ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2001
Seria: AUTOMATYKA z. 133 Nr kol. 1526

Adam GAŁUSZKA, Andrzej ŚWIERNIAK

WIELOMIANOWY ALGORYTM GENERUJĄCY OPTYMALNY PLAN DLA PROBLEMÓW Z DZIEDZINY „ŚWIATA KLOCKÓW"

Streszczenie. W artykule rozważane są charakterystyczne dla Sztucznej Inteligencji problemy planowania zadań i osiągania celów. Rozważania dotyczące złożoności obliczeniowej umożliwiły zastosowanie strategii zachłannej do wyboru podcelu spośród podcelów wynikających z grafu ograniczeń kolejnościowych. Strategia zachłanna w przypadku dziedziny „świata klocków" prowadzi do znalezienia optymalnego (w sensie najmniejszej liczby operatorów rozwiązujących problem) planu. Przykład wyjaśnia istotę zachłanności wyborów w środowisku tzw. „świata klocków". We wszystkich przypadkach problem planowania zamodelowany został przy użyciu reprezentacji STRIPS.

POLYNOMIAL TIME COMPLETE OPTIMAL ALGORITHM FOR „BLOCK WORLD" PROBLEMS

Summary. In this paper planning and goal persuit problems in artificial intelligence are considered. The analysis of computational complexity allows to apply greedy strategy to choose subgoal which respect precedence constraints graph of the problem. In the case of „block world" environment the greedy strategy leads to finding optimal problem solution. The essence of this approach is explained by the example in block world environment. In all cases planning problems have been modelled using STRIPS representation.

1. Wstęp

W artykule prezentowane jest zastosowanie algorytmu zachłannego (Cormen i inni 1997) do rozwiązania klasycznego problemu planowania zadań robota (Gałuszka 1998, Gałuszka 1998a). Algorytmy zachłanne często znajdują zastosowanie w teorii grafów do optymalnego przeszukiwania ścieżek w grafie (Cormen i inni 1997), np. do rozwiązania problemu minimalnego drzewa rozpinającego, szeregowania zadań (Cohen i inni 1999) lub znajdowania dolnego ograniczenia do rozwiązania problemu komiwojażera (Wilson 1998). Prezentowany algorytm jest wielomianowo złożony. Artykuł zawiera także uzasadnienie faktu, że strategia zachłanna generuje plan rozwiązujący problem w czasie wielomianowym.

Algorytm zachłanny został zaimplementowany w języku PROLOG i jest wykorzystywany do sterowania robotem Irp-6 w laboratorium Instytutu Automatyki (Jaskot 2000). Zadaniem robota jest osiąganie pożądanej sytuacji docelowej w środowisku „świata klocków".

2. Definicja problemu i kryterium optymalności

W przypadku ogólnym zadanie ze „świata klocków" jest reprezentowane poprzez trzy listy <*O, I, G*> (Nilson 1980):

- zbiór operatorów (*O*);
- zbiór predykatów opisujących początkowy stan świata w zadaniu (*I*);
- zbiór predykatów opisujących docelowy stan świata w zadaniu (*G*).

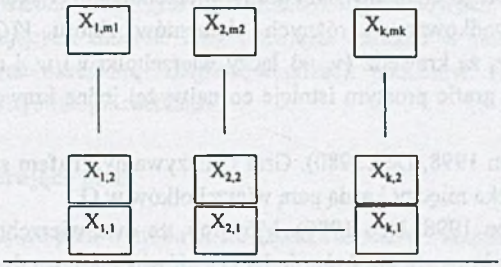
Predykaty opisujące stany początkowy i docelowy powinny opisywać jedną, fizycznie możliwą konfigurację klocków względem siebie. Ponadto opis stanu początkowego powinien być kompletny, tzn. powinien zawierać każdy prawdziwy predykat wynikający z początkowego ułożenia klocków.

Rezultatem działania algorytmu jest zbiór operatorów przekształcających stan początkowy w stan docelowy świata w zadaniu. **Optymalnym** rezultatem jest osiągnięcie stanu docelowego w minimalnej liczbie kroków, tzn. przy użyciu **minimalnej** liczby operatorów. Operatory w tak zdefiniowanym systemie STRIPS składają się z trzech podlist: listy warunków stosowalności *(precondition list)*, listy skreśleń *(delete list)* oraz listy dopisków *(add list)*. Lista warunków stosowalności jest zbiorem predykatów, które muszą być prawdziwe w opisie bieżącego stanu świata, aby móc zastosować dany operator. Lista skreśleń jest zbiorem predykatów, które przestają być prawdziwe po zastosowaniu danego operatora, natomiast lista dopisków to zbiór predykatów, które stają się prawdziwe po zastosowaniu danego operatora. Innymi słowy ostatnie dwie listy opisują efekt zastosowania operatora do bieżącego stanu świata w zadaniu.

