ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1988

Seria:AUTOMATYKA z.94 Nr kol.970

Stanisław Kryński, Marek Libura
Instytut Badań Systemowych PAN,Warszawa

ALGORYTMY PRZYBLIŻONE DLA ZADANIA KOMIWOJAŻERA NA PŁASZCZYŹNIE OPARTE NA KONSTRUKCJI KRZYWEJ WYPEŁNIAJĄCEJ KWADRAT

Streszczenie. W pracy badane są algorytmy przybliżone dla zadania komiwojażera na płaszczyźnie wykorzystujące nową heurystykę zaproponowaną przez Platzmana i Bartholdiego. Polega ona na wykorzystaniu krzywej wypełniającej kwadrat do skonstruowania obwodu Hamiltona łączącego n punktów płaszczyzny i charakteryzuje się wyjątkowo niską złożonością obliczeniową O(n log n). W pracy podano opis tego podejścia oraz zaproponowano i zbadano jego modyfikacje. Przedstawiono wyniki eksperymentów obliczeniowych dla dwuczłonowych heurystyk, w których rozwiązanie początkowe jest konstruowane z użyciem krzywej wypełniającej kwadrat, a następnie jest poprawiane przez dołączenie różnych algorytmów opartych na idei lokalnego przeglądu.

1. Wprowadzenie

Zadanie komiwojażera (inaczej: zadanie znajdowania najkrótszego obwodu Hamiltona w grafie) polega na znalezieniu dla danego grafu takiego obwodu łączącego wszystkie jego wierzchołki, którego długość jest minimalna. Zagadnieniu temu jest poświęcona ogromna literatura (patrz np. monografia [4]). Rozważane w niniejszej pracy zadanie komiwojażera na płaszczyźnie jest ważnym szczególnym przypadkiem ogólnego problemu.

Zakładamy, że w R^2 dany jest zbiór n-elementowy $A=\{a_1,\ldots,a_n\}$. Niech $\Pi(n)$ oznacza rodzinę permutacji zbioru $\{1,\ldots,n\}$. Dla $p\in\Pi(n)$ definiujemy:

$$D_A(p) = d(a_{p(n)},a_{p(1)}) + \sum_{i=1}^{n-1} d(a_{p(i)},a_{p(i+1)}) \qquad (1)$$

gdzie d jest ustaloną metryką w R^2. $D_A(p)$ jest długością obwodu łączącego wszystkie punkty zbioru A przy kolejności obchodzenia punktów wyznaczonej przez permutację p. Problem polega na znalezieniu takiej permutacji $p^*\in\Pi(n)$, dla której

$$D_A(p^*) = \min\{D_A(p): p\in\Pi(n)\}. \qquad (2)$$

W przypadku gdy d jest odległością euklidesową, mