STUDIA INFORMATICA 2000
Volume 21 Number 3 (41)

Marcin CIURA
Politechnika Śląska, Instytut Informatyki

ROZWIĄZYWANIE PROBLEMU DOSTAWY ZA POMOCĄ ALGORYTMU GENETYCZNEGO

Streszczenie. W pracy przedstawiono algorytm genetyczny rozwiązywania zadania dostawy, oparty na algorytmie podziału zbioru, opisanym przez Chu i Beasleya [2]. Działa on w czasie zbliżonym do innych algorytmów heurystycznych [3,4,7,8], dając średnio lepsze od nich rozwiązania.

SOLVING A DELIVERY PROBLEM WITH A GENETIC ALGORITHM

Summary. This paper presents a genetic algorithm for solving a delivery problem, based upon an algorithm for solving the set partitioning problem, developed by Chu and Beasley [2]. Being as fast as other heuristic algorithms [3,4,7,8], it produces better solutions on the average.

1. Wprowadzenie

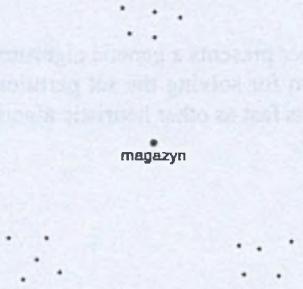
W zadaniu dostawy (ang. *delivery problem*) dana jest skończona przestrzeń metryczna, czyli zbiór punktów i odległości między nimi. Punktami tej przestrzeni są *odbiorcy* i *magazyn*. *Trasą*, do której należą odbiorcy $o_1,...,o_k$, nazywamy k-elementowy ciąg $(o_1,...,o_k)$. *Długością trasy* nazywamy sumę odległości między parami odbiorców $(o_1,o_2),...,(o_{k-1},o_k)$, między magazynem a odbiorcą o_1, i między magazynem a odbiorcą o_k.

Należy znaleźć zbiór tras, do których należą wszyscy odbiorcy, mający jak najmniejszą sumaryczną długość. Korzystając z definicji metryki, można udowodnić, że istnieje rozwiązanie o minimalnej sumarycznej długości, w którym nie ma odbiorców należących do więcej niż jednej trasy. Dlatego w praktyce rozwiązuje się zadanie sformułowane mniej ogólnie: poszukiwanie takiego zbioru tras, w którym każdy odbiorca należy do dokładnie jednej trasy.

W zadaniu dostawy liczba odbiorców należących do każdej trasy jest ograniczona od góry. W niniejszej pracy to ograniczenie jest równe 3, co oznacza, że każda trasa jest drogą zamkniętą, do której należy magazyn i od jednego do trzech odbiorców.

Znane dokładne algorytmy rozwiązywania tego zadania mają wykładniczą złożoność czasową, co dla dużej liczby odbiorców czyni je nieprzydatnymi w praktyce. Dlatego poszukuje się algorytmów przybliżonych, które pozwalają uzyskać w stosunkowo krótkim czasie rozwiązania optymalne lub do optimum zbliżone. Przybliżone algorytmy rozwiązywania zadania dostawy były przedmiotem prac [3,4,7,8].

W dalszej części niniejszej pracy odbiorcy i magazyn będą punktami równomiernie losowo rozmieszczonymi wewnątrz płaskiej figury wypukłej, a odległości między nimi będą wyznaczane według metryki euklidesowej. Optymalne rozwiązanie tak sformułowanego zadania zazwyczaj nie zawiera tras odwiedzających tylko jednego odbiorcę, i jednocześnie zawiera największą możliwą liczbę tras odwiedzających trzech odbiorców. To heurystyczne spostrzeżenie zostanie wykorzystane przy tworzeniu algorytmu genetycznego rozwiązywania zadania dostawy, chociaż nie dla każdej konfiguracji punktów jest prawdziwe (kontrprzykład przedstawia rysunek 1*). Jednak prawdopodobieństwo takiego ułożenia punktów o rozkładzie równomiernym dąży do zera wraz ze wzrostem liczby odbiorców.



Rys. 1. Przykład ułożenia odbiorców, dla którego optymalny zbiór tras nie zawiera największej możliwej liczby tras odwiedzających trzech odbiorców
Fig. 1. A configuration of customers, for which the optimal set of routes does not contain the greatest possible number of three-customer routes

Część 2. niniejszej pracy poświęcono sprowadzeniu zadania dostawy do zadania podziału zbioru. Część 3. przedstawia podstawowe wiadomości o algorytmach genetycznych, część 4. zawiera opis algorytmu genetycznego podziału zbioru na podstawie pracy [2], a część 5. zawiera algorytm rozwiązywania zadania dostawy sprowadzonego do zadania podziału

* Niestety, jego twórca pozostaje anonimowy. Autor niniejszego artykułu spotkał go 15 maja 1999 roku na Placu Grunwaldzkim we Wrocławiu.

zbioru. Odpowiedni wybór parametrów tego algorytmu jest przedmiotem części 6. Wyniki pracy podsumowano w części 7.

