawdopodobieństwa błędnej klasyfikacji. Dualnie, zamiast minimalizować prawdopodobieństwo błędnej klasyfikacji, możemy zdefiniować regułę decyzyjną, która maksymalizuje prawdopodobieństwo poprawnej klasyfikacji dla każdej obserwacji. W naturalny sposób wyraża to prawdopodobieństwo warunkowe a posteriori $p(\Omega_i \mid \mathrm{c})$. Czyli reguła decyzyjna optymalnego klasyfikatora stochastycznego obserwacji c przyporządkowuje klasę Ω_i o największej wartości

$$p(\Omega_i \mid \mathrm{c}) = \mathrm{p}(\mathrm{c} \mid \Omega_i)\ \mathrm{p}(\Omega_i), \ , i = 1; \ldots , K.$$

Taką regułę decyzyjną stosuje *klasyfikator Bayesa*. Dla binarnej funkcji kosztu jest to optymalny klasyfikator w sensie minimalizacji *V*, a każdy dobrze określony klasyfikator numeryczny aproksymuje z góry błąd klasyfikatora Bayesa.

Jednak zastosowanie klasyfikatora Bayesa nie zawsze jest możliwe, gdyż wymaga on pełnej informacji statystycznej o analizowanej dziedzinie.

15.2. Podstawowe rodzaje klasyfikatorów numerycznych

Klasyfikator według funkcji potencjału („maszyna liniowa")

Należy określić postać parametrycznej funkcji (np. liniowa funkcja, wielomian 2-go stopnia) właściwej dla charakteru zbioru próbek (np. liniowo separowalne reprezentacje – cechy - wzorców różnych klas). Następnie w wyniku procesu uczenia klasyfikatora należy związać z każdą klasą jej konkretną funkcję wyrażającą tzw. potencjał w przestrzeni cech (uzyskujemy właściwe wartości parametrów funkcji potencjału dla każdej klasy). Wreszcie podczas aktywnej pracy klasyfikatora należy klasyfikować kolejne wektory cech zgodnie z kryterium maksymalizacji wartości funkcji potencjału dla danego wektora cech. Klasyfikator o liniowych funkcjach potencjału zwykle optymalizuje kryterium złożoności obliczeniowej dla klasyfikatorów.

Klasyfikator SVM - maszyna wektorów wspierających

Zakłada się w nim, że próbki uczące mają skończony charakter, tzn. w ogólności nie są reprezentatywne dla całości rozkładu. Klasyfikacja *k* klas ma charakter wielokrotnej klasyfikacji binarnej.

Klasyfikator SVM jest optymalny z punktu widzenia kryterium zdolności do uogólniania dla nieistniejących próbek.

Klasyfikator stochastyczny Bayesa

Zakładamy, że rozkłady prawdopodobieństwa klas nad przestrzenią cech mają znaną postać (np. rozkład normalny) albo mogą być w pełni określone. Zadaniem konstrukcji (procesu uczenia) jest jedynie określenie parametrów tego rozkładu lub całego rozkładu (o nieparametrycznej postaci) dla każdej z rozpatrywanych klas. Zakłada się, że próbki uczące mają charakter reprezentatywny dla całości rozkładu. Wtedy klasyfikator Bayesa jest optymalny z punktu widzenia minimalizacji błędu klasyfikacji. Klasyfikator statystyczny może dysponować parametryczną lub nie-parametryczną funkcją gęstości prawdopodobieństwa.

Jako przypadki szczególne klasyfikatora Bayesa rozpatruje się klasyfikator „największej wiarygodności" i klasyfikator geometryczny minimalnej odległości.

Klasyfikator neuronowy - wielowarstwowy perceptron

Warto zastosować sieć neuronową do aproksymacji funkcji decyzyjnych w sytuacji, gdy nie wymagamy jawnych postaci tych funkcji a znalezienie ich na drodze analitycznej jest żmudne lub niemożliwe. Klasyfikator neuronowy dysponuje wysoką zdolnością do uczenia się funkcji decyzyjnej.

15.3. Sekwencja decyzji prostych

Dana jest sekwencja $\mathbf{O}$ prostych obserwacji:

$$\mathbf{O} = (o^1, o^2, ..., o^N) .$$

Należy przypisać sekwencji $\mathbf{O}$ odpowiadającą jej sekwencję prostych klas (decyzji):

$$\mathbf{\Omega} = (\Omega^1, \Omega^2, ..., \Omega^N) .$$

Każda obserwacja o^i w tej sekwencji reprezentowana jest wektorem cech c^i, co daje sekwencję wektorów cech (wartości zmiennych obserwowanych):

$$C = (c^1, c^2, ..., c^N) .$$

Problem znalezienia optymalnej sekwencji można rozwiązać stosując klasyfikator Bayesa, czyli określając maksimum prawdopodobieństwa a posteriori:

$$p(\mathbf{\Omega} \mid \mathbf{C}) = \frac{p(\mathbf{\Omega}) p(\mathbf{C} \mid \mathbf{\Omega})}{p(\mathbf{C})} = \alpha \cdot p(\mathbf{\Omega}) p(\mathbf{C} \mid \mathbf{\Omega}) \tag{15.3}$$

Niech k oznacza liczbę prostych klas, a n – długość wektora cech. W ogólnym przypadku określenie prawdopodobieństw (11.3) wymaga wyznaczenia: k^N wartości a priori prawdopodobieństw N-elementowych sekwencji klas, k^N gęstości a priori prawdopodobieństw warunkowych o rozmiarze n x N.

W praktyce wyznaczenie wielowymiarowych rozkładów jest trudnym zadaniem. Zmniejszenie rozmiaru problemu uzyskuje się dzięki dwóm uproszczeniom. Po pierwsze, zakłada się, że poszczególne obserwacje są statystycznie niezależne:

$$p(C \mid \Omega) = \prod_{i=1}^{N} p(c^i \mid \Omega^i = \Omega_k) . \tag{15.4}$$

Zamiast k^N *rozkładów* $n \times N$-wymiarowych potrzebujemy tylko k rozkładów n-wymiarowych. Po drugie, założenie odpowiadające modelowi procesu Markowa pierwszego rzędu, klasyfikacja każdego prostego problemu zależy tylko od bezpośrednio poprzedzającego problemu. Zamiast

$$p(\Omega) = p(\Omega^1, \Omega^2, ..., \Omega^N) = p(\Omega^1) p(\Omega^2 \mid \Omega^1) p(\Omega^3 \mid \Omega^1, \Omega^2) \cdots p(\Omega^N \mid \Omega^1 \cdots \Omega^{N-1})$$

wystaczy policzyć zależność:

$$p(\Omega) = p(\Omega^1)p(\Omega^2 \mid \Omega^1)p(\Omega^3 \mid \Omega^2)\cdots p(\Omega^N \mid \Omega^{N-1}). \quad (15.5)$$

W miejsce k^N wartości a-priori prawdopodobieństw sekwencji klas wystarczy k^2 wartości prawdopodobieństw przejść $p(\Omega_i \mid \Omega_j)$ i k wartości $p(\Omega_k)$.

