ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2006
Seria: AUTOMATYKA z. 144 Nr kol. 1727

Rafał SZKLARCZYK
Politechnika Śląska

MDMGVRP - NOWY WARIANT PROBLEMU MARSZRUTYZACJI POJAZDÓW. CZĘŚĆ 2 - ROZWIĄZANIE METODĄ CLP

Streszczenie. W artykule przedstawiono dwa programy do rozwiązywania MDMGVRP, problemu opisanego w artykule [1]. Artykuł niniejszy zawiera opis programu, krótko scharakteryzowane różnice pomiędzy dwoma użytymi narzędziami: CHIP'em oraz GNU Prologiem. Na koniec zamieszczono przykład danych i rozwiązania.

MDMGVRP - A NEW VARIANT OF VEHICLE ROUTING PROBLEM. PART 2 - THE CLP SOLUTION

Summary. The paper presents two CLP programs solving MDMGVRP, described in the paper [1]. The two programs are in CHIP and GNU Prolog. The programs are compared and their main differences highlighted. Finally, they are both applied to a representative MDMGVRP problem.

1. Wstęp

CLP jest skrótem pochodzącym od angielskiego *constraint logic programming*, czyli programowania w logice z ograniczeniami. Podejście CLP jest stosunkowo nowe i wciąż mało popularne. CLP polega na przeglądzie przestrzeni rozwiązań z zastosowaniem metod ograniczenia zbioru poszukiwań typu "branch and bound" z dodatkową propagacją ograniczeń *(ang. constraints)*. Propagacja ta redukuje - w większym lub mniejszym stopniu - domeny zmiennych dyskretnych, czyli zbiory możliwych wartości przyjmowanych przez zmienne. W ten sposób uzyskujemy znaczną redukcję czasu poszukiwań rozwiązania. CLP umożliwia uzyskanie rozwiązania optymalnego. Jeśli zatrzymamy poszukiwania przed znalezieniem optimum, narzędzia CLP zwracają najlepsze rozwiązanie znalezione przed zatrzymaniem. Efektywność metody zależy w sposób krytyczny od umiejętności zmniejszenia rozmiarów drzewa poszukiwań na drodze właściwej propagacji ograniczeń problemu.

Do najpopularniejszych narzędzi CLP należy CHIP. Innym narzędziem jest GNU Prolog. CHIP jest narzędziem komercyjnym francuskiej firmy COSYTEC

[3, 1997]. GNU Prolog jest narzędziem należącym do rodziny oprogramowania *opensource* i jest dystrybuowany za darmo na zasadach licencji GPL. Narzędzie to zostało stworzone głównie przez Daniela Diaza w 2002 r [2, 2002]. Do dyskusji wybrano te narzędzia z uwagi na fakt, że jedno z nich należy do grupy narzędzi komercyjnych, a drugie jest oprogramowaniem darmowym rozwiniętym przez środowisko akademickie. Innym powodem jest subiektywny wybór autora niniejszego artykułu. Nie są to jedyne narzędzia CLP. Warto zwrócić uwagę na Eclipse. Jest to produkt zbliżony co CHIP. Innym przykładem jest Ciao Prolog o modułowej budowie i dużej gamie dostępnych rozszerzeń (między innymi CLP).

Składnie rozważanych narzędzi zaczerpnięto z języka Prolog, obydwa narzędzia stanowią jego dialekty.

W CHIP'ie mamy do dyspozycji szerszy wachlarz wbudowanych predykatów, które pozwalają na wprowadzenie warunków umożliwiających rozwiązywanie skomplikowanych problemów kombinatorycznych.

GNU Prolog pod tym względem jest znacznie uboższy. Oferuje jednak możliwość dołączania predykatów napisanych samodzielnie. Pakiet instalacyjny zawiera przykład w języku C. Dodatkowo GNU Prolog pozwala na tworzenie nieliniowych ograniczeń. Przy rozwiązywaniu problemu MDMGVRP właściwość ta okazała się ważna.

Wreszcie, w przypadku gdy istnieje więcej niż jedno rozwiązanie optymalne, GNU Prolog oferuje możliwość obejrzenia wszystkich znalezionych rozwiązań, podczas gdy CHIP wskazuje tylko pierwsze znalezione. Uzyskanie pozostałych w CHIP'ie wymaga tworzenia dodatkowego kodu.

2. Konstrukcja programu

Program CLP do rozwiązywania MDMGVRP zbudowany jest według następującego schematu:

1. Dane wejściowe i ograniczenia
2. Część zasadnicza programu
3. Predykaty pomocnicze
4. Definicja macierzy odległości

W pierwszej części definiowane są kolejno ładowności samochodów oraz jednostkowe koszty przejechania odległości dla wszystkich samochodów, które są dane do rozwiązania problemu. Otrzymujemy:

```
L1=15, C1=3, MS1=4,
```

gdzie L oznacza ładowność, C – jednostkowy koszt przejechania odległości, MS – miasto startowe samochodu. Liczby, które znajdują się za literami, to indeksy. W powyższym oznaczają numer samochodu, dla którego definiowane są parametry.