2. Zadanie podziału zbioru

Zadanie podziału zbioru (ang. *set partitioning problem*) można sformułować następująco: dana jest macierz zerojedynkowa A; każdej kolumnie tej macierzy odpowiada stały koszt cząstkowy; koszt podziału jest równy sumie kosztów cząstkowych kolumn wchodzących w jego skład. Należy wyznaczyć taki zbiór kolumn, by wiersze macierzy A były *dokładnie pokryte* przy minimalnym koszcie. Dokładne pokrycie (ang. *exact coverage*) oznacza, że w każdym wierszu macierzy złożonej z wybranych kolumn macierzy A występuje dokładnie jedna jedynka.

Niech n i r będą odpowiednio liczbą wierszy i kolumn macierzy $A = [a_{ij}]$. Niech $C = [c_j]$ oznacza r-elementowy wektor kosztów cząstkowych związanych z kolumnami macierzy A, a $X = [x_j]$ – poszukiwany r-elementowy wektor, którego j-ty element jest równy 1, jeżeli j-ta kolumna wchodzi w skład rozwiązania, i równy 0 w przeciwnym przypadku. Przy tej notacji zadanie podziału zbioru polega na minimalizacji wartości wyrażenia

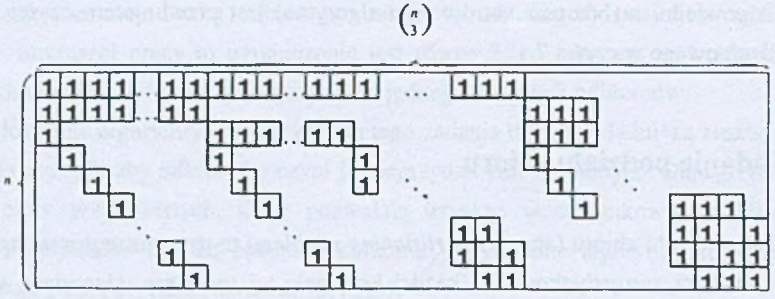
$$CX^T = \sum_{j=1}^{r} c_j x_j \tag{1}$$

przy spełnieniu warunku dokładnego pokrycia:

$$AX^T = [1\,1\ldots1]^T, \text{ czyli dla każdego } i \in \{1,\ldots,n\} \text{ zachodzi równość } \sum_{j=1}^{r} a_{ij} x_j = 1. \tag{2}$$

Jeśli n jest podzielne przez 3, to zadanie dostawy dla n odbiorców można sprowadzić do zadania podziału zbioru, w którym macierz A ma wymiary $n \times \binom{n}{3}$ i jest wypełniona na przykład w odwrotnym porządku leksykograficznym kolumn, zgodnie ze schematem przedstawionym na rysunku 2. Gdy liczba odbiorców n nie jest podzielna przez 3, to zaokrągla się ją w górę do najbliższej wielokrotności trzech, wprowadzając jednego lub dwóch *pseudoodbiorców*, położonych w zerowej odległości od magazynu.

Dla odbiorców $\{a, b, c\}$ odpowiednim elementem wektora C jest długość najkrótszej spośród tras (a, b, c), (b, c, a), (c, b, a).



Rys. 2. Schemat wypełnienia macierzy A dla zadania podziału zbioru równoważnego zadaniu dostawy (puste miejsca zawierają zera)

Fig. 2. Contents of matrix A in a set partitioning problem resulting from a delivery problem (empty cells contain zeroes)

3. Algorytmy genetyczne

Zgodnie z teorią ewolucji, rozwojem populacji organizmów żywych rządzą zasady doboru naturalnego. Osobniki lepiej przystosowane (dopasowane) do środowiska mają większe szanse przeżycia i posiadania potomstwa od osobników gorzej przystosowanych. Oznacza to, że w następnym pokoleniu zwiększy się udział ich genów w populacji. Z krzyżowania genów dobrze dopasowanych rodziców może powstawać jeszcze lepiej dopasowane potomstwo. W ten sposób gatunek jako całość coraz lepiej dopasowuje się do środowiska.

Algorytmy genetyczne (ang. *genetic algorithms*) symulują ten proces, z tym że celem większości ich zastosowań w zadaniach optymalizacji nie jest dobrze przystosowana populacja, tylko jak najlepsze indywidualne osobniki – rozwiązania danego zadania. Wprowadzenie do algorytmów genetycznych i przegląd ich zastosowań można znaleźć w pracach [5,6]. Ogólny schemat algorytmu genetycznego przedstawia rysunek 3.

```
stwórz populację początkową;
oblicz wartości funkcji dopasowania osobników;
powtarzaj
      wybierz osobniki rodzicielskie;
      krzyżuj osobniki rodzicielskie uzyskując osobniki potomne;
      zastosuj mutację do osobników potomnych;
      oblicz wartości funkcji dopasowania osobników potomnych;
      zastąp część lub całość populacji przez osobniki potomne;
przerwij, gdy został spełniony warunek zakończenia.
```