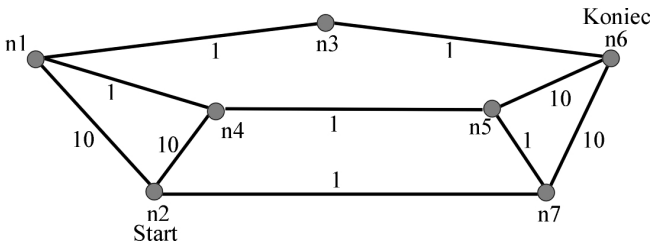
15.4. Programowanie dynamiczne

Do poszukiwania sekwencji klas najlepiej dopasowanej do sekwencji obserwacji posłuży nam metoda tzw. *programowania dynamicznego* (podana przez Bellmana). Założenia *programowania dynamicznego* są dualne do założeń przyjętych w bayesowskim modelu dla procesu Markowa 1 rzędu, z tym, że zamiast miary prawdopodobieństwa metoda posługuje się funkcją kosztu. Zakłada się w niej, że istnieje funkcja kosztu elementarnych decyzji, która jest *monotoniczna* (wartość kosztu nie maleje po każdej kolejno dodawanej decyzji) i *addytywna* (sumujemy dotychczasowe koszty już podjętych decyzji i nowe koszty związane z kolejną decyzją).

Należy zdefiniować graf problemu (jak np. podany na rys. 15.1), w którym:

- węzły reprezentują pojedyncze decyzje, inaczej mówiąc – możliwe wyniki prostej klasyfikacji pojedynczej obserwacji, mierzone miarą kosztu dualnie do $p(C|\Omega^i)$ (w równaniu 15.5), a
- łuki pomiędzy węzłami i ich etykiety – reprezentują koszty przejść, dualnie do rozkładów warunkowych, $p(\Omega^i|\ \Omega^{i-1})$ (w równaniu 15.5).

Istotna zasada programowania dynamicznego to: każda uznana za optymalną pod-sekwencja nie wymaga zmiany po znalezieniu całej sekwencji (jest już optymalnie wybrana).



Rys. 15.1: Przykład grafu problemu, w którym każdy węzeł akceptuje pojedynczą obserwację (koszty prostych klasyfikacji związane są z węzłami i generalnie zależą od obserwacji).

W grafie problemu należy wyznaczyć stan (lub stany) początkowy i stan (stany) końcowy.

Przestrzeń decyzji w programowaniu dynamicznym przedstawimy w postaci specyficznego grafu decyzyjnego – węzły takiego grafu (nazywane stanami) wyznaczają 2-wymiarową tablicę – stany są uporządkowane przy pomocy dwóch indeksów – indeksu węzła (grafu problemu) i indeksu kroku obliczeń.

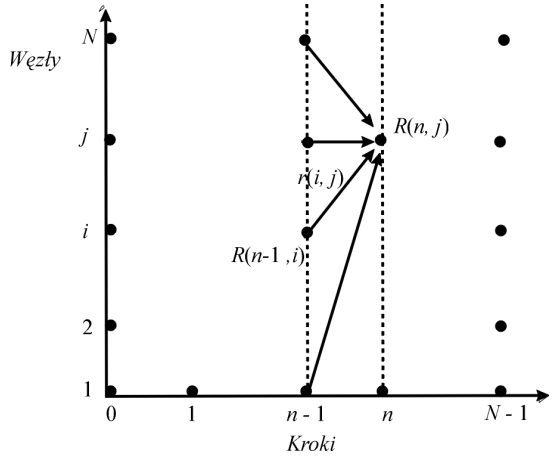
Koszt $R(n;\ j)$ obliczony dla stanu odpowiadającego węzłowi (klasie prostej) *j* osiąganemu w kroku obliczeń n , zgodnie z rekurencyjną formułą wynosi (rys. 15.2)

$$R(n, j) = \min_i \{R(n-1, i) + d(n, j) + r(i, j)\} \quad (15.6)$$

gdzie: $R(n\text{-}1;\ i)$ oznacza sumaryczny koszt decyzji poniesionych od momentu startu *idąc do przodu* do poprzedniego węzła i osiąganego po n-1 decyzjach prostych, czyli są to koszty osiągnięcia stanu $(n\text{-}1;\ i)$ w przestrzeni decyzji; $d(n,j)$ to koszt pojedynczej klasyfikacji n-tej obserwacji jako klasy j, $r(i,j)$ to koszt elementarnego przejścia pomiędzy węzłami i, j grafu problemu.

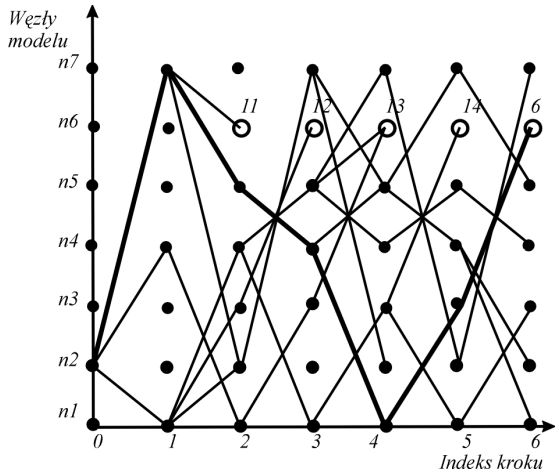
Liczba kroków decyzyjnych, które musimy wykonać w programowaniu dynamicznym, odpowiada większej z dwóch liczb - najdłuższej ścieżce (bez zapętleń) w grafie problemu prowadzącej od

stanu początkowego do jednego ze stanów końcowych lub liczbie symboli w aktualnej sekwencji danych.



Rys. 15.2: Przykład rozszerzania ścieżki w przestrzeni decyzji. Jeśli wybraliśmy w kroku n stan (n, j), to jest to optymalny wybór, tzn. wszystkie inne alternatywne dojścia do stanu (n, j) są „gorsze"(mają wyższy koszt).

Przykład. Przykład przestrzeni decyzji w programowaniu dynamicznym dla problemu z rys. 15.1 i przy braku obserwacji (co jest równoważne zerowym kosztom klasyfikacji prostych) pokazuje rys. 15.3.



Rys. 15.3: Przykład przestrzeni decyzji w programowaniu dynamicznym. Znaleziono 5 alternatywnych sekwencji decyzyjnych, z których jedna (najdłuższa sekwencja) jest najlepsza (o koszcie 6).

