ZESZYTY NAUKOWE POLITECHNIKI ŚLASKIEJ 2006
Seria: AUTOMATYKA z. 145 Nr kol. 1728

Tomasz TATOŃ
Studium Doktoranckie Informatyki, Politechnika Śląska

ZASTOSOWANIE PROGRAMOWANIA OGRANICZENIOWEGO W LOGICE DLA PROBLEMU KONFIGUROWANIA ZESPOŁÓW PRACOWNIKÓW

Streszczenie. Przedstawiono sformułowanie problemu konfiguracji zespołu pracowników. Zaprezentowano rozwiązanie tego problemu za pomocą programowania ograniczeniowego w logice, zaimplementowanego w języku CHIP. Przedyskutowano możliwość wprowadzenia dodatkowych ograniczeń.

CONSTRAINT LOGIC PROGRAMMING FOR A TEAM MANAGEMENT PROBLEM

Summary. The paper starts with a formulation of a team management problem. Next, the problem is solved using constraint logic programming implemented in CHIP. Obtained results were interpreted.

1. Wprowadzenie

Zakłada się, że dany jest zbiór pracowników o różnych kwalifikacjach, różnych wydajnościach oraz różnych kosztach ich pracy w jednostce czasu. Dana jest praca scharakteryzowana całkowitą pracochłonnością i wymagająca pewnej liczby osób o określonych kwalifikacjach. Należy wybrać spośród danego zbioru pracowników zespół wykonawców, który wykona daną pracę nie przekraczając jej całkowitego zadanego kosztu (rozwiązanie dopuszczalne) lub w sposób minimalizujący całkowity koszt pracy (rozwiązanie optymalne). Kombinatoryczna natura problemu sugeruje, że najprościej będzie go można rozwiązać za pomocą programowania w logice z ograniczeniami. Do tego celu użyto oprogramowania *CHIP* (*Constraint Handling In Prolog*) firmy *COSYTEC*

2. Model matematyczny zagadnienia konfigurowania pracowników

Stosuje się następujące oznaczenia:

- Zbiór nazw $P_i = \{P_{i1}, P_{i2}, ..., P_{il}, ..., P_{iL_i}\}$ jest i-tym zespołem nazw wykonawców pracy, jest on podzbiorem wszystkich możliwych nazw wykonawców, oznaczonym symbolem *KADRA*, $P_i \in KADRA$. P_{il} jest nazwą stanowiska l-tego pracownika

z zespołu *i*-tego. Jeżeli w zespole jest więcej pracowników zatrudnionych na tym samym stanowisku, każdy z nich powinien występować z sobie tylko przyporządkowaną nazwą. L_i jest liczbą wszystkich pracowników zespołu P_i.

- Zbiór liczb naturalnych $N_i = \{N_{i1}, N_{i2}, ..., N_{il}, ..., N_{iL_i}\}$ jest zbiorem norm wykonawców z zespołu P_i. Norma *l*-tego pracownika N_{il} z tego zespołu określa możliwą do wykonania przez niego pracę w jednostce czasu, wspólnej dla wszystkich wykonawców z zespołu.
- Zbiór liczb rzeczywistych $K_i = \{K_{i1}, K_{i2}, ..., K_{il}, ..., K_{iL_i}\}$ jest zbiorem kosztów wykonania norm pracy dla wykonawców z zespołu P_i. K_{il} jest kosztem wykonania normatywnej pracy przez l-tego pracownika i-tego zespołu.

Problem można opisać następującymi ograniczeniami:

2.1. Ograniczenie na wykonanie pracy

Wykonawcy muszą wykonać pracę, a więc normy N_{il} wykonawców P_{il} muszą spełnić ograniczenie:

$$\sum_{l=1}^{L_i} N_{il} \geq N_C \tag{1}$$

gdzie N_c – całkowita praca do wykonania, wyrażona w jednostkach normy.

2.2. Rozwiązanie dopuszczalne - ograniczenie na koszty pracy

W przypadku rozwiązania dopuszczalnego suma kosztów K_{il} wykonawców P_{il} nie powinna być większa od pewnych kosztów granicznych:

$$\sum_{l=1}^{L_i} K_{il} \leq K_C \tag{2}$$

gdzie:
K_c – jest granicznym całkowitym kosztem wykonania pracy.