Rys. 3. Szkielet algorytmu genetycznego

Fig. 3. Skeleton of a genetic algorithm

4. Algorytm genetyczny podziału zbioru

W pracy [2] przedstawiono algorytm genetyczny rozwiązywania zadania podziału zbioru, w którym ocenia się osobniki wchodzące w skład populacji, korzystając z oddzielnych funkcji dopasowania (ang. *fitness function*) i niedopasowania (ang. *unfitness function*). Poza tym zastosowano w nim kojarzenie osobników na podstawie wzajemnego dopasowania (ang. *matching selection*), mutację statyczną (ang. *static mutation*) i adaptacyjną (ang. *adaptive mutation*), heurystyczny operator ulepszania rozwiązań (ang. *heuristic improvement operator*) i oparty na porządkowaniu osobników sposób ich usuwania z populacji (ang. *ranking replacement*). Wymienionych składników algorytmu można używać również do rozwiązywania innych zagadnień optymalizacji kombinatorycznej, ale – jak się okaże w dalszej części pracy – większość z nich jest niepotrzebna przy rozwiązywaniu zadania dostawy.

Algorytm przedstawiony w pracy [2] należy do klasy *przyrostowych algorytmów genetycznych* (ang. *incremental genetic algorithms*). Oznacza to, że w każdym kroku (pokoleniu) powstaje jeden osobnik potomny, który zastępuje jednego członka populacji. Jeżeli osobnik identyczny z nowo powstałym już występuje w populacji, to przeprowadza się ponownie losowanie osobników rodzicielskich, krzyżowanie, mutację i ulepszanie. Kryterium zakończenia działania algorytmu jest wykonanie zadanej z góry liczby kroków.

Reprezentacją osobników w tym algorytmie jest wektor zerojedynkowy X. Osobniki potomne powstają na drodze *krzyżowania jednorodnego* (ang. *uniform crossover*). Polega ono na tym, że każdy bit reprezentacji osobnika potomnego przyjmuje wartość pewnej funkcji losowej f, zależnej od odpowiadających mu bitów reprezentacji osobników rodzicielskich. Funkcja ta spełnia warunki: $f(0,0)=0$ oraz $f(1,1)=1$. Jeśli wartości tych bitów w osobnikach rodzicielskich są różne, to f przyjmuje wartość 1 z prawdopodobieństwem p i wartość 0 z prawdopodobieństwem $1-p$, gdzie $p\in[0,1]$ jest ustalone z góry. (W literaturze rozpatruje się zazwyczaj tylko przypadek, gdy $p=1/2$. Powyższe uogólnienie pochodzi od autora.)

Po krzyżowaniu osobnik potomny podlega mutacji, której celem jest rozszerzenie przestrzeni poszukiwań, zapobiegające przypadkowym stratom wartościowej informacji genetycznej przy krzyżowaniu. Mutacja zmienia wartość M_s+M_a losowo wybranych bitów reprezentacji rozwiązania na przeciwną. W algorytmie używa się mutacji statycznej, o stałej wartości parametru M_s, oraz adaptacyjnej, którą stosuje się wtedy, gdy pewne wiersze są pokryte w niewielkiej liczbie osobników populacji (np. mniej niż w połowie jej liczebności). Ten rodzaj mutacji dodaje do nowo utworzonego rozwiązania M_a kolumn pokrywających te wiersze.

Rozwiązanie utworzone w procesie krzyżowania i mutacji najczęściej nie spełnia warunku dokładnego pokrycia (2). W pracy [2] zaproponowano heurystyczny operator ulepszania rozwiązań, który przybliża pokrycie wierszy w uzyskanym rozwiązaniu do nałożonych ograniczeń. Pierwszą część operacji ulepszania stanowi wybór tych wierszy, które są pokryte ponad jeden raz i losowe usuwanie z rozwiązania pokrywających je kolumn tak długo, aż wszystkie wiersze pokrywa co najwyżej jedna kolumna. W drugiej części wybiera się losowo jeden niepokryty wiersz i dodaje do rozwiązania tę spośród kolumn go pokrywających i zarazem nie powodujących nadmiernego pokrycia innych wierszy, której odpowiada najmniejszy koszt. Operację tę powtarza się dopóty, dopóki można poprawić niedostateczne pokrycie wierszy, nie wprowadzając nadmiernego pokrycia innych wierszy. Algorytm ten przedstawiono na rysunku 4.

Pomocnicze oznaczenia:
S – zbiór kolumn rozwiązania podlegającego ulepszaniu;
$w_i = \sum_{j=1}^{r} a_{ij} x_j$ – liczba kolumn pokrywających wiersz i;
$\beta_j = \{k : a_{kj} = 1\}$ – zbiór wierszy pokrywanych w tym rozwiązaniu przez kolumnę j.