Znaleziona ścieżka [(0, n2)-(1, n7)-(2, n5)-(3, n4)-(4, n1)-(5, n3)-(6, n6)] jest optymalna. Jej koszt wynosi (6), podczas gdy inne alternatywne przejścia pomiędzy stanami (0, n2) a (*i*, n6) (*i*=2,3,4,5,6) posiadają wyższe koszty: 11,12,13 i 14 (mimo, iż same ścieżki są krótsze).

15.5. Pytania

1. Wyjaśnić pojęcie klasyfikatora numerycznego i podać główne odmiany takich klasyfikatorów.
2. W jakich warunkach klasyfikator Bayesa jest optymalny?

3. W jaki sposób model procesu Markowa 1 rzędu pozwala uprościć klasyfikację sekwencji obserwacji?
4. Na czym polega programowanie dynamiczne?

15.6. Zadania

Zad. 15.1

Zdefiniować graf problemu dla programowania dynamicznego, które ma dopasować dwa łańcuchy liter*:* modelową sekwencję (*d z i e ń*) z aktualnie analizowaną sekwencją z przekłamaniami (*a d z e t e*).

Zad. 15.2

Przeprowadzić symulację metody programowania dynamicznego dla grafu problemu zaprojektowanego w zad. 15.1 i kosztów klasyfikacji liter w obserwowanej sekwencji podanych w tabeli 15.1.

Tab. 15.1

Indeks obserwacji	1	2	3	4	5	6
Wybrana litera	a	d	z	e	t	e
Koszt klasyfikacji	8	2	2	3	5	8

Zad. 15.3

Zmienić graf problemu zdefiniowany w zad. 15.1 do postaci modelu HMM dla słowa „dzień” i możliwych obserwacji (liter): „a”, „i”, „e”, „d”, „ń”, „t”, „z”, tolerujący przekłamania na tyle, aby możliwe było zaakceptowanie sekwencji obserwowanej „adzete”.

Zad. 15.4

Przeprowadzić symulację „przeszukiwania Viterbiego” dla modelu HMM zaprojektowanego w zad. 15.3 i rozkładów dyskretnych prawdopodobieństwa zmiennej obserwacji (liter) w kolejnych chwilach czasu $P(\Omega_k|O^t)$ podanych w tabeli 15-2.

Tab. 15.2

czas / *litera*	1	2	3	4	5	6
a	0.5	0	0	0	0	0
i	0	0	0.5	0.5	0	0
e	0	0	0	0.5	0.3	0.5
d	0.5	0.5	0	0	0	0
ń	0	0	0	0	0.2	0.2
t	0	0	0	0	0.5	0
z	0	0.5	0.5	0	0	0

16. Uczenie statystyczne

- Klasyfikator Bayesa
- Uczenie modelu HMM
- Klasteryzacja (uczenie bez nadzoru)
- Zadania

16.1. Klasyfikator stochastyczny Bayesa

16.1.1. Definicja

Zakładamy istnienie statystyki klas w przestrzeni cech w postaci znanych nam:

- a priori prawdopodobieństw klas $p(\Omega) = [P(\Omega_1), P(\Omega_2), \ldots, p(\Omega_K)]$

- a priori gęstości prawdopodobieństw warunkowych $p(c \mid \Omega_\kappa)$, dla wszystkich $\Omega_\kappa \in \Omega$.

Zgodnie ze wzorem Bayesa prawdopodobieństwo warunkowe a posteriori wynosi:

$$p(\Omega_k \mid c) = \frac{p(\Omega_k)p(c \mid \Omega_k)}{p(c)} = \frac{p(\Omega_k)p(c \mid \Omega_k)}{\sum_{j=1}^{K} p(\Omega_j)p(c \mid \Omega_j)} \tag{16.1}$$

Reguła decyzyjna

Klasyfikator Bayesa poszukuje maksymalnego prawdopodobieństwa a posteriori:

$$\varsigma(c) = \arg\max_{\lambda} p(\Omega_\lambda \mid c) = \arg\max_{\lambda} p(\Omega_\lambda)p(c \mid \Omega_\lambda). \tag{13.2}$$

Reguła ML

Dla jednorodnego rozkładu klas ($p(\Omega_\kappa) = p(\Omega_\lambda)$ dla wszystkich par klas) reguła decyzyjna Bayesa (16.2) sprowadza się do reguły decyzyjnej największej wiarygodności (ang. *maximum likelihood,* ML):

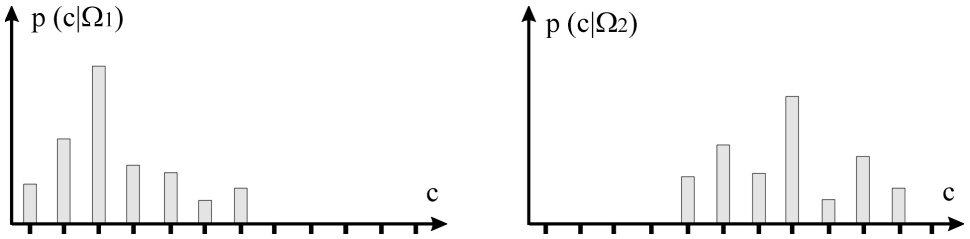
$$\varsigma(c) = \arg\max_{\lambda} p(\Omega_\lambda \mid c) = \arg\max_{\lambda} p(\Omega_\lambda)p(c \mid \Omega_\lambda) = \arg\max_{\lambda} p(c \mid \Omega_\lambda). \tag{16.3}$$

16.1.2. Uczenie się rozkładów prawdopodobieństwa

W procesie uczenia klasyfikatora Bayesa należy wyznaczyć rozkłady prawdopodobieństwa a priori.

A priori prawdopodobieństwa klas, $P(\Omega_k)$, $1 \le k \le K$, można wyliczyć na podstawie relatywnej częstości klas w zbiorze próbek.

Modele funkcji gęstości prawdopodobieństwa $p(c \mid \Omega_k)$ dzielimy na parametryczne i nieparametryczne. Nieparametryczne modele polegają na zastosowaniu dyskretnego rozkładu prawdopodobieństwa w jawnej postaci (np. histogram) (rys. 16.1). Parametryczne modele zakładają istnienie odpowiedniej rodziny funkcji gęstości prawdopodobieństwa i wymagają oszacowania ich parametrów na podstawie próbek uczących.



Rys. 16.1: Histogramy cech jako nieparametryczne modelowanie gęstości prawdopodobieństwa.

Parametryzacja gęstości rozkładu w klasyfikatorach statystycznych.