Przyjęto taką zasadę, że jeśli jest więcej niż jeden indeks, pierwsza cyfra oznacza samochód, druga numer asortymentu, trzecia miasto.

Dalej zdefiniowano stany magazynowe:

```
S=[S1,S2,S3,S4],
SA1=[SA11,SA12,SA13,SA14],
SA2=[SA21,SA22,SA23,SA24],
[...]
S1#=SA11+SA21+SA31+SA41+SA51+SA61,
S2#=SA12+SA22+SA32+SA42+SA52+SA62,
[...]
SA1=[0,0,0,7],
SA2=[0,0,4,2],
```

Zmienne $S1, S2...$ oznaczają sumaryczne stany magazynowe w poszczególnych miastach. $SA21$ oznacza stan magazynowy pojemników z asortymentem 2 w mieście 1.

W dalszej części programu zdefiniowano zapotrzebowania na poszczególne asortymenty w poszczególnych miastach:

```
D=[D1,D2,D3,D4],
DA1=[DA11,DA12,DA13,DA14],
DA2=[DA21,DA22,DA23,DA24].
```

Oznaczenia są analogiczne do oznaczeń stanów magazynowych.

Ostatnią daną w problemie są odległości między miastami. W opisywanym programie definicja odległości znajduje się pod koniec kodu. Sposób zapisu to predykat `odl`. Wygląda to następująco:

```
odl(_,0,0).
odl(1,1,0).
odl(1,2,1).
odl(1,3,3).
[...]
```

Predykat `odl` ma trzy parametry. Pierwszy to miasto początku przejazdu, drugi to miasto docelowe, trzeci to odległość do przejechania między nimi.

Pierwsza linia powyższego kodu staje się zrozumiała dopiero po wyjaśnieniu zapisu szukanej trasy samochodu. Linia ta pozwala na prawidłowe przeliczenie funkcji celu w przypadku, gdy samochód stoi.

Druga linia oznacza, że z miasta 1 do 1 odległość wynosi 0. Trzecia zacytowana linia oznacza, że odległość z miasta 1 do miasta 2 wynosi 1.

Po definicji zapotrzebowań następują definicje zmiennych potrzebnych do określenia rozwiązania, to jest załadunków, rozładunków oraz wektorów tras.

Definicje poprzeplatane są ograniczeniami (ang. constraints). Ograniczenia te pozwalają wykluczyć przypadki niedopuszczalne z punktu widzenia modelu matematycznego [1]. Przykładem takiego ograniczenia jest:

```
ZA123#=<SA23.
```

Oznacza to, że załadunek asortymentu numer 2 w mieście numer 3 do samochodu numer 1 jest mniejszy lub równy stanowi magazynowemu asortymentu 2 w mieście 3.

Jednym z ważniejszych ograniczeń jest:

```
R12*D2#=R12*R12.
```

Powyższe określa, że rozładunek samochodu 1 w mieście 2 jest równy zapotrzebowaniu w tym mieście lub, że jest równy 0. Takiego zapisu można użyć w GNU Prologu, który akceptuje w warunkach iloczyny zmiennych decyzyjnych. W CHIP'ie trzeba zapisać to inaczej. Na przykład:

```
if (R12#\=D2) then (R12#=0),
```

a więc, jeśli rozładunek jest różny od zapotrzebowania, to rozładunek jest równy 0. Umieszczenie takich warunków gwarantuje, że zapotrzebowania z danego miasta będą zaspokojone przez jeden samochód, co jest warunkiem modelu.

Dalej w programie zdefiniowane są trasy. Trasa w programie to wektor T z indeksem określającym numer samochodu o wymiarze równym liczbie miast. Inaczej niż ogólna postać przedstawiona w modelu za pomocą macierzy X, która posiada tyle wierszy i tyle kolumn, ile wynosi liczba miast. Jeśli jej element jest równy 1, to oznacza, że dany samochód podróżuje z miasta o numerze równym wierszowi macierzy do miasta o numerze równym kolumnie macierzy.