$T \leftarrow S$;
powtarzaj
wybierz losową kolumnę $j \in T$; $T \leftarrow T \setminus \{j\}$;
jeśli istnieje $i \in \beta_j : w_i > 1$ **to** $S \leftarrow S \setminus \{j\}$;
przerwij, gdy $T = \emptyset$;
utwórz zbiór niepokrytych wierszy $U = \{k : w_k = 0\}$;
powtarzaj
wybierz losowy wiersz $i \in U$;
wybierz ze zbioru kolumn $\{j : a_{ij} = 1, \beta_j \subseteq U\}$
kolumnę j o minimalnym koszcie c_j ;
$S \leftarrow S \cup \{j\}$; $U \leftarrow U \setminus \beta_j$;
przerwij, gdy $U = \emptyset$.

Rys. 4. Algorytm ulepszania rozwiązań
Fig. 4. The improvement algorithm

5. Algorytm genetyczny rozwiązywania zadania dostawy

Powodem wprowadzenia operacji wymienionych na początku części 4. jest to, że w ogólnym przypadku trudne jest znalezienie nie tylko optymalnego, lecz *jakiegokolwiek* podziału zbioru, czyli zbioru kolumn, który spełnia warunek dokładnego pokrycia (2). Jednak

w tej wersji zadania podziału zbioru, która wynika z zadania dostawy, algorytm ulepszania dokonuje pełnej naprawy rozwiązań i wszystkie osobniki populacji spełniają warunek (2).

Stosowanie wskaźnika niedopasowania, skomplikowanego sposobu kojarzenia osobników rodzicielskich i wyboru osobników usuwanych z populacji oraz mutacji adaptacyjnej okazało się niepotrzebne. Wskaźnik niedopasowania równał się zeru dla wszystkich osobników populacji, sposób kojarzenia był równoważny z wyborem turniejowym (patrz poniżej), sposób zastępowania równoważny z zastępowaniem osobnika o najgorszej (największej) wartości funkcji celu, a mutacja adaptacyjna nigdy nie była wykonywana.

Wybór turniejowy (ang. *tournament selection*) polega na wylosowaniu dwóch osobników z populacji i wyborze jako osobnika rodzicielskiego tego z nich, który ma lepszą (niższą) wartość funkcji dopasowania. Podobnie wybiera się drugiego osobnika rodzicielskiego, z tym że losowanie dotyczy wszystkich osobników poza już wybranym.

Na wzmiankę zasługują poza tym następujące szczegóły implementacji algorytmu.

Jeżeli macierz A jest wypełniona zgodnie z rys. 2, to nie trzeba jej przechowywać w pamięci, znając bowiem numer kolumny, można określić numery wierszy pokrywanych przez nią. Ponadto znając numery trzech wierszy można obliczyć numer kolumny, która je pokrywa. W programie realizującym algorytm genetyczny korzystając z tego faktu przechowuje się jedynie zawartość wektora C.

Ponieważ wektor X reprezentujący osobniki populacji jest rzadki (tylko $n/3$ z $\binom{n}{3}$ elementów jest różne od zera), dalsze oszczędności uzyskuje się, przechowując w pamięci tylko numery jego niezerowych elementów.

Największy udział czasowy w rozwiązywaniu zadania dostawy ma druga część operacji ulepszania rozwiązań: wybór kolumny o minimalnym koszcie spośród kolumn pokrywających dany wiersz. Jeżeli liczba niepokrytych wierszy jest równa 3 lub 6, dokonuje się tego przez porównanie kosztu pokrycia wszystkich możliwych trójek niepokrytych wierszy zawierających dany wiersz. Jeżeli liczba niepokrytych wierszy jest większa, to stosuje się inny algorytm, zaproponowany przez Sebastiana Deorowicza: na początku działania programu tworzy się trójwymiarową tablicę o rozmiarach $n \times n \times (n-2)$, której elementy na pozycjach $[a][b][1 \ldots n-2]$ są numerami wierszy c, uszeregowanymi według rosnących kosztów pokrycia wierszy a, b i c. Najtańsze pokrycie danych wierszy a i b daje pierwszy niepokryty wiersz w podtablicy $[a][b][1 \ldots n-2]$. Najtańsze pokrycie danego wiersza a jest minimum najtańszych pokryć a i b, gdzie b przebiega wszystkie niepokryte wiersze różne od a.

Reprezentacja populacji powinna umożliwiać szybkie usuwanie najgorszych osobników i sprawdzanie, czy należy do niej osobnik identyczny do potomnego. W tym celu zastosowano drzewo-kopce (ang. *treaps*) [1], strukturę danych będącą kopcem maksymalnym względem wartości funkcji dopasowania (tj. korzeń kopca jest najgorszym osobnikiem

populacji) i drzewem binarnym względem porządku leksykograficznego wektorów X. W drzewie-kopcu średni czas wstawiania (połączonego z usuwaniem) i wyszukiwania elementów jest proporcjonalny do logarytmu liczby jego elementów.