Najczęściej staramy się dopasować dla *n*–wymiarowego wektora cech rozkład normalny gęstości prawdopodobieństwa, czyli *n*-wymiarową funkcję Gaussa:

$$p(c \mid \Omega_k) = \frac{1}{\sqrt{|\det(2\pi \sum_k)}} \cdot \exp\left(-\frac{(c-\mu_k)^T \cdot \sum_k^{-1} \cdot (c-\mu_k)}{2} \right) \tag{16.4}$$

Wartości parametrów μ_k i Σ_k (dla każdej klasy Ω_k) mogą być oszacowane zgodnie z zasadą estymatora największej wiarygodności ML (ang. *maximum likelihood*).

Niech $C^{(k)}$ oznacza zbiór próbek uczących (wektorów cech) dla wzorców należących do klasy Ω_k, a $\theta^{(k)}$ będzie zbiorem parametrów rozkładu prawdopodobieństwa $p(c \mid \Omega_k)$ (dla rozkładu normalnego $\theta^{(k)}{}_N = \{\mu_k, \Sigma_k\}$). Estymatorem największej wiarygodności dla klasy Ω_k jest każdy taki zestaw parametrów $\theta^{(k)}$, który maksymalizuje wartość prawdopodobieństwa obserwowanych cech:

$$\max_{\theta^{(k)} \in \Theta} \sum_{c_j \in C^{(k)}} P(c_j \mid \theta^{(k)}) \tag{16.5}$$

Można sprawdzić, że dla postaci rozkładu normalnego rozwiązanie powyższego problemu (16.5) jest postaci:

$$\mu_k \cong \frac{1}{N_k} \sum_{j=1}^{N_k} c_j; \quad c_j \in C^{(k)} \tag{16.6}$$

$$\sum_k \cong \frac{1}{N_k} \sum_{j=1}^{N_k} (c_j - \mu_k)(c_j - \mu_k).$$

Dla innych postaci rozkładów niż rozkład normalny często nie jest możliwe uzyskanie dokładnych analitycznych postaci dla estymatorów największej wiarygodności. Wtedy należy zastosować iteracyjny sposób szacowania parametrów rozkładu, czyli rozwiązać „równania wiarygodności” o postaci:

$$\frac{\partial}{\partial \theta_i} \log \sum_{c_j \in C^{(k)}} P(c_j \mid \theta^{(k)}) = 0 \quad , \quad \theta_i \in \theta^{(k)} \tag{16.7}$$

W tym celu można zastosować metodę Newtona lub inną gradientową metodę iteracyjną. Przy takim podejściu wyznaczany jest w każdej iteracji optymalny kierunek w przestrzeni parametrów dla poszukiwania lokalnego maksimum „funkcji wiarygodności”.

Algorytm EM

Przykładem prostego iteracyjnego algorytmu, który nie posługuje się kierunkiem w przestrzeni parametrów a poszukuje rozwiązania problemu metodą kolejnych przybliżeń jest algorytm maksymalizacji oczekiwań EM (ang. *expectation maximization*).

Algorytm EM dla prawdopodobieństwa o rozkładzie Gaussa

Inicjalizacja parametrów w sposób losowy.

REPEAT kroki 1 i 2:

(Krok estymacji). Oblicz aktualne oszacowanie rozkładu

$$P_j^{(k)} = \alpha\, P(c_j \mid \Omega_k)\text{, dla } c_j \in C^{(k)};$$

$$P^{(k)} = \sum_j P_j^{(k)}\,.$$

(Krok maksymalizacji). Oblicz aktualne parametry szukanego rozkładu prawdopodobieństwa:

$$\mu_k = \sum_j (P_j^{(k)} \cdot c_j) / P^{(k)}$$

$$\Sigma_k = \sum_j [P_j^{(k)} \cdot (c_j - \mu_k) \cdot (c_j - \mu_k)^T] / P^{(k)}\,.$$

UNTIL nie zachodzi zbieżność parametrów rozkładu.

16.1.3. Klasyfikator geometryczny według minimalnej odległości

Klasyfikator geometryczny jest to uproszczona wersja klasyfikatora Bayesa przy zastosowaniu uproszczeń rozkładów priori i wyrażeniu miary odległości prawdopodobieństwa przez odległość Euklidesa w przestrzeni cech.

Reguła decyzyjna klasyfikatora według minimalnej odległości

Zamiast klasy o maksymalnym a posteriori prawdopodobieństwie $p(\Omega_k \mid \mathrm{c})$ wybierz klasę o minimalnej odległości:

$$\mathrm{d}(\Omega_k\,, \mathrm{c}) = -\log p(\Omega_k \mid \mathrm{c}).$$

Prześledzimy wyprowadzenie powyższej reguły z reguły klasyfikatora Bayesa dla przypadku 2 klas.

Dla każdej pary klas reguła decyzyjna Bayesa – „wybierz klasę Ω_1, gdy

$$p(\Omega_1)\, p(\mathrm{c} \mid \Omega_1) > p(\Omega_2)\, p(\mathrm{c} \mid \Omega_2) \tag{16.8}$$

dla postaci rozkładu Gaussa po normalizacji obu rozkładów, tzn. po przyjęciu:

$$\Sigma_1 = \Sigma_2 \tag{16.9}$$

i po jej logarytmowaniu, odpowiada regule decyzyjnej według odległości w przestrzeni:

$$\log p(\Omega_1) + \mu_1^T \Sigma^{-1} c - \frac{1}{2}\mu_1^T \Sigma^{-1} \mu_1 > \log p(\Omega_2) + \mu_2^T \Sigma^{-1} c - \frac{1}{2}\mu_2^T \Sigma^{-1} \mu_2 \tag{16.10}$$

Specjalna postać powyższej reguły decyzyjnej otrzymana przy założeniu $p(\Omega_1) = p(\Omega_2)$, to reguła decyzyjna:

$$\mu_1^T \Sigma^{-1} c - \frac{1}{2}\mu_1^T \Sigma^{-1} \mu_1 > \mu_2^T \Sigma^{-1} c - \frac{1}{2}\mu_2^T \Sigma^{-1} \mu_2$$

$$(c - \mu_1)^T \Sigma^{-1} (c - \mu_1) < (c - \mu_2)^T \Sigma^{-1} (c - \mu_2) \tag{16.11}$$

Po kolejnym uproszczeniu rozkładów a priori – normalizacji przestrzeni cech tak, aby zachodziło $\Sigma = \mathbf{1}$ – otrzymujemy ostatecznie:

$$(c - \mu_1)^T (c - \mu_1) < (c - \mu_2)^T (c - \mu_2) \quad \text{czyli} \quad |\, c - \mu_1 \,|^2 < |\, c - \mu_2 \,|^2 \tag{16.12}$$

Reguła decyzyjna (16.12) porównuje kwadratowe miary odległości Euklidesowej aktualnego wektora cech od środków rozkładów klas w przestrzeni cech i działa zgodnie z zasadą wyboru najmniejszej odległości.