Przykładowo macierzy X:

$$X = \begin{bmatrix} 0, & 1, & 0, & 0 \\ 0, & 0, & 0, & 1 \\ 1, & 0, & 0, & 0 \\ 0, & 0, & 1, & 0 \end{bmatrix}$$

odpowiada wektor T:

$$T = [2, 4, 1, 3].$$

Po zdefiniowaniu tras znajduje się główna część programu:

```
statistics(real_time,[Czas_s,_]),
fd_minimize((
  fd_labeling(RA11),  fd_labeling(RA12),  fd_labeling(RA21),
  fd_labeling(RA22),

  fd_labeling(ZA11),  fd_labeling(ZA12),  fd_labeling(ZA21),
```

```
  fd_labeling(ZA22),

  niezaladowane_stoja(Z1,T1),  niezaladowane_stoja(Z2,T2),
  jedz_gdzie_trzeba(Z1,R1,T1,MS1),
  jedz_gdzie_trzeba(Z2,R2,T2,MS2),

  fd_labeling(T1),  fd_labeling(T2),

  /* Sprawdzanie poprawności trasy */
    /* Ciągłość trasy, bilans i przekroczenie ładowności */
    sprawdz_przeladowanie(MS1,T1,Z1,R1,0,L1),
      bilans(MS1,T1,ZA11,RA11,0), bilans(MS1,T1,ZA12,RA12,0),

    sprawdz_przeladowanie(MS2,T2,Z2,R2,0,L2),
      bilans(MS2,T2,ZA21,RA21,0), bilans(MS2,T2,ZA22,RA22,0),

  /* Liczenie funkcji celu */
  dlugosc_trasy(T1,1,DT1),  dlugosc_trasy(T2,1,DT2),
  CEL is (DT1*C1)+(DT2*C2),

statistics(real_time,[Czas_1,_]),

  write('Cel'),nl,
  write(CEL),nl,
  write('EUREKA!'),nl,
  write('Zaladunki'),nl,
  write('Z1='),write(Z1),write(', ZA11='),write(ZA11),
  write(', ZA12='),  write(ZA12),nl,write('Z2='),
  write(Z2),write(', ZA21='),write(ZA21),
  write(', ZA22='),write(ZA22),nl,
  write('Rozladunki'),nl,
  write('R1='),write(R1),write(', RA11='),write(RA11),
  write(', RA12='),write(RA12),nl, write('R2='),
  write(R2),write(', RA21='),write(RA21),
  write(', RA22='),write(RA22),nl,
  write('Trasy'),nl,
  write('T1='),write(T1),write(', T2='),write(T2),nl,
  write(''),nl,
  Czas is Czas_1-Czas_s,
  write('Czas liczenia='),write(Czas),write(' ms'),nl,
  write(''),nl
  ),
  CEL
),
```

W GNU Prologu zasadniczą rolę odgrywa predykat `fd_minimize`, który wykonuje znajdujący się wewnątrz kod jako pierwszy parametr tak długo, aż wartość zmiennej podanej jako drugi parametr (w powyższym przypadku zmiennej `CEL`) osiągnie minimum.

W przedstawionym programie obliczenia dokonywane są dla dwóch samochodów i dwóch asortymentów. Na początku dokonywane jest ukonkretnienie załadunków i rozładunków za pomocą predykatu `fd_labeling`. Dalej nakładane są dodatkowe warunki w postaci predykatów `niezaladowane_stoja` i `jedz_gdzie_trzeba`. W momencie gdy znane są już załadunki i rozładunki, możemy wymusić, by samochód, który nie uczestniczy w transportach, załadunki i rozładunki są równe 0, miał zerową trasę. To właśnie zapewnia predykat `niezaladowane_stoja`. Dalej, gdy wiemy, że samochód ładuje i rozładowuje w pewnych miastach, możemy wymusić, by wspomniane miasta znalazły się na trasie. Do tego służy predykat `jedz_gdzie_trzeba`.

W kolejnym kroku ukonkretniane są trasy samochodów i dalej sprawdzane jest, czy trasa jest ciągła, to znaczy czy wektor T nie zawiera kilku niezależnych obiegów na przykład:

$$T = [2, 1, 0, 5, 4]$$

oraz czy w każdym momencie trasy samochód nie został przeładowany (ładunek większy niż ładowność samochodu) lub czy nie wyładowano więcej niż w danym momencie znajdowało się na samochodzie. Te czynności wykonuje predykat `sprawdz_przeladowanie`, kolejne predykaty `bilans` sprawdzają czy nie nastąpiło wyładowanie większej ilości pojemników z poszczególnymi asortymentami niż znajdującej się na samochodzie w każdym z miast trasy. Na koniec wyświetlane jest rozwiązanie. To daje nam możliwość podglądu kolejnych najlepszych rozwiązań dopuszczalnych. W CHIP'ie nie trzeba zamieszczać w programie instrukcji `write`, system sam wyświetla na ekranie rozwiązania dopuszczalne otrzymywane przed rozwiązaniem optymalnym.