6. Optymalne parametry algorytmu

Po zaimplementowaniu algorytmu przeprowadzono pewną liczbę doświadczeń. Ich celem był wybór takich parametrów algorytmu, które pozwalają uzyskać jak najlepsze rozwiązanie w jak najkrótszym czasie. Jako kryterium jakości przyjęto średnią arytmetyczną kosztów najlepszych rozwiązań otrzymanych w stu niezależnych wykonaniach programu przy danym zestawie parametrów i stałym położeniu odbiorców. Pomocniczym kryterium była liczba pokoleń, wymaganych do otrzymania najlepszego rozwiązania w danym uruchomieniu (które może, ale nie musi być rozwiązaniem optymalnym). Wprawdzie liczba operacji potrzebnych do generowania osobników potomnych nie jest stała – przeciętnie jest większa w początkowym okresie ewolucji, kiedy osobniki są bardziej zróżnicowane, i zmniejsza się w następnych pokoleniach – to jednak licznik pokoleń jest przybliżoną miarą czasu wykonywania algorytmu.

Badania przeprowadzono dla ustalonych danych obejmujących 40, 60, 80 i 100 odbiorców, ograniczając liczbę pokoleń do 200000.

Badaniu podlegały następujące parametry:

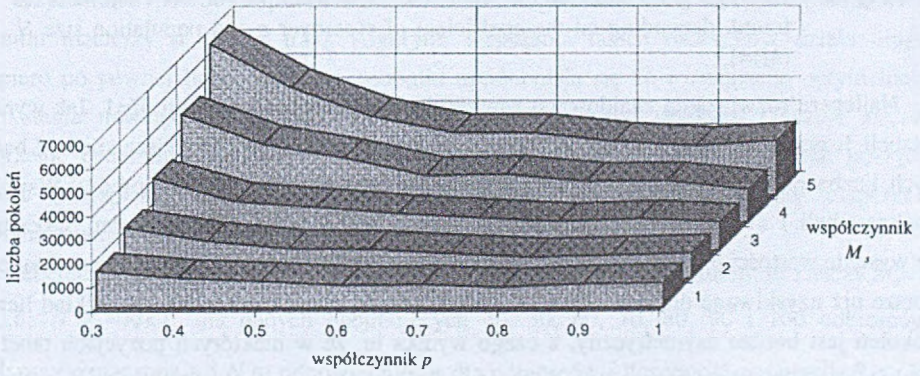
- Współczynnik krzyżowania jednorodnego p: prawdopodobieństwo, z którym rozwiązanie potomne dziedziczy po rozwiązaniach rodzicielskich bity równe 1. Autorzy pracy [2] rozpatrywali dwie jego wartości: 0,5 i 1. Druga z nich odpowiada sytuacji, gdy wszystkie bity równe 1 w rozwiązaniu potomnym odpowiadają bitom o wartości 1 w którymkolwiek z jego rozwiązań rodzicielskich (logiczne LUB). Rozwiązanie potomne jest więc nadmiernym pokryciem macierzy A, ale to nadmierne pokrycie jest eliminowane w operacji ulepszania tego rozwiązania. Z doświadczeń opisanych w pracy [2] wynika, że dla ogólnego zadania podziału zbioru lepsze wyniki uzyskuje się przy p=0,5. Nie musi to jednak być prawdą dla zadania dostawy, dlatego rozpatrywano różne wartości tego parametru.
- Współczynnik mutacji statycznej M_s: liczba bitów w rozwiązaniu potomnym, których wartość jest zmieniana na przeciwną. W pracy [2] zaleca się przyjęcie M_s=3. Z powodów tych samych, co w powyższym punkcie, a także dlatego, że najlepsza wartość tego parametru może zależeć od rozmiaru zadania, to jest liczby dostawców, badano kilka wartości M_s.
- Liczebność populacji N.

Dla danych obejmujących 40 odbiorców przeprowadzono testy z następującymi kombinacjami wartości parametrów: $p \in \{0,3; 0,4;...;0,9; 1,0\}$; $M_s \in \{0, 1,...,8\}$; $N \in \{400, 300, 200, 100, 50, 25, 12\}$. Uzyskane wyniki wskazują, że wartości $p<0,5$; $M_s=0$, a także $N<50$ są zdecydowanie niekorzystne dla działania algorytmu. Dla tych wartości parametrów konsekwentnie uzyskiwano gorsze rozwiązania niż dla pozostałych badanych wartości, kiedy to zwykle w 100 wykonaniach programu na 100 znajdowano to samo, optymalne rozwiązanie.

Średnia liczba pokoleń koniecznych do znalezienia najlepszego rozwiązania była najmniejsza, gdy liczba mutacji M_s wynosiła 1. Pod tym samym względem najkorzystniejsza spośród badanych wartości liczebność populacji N mieści się między 100 a 300. Wartości współczynnika krzyżowania p, o ile należą do przedziału [0,5; 1,0], mają niewielki wpływ na średnią liczbę pokoleń pozwalającą uzyskać najlepsze rozwiązanie, ale daje się zauważyć, że wyższe wartości powodują nieznaczne jej zmniejszenie.