16.1.4. Klasyfikator „według k sąsiadów”

Zakładamy istnienie zbioru próbek uczących i wcześniej klasyfikowanych wektorów cech - $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$, z których każda próbka względnie wektor cech należy do klasy zgodnej z regułą decyzyjną $\zeta(c_i)$ tego klasyfikatora. Każda próbka względnie każdy kolejny wektor cech staje się reprezentantem swojej klasy.

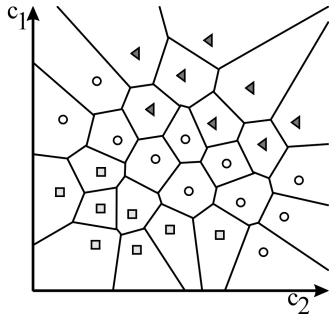
Reguła decyzyjna klasyfikatora według k sąsiadów.

Przyporządkuj nowemu wektorowi cech $\mathbf{c}$ tę klasę, do której należy jego najbliższy sąsiad (NN-klasyfikator, k=1), względnie większość spośród k najbliższych sąsiadów (kNN-klasyfikator, k>1).

$$\varsigma(\mathbf{c}) = \arg\min_{\varsigma(\mathbf{c}_i)} |\{|\,\mathbf{c} - \mathbf{c}_i\,|\,, i = 1,2,\ldots,n\} \tag{16.13}$$

W praktyce stosuje się k = 3-7.

Zbiór punktów przestrzeni cech, dla których najbliższym sąsiadem spośród istniejących cech jest c_i nazywany jest komórką Voronoia dla c_i. Tym samym aktualny zbiór C wyznacza podział Voronoia całej przestrzeni cech (rys. 16.2).



Rys. 16.2: Ilustracja zbioru Voronoia dla 2-wymarowych wektorów cech.

Zauważmy, że powyższy podział przestrzeni cech można utożsamić z szacowaniem a posteriori prawdopodobieństw $p(\Omega_k \mid \mathrm{c})$ i tym samym można oszacować błąd klasyfikacji w terminach błędu dla optymalnego klasyfikatora Bayesa. Prawdopodobieństwo błędnej klasyfikacji dla klasyfikatora według k najbliższych sąsiadów (oznaczmy je jako P_{NN}) zbiega do prawdopodobieństwa błędu Bayesa (oznaczmy je jako P_B) wraz ze wzrostem liczby próbek uczących N i liczby badanych sąsiadów k. Prawdopodobieństwo błędnej klasyfikacji dla *kNN*-klasyfikatora (przy K klasach) można bowiem oszacować jako:

$$P_B \le P_{kNN} \le P_B\left(2 - P_B \frac{k}{k-1}\right) \tag{16.14}$$

czyli dla dostatecznie małych P_B w przybliżeniu zachodzi: $P_B \le P_{kNN} \le 2\,P_B$.

16.2. Uczenie modelu HMM

Wartości prawdopodobieństw obserwacji (akceptacji symboli) zależą od przyjętych parametrów modelu Markowa. W procesie uczenia modelu dokonujemy estymacji *parametrów modelu*, według zasady EM (ang. *expectation maximization*), stosując algorytm *Bauma-Welcha* lub *trening Viterbiego*.

16.2.1. Zadanie algorytmu Bauma-Welcha

Dane: aktualny dyskretny model HMM – struktura i parametry λ, sekwencja obserwacji $\mathbf{O} = (\mathrm{O}_1 \ldots \mathrm{O}_T)$

Szukamy: prawdopodobieństw obserwacji pod warunkiem modelu.

Stosując podejście ML (maksymalizacji wiarygodności) powinniśmy obliczyć i maksymalizować prawdopodobieństwo:

$$\mathrm{P}_{HMM}(\boldsymbol{\lambda}) = P(\mathbf{O} \mid \boldsymbol{\lambda}) = \sum_{\mathbf{q} \in \mathbf{S}^T} P(\mathbf{O}, \mathbf{q} \mid \boldsymbol{\lambda}) \tag{16.15}$$

gdzie $\mathbf{q}$ – każda możliwa sekwencja stanów o długości T.

16.2.2. Określenie prawdopodobieństw "wprzód i wstecz"

Algorytm "wprzód-wstecz" oblicza prawdopodobieństwa akceptacji symboli wyjściowych (obserwacji) w każdym stanie modelu HMM. Prawdopodobieństwo “wprzód” – prawdopodobieństwo generowania sekwencji częściowej kończonej w stanie o indeksie j w chwili t :

$$\alpha_t(j) = P(\mathrm{O}_1 \cdots \mathrm{O}_t, q_t = s_j \mid \boldsymbol{\lambda});$$

Prawdopodobieństwo “wstecz” – prawdopodobieństwo generowania reszty sekwencji, począwszy od stanu o indeksie i w chwili t :

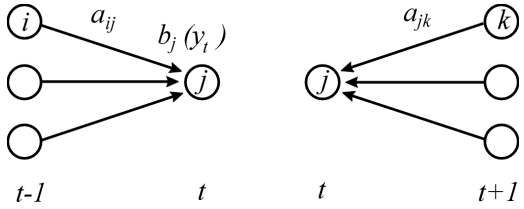
$$\beta_t(i) = P(\mathrm{O}_{t+1} \cdots \mathrm{O}_T, q_t = s_i \mid \boldsymbol{\lambda}).$$

Tym samym aktualnie szacowane (pod warunkiem λ) prawdopodobieństwo wizytowania stanu o indeksie j w chwili t dla całej obserwacji O jest iloczynem prawdopodobieństw „wprzód i wstecz” dla danego stanu i chwili t, czyli postaci:

$$\alpha_t(j) \cdot \beta_t(j) = P(\mathbf{O}, q_t = s_j \mid \boldsymbol{\lambda}) \tag{16.16}$$

16.2.3. Algorytm estymacji prawdopodobieństw “wprzód i wstecz”

Pary $\{\alpha_t(j), \beta_t(j)\}$ obliczymy iteracyjnie przechodząc obserwację w obu kierunkach (rys. 16.3).



Rys. 16.3: Idea obliczania prawdopodobieństwa w przód i wstecz.

Jeśli dysponujemy wartością $\alpha_{t-1}(i)$, dla wszystkich i , w chwili t-1, to obliczymy $\alpha_t(j)$ uwzględniając prawdopodobieństwo przejścia do stanu j z każdego stanu i gdy akceptujemy symbol y_t.

Podobnie $\beta_t(j)$ obliczamy na podstawie prawdopodobieństw stanów-następców $\beta_{t+1}(k)$.