Założenie. Można założyć, że istnieją cztery operatory, które pozwalają przekształcać stan świata w zadaniu (Nilson 1980):

1) **pickup(x)** - oznacza podniesienie klocka X ze stołu;
   Lista warunków stosowalności: ONTABLE(X), CLEAR(X), HANDEMPTY
   Lista skreśleń: ONTABLE(X), CLEAR(X), HANDEMPTY
   Lista dopisków: HOLDING(X)
2) **putdown(x)** - oznacza położenie klocka X na stół;
   Lista warunków stosowalności: HOLDING(X)
   Lista skreśleń: HOLDING(X)
   Lista dopisków: ONTABLE(X), CLEAR(X), HANDEMPTY
3) **stack(x,y)** - oznacza położenie klocka X na klocku Y;
   Lista warunków stosowalności: HOLDING(X), CLEAR(Y)
   Lista skreśleń: HOLDING(X), CLEAR(Y)
   Lista dopisków: HANDEMPTY, ON(X,Y), CLEAR(X)
4) **unstack(x,y)** - oznacza ściągnięcie klocka X z klocka Y;
   Lista warunków stosowalności: HANDEMPTY, CLEAR(X), ON(X,Y)
   Lista skreśleń: HANDEMPTY, CLEAR(X), ON(X,Y)
   Lista dopisków: HOLDING(X), CLEAR(Y)

Przypadek ogólny zadania ze „świata klocków", w którym celem jest ułożenie zbioru wież, przedstawia rys. 1. Stan początkowy zadania może być dowolną kombinacją wszystkich klocków występujących w opisie sytuacji docelowej.



- k jest liczbą wież;
- m_i dla i = 1, 2 ... k jest liczbą klocków w i-tej wieży;
- $\sum_{i=1}^{k} m_k = n$ jest liczbą wszystkich klocków w zadaniu.

Rys. 1. Zadanie ze „świata klocków" - stan docelowy
Fig. 1. Goal state in block world environment

Łatwo można sobie wyobrazić, że graf przestrzeni stanów zadania, który opisuje wszystkie możliwe stany zadania pomiędzy stanami początkowym i docelowym, jest bardzo skomplikowany. Praca (Bylander 1994) przedstawia dowód wykazujący, że problem generacji optymalnego planu dla zadania ze "świata klocków" przez przeszukiwanie grafu przestrzeni stanów zadania może być zredukowany w czasie wielomianowym do NP_t złożonego problemu 3-wymiarowego dopasowania (Garey i Johnson 1979).

3. Algorytm zachłanny

W algorytmie zachłannym w każdym kroku jest podejmowany wybór, który wydaje się być najlepszy w danej chwili, a następnie rozwiązywane są podproblemy, które wynikają z podjętej decyzji. Wybory podejmowane w algorytmie zachłannym mogą zależeć od dotychczasowych decyzji, lecz nie mogą być uzależnione od przyszłych wyborów lub od rozwiązań podproblemów. Sprawia to, że w przeciwieństwie do algorytmów opartych na programowaniu dynamicznym, rozwiązujących problem w sposób wstępujący, algorytmy zachłanne znajdują rozwiązanie w sposób zstępujący, podejmując kolejno decyzje zachłanne, stopniowo redukując podproblemy do coraz mniejszych (Cormen i inni 1997).

3.1. Graf ograniczeń kolejnościowych

Analiza sytuacji docelowej problemu planowania przed przystąpieniem do poszukiwania planu wpływa na efektywność algorytmów planowania oraz może w ogóle gwarantować znalezienie planu. Jedną z dróg analizy może być znalezienie *grafu ograniczeń kolejnościowych* dla danej sytuacji docelowej. Poniżej przedstawione są istotne definicje dotyczące struktury grafu, wykorzystywane w dalszym ciągu pracy.