Jeżeli zatem $P_i \in KADRA$ i spełni (1) i (2), to zbiór P_i nazywamy rozwiązaniem dopuszczalnym.

2.3. Rozwiązanie optymalne – minimalizacja kosztów

Zbiór P_i nazywamy optymalnym rozwiązaniem, jeżeli odpowiadające mu koszty K_i minimalizują sumę kosztów dla wszystkich możliwych zbiorów wykonawców ze zbioru *KADRA* przy spełnieniu ograniczenia (1):

$$\min_{P_i \in KADRA} \sum_{l=1}^{L_i} K_{il} \tag{3}$$

3. Przestrzeń rozwiązań

Prezentowane zagadnienie jest kombinatorycznym problemem decyzyjnym dającym się opisać za pomocą zmiennych dyskretnych przyjmujących wartości:

- 0 – osoba nie wchodzi w skład zespołu;
- 1 – osoba wchodzi w skład zespołu.

Charakterystyczna dla omawianego zagadnienia jest skończona przestrzeń dopuszczalnych rozwiązań. Rozmiar tej przestrzeni uzależniony jest od liczby danych wejściowych i postawionych ograniczeń. W szczególnym przypadku przestrzeń ta może być zbiorem pustym, gdy żadne z rozwiązań nie spełni wszystkich postawionych ograniczeń. Problem konfigurowania zespołów pracowników jest, jak większość problemów kombinatorycznych, obarczony eksplozją obliczeniową. Występuje ona przy wzroście rozmiaru problemu powodując bardzo duże zwiększanie się liczby możliwych kombinacji. Liczbę wszystkich możliwych kombinacji problemu można wyznaczyć według wzoru:

$$\sum_{k=1}^{n} \frac{n!}{k!(n-k)!} = 2^n - 1 \qquad (4)$$

gdzie: n – jest liczbą elementów zbioru *KADRA*,
k – jest liczbą wykonawców kolejnych zbiorów wykonawców.

Dla omawianego przykładu (pkt 6) suma wszystkich możliwych kombinacji wynosi 32768 rozwiązań.

4. Programowanie ograniczeniowe w logice

Wyznaczenie rozwiązania optymalnej konfiguracji zespołu pracowników jest w związku z bardzo dużą liczbą możliwych kombinacji procesem niesamowicie absorbującym czas pracy komputera. Nawet przy niewielkich rozmiarach zagadnienia algorytmy mające za zadanie wyznaczyć wszystkie możliwe kombinacje wymagają zbyt dużego nakładu czasu. Zastosowanie programowania ograniczeniowego w logice miało na celu:

- uczynienie programu rozwiązującego problem programem deklaratywnym: odpowiedni opis problemu jest programem rozwiązującym problem;
- możliwość prostego uwzględnienia dodatkowych ograniczeń problemu;
- ograniczenie czasu, jaki aplikacja potrzebowałaby na wyznaczenie optymalnej konfiguracji zespołu pracowników. Jest to możliwe dzięki zastosowaniu szeregu heurystyk (*Looking Ahead*, *Forward Checking*) wbudowanych w algorytm *Branch and Bound* zaimplementowany w kompilatorze języka Chip oraz dzięki efektywnemu analizowaniu ograniczeń modelu. Dlatego aplikacje pisane w oparciu o ten typ programowania doskonale sprawdzają się w rozwiązywaniu zagadnień kombinatorycznych, skracając czas potrzebny na wyznaczenie rozwiązań. Na dodatkowe podkreślenie zasługuje deklaratywny charakter programu napisanego w Chipie: program ten jest odpowiednio sporządzonym opisem problemu, nie zawiera zaś algorytmu rozwiązania problemu.