Inicjalizacja:

$\alpha_0(\#_0) = 1$; // stan początkowy

FOR $j = 1, \ldots, \mathrm{N}$ DO

$\beta_{T+1}(\#_{T+1}) = 1$; // stan końcowy

FOR $i = 1, \ldots, \mathrm{N}$ DO

$\alpha_1(j) = \pi_j \cdot b_j(\mathrm{O}_1)$; $\beta_T(i) = 1$;

Iteracja

FOR ($t > 1$ AND $j = 1, ..., \mathrm{N}$) DO FOR ($t < T$ AND $i = 1, ..., \mathrm{N}$) DO

$$\alpha_t(j) = \left(\sum_{i=1}^{N} \alpha_{t-1}(i) \cdot a_{ij} \right) \cdot b_j(O_t) \qquad \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

Zwracany wynik – prawdopodobieństwo łączne całej sekwencji obserwacji jest alternatywnie dane jako:

$$P(\mathbf{O} \mid \boldsymbol{\lambda}) = \sum_{j=1}^{N} \alpha_T(j) \qquad P(\mathbf{O} \mid \boldsymbol{\lambda}) = \sum_{j=1}^{N} \pi_j b_j(\mathrm{O}_1) \beta_1(j)$$

W ogólnej postaci wyrazimy to też jako::

$$P(\mathbf{O} \mid \boldsymbol{\lambda}) = \sum_{j=1}^{N} \alpha_T(j) \beta_T(j) \qquad P(\mathbf{O} \mid \boldsymbol{\lambda}) = \sum_{j=1}^{N} \alpha_1(j) \beta_1(j)$$

16.2.4. Estymacja parametrów modelu HMM

Wartości prawdopodobieństw obserwacji (akceptacji symboli) zależą od przyjętych parametrów modelu Markowa. W procesie uczenia modelu dokonujemy estymacji *parametrów modelu*, według zasady EM (ang. *expectation maximization*), stosując algorytm *Bauma-Welcha* lub *trening Viterbiego*.

„Uczenie Bauma-Welcha”

POWTARZAJ poniższe kroki (1)-(3) zadaną liczbę razy:

1) Dla aktualnej estymacji zbioru parametrów: $\lambda = \{\pi_i, a_{ij}, b_{jk} \mid i,j=1,...,N; k=1,...,M\}$ i dla $(t=1,\ldots,T)$; obliczyć wszystkie prawdopodobieństwa $\alpha_t(j), \beta_t(i)$ za pomocą wygładzania “w przód-wstecz”.

2) (Krok E)

Obliczamy oczekiwane prawdopodobieństwo przejść $s_i \rightarrow s_j$ w chwili t mając daną sekwencję wyjść $\mathbf{O} = (\mathrm{O}_1, \ldots, \mathrm{O}_T)$:

$$\xi_t(i,j) = P(q_t = s_i, q_{t+1} = s_j \mid \mathbf{O}, \boldsymbol{\lambda})$$

$$= \frac{P(q_t = s_i, q_{t+1} = s_j, \mathbf{O} \mid \boldsymbol{\lambda})}{P(\mathbf{O} \mid \boldsymbol{\lambda})} = \frac{\alpha_t(i) \cdot a_{ij} \cdot b_j(\mathrm{O}_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_t(i) \cdot \beta_t(i)}. \tag{16.17}$$

Oznaczmy przez $\gamma(i, j)$ oczekiwane prawdopodobieństwo przejścia ze stanu i do stanu j, w dowolnej chwili t od 1 do T:

$$\gamma(i,j) = \sum_t \xi_t(i,j). \tag{16.18}$$

Sumując te prawdopodobieństwa po wszystkich stanach j, obliczymy $\gamma(i)$, które reprezentuje oczekiwane prawdopodobieństwo tego, że stan i będzie wizytowany, w dowolnej chwili t od 1 do T:

$$\gamma(i) = \sum_j \sum_t \xi_t(i,j)\,. \tag{16.19}$$

Wybierając tylko te sytuacje, gdy stan i akceptuje symbol klasy Ω_k, uzyskamy oczekiwane prawdopodobieństwo takich sytuacji jako:

$$\gamma(i,\Omega_k) = \sum_{t:\mathrm{O}_t \in \Omega_k} \sum_j \xi_t(i,j)\,. \tag{16.20}$$

Oczekiwane prawdopodobieństwo wizytowania stanu s_i w chwili t wynosi:

$$\gamma_t(i) = P(q_t = s_i \mid \mathbf{O}, \boldsymbol{\lambda}) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} = \sum_{j=1}^{N} \xi_t(i,j) \tag{16.21}$$

3) (Krok M)

W tym kroku ponawiamy estymację parametrów modelu HMM – wektora początkowego π i macierzy **A** i **B**:

$$\hat{\pi}_i = \gamma_1(i) = \frac{\alpha_1(i)\beta_1(i)}{\sum_{j=1}^{N} \alpha_1(j)\beta_1(j)} = \sum_{j=1}^{N} \xi_1(i,j) \tag{16.22}$$

$$\hat{a}_{ij} = \frac{\gamma(i,j)}{\gamma(i)} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1}^{T} \alpha_t(i) \cdot a_{ij} \cdot b_j(\mathrm{O}_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{t=1}^{T} \alpha_t(i) \cdot \beta_t(i)} \tag{16.23}$$

$$\hat{b}_{jk} = \frac{\gamma(i,\Omega_k)}{\gamma(i)} = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot \chi(\mathrm{O}_t \in \Omega_k)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1}^{T} \alpha_t(j) \cdot \beta_t(j) \cdot \chi(\mathrm{O}_t \in \Omega_k)}{\sum_{t=1}^{T} \alpha_t(j) \cdot \beta_t(j)} \tag{16.24}$$

Oznaczenie: $\chi(a \in \Omega)$ to predykat, który zachodzi ($\chi(a \in \Omega)=1$) wtedy, gdy formuła w nawiasie jest spełniona, lub $\chi(.)=0$ – przeciwnym razie.

16.3. Uczenie bez nadzoru

Jak dotąd zajmowaliśmy się uczeniem z nadzorem, tzn. wszystkie próbki uczące były etykietowane przynależnością do klasy. Teraz zajmiemy się problemem grupowania (klasteryzacji) próbek uczących pozbawionych etykiet klas. Jest to problem uczenia bez nadzoru, często też stosowany jako przykład samoorganizacji systemu agentowego.

16.3.1. Klasteryzacja "*k*-średnich"

Jest to prosty algortym klasteryzacji i organizacji danych. Dlatego stanowi on nasz punkt wstępu do tych zagadnień.

Dane

Zakładamy, że istnieje pewna liczba obserwacji o postaci (wektorów cech) c_j (j=1,2 ..., n). Niech spodziewana liczba klas wynosi K.