Definicja 1 (Wilson 1998, Deo 1980). **Graf prosty** G składa się z niepustego zbioru skończonego $V(G)$, którego elementy nazywamy wierzchołkami, i skończonego zbioru $E(G)$ różnych par nieuporządkowanych różnych elementów zbioru $V(G)$, które nazywamy krawędziami. Mówimy, że krawędź $\{v, w\}$ łączy wierzchołki v i w, i na ogół oznaczamy ją krócej vw. W każdym grafie prostym istnieje co najwyżej jedna krawędź łącząca daną parę wierzchołków.

Definicja 2 (Wilson 1998, Deo 1980). Graf G nazywamy **grafem spójnym**, jeśli istnieje co najmniej jedna ścieżka między każdą parą wierzchołków w G.

Definicja 3 (Wilson 1998, Deo 1980). Mówimy, że dwa wierzchołki v i w grafu G są **sąsiednie**, jeśli istnieje krawędź vw, która je łączy. Mówimy też wtedy, że wierzchołki v i w są **incydentne** z tą krawędzią.

Definicja 4 (Deo 1980). **Łańcuch** jest to skończony ciąg występujących na przemian wierzchołków i krawędzi, rozpoczynający się i kończący wierzchołkami taki, że każda krawędź jest incydentna z wierzchołkami poprzedzającymi ją i następującymi po niej.

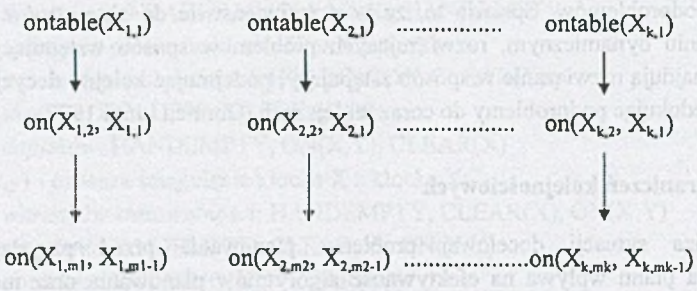
Definicja 5 (Deo 1980). Łańcuch nazywamy **łańcuchem zamkniętym**, jeśli rozpoczyna się i kończy w tym samym wierzchołku.

Definicja 6 (Wilson 1998). Łańcuch zamknięty, w którym żaden z wierzchołków (z wyjątkiem początkowego i końcowego) nie pojawia się więcej niż jeden raz, nazywamy **cyklem**.

Definicja 7 (Wilson 1998). **Lasem** nazywamy graf nie zawierający cykli, a **drzewem** - las spójny.

Forma grafu ograniczeń kolejnościowych przedstawionego na rys. 2 wynika z charakterystycznej formy sytuacji docelowej zadania z rys. 1. Graf ten pokazuje, które z podcelów muszą być wykonane przed innymi, aby osiągnąć pożądany stan docelowy. Graf ten ponadto musi zawierać predykaty ontable(X). Brak rozszerzenia opisu sytuacji docelowej o predykaty ontable(X) powoduje, że stan docelowy zadania staje się nieszeregowalny (Joslin i Roach 1989). Może to uniemożliwić uszeregowanie podcelów w sposób prowadzący do rozwiązania zadania. Sytuacja taka znana jest jako anomalia Sussmana (Joslin i Roach 1989, Barret i Weld 1994).

Wniosek 1. Graf ograniczeń kolejnościowych dla zadań ze "świata klocków" jest lasem, którego składowe spójności są łańcuchami.



Rys. 2. Graf ograniczeń kolejnościowych
Fig. 2. Precedence constraints graph

Należy zauważyć (Gałuszka 1997), że jakakolwiek kolejność realizacji podcelów, która spełnia warunki narzucone przez graf ograniczeń kolejnościowych, prowadzi do generacji operatorów rozwiązujących zadanie. Aby rozwiązać zadanie w mniejszej liczbie kroków, należy jednak wybrać określoną kolejność realizacji podcelów. Kolejność ta może być generowana za pomocą strategii zachłannej.

3.2. Procedura generująca plan

Procedura generująca plan w środowisku „świata klocków" została zaimplementowana za pomocą języka programowania PROLOG. Język ten w zasadniczej części jest podzbiorem języka klauzulowej postaci logiki. Język klauzul składa się z następujących wyrażeń: **termów**, **formuł** i **klauzul**.