5. Ograniczenia składu tworzonego zespołu

Oprócz ograniczeń naturalnych problemu opisanych w punkcie (2.1, 2.2, 2.3) wpływ na skład zespołu pracowników można uzyskać poprzez dodatkowe ograniczenia. Mają one na celu odzwierciedlenie preferencji dotyczącej składu zespołu przez osobę tworzącą konfigurację, a także ograniczenie czasu potrzebnego do wyznaczenia rozwiązania. Ograniczenia te można podzielić na kilka typów:

a) Ograniczenia związane z całkowitym kosztem projektu - jeżeli koszt całkowity projektu K_c mieści się w pewnych określonych wartościach granicznych, to w zespole P_i musi znaleźć się pracownik P_{ix}.
b) Ograniczenia związane z całkowitą normą projektu - jeżeli norma całkowita projektu N_c mieści się w pewnych określonych wartościach granicznych, to w zespole P_i musi znaleźć się pracownik P_{ix}.
c) Ograniczenia związane ze składem personalnym zespołu - jeżeli $P_{ix} \in P_i$, to w zespole musi znaleźć się P_{iy}. Ograniczenie to może również przybrać postać odwrotną, a mianowicie jeżeli $P_{ix} \in P_i$, to w zespole nie może znaleźć się P_{iy}.

Podczas konstruowania ograniczeń należy pamiętać o tym, aby nie wykluczały się one wzajemnie, gdyż nie miałyby one sensu, a otrzymany wynik byłby przekłamany.

6. Przykład

Określić skład zespołu pracowników (informatyków) w zależności od przewidywalnej liczby linii kodu źródłowego projektu na podstawie danych (tabela 1) oraz postawionych dodatkowych ograniczeń.

Tabela 1

Dane charakteryzujące pracowników

Nr	Nazwa stanowiska	Koszt	Norma	Nr	Nazwa stanowiska	Koszt	Norma
1	Stażysta I	250	400	9	Specjalista ds. aplikacji	1000	1100
2	Stażysta II	350	600	10	Specjalista ds. dokumentacji	900	200
3	Młodszy programista I	700	800	11	Projektant aplikacji	700	800
4	Młodszy programista II	600	900	12	Analityk aplikacji	850	600
5	Programista I	900	750	13	Tester oprogramowania	600	300
6	Programista II	650	600	14	Koordynator projektu	1200	600
7	Starszy programista I	800	900	15	Kierownik projektu	1400	600
8	Starszy programista II	1100	800				

Ograniczenia:

a) Jeżeli liczba linii kodu >= 1500, w zespole musi się znaleźć Projektant aplikacji.

b) Jeżeli liczba linii kodu >= 3000, w zespole musi się znaleźć Koordynator projektu.

c) Jeżeli liczba linii kodu >= 5000 w zespole musi się znaleźć Kierownik projektu.

d) Jeżeli w zespole znajdzie się stażysta, to musi się w nim znaleźć również Starszy programista II.

Problem:

Wyznaczyć optymalną konfigurację zespołu dla projektu o łącznej przewidywanej liczbie linii kodu nie mniejszej niż 4700 linii.

Rozwiązanie:

Optymalne rozwiązanie obejmuje dwie konfiguracje:

a) Młodszy programista I, Młodszy programista II, Programista II, Specjalista ds. aplikacji, Projektant aplikacji, Koordynator projektu,

b) Stażysta I, Stażysta II, Młodszy programista II, Programista II, Starszy programista II, Projektant aplikacji, Koordynator projektu.

Tabela 2

Otrzymane wyniki

Rozwiązanie	Koszt zespołu	Norma zespołu	Liczba osób w zespole	Numery stanowisk
a)	4850	4800	6	3,4,6,9,11,14
b)	4850	4700	7	1,2,4,6,8,11,14