Obliczenia – algorytm *k-średnich*

1. Inicjalizuj dowolnie K środków klastrów: $\mu^{(i)}$.

2. FOR każda obserwacja c_j DO

   przypisz go do najbliższego klastra, tzn.

   ustaw $\xi(c_j) \leftarrow i$, wtedy gdy $\mathrm{d}(\mu^{(i)}, c_j) = \min_k \mathrm{d}(\mu^{(k)}, c_j)$,

   gdzie $\mathrm{d}(,)$ jest pewną miarą odległości (zwykle miarą Euklidesa).

3. FOR każda klasa DO

   wyznacz wektor średni obserwacji przypisanych do danej klasy

$$\mu^{(i)} = \frac{1}{n^{(i)}} \sum_{j,\xi(c_j)=i} c_j$$

4. REPEAT kroki 2 i 3 zadaną liczbę iteracji.

Podstawowym problemem projektowym dla algorytmu *k*-średnich jest właściwa inicjalizacja środków klas. Najczęściej wykonuje się go wielokrotnie dla różnych wartości początkowych i wybiera najlepszy wynik.

Wadą algorytmu *k*-średnich jest brak gwarancji zbieżności do stanu ustalonego

16.3.2. Klasteryzacja EM ("Expectation Maximization")

EM (ang. expectation-maximization) jest ogólną techniką statystyczną radzenia sobie z brakującą informacją . Możemy go zastosować m.in. w problemie klasteryzacji. Wtedy dostarcza on szacowanie największej wiarygodności (ML) dla mieszaniny rozkładów Gaussa:

$$p(c \mid \theta) = \sum_{j=1}^{K} \alpha^j N(c \mid \mu^j, \Lambda^j) \tag{16.25}$$

gdzie $N(.)$ oznacza funkcję Gaussa gęstości prawdopodobieństwa (tzw. rozkład normalny). Każdy klaster j (=1, ..., K) charakteryzowany jest rozkładem Gaussa o parametrach (μ^j, Λ^j). Te parametry i wagi rozkładów, α^j , są tak dobrane w procesie klasteryzacji EM, aby maksymalizować sumę prawdopodobieństw (16.25) liczoną po wszystkich próbkach uczących.

Poza inicjalizacją, algorytm klasteryzacji EM składa się z dwóch kroków wykonywanych iteracyjnie, aż do spełnienia warunku stopu:

- krok E generuje aktualną "miękką" klasteryzację, tzn. wartość P^i_j oznacza prawdopodobieństwo przynależności do klastra i próbki c_j ;
- krok M jest aktualnym oszacowaniem wszystkich parametrów rozkładu: α^j, μ^j, Λ^i .

Klasteryzacja EM

Dane:

obserwacje (wektory cech) c_j (j=1,2 ..., n), liczba klas wynosi K.

Obliczenia – *klasteryzacja EM*

1. Inicjalizuj parametry: θ ={ α^j, μ^j , Λ^i | *i=1,...,K* }.

REPEAT kroki E i M:

2. Krok E:

FOR każda próbka c_j DO

FOR każda klasa *i* DO

$$P_j^i = \frac{p(c_j \mid \xi(c_j) = i, \theta)\alpha^i}{\sum_i p(c_j \mid \xi(c_j) = i, \theta)\alpha^i} \tag{16.26}$$

3. Krok M:

FOR każda klasa *i* DO

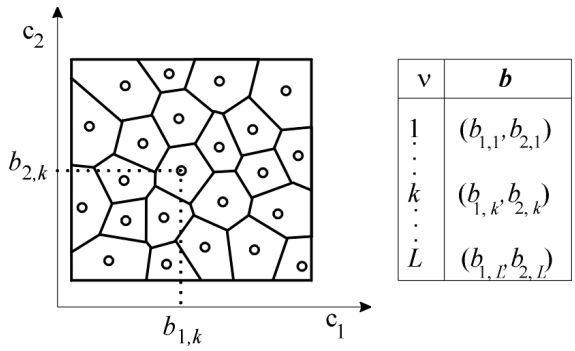
$$\mu^i = \frac{\sum_j c_j P_j^i}{\sum_j P_j^i}\ , \qquad \Lambda^i = \frac{\sum_j (c_j - \mu^i)(c_j - \mu^i)^T P_j^i}{\sum_j P_j^i}\ , \qquad \alpha^i = \frac{\sum_j P_j^i}{\sum_i \sum_j P_j^i} \tag{16.27}$$

UNTIL nie wykonano zadanej z góry liczby iteracji.

16.3.3. Wyznaczanie liczby klastrów

Kwantyzacja wektorowa dla K klas

Algorytmy *kwantyzacji wektorowej* często stosują metodę „*k*-średnich" dla wyznaczenia podziału przestrzeni reprezentacji na podobszary odpowiadające klastrom i ich reprezentowania przez ich wektory średnie. W ten sposób w procesie uczenia wyznaczany jest słownik kodowy złożony z reprezentantów *K* klas. Następnie na etapie kodowania *kwantyzer* zastępuje wektor cech przez indeks reprezentanta w słowniku kodowym.



Rys. 16.4: Idea 2-wymiarowej kwantyzacji wektorowej: klasteryzacja przestrzeni cech i słownik kodowy.

Zazwyczaj kwantyzator rozszerza ideę klasteryzacji *k*-średnich o adaptacyjny warunek stopu. W poniższym algorytmie w każdej iteracji badana jest zbieżność aktualnego błędu kwantyzacji próbek uczących. Jeśli błąd przestaje się istotnie zmniejszać iteracja jest przerywana.

Dane są:

- zbiór wektorów cech – próbek uczących: $\omega = \{c^i \mid i = 1,..., N\}$,
- liczba klas K,
- początkowy słownik $Z^{(0)}$ złożony z wektorów prototypów $Z^{(0)}=\{z^{(0)}_\kappa, \kappa=1,...,K\}$,
- początkowy błąd aproksymacji $\varepsilon^{(0)}$,
- zadany próg Θ dla względnego błędu aproksymacji.

Obliczenia – *kwantyzacja wektorowa k-średnich*:

1. Iteruj kroki (a)-(d) dla $I = 1,2,3, ...$

(a) zaklasyfikuj wektory próbek $\mathrm{c}^i \in \omega$ zgodnie z aktualnym słownikiem kodowym $Z^{(I-1)}$ – wybierając klasę odpowiadającą najmniejszej odległości próbki od prototypu klasy – i wyznacz sumaryczny błąd klasyfikacji

$$\varepsilon^{(I)} = \sum_{i=1}^{N} \min_{\kappa} | \mathbf{c}_i - z_{\kappa}^{(I-1)} | . \tag{16.31}$$

(b) jeśli względna różnica błędów jest poniżej zadanego progu, tzn.

$$\frac{| \varepsilon^{(I)} - \varepsilon^{(I-1)} |}{\varepsilon^{(I-1)}} < \Theta \tag{16.32}$$

to przejdź do kroku KONIEC.