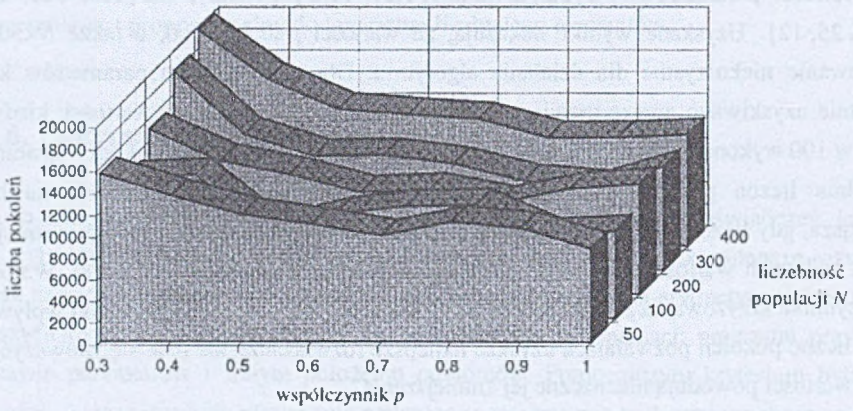
Ilustrują to rysunki 5 i 6, przedstawiające odpowiednio zależność średniej liczby pokoleń od M_s i p przy ustalonym N=200 oraz od N i p przy ustalonym M=1.

Korzystając z otrzymanych wyników, wykonano program dla danych testowych obejmujących 60, 80 i 100 odbiorców z wartościami parametrów z następujących zbiorów: $p \in \{1/2, 2/3, 5/6, 1\}$, $M_s \in \{1,2,3,4\}$, $N \in \{100, 200, 400, 800$ (dla 100 odbiorców również 1600)$\}$.



Rys. 5. Średnia liczba pokoleń wymagana do znalezienia najlepszego rozwiązania zadania dla 40 odbiorców w zależności od liczby mutacji M_s i wartości współczynnika krzyżowania p (N=200)

Fig. 5. Average generation count when the best solution for 40 customers is found, depending on the number of mutations M_s and coefficient of crossover p (N=200)



Rys. 6. Średnia liczba pokoleń wymagana do znalezienia najlepszego rozwiązania zadania dla 40 odbiorców w zależności od wartości współczynnika krzyżowania p i liczebności populacji N (M_s=1)

Fig. 6. Average generation count when the best solution for 40 customers is found, depending on the coefficient of crossover p and population size N (M_s=1)

Najlepsze rozwiązania znajdowano zawsze dla współczynnika mutacji M=1. Jak wynika z tabeli 1, przedstawiającej jakość uzyskiwanych rozwiązań, najkorzystniejsza spośród badanych liczba osobników populacji N to 400 dla 60; 800 dla 80 i 100 odbiorców. Wspólna analiza tabeli 1 i tabeli 2, która przedstawia średnią liczbę wymaganych pokoleń, wskazuje, że wysokie wartości p powodują przyspieszenie znajdowania rozwiązań, ale rozwiązania te są gorsze niż uzyskiwane dla p bliskich 1/2. Warto w tym miejscu wyjaśnić, że rozkład liczby pokoleń jest bardzo asymetryczny, z czego wynika to, że w niektórych pozycjach tabeli 2 odchylenie standardowe przewyższa wartość średnią.

Podsumowując opisane powyżej eksperymenty, stwierdzamy, że za parametry algorytmu można przyjąć: współczynnik mutacji M_s=1; współczynnik krzyżowania p=0,6. W celu otrzymania pełnego opisu algorytmu dla dowolnej liczby odbiorców n pozostaje ustalić warunek zakończenia jego działania i liczebność populacji N.

Na podstawie tabel 1 i 2 oraz wybiórczych eksperymentów z różnymi odwzorowaniami $N=f(n)$ autor przyjął $N = \left\lfloor 40\sqrt{n} \right\rfloor$. Należy zaznaczyć, że przy bardzo małej liczbie odbiorców ($n \leq 10$) tak ustalona liczebność populacji przewyższa liczbę możliwych rozwiązań zadania i algorytm wykonuje nieskończone iteracje przy próbie wygenerowania N różnych rozwiązań.

Jednak stosowanie algorytmów genetycznych mija się z celem przy tak małych rozmiarach zadania, gdy rozwiązanie optymalne można wyznaczyć sprawdzając wszystkie możliwe rozwiązania.

W powyższych eksperymentach warunek zakończenia działania algorytmu był stały: liczba pokoleń równa 200000. Jednak, jak wynika z tabeli 2, średnia liczba pokoleń przed znalezieniem najlepszego znanego rozwiązania zależy od n i dla rozpatrywanego przedziału ($n \leq 100$) jest mniejsza od 200000. Z drugiej strony, zwiększenie liczby pokoleń może umożliwić znajdowanie lepszych rozwiązań. Dlatego każde ograniczenie na liczbę pokoleń jest kompromisem między czasem działania programu a jakością znajdowanych rozwiązań. W swoich kolejnych testach autor przyjął, że algorytm zakończy działanie, gdy skład populacji pozostawał stały przez N kolejnych pokoleń, gdzie N jest liczebnością populacji. Oznacza to, że żaden osobnik potomny powstały w tych pokoleniach nie wszedł w skład populacji. Osobniki te były identyczne z pewnymi członkami populacji lub wartości ich funkcji dopasowania były gorsze niż u najgorszego osobnika populacji.