7. Aplikacja w CHIP'ie rozwiązująca problem konfigurowania zespołów pracowników

Zaprezentowane w tabeli 2 wyniki zostały wygenerowane przez aplikację napisaną w *CHIP*'ie. Bardzo efektywne wyznaczanie rozwiązań aplikacja realizuje dzięki możliwości ukonkretniania domeny zmiennych dyskretnych przez standardowy predykat wbudowany o nazwie *labeling*. Dokonuje on ukonkretniania zmiennych tak długo, dopóki wszystkie ograniczenia są spełniane. W przypadku naruszenia któregoś z ograniczeń (first_fail) przez ukonkretnianą wartość zmiennej, *labeling* powraca do najbliższej ukonkretnionej już zmiennej, dla której istnieje możliwość innego ukonkretnienia. Podstawowy fragment kodu aplikacji realizujący ukonkretnianie zmiennych, a zatem i powstanie optymalnej konfiguracji zespołu pracowników został przedstawiony poniżej.

```
wyznacz:-
min_norma_zespolu(Minimalna_norma),
nr_stanowisk(Numery),
lista_kosztow(Koszty),
```

```
lista_norm(Normy),
lista_decyzji(Decyzje),
assert(konfiguracja_najtansza(50000,50000,[])),
assert(temp(50000)),
Decyzje::0..1,
labeling(Decyzje,0,first_fail,indomain),
tworz_kofiguracje(Numery,Koszty,Normy,Decyzje,[],[],[],Minimalna_norma),fail.

wyznacz.

tworz_ kofiguracje([G1|O1],[G2|O2],[G3|O3],[G4|O4],L,L1,L2,Minimalna_norma):-
G is G1*G4,
GN is G2*G4,
GW is G3*G4,
tworz_pare(O1,O2,O3,O4,[G|L],[GN|L1],[GW|L2],Minimalna_norma).

tworz_ kofiguracje([],[],[],[],L,L1,L2,Minimalna_norma):-
suma(Koszt_zespolu,L1),
suma(Norma_zespolu,L2),
Norma_zespolu >= Minimalna_norma,
sprawdz_ograniczenia_norm(Koszt_zespolu,Norma_zespolu,L),
sprawdz_ograniczenia_osobowe(Koszt_zespolu,Norma_zespolu,L).
```

W omawianym fragmencie kodu aplikacji ukonkretnianiu ulegają zmienne z domeny o nazwie *„Decyzje"*, przyjmując wartości 0 lub 1. Ukonkretnianie to odbywa się za pomocą mnożenia poszczególnych elementów ze zbioru *KADRA* przez ukonkretnianą wartość decyzji tworząc w ten sposób zbiór P_i. W podobny sposób powstają listy zawierające odpowiednio zbiory K_i oraz N_i. Ograniczenia opisane w pkt 5 zostały podzielone na trzy grupy. Implementacja poszczególnych ograniczeń w omawianej aplikacji wygląda następująco:

a) Ograniczenia związane z całkowitym kosztem projektu:

```
sprawdz_ograniczenia_kosztu(Koszt_zespolu,Norma_zespolu,Lista):-
lista_ograniczen_kosztu(G,O),
Koszt_zespolu #>=G,
nalezy_do_listy(O,Lista).
```

b) Ograniczenia związane z całkowitą normą

```
sprawdz_ograniczenia_norm(Koszt_zespolu,Norma_zespolu,Lista):-
lista_ograniczen_norm_wieksze(G,O),
Norma_zespolu #>=G,
nalezy_do_listy(O,Lista).
```

c) Ograniczenia związane ze składem personalnym zespołu

```
sprawdz_ograniczenia_osobowe(Koszt_zespolu,Norma_zespolu,Lista):-
lista_ograniczen_personalnych(G,O),
 nalezy_do_listy(G,Lista),
not(nalezy_do_listy(O,Lista)),
!.
```

sprawdz_ograniczenia_osobowe(Koszt_zespolu,Norma_zespolu,Lista):-
wyznacz_min_kosztu(Koszt_zespolu,Norma_zespolu,Lista).

Zbudowane w ten sposób ograniczenia są bardzo efektywne i uniwersalne w zapisie. Wszystkie dane ograniczające konfiguracje można zaimplementować do aplikacji w postaci list np.: [[[2,1],8]]. Zapis taki mówi, że jeżeli w zespole znajdzie się pracownik o numerach 2 i 1 (Stażysta I i Stażysta II), to musi się w nim znaleźć również pracownik nr 8 (Starszy programista II) – ograniczenie d) w pkt 6.

Aby móc określić czasochłonność aplikacji, zastosowano standardowy predykat o nazwie utime(). Pozwala on na pomiar czasu działania konkretnych elementów składowych aplikacji. Po przeprowadzonym wielokrotnym teście uruchomienia aplikacji można określić czas, jaki aplikacja potrzebuje na wygenerowanie optymalnego wyniku. Przy liczbie ograniczeń równej 4 i liczbie stanowisk pracowników równej 15 oscylował on na poziomie około 2200 ms. Test został przeprowadzony na komputerze z procesorem Intel Celeron 1.5M, oraz z 512 Mb pamięci RAM. Biorąc pod uwagę liczbę możliwych do utworzenia kombinacji (32768), wynik ten jest bardzo dobry.

8. Wnioski

Programowanie ograniczeniowe w logice z powodzeniem można zastosować do modelowania i rozwiązywania zagadnień konfigurowania zespołów pracowników. Kompilator programu *CHIP v5.2* dzięki analizie problemu i ograniczeń oraz zależności logicznych między nimi zawartych pozwala na dość znaczne ograniczenie czasu potrzebnego na wyznaczenie rozwiązań. Dzięki deklaratywności programu napisanego w konwencji ograniczeniowej eliminujemy konieczności tworzenia algorytmu rozwiązań. Opisany przykład konfigurowania zespołu pracowników jest jednym z wielu analizowanych przeze mnie. Wnioski nasuwające się z otrzymanych wyników są następujące:

a) liczba możliwych kombinacji, a zatem i decyzji nawet przy niewielkiej liczbie danych może być bardzo duża,
b) liczbę rozwiązań można ograniczać za pomocą dodatkowych ograniczeń stawianych podczas konfigurowania zespołu,
c) wprowadzenie nawet niewielkiej zmiany w jednym z ograniczeń może spowodować bardzo dużą zmianę w otrzymanych wynikach,
d) w programowaniu ograniczeniowym zwiększanie liczby ograniczeń powoduje zmniejszanie się czasu potrzebnego na wyznaczenie końcowych rozwiązań,
e) rozwiązań optymalnych o jednakowych kosztach może być więcej niż jedno,
f) stworzenie zespołu o mniejszej liczbie pracowników może owocować większą liczbą linii kodu – patrz tabela 2.

Omówiony w pkt 6 program można zastosować jako pomoc w podejmowaniu decyzji odnośnie składu personalnego zespołu pracowników. Duża i prosta konfigurowalność oraz szybkie działanie aplikacji sprawiają, że można ją zaimplementować w bardzo wielu dziedzinach życia związanych z optymalizacją parametru, na przykład kosztu przy postawionych dodatkowych ograniczeniach.

W [1] można znaleźć informacje o modelu COCOMO II umożliwiającym estymację kosztu projektu, lecz nie pozwalającym na stawianie dodatkowych ograniczeń dotyczących personalnego składu zespołu i preferencji osoby tworzącej zespół. Proponowane przeze mnie rozwiązanie problemu pozwala na tworzenie ograniczeń komplementarnych i substytucyjnych związanych z obecnością lub nieobecnością danych pracowników w zespole.

LITERATURA

1. Boehm W.B.: (2002) Software cost estimation with COCOMO II. http://csse.usc.edu/research/COCOMOII.
2. Lipski W.:Kombinatoryka dla programistów. WNT, Warszawa 2004.
3. Niederliński A.: Constraint logic programming - from Prolog to Chip, Proceedings of the PDC'99 Workshop on Constraint Programming for Decision and Control. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 1999, s. 27-34.
4. Niederliński A.: Making Backtracking and Branch-and-Bound More Effective. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 2001, s. 47-52.
5. Niederliński A.: A Tutorial CLP Example In CHIP. Proceedings of the CPDC'2003 Workshop on Constaint Programming for Decision and Control. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 2003, s. 31-34.
6. Sysło M.M., Deo N., Kowalik J.S.: Algorytmy optymalizacji dyskretnej. Wydawnictwo Naukowe PWN, Warszawa 1995.
7. Tatoń T.: Konfigurowanie zespołów ludzkich. Praca dyplomowa magisterska, Wyższa Szkoła Biznesu, Dąbrowa Górnicza 2005.

Podziękowania
Autor chciałby podziękować Panu Prof. A. Niederlińskiemu za pomoc przy opracowaniu tego artykułu.

Recenzent: Prof. dr hab. inż. Zbigniew Banaszak

Abstract

The paper presents a Constraint Logic Programming solution to a team management problem: it is concerned with assigning specialists from a known set of specialists with different qualifications, different job-related efficiency and different wages to a job-oriented team. The assignment should be done in such a way, that the team does the job either staying within a prescribed overall cost limit or minimizing the overall cost, while satisfying constraints relating the job size to the qualification, efficiency and wages of assigned specialists. A Constraint Logic Program in *CHIP v. 5.2* is presented for solving the team management problem. The program is used to derive a solution to a team management example. Attention is drawn to the easy with which additional constraint may be incorporated into the program.