(c) oblicz nowy słownik kodowy $Z^{(I)}$ z prototypami klas $z^{(I)}{}_{\kappa}$:

$$\mathbf{z}_k^{(I)} = \frac{1}{N_k^{(I)}} \sum_{c \in \Omega_k^{(I)}} \mathbf{c} \tag{16.33}$$

(d) ustaw $I = I+1$ i ponów iterację od kroku (a).

(KONIEC) Znaleziony słownik kodowy to $Z^{(I)}$ przy błędzie aproksymacji $\varepsilon^{(I)}$.

Jako wynik działania algorytmu kwantyzacji otrzymuje się zbiór klastrów i przydzielonych do nich reprezentantów (punkty centralne w każdym klastrze, określane mianem centroidów), spełniających następujące warunki:

3. Każdy punkt należy do klastra, którego centroid jest położony najbliżej tego punktu.
4. Każdy centroid jest punktem, który jest najmniej oddalony od wszystkich pozostałych punktów klastra (suma odległości tego punktu od wszystkich pozostałych jest najmniejsza, wg przyjętej miary odległości, np. Euklidesa).

Pełna kwantyzacja wektorowa (z wyznaczaniem liczby klas)

Pełna kwantyzacja polega na nienadzorowanym uczeniu zarówno **liczby** klas jak i **rozkładu** klastrów w przestrzeni cech.

DANE wejściowe:

- zbiór wektorów cech – próbek uczących: $\omega = \{\mathrm{c}_i \mid i = 1,..., N\}$,
- próg dla minimalnego błędu Γ.

OBLICZENIA – pełna kwantyzacja z wyznaczaniem liczby klas

1) Oblicz centroidę całego zbioru próbek ω :

$$\boldsymbol{\mu}^{(0)} = \frac{1}{N} \sum_i \mathbf{c}_i \tag{16.34}$$

2) Ustaw początkowy słownik $Z^{(0)}$ dla $K_0 = 2$ klas i posiadających prototypy:

$$\mathrm{z}_{1,2} = (1 \pm \delta)\, \mu \text{ , gdzie } \delta < 1. \tag{16.35}$$

3) Iteruj kroki (4)-(7) dla $K = K_0$, $K_0 + 1$, $K_0 + 2$, …

4) Wykonaj *kwantyzację wektorową k-średnich* dla $(K, Z^{(0)})$. Uzykujemy nowy słownik $Z^{(I)}$ obarczony błędem $\varepsilon^{(I)}$.

5) IF ($\varepsilon^{(I)} < \Gamma$) THEN STOP.

6) Ustaw nowy słownik $Z^{(0)}$ dla $K+1$ klas o prototypach:

- wybierz klaster K_κ o najwiekszej wariancji próbek tej klasy i podziel klaster na 2 części o centroidach:

$$z_{\kappa,K+\kappa} = (1 \pm \delta)\, z_\kappa \text{ , gdzie } \delta < 1. \tag{16.36}$$

- pozostałe klastry przechodzą bez zmian do nowego zbioru klastrów.

7) Przejdź do (3) i wykonuj następną iterację kroków (4-7).

16.4. Pytania

1. Omówić regułę decyzyjną klasyfikatora Bayesa.
2. Omówić uczenie ML i EM dla klasyfikatora Bayesa.
3. Omówić klasyfikator *k-NN* (klasyfikacja według *k*-sąsiadów).
4. Omówić uczenie modelu HMM metodą Bauma-Welcha.
5. Na czym polega klasteryzacja „*k*-średnich”?
6. Na czym polega klasteryzacja EM?
7. Jaki jest cel i jak realizuje się kwantyzację wektorową?
8. Jak ustalana jest liczba szukanych klas?

16.5. Zadania

Zad. 16.1

Nad przestrzenią cech $\Re^2$ ustalić należy niezbędne rozkłady 4 klas, mając 11 próbek uczących, podanych w tabeli 16-1.

Tab. 13-1

(x,y)	(1;2,5)	(1; 3)	(1;3,5)	(2; 3)	(4; 3)	(2; 2)	(1; 1)	(2; 0)	(3; 1)	(3,5;1,5)	(4;1,5)
Klasa	1	1	1	2	2	3	3	3	3	4	4

Podać rozkłady uzyskane dla klasyfikatorów: stochastycznego klasyfikatora Bayesa (założyć rozkład Gaussa) i klasyfikatora geometrycznego minimalnej odległości,

Podać wyniki klasyfikacji wektora cech (2; 2,5) w obu klasyfikatorach.

Zad. 16.2

Dany jest zbiór 15 próbek uczących o postaci 3-wymiarowych wektorów cech (tab. 13-2). Załóżmy najpierw, że ich przynależność do klasy nie jest znana.

Tab. 16-2

Indeks	0	1	2	*Klasa*
1	16	27	10	1
2	16	27	12	1
3	18	29	8	2
4	17	28	8	2

5	18	29	8	2
6	27	43	5	3
7	31	48	7	3/
8	23	35	7	4
9	22	34	5	5
10	28	44	6	6
11	29	45	6	6
12	27	43	7	7
13	28	44	5	8
14	28	44	7	8
15	28	44	8	8

Dla powyższego zbioru 15 próbek zastosować algorytm pełnej klasteryzacji (wybór liczby klas i kwantyzacja wektorowa) – wykonać jedynie 2 iteracje wewnętrzne w tym algorytmie.

Przedstawić wyniki wykonania tego algorytmu. Z uwagi na całkowite wartości cech mogą wystąpić zerowe wariancje – w takiej sytuacji przyjąć wartość wariancji za równą 0.25.

Podać warunek przy jakim klasteryzacja podanego zbioru cech będzie kontynuowana po zakończeniu 2 iteracji.

Zweryfikować poprawność wyników klasteryzacji porównując liczbę klastrów i przynależność każdej próbki do klastra z jej rzeczywistą klasą, podaną w ostatniej kolumnie powyższej tabelki

Bibliografia

[1] Russell S., Norvig P.: *Artificial Intelligence: A Modern Approach*. New Jersey, Prentice Hall, 2002, 2010.

[2] Flasiński M.: *Wstęp do Sztucznej Inteligencji*. Wydawnictwo Naukowe PWN, Warszawa, 2011.

[3] Rutkowski R.: *Metody i techniki sztucznej inteligencji*. Warszawa, Wydawnictwo Naukowe PWN, 2005.

[4] Duda R.O., Hart P.E., Stork D.G.: *Pattern Classification. Second Edition*. John Wiley, 2000.

[5] Kasprzak W.: *Rozpoznawanie obrazów i sygnałów mowy*. Warszawa, Oficyna Wydawnicza Politechniki Warszawskiej, 2009.