Definicja 8 (Szajna i inni 1991). Zbiór **termów** jest zbiorem wyrażeń spełniających następujące warunki:

- stałe są termami,
- zmienne są termami,
- jeżeli *f* jest *n*-argumentowym symbolem funkcyjnym, a *t1, t2, ..., tn* są termami, to *f(t1, t2, ..., tn)* jest również termem.

Definicja 9 (Szajna i inni 1991). Predykat składa się z symbolu predykatowego, po którym następuje ujęty w nawiasy ciąg oddzielonych przecinkami argumentów – dowolnych termów. W predykatach zeroargumentowych nawiasy pomija się. Predykat jest nazwą relacji. Jeżeli *p* jest symbolem *n*-argumentowej relacji ($n \geq 0$), to predykat *p(t1, t2, ..., tn)* jest nazywany **formułą atomową**. **Formuły molekularne** są zbudowane z formuł atomowych połączonych spójnikami logicznymi *and* i *or*.

"Relację przedstawioną predykatem *p(t1, t2, ..., tn)* definiuje się również jako odwzorowanie między każdym *n*-elementowym podzbiorem elementów dziedziny a wartościami logicznymi należącymi do zbioru {*true, false*}, czyli {*prawda, fałsz*}. Obydwa te pojęcia są równoważne, jeśli założyć, że nie wymienionym *n*-elementowym podzbiorem zbioru elementów dziedziny odpowiada wartość logiczna *false*; jest to tzw. założenie o zamkniętym świecie – CWA (*Close World Assumption*). W Prologu przyjmuje się, że baza danych jest przedstawiona przy założeniu CWA" (Szajna i inni 1991).

Definicja 10 (Szajna i inni 1991). **Klauzula warunkowa** jest wyrażeniem o postaci:

B1, B2, ..., Bm if *A1, A2, ..., An* $\quad n \geq 0, m \geq 0,$

w którym "if" jest symbolem logicznym implikacji, natomiast *B1, B2, ..., Bm* oraz *A1, A2, ..., An* są formułami atomowymi (predykatami). Przecinki po lewej stronie klauzuli są interpretowane jako spójniki or, a po prawej – jako spójniki and. Ciąg formuł *B1, B2, ..., Bm* nosi nazwę następnika klauzuli, a ciąg *A1, A2, ..., An* – poprzednika klauzuli.

Formuły *B1, B2, ..., Bm* w następniku tworzą alternatywę konkluzji klauzuli, podczas gdy formuły *A1, A2, ..., An* w poprzedniku tworzą koniunkcje przesłanek (warunków) klauzuli warunkowej. W Prologu korzysta się wyłącznie z **klauzul Horna**.

Definicja 11 (Szajna i inni 1991). **Klauzula Horna** to klauzula warunkowa, dla której m = 0 lub m = 1. Klauzula Horna nosi również nazwę reguły i przyjmuje postać:

B1 if *A1, A2, ..., An.*

Jeśli m = 1 i n = 0, to klauzula warunkowa nosi nazwę **faktu**. Jeśli m = 0 i n = 0, to klauzula nosi nazwę **klauzuli pustej** i oznacza sprzeczność, czyli jest zawsze fałszywa.

Istotą procedury opisanej poniżej jest przetwarzanie dwóch **list**. Pierwsza z nich (nazwana *Stan_biezacy*) zawiera zbiór predykatów modelujących stan bieżący problemu i jest nieuporządkowana, tzn. zbiór predykatów może wystąpić w dowolnej kolejności w liście i jest on traktowany jako koniunkcja predykatów. Druga lista (nazwana *Cele*) zawiera **uszeregowany** zbiór predykatów i operatorów modelujących przetwarzanie celu głównego problemu.

Uwaga 1. Sposób uszeregowania ma istotny wpływ na wielkość przeszukiwanej przestrzeni stanów problemu. Dla operatorów zdefiniowanych w założeniu 3.1 można znaleźć takie uszeregowanie warunków stosowalności, że spełniając je w znalezionej kolejności zawsze spełniona będzie ich koniunkcja (podobnie jak dla celu szeregowalnego). Wynikiem takiego działania jest bardzo ważna (ze względów efektywności algorytmu) własność algorytmu, która pozwala na znalezienie operatora bez konieczności wykonywania operacji nawrotów (tzw. backtrackingu). Technika taka umożliwia zrezygnowanie z przeszukiwania przestrzeni stanów zadania, co zasadniczo wpływa na efektywność algorytmu poszukującego rozwiązania problemu. Mówiąc precyzyjnie, plan rozwiązujący pojedynczy podcel znajdowany jest w czasie wielomianowym, co w pierwszej chwili wydaje się sprzeczne z rezultatem wykazującym NP lub PSPACE –zupełność problemów planowania STRIPS dla złożonych operatorów. W istocie dla środowiska „świata klocków" czas znalezienia planu jest ograniczony od góry jednomianem. Problem ten jest opisany w (Weld 1999).