3. Przykład

Rozważmy przykład MDMGVRP, gdzie dane są dwa samochody, dwa rodzaje asortymentów, 6 miast. W dwóch z nich znajdują się magazyny, a w czterech klienci. Stany magazynowe dla poszczególnych asortymentów określone są następująco:

```
SA1=[0,0,0,7,2,0],
SA2=[0,0,0,4,0,0].
```

Zamówienia zapisane są poniżej:

```
DA1=[2,4,1,0,0,0],
DA2=[1,2,0,0,0,0].
```

Samochody startują z miasta 6, w którym nie ma ani magazynu, ani klienta, tylko baza transportowa. Parametry samochodów wyglądają następująco:

```
L1=10, C1=3, MS1=6,
L2=6, C2=1, MS2=6,
```

gdzie L oznacza ładowność, C – koszt przejechania jednostki odległości, MS to miejsce startu samochodu.

Macierz odległości dla danego przykładu wygląda następująco:

$$O = \begin{bmatrix} 0, & 1, & 3, & 2, & 4, & 5 \\ 1, & 0, & 1, & 2, & 3, & 4 \\ 3, & 1, & 0, & 1, & 3, & 3 \\ 2, & 2, & 1, & 0, & 1, & 2 \\ 4, & 3, & 3, & 1, & 0, & 1 \\ 5, & 4, & 3, & 2, & 1, & 0 \end{bmatrix}.$$

GNU Prolog na komputerze z architekturą Intela (Celeron 1.4GHz, 256 MB ram, system operacyjny MS Windows XP Home) rozwiązuje zadanie w czasie 32397 milisekund i podaje rozwiązanie:

```
Cel
15
EUREKA! Rozwiazanie ostateczne:
Zaladunki
Z1=[0,0,0,8,2,0], ZA11=[0,0,0,5,2,0], ZA12=[0,0,0,3,0,0]
Z2=[0,0,0,0,0,0], ZA21=[0,0,0,0,0,0], ZA22=[0,0,0,0,0,0]
Rozladunki
R1=[3,6,1,0,0,0], RA11=[2,4,1,0,0,0], RA12=[1,2,0,0,0,0]
R2=[0,0,0,0,0,0], RA21=[0,0,0,0,0,0], RA22=[0,0,0,0,0,0]
Trasy
T1=[0,1,2,3,4,5], T2=[0,0,0,0,0,0]
Czas liczenia=32397 ms
```

Dodatkowo wyświetla informację, że nie jest to jedyne rozwiązanie i umożliwia wyświetlenie pozostałych. Na przykład:

```
Cel
15
EUREKA! Rozwiazanie ostateczne:
Zaladunki
Z1=[0,0,0,10,0,0], ZA11=[0,0,0,7,0,0], ZA12=[0,0,0,3,0,0]
Z2=[0,0,0,0,0,0], ZA21=[0,0,0,0,0,0], ZA22=[0,0,0,0,0,0]
Rozladunki
R1=[3,6,1,0,0,0], RA11=[2,4,1,0,0,0], RA12=[1,2,0,0,0,0]
R2=[0,0,0,0,0,0], RA21=[0,0,0,0,0,0], RA22=[0,0,0,0,0,0]
```

```
Trasy
T1=[0,1,2,3,0,4], T2=[0,0,0,0,0,0]
Czas liczenia=42712 ms
```

CHIP to samo rozwiązanie daje po czasie 38505 milisekund, z tym, że podaje tylko pierwsze rozwiązanie optymalne.

LITERATURA

1. Szklarczyk R.: MDMGVRP - nowy wariant problemu marszrutyzacji pojazdów. Część 1 - Sformułowanie problemu. XV Krajowa Konferencja Automatyzacji Procesów Dyskretnych, Zakopane 2006.
2. Diaz D.: Dokumentacja programu GNU Prolog. Paryż 2002.
3. COSYTEC SA. Dokumentacja programu CHIP. Orsay Cedex 1997.

Podziękowania

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

Recenzent: Prof. dr hab. inż. Zbigniew Banaszak

Abstract

The paper presents two CLP (Constraint Logic Programing) programs for solving the MDMGVRP problem defined in the accompaning paper [1]. CLP an exact method for solving combinatorial problems. Its practical effectiveness depends largely upon our ability to introduce such sets of constraints, that prune large parts of the search tree as compared with the "branch and bound" bruteforce methods.

Two CLP solver tools are discused. One is COSYTEC CHIP and the other GNU Prolog. Both tools have been used to formulate CLP programs for the MDMGVRP problem. The program structure has been discused. It consist of 4 main sections: 1. Entry data and constraints, 2. Main (computing) part, 3. Additional predykates, 4. Distance matrix definition.

A demonstration MDMGVRP problem has been solved using both CHIP and GNU Prolog programs.