Po zaimplementowaniu algorytmu okazało się, że stosując inny sposób tworzenia populacji początkowej, niż opisany w pracy [2], można przyspieszyć jego działanie bez pogarszania jakości otrzymywanych rozwiązań. Autorzy [2] inicjowali populację losowymi osobnikami. Ich krzyżowanie powoduje powstawanie osobników potomnych, w których brakuje wielu kolumn macierzy A (wielu tras). Algorytm ulepszania takich osobników działa długo. Dopiero po pewnej liczbie pokoleń osobniki upodobniają się do rozwiązania optymalnego i do siebie nawzajem, i algorytm ulepszania działa znacznie szybciej. Okazuje się, że tworzenie osobników populacji początkowej za pomocą zachłannego algorytmu pokrycia zbioru przyspiesza działanie algorytmu. Algorytm ten jest równoważny ulepszaniu osobników bez żadnych tras za pomocą algorytmu z rys. 4.

Tabela 3. przedstawia wyniki działania programu z takimi parametrami, to znaczy M_r=1; p=0,6; $N = \lfloor 40\sqrt{n} \rfloor$, dla danych obejmujących 40, 50, 60, 70, 80, 90 i 100 odbiorców. Kolumny oznaczone n i N to odpowiednio liczba odbiorców i liczebność populacji. R_{min}, $R_{śr}$ i σ_R oznaczają najkrótszą i średnią długość rozwiązań uzyskanych w 100 wykonaniach programu oraz jej odchylenie standardowe; $P_{śr}$ i σ_P to średnia liczba pokoleń i jej odchylenie standardowe. W kolumnie h przedstawiono, ile razy na 100 wykonań programu uzyskano rozwiązanie o długości R_{min}, a średni czas jednego wykonania programu na komputerze z procesorem Pentium 133 MHz zawiera kolumna oznaczona $t_{śr}$.

Tabela 1

Długość najlepszej znalezionej trasy dla 60, 80 i 100 odbiorców, przy M_r=1, różnych liczebnościach populacji N i wartościach współczynnika p; średnia i odchylenie standardowe ze 100 wykonań programu

n	N	Wartość współczynnika krzyżowania p			
		1/2	2/3	5/6	1
60	100	1173,764±1,915	1173,813±2,537	1173,914±2,227	1174,323±2,757
	200	1173,352±0	1173,402±0,495	1173,482±0,939	1173,402±0,495
	400	1173,352±0	1173,352±0	1173,352±0	1173,402±0,495
	800	1173,352±0	1173,352±0	1173,352±0	1173,352±0
80	100	1504,553±3,275	1503,917±2,216	1504,453±3,542	1504,738±3,600
	200	1503,319±0,800	1503,486±2,071	1503,288±0,794	1503,422±1,741
	400	1503,072±0,449	1503,073±0,530	1503,111±0,554	1503,071±0,449
	800	1503,018±0,285	1502,987±0,229	1502,975±0,206	1502,992±0,237
100	100	1841,164±2,944	1841,176±3,297	1841,406±3,544	1841,286±3,066
	200	1840,386±2,723	1840,605±2,482	1841,214±3,288	1840,233±2,354
	400	1839,924±1,441	1840,001±1,845	1839,972±1,735	1839,771±1,547
	800	1839,948±1,384	1839,719±1,152	1839,572±1,145	1839,532±1,055
	1600	1872,176±11,124	1852,709±9,343	1844,666±6,623	1843,451±5,710

Tabela 2

Liczba pokoleń wymaganych do znalezienia najlepszego rozwiązania dla 60, 80 i 100 odbiorców, przy M_r=1, różnych liczebnościach populacji N i wartościach współczynnika p; średnia i odchylenie standardowe ze 100 wykonań programu

n	N	Wartość współczynnika krzyżowania p			
		1/2	2/3	5/6	1
60	100	49100±52400	37600±42800	39100±45800	50000±54900
	200	28700±27100	31300±28900	28000±26100	27400±25900
	400	34300±24300	30900±21100	28500±15700	25600±14600
	800	51900±23000	44700±16600	39700±13100	39300±13900
80	100	49500±45200	46500±49600	44500±45100	45000±49400
	200	56600±45300	55900±47500	47600±43600	52600±47600
	400	78400±44300	65000±41400	68700±46500	61100±40700
	800	110700±41000	104600±42200	86700±36000	91000±37200
100	100	57700±48400	65200±52500	60300±49400	62700±53700
	200	81500±49400	76400±46700	67800±50300	68400±45500
	400	95300±36800	86700±37700	85900±40000	81100±39700
	800	149800±25100	131900±26400	126200±29700	119700±25800
	1600	188000±15400	195100±6300	195500±4900	194900±5500