Definicja 12 (Szajna i inni 1991). **Listą** nazywamy:

a) element określonego typu połączony z listą elementów tego samego typu,

b) listę pustą, tzn. listę nie zawierającą żadnych elementów.

Procedura składa się z dwóch rekurencyjnych reguł (klauzul Horna) przedstawionych na rys. 3.

```
1. generuj_plan(Cele, Stan_biezacy) if
      Cele = [predykat | Lista],
      Cel_zmodyfikowany = [warunki_stosowalnosci_operatora,
                           operator_ktorego_lista_dopiskow_zawiera_predykat |
                           Lista],
                           generuj_plan(Cel zmodyfikowany, Stan_biezacy).

2. generuj_plan(Cele, Stan_biezacy) if
      Cele = [operator | Lista],
      wykonaj(operator),
      zmodyfikuj_stan_biezacy_zgodnie_z_definicja_operatora(operator, Stan_biezacy,
                                                            Nowy_stan_biezacy),
      generuj_plan(Lista, Nowy_stan_biezacy).
```

Rys. 3. Procedura generująca plan
Fig. 3. Plan generation procedure

3.3. Wybory zachłanne

Algorytm zachłanny podejmuje decyzje o wyborze kolejnego podcelu do realizacji według następującego schematu (Gałuszka 1998, Gałuszka 1998a):

- istnieje k różnych podcelów, które można wybrać do realizacji w każdym kroku (k jest liczbą wież); wynika to bezpośrednio z grafu ograniczeń kolejnościowych;
- spośród nich wybierz ten podcel, do którego osiągnięcia potrzeba najmniejszej liczby operatorów;
- jeśli liczba operatorów potrzebnych do osiągnięcia pojedynczego podcelu jest taka sama dla kilku podcelów, wybierz pierwszy z nich.

Liczba operatorów potrzebna do osiągnięcia pojedynczego podcelu (L) może być łatwo obliczona z następującej formuły:

- jeśli wybrany podcel ma postać ontable(X) wtedy:

L = (Σ wszystkich klocków na klocku X w opisie stanu bieżącego * 2) + 2 (1)

- jeśli wybrany podcel ma postać on(X, Y), wtedy:
  - jeśli obydwa elementy X oraz Y znajdują się na tej samej wieży, to:

L = (Σ wszystkich klocków na klocku X w opisie stanu bieżącego * 2) + 2 (2)

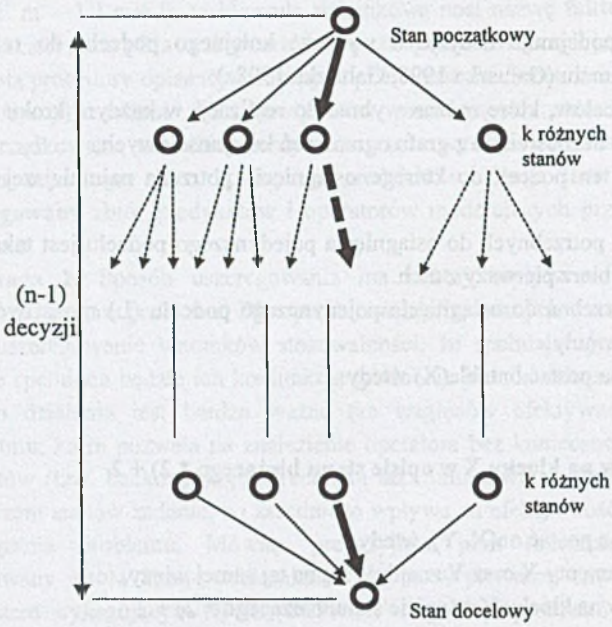
  - jeśli obydwa elementy X oraz Y znajdują się na różnych wieżach, to:

L = (Σ wszystkich klocków na klocku X w opisie stanu bieżącego * 2)
+ (Σ wszystkich klocków na klocku Y w opisie stanu bieżącego * 2) + 2 (3)

- jeśli wybrany podcel jest już osiągnięty w opisie stanu docelowego, wtedy L = 0.

Graf przestrzeni stanów wynikający z różnych możliwych kolejności realizacji podcelów, będących rezultatem grafu ograniczeń kolejnościowych (patrz rys.2), jest przedstawiony na rys. 4. Za pomocą strategii zachłannej wyznaczona jest jedna kolejność podcelów rozwiązująca problem (zaznaczona linią pogrubioną).

Sytuacja docelowa składa się z n podcelów (n jest liczbą klocków). Ze względu na to, że za pomocą decyzji zachłannej wybierany jest zawsze jeden podcel do realizacji, algorytm zachłanny podejmuje (n-1) decyzji zachłannych. Graf przestrzeni stanów dopuszcza k różnych stanów w pierwszym kroku (k jest liczbą wież w sytuacji docelowej), k^2 różnych stanów w drugim kroku itd. Od pewnego momentu liczba dozwolonych stanów zadania zmniejsza się (coraz więcej podcelów jest spełnionych). W przedostatnim kroku liczba dozwolonych stanów znów kurczy się do k (tylko jeden podcel nie jest jeszcze osiągnięty).



Rys. 4. Graf przestrzeni stanów wynikający z grafu ograniczeń kolejnościowych
Fig. 4. State space graph conditioned by precedence constraints graph

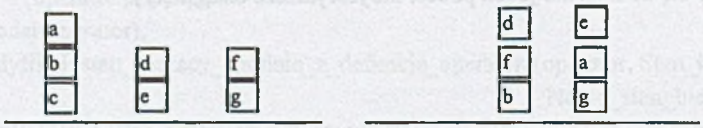
Wniosek 2. Liczba wierzchołków grafu przestrzeni stanów jest więc ograniczona przez $L_{vgps} < k^{(n-1)}$ dla $k > 1$ (w rzeczywistości liczba ta jest mniejsza, ale zachowuje wykładniczą zależność).

Wniosek 3. Liczba wierzchołków grafu przestrzeni stanów dla $k = 1$ jest równa $L_{vgps} = n$. Graf przestrzeni stanów jest w tym przypadku równy grafowi ograniczeń kolejnościowych.

Wykładnicza zależność rozmiaru przestrzeni stanów ograniczonej przez graf ograniczeń kolejnościowych od rozmiaru problemu wyjaśnia wykładniczą efektywność algorytmu znajdującego optymalny plan rozwiązujący problem z wykorzystaniem wielomianowych metod znajdowania najkrótszej ścieżki w grafie. Z liniowej zależności rozmiaru przestrzeni stanów od rozmiaru problemu dla $k = 1$ wynika natomiast, że znalezienie optymalnego planu dla tego przypadku jest problemem wielomianowo złożonym.

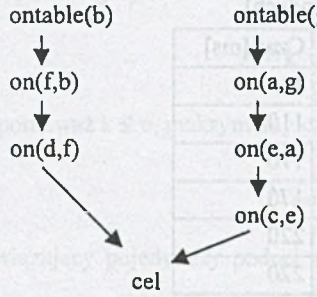
Przykład 1

Sytuacja początkowa i docelowa przedstawiona jest na rysunku.



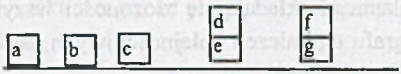
Sytuacja początkowa Sytuacja docelowa

Graf ograniczeń kolejnościowych, odpowiadający sytuacji docelowej z przykładu 1, jest dany poniżej:

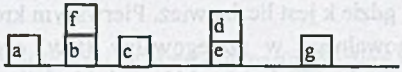


Kolejne wybory zachłanne oraz odpowiadające im dekompozycje stanu bieżącego są przedstawione poniżej:

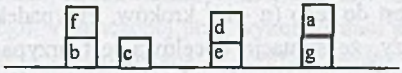
1. ontable(b)∨ontable(g) ⇒ ontable(g) L = 0
2. ontable(b)∨on(a,g) ⇒ ontable(b) L = 4



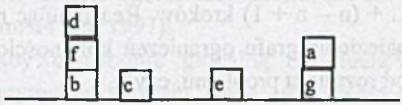
3. on(f,b)∨on(a,g) ⇒ on(f,b) L = 2



4. on(d,f)∨on(a,g) ⇒ on(a,g) L = 2



5. on(d,f)∨on(e,a) ⇒ on(d,f) L = 2



6. on(e,a) L = 2
7. on(c,e) L = 2

W tabeli 1 prezentowana jest szybkość algorytmu zachłannego w zależności od rozmiaru problemu. Symulacje zostały wykonane na komputerze wyposażonym w procesor Pentium 150 MHz, 16 Mb RAM.

Tabela 1
Czas trwania generacji planu
(w milisekundach)

Liczba klocków	Czas [ms]
3	110
4	110
5	170
6	170
7	220
8	220
9	270
10	330
11	330

3.4. Złożoność algorytmu zachłannego

Na złożoność obliczeniową algorytmu zachłannego składają się złożoności wszystkich etapów rozwiązywania problemu: znajdowania grafu ograniczeń kolejnościowych, wyborów zachłannych oraz znajdowania planu do spełnienia wybranego podcelu.

Zbiorem danych wejściowych dla podproblemu znalezienia grafu ograniczeń kolejnościowych jest koniunkcja predykatów $on(X_1, X_2)$. Jeśli w zadaniu jest n klocków, to liczba predykatów $on(X_1, X_2)$ jest równa (n – k), gdzie k jest liczbą wież. Pierwszym krokiem jest operacja przekształcenia celu z nieszeregowalnego w szeregowalny (tzw. operacja linearyzacji celu), czyli znalezienie predykatów $ontable(X)$. Jest to zbiór tych klocków, które nie stoją na żadnym innym, czyli zbiór tych, które występują jako pierwsza zmienna w predykacie $on(X_1, X_2)$ i nie występują jako druga. Wobec tego dla każdego elementu X_1 należy przeszukać wszystkie X_2. Potrzebnych jest do tego $(n - k)^2$ kroków. Przypadek, gdy takie elementy nie zostaną znalezione, świadczy, że sytuacja docelowa jest przypadkiem cyklu i nie posiada fizycznej interpretacji. Operacja szeregowania celu służy też wobec tego jako weryfikacja istnienia rozwiązania problemu. Następnym krokiem jest znalezienie klocków leżących na nich, a więc predykat, którego druga zmienna jest klockiem leżącym na stole itd. Wymaga to (n – k) + (n – k – 1) + ... + (n – n + 1) kroków. Reasumując można stwierdzić, że liczba kroków potrzebnych do znalezienia grafu ograniczeń kolejnościowych jest ograniczona wielomianem drugiego stopnia od rozmiaru problemu, czyli:

$$T_{1\,max} = O(n^2) \tag{4}$$

Przyjmując, że n jest liczbą wszystkich klocków w zadaniu zawsze istnieje n podcelów do uszeregowania (patrz graf ograniczeń kolejnościowych). Łatwo można zauważyć, że maksymalna liczba decyzji o wyborze kolejnego podcelu do realizacji jest równa n-1. Ponadto, w celu ustalenia liczby L, algorytm musi w każdym kroku zliczyć wszystkie klocki występujące w zadaniu. Przyjmując, że k jest liczbą wież występujących w stanie docelowym,

w każdym kroku istnieje k podcelów do wyboru. Po zsumowaniu maksymalna liczba operacji do wykonania przez algorytm zachłanny jest równa:

$$N = k * n * (n-1)$$

Ostatecznie, ponieważ $k \leq n$, maksymalny czas pracy algorytmu zachłannego T_{max} wynosi:

$$T_{2\,max} = O(n^3) \tag{5}$$

Plan rozwiązujący pojedynczy podcel znajdowany jest w czasie wielomianowym (patrz uwaga 3.1).

$$T_{3\,max} = O(n) \tag{6}$$

Złożoność obliczeniowa algorytmu zachłannego wynosi zatem (wykorzystując (4), (5), oraz (6)):

$$T_{max} = O(n^6) \tag{7}$$

Wynika z tego, że algorytm zachłanny w „świecie klocków" jest wielomianowo złożony w czasie.

4. Dyskusja - optymalność algorytmu zachłannego w „świecie klocków"

Algorytmy zachłanne generują rozwiązania optymalne dla specjalnej klasy problemów. Problemy te charakteryzują się właściwością zwaną właściwością wyboru zachłannego (Cormen i inni 1997): globalne optymalne rozwiązanie problemu jest funkcją lokalnych optymalnych rozwiązań. Właściwość ta w zasadniczy sposób odróżnia strategię zachłanną od strategii wyznaczonej przy wykorzystaniu programowania dynamicznego, w którym decyzje są także podejmowane w każdym kroku, natomiast właściwy wybór zależy również od rozwiązań podproblemów. Wybory podejmowane w algorytmie zachłannym mogą zależeć od dotychczasowych decyzji, lecz nie mogą być uzależnione od przyszłych wyborów.

Nie istnieje ogólna metoda służąca do wykazywania optymalności strategii zachłannych (Cormen i inni 1997).

Aby udowodnić globalną optymalność strategii zachłannej wykażemy, że decyzja nieoptymalna lokalnie (inna niż zachłanna) nie prowadzi do polepszenia rozwiązania globalnego.

Twierdzenie 1. Strategia zachłanna wykorzystująca graf ograniczeń kolejnościowych wraz z fizyczną interpretacją sytuacji bieżącej prowadzi do znalezienia optymalnego planu rozwiązującego problem w środowisku „świata klocków" w czasie wielomianowym.

Dowód

Można założyć, że istnieje problem wyboru podcelu do realizacji:

1. on(X1, Y1) lub 2. on(X2, Y2)

Wykorzystując fizyczną interpretację sytuacji docelowej można wyznaczyć liczbę operatorów potrzebnych do realizacji każdego z podcelów:

dla $1 - L = b_1$, dla $2 - L = b_2$.

Przy założeniu, że $b_1 < b_2$, przyjmijmy, że została podjęta decyzja nieoptymalna lokalnie, czyli do realizacji został wybrany podcel 2. Po realizacji tego podcelu na stole będzie położonych (przekształcając zależności (2) i (3)): $(b_2 - 2)/2$ klocków, które znajdowały się na X2 i Y2 oraz powyżej nich. Żaden z nich nie znajduje się na swojej pozycji w sytuacji docelowej, co wynika z grafu ograniczeń kolejnościowych. Wobec tego w przyszłości wymagają one $2*(b_2 - 2)/2$ operatorów, aby spełnić $(b_2 - 2)/2$ podcelów. Wynika z tego, że sytuacja została zdekomponowana w większym stopniu, niż gdyby została podjęta decyzja o realizacji podcelu 1. Decyzja nieoptymalna lokalnie nie prowadzi zatem do polepszenia rozwiązania globalnego (w przyszłości będzie trzeba większej liczby operatorów do powtórnego skomponowania problemu). Wynika z tego, że strategia zachłanna prowadzi do rozwiązania optymalnego globalnie. cnd.

5. Wnioski

Uzyskany wynik nie jest sprzeczny z twierdzeniem, które mówi o NP – zupełności problemu decyzyjnego, do którego można sprowadzić „świat klocków". W istocie algorytm zachłanny nie jest bowiem problemem planowania typu STRIPS. Algorytm wykorzystuje interpretację fizyczną sytuacji docelowej i bieżącej zarówno do wygenerowania grafu ograniczeń kolejnościowych, jak i do poszukiwania liczby operatorów spełniających dany podcel (zależności (1), (2) i (3)). Reprezentacja STRIPS oraz wnioskowanie oparte na definicji operatorów jest wykorzystywane do znalezienia planu spełniającego pojedynczy podcel, co jest problemem wielomianowo złożonym. Dziedzina „świata klocków" jest szczególna wśród problemów typu STRIPS, ponieważ mimo stosunkowo złożonej budowy operatorów istnieje dla tych problemów prosta interpretacja fizyczna, która może być wykorzystana podczas procesu poszukiwania planu. Analiza sytuacji docelowej oraz poszukiwanie interpretacji problemu (np. fizycznej) jest jednym ze sposobów poszukiwania algorytmów efektywnych dla danej dziedziny planowania (Kirsh1995, Weld 1999).

Wadą takiego podejścia jest brak uniwersalności zastosowanej metody (zastosowanie do innych dziedzin).

Niniejsza praca finansowana była z funduszu badań własnych BW 494/Rau1/2001 t.1.

LITERATURA

1. Barret, A.; D.S. Weld. 1994. „Partial-Order Planning: Evaluating Possible Efficiency Gains." Artificial Intelligence, 67: 71-112.
2. Bylander, T. 1994. „The Computational Complexity of Propositional STRIPS Planning." Artificial Intelligence, 69:165-204.