Tabela 3

Jakość rozwiązań zadania dostawy dla 40–100 odbiorców

n	N	R_{min}	$R_{śr}$	σ_R	$P_{śr}$	σ_P	h [%]	$t_{śr}$ [s]
40	252	819,010	819,767	2,071	7916	3065	87	1,51
50	282	982,229	982,574	0,651	10263	3426	69	2,23
60	309	1173,352	1173,505	0,281	17317	5286	58	4,64
70	334	1364,715	1370,570	3,822	26170	7205	14	8,19
80	357	1502,933	1503,951	1,129	28831	7604	28	10,53
90	379	1691,076	1696,672	3,681	59372	14775	12	26,67
100	400	1836,501	1840,586	2,026	70228	15917	1	35,07

7. Podsumowanie wyników

Pod względem jakości uzyskiwanych rozwiązań porównanie algorytmów rozwiązywania zadania dostawy przedstawionych w pracach [3,4,7,8] z przedstawionym w niniejszym artykule wypada na korzyść algorytmu genetycznego. Natomiast jego wadą jest wysoka złożoność pamięciowa, która jest rzędu sześcianu liczby odbiorców, a dla algorytmów k-optymalnych, mrówkowych, wyżarzania i przeszukiwania tabu jest kwadratowa. Również dla algorytmu przedstawionego w niniejszej pracy można ją zmniejszyć do rzędu kwadratowego, ale za cenę zwiększenia czasu działania. W tym celu należy nie przechowywać w pamięci wektora kosztów C, lecz obliczać od nowa koszty trasy przechodzącej przez daną trójkę odbiorców za każdym razem, gdy jest to wymagane przez algorytm, oraz zmienić implementację algorytmu ulepszania rozwiązań.

Możliwe są modyfikacje algorytmu, polegające na uzmiennieniu jego parametrów. Wartości współczynników krzyżowania i mutacji można uzależnić od numeru pokolenia. W początkowych pokoleniach miałyby one wartości przyspieszające zbieżność, a następnie zmieniały się tak, by preferować znajdowanie lepszych rozwiązań zadania. Możliwe byłoby też uzmiennienie liczebności populacji, chociaż korzyści z tego nie są jasne i skomplikowałoby to program.

8. Podziękowania

Autor dziękuje Zbigniewowi Czechowi za sugestię użycia drzewo-kopców i uwagi na temat niniejszego artykułu, Sebastianowi Deorowiczowi za dyskusje o algorytmach genetycznych i usprawnienie algorytmu ulepszania, oraz twórcy kontrprzykładu z rysunku 1. Część obliczeń wykonano na serwerze sieciowo-obliczeniowym Śląskiej Akademickiej Sieci Komputerowej zainstalowanym w Centrum Komputerowym Politechniki Śląskiej.

LITERATURA

1. Aragon C.R, Seidel R.B.: Randomized search trees (extended abstract). 30th IEEE Symposium on Foundations of Computer Science, 1989, s. 540–546.
2. Chu P.C., Beasley J.E.: Constraint handling in genetic algorithms: the set partitioning problem. Journal of Heuristics, 1998, t. 11, s. 323–357.
3. Cichoński S.: Sekwencyjne i równoległe algorytmy symulowanego wyżarzania. Praca magisterska, Instytut Informatyki Politechniki Śląskiej, Gliwice 1999.
4. Deorowicz S.: Rozwiązywanie problemu dostawy za pomocą algorytmów k-optymalnych. ZN Pol. Śl., s. Informatyka, z. 33, Gliwice 1997, s. 139–146.
5. Goldberg D.E.: Algorytmy genetyczne i ich zastosowania. WNT, Warszawa 1995.
6. Michalewicz Z.: Algorytmy genetyczne + struktury danych = programy ewolucyjne. WNT, Warszawa 1997.
7. Skórczyński A.: Rozwiązywanie problemu dostawy za pomocą algorytmów mrówkowych. ZN Pol. Śl., s. Informatyka, z. 37, Gliwice 1999, s. 287–295.
8. Szołtysek M.: Rozwiązywanie problemu dostawy za pomocą przeszukiwania tabu. ZN Pol. Śl., s. Informatyka, z. 37, Gliwice 1999, s. 209–221.

Recenzent: Dr inż. Mariusz Boryczka

Wpłynęło do Redakcji 20 lipca 2000 r.

Abstract

This paper presents a genetic algorithm for solving the delivery problem, based upon a genetic algorithm for solving the set partitioning problem by Chu and Beasley [2]. Due to idiosyncrasies of the delivery problem, some components of the algorithm proved useless, and it has been established that better results are obtained when the population size and the mutation coefficient have different values than in the original algorithm [2]. The author tested empirically a generalization of uniform crossover, which improves the process of finding the solutions. The genetic algorithm is as fast as other heuristic algorithms for solving the delivery problem, yet it produces better solutions on the average. However, its memory requirements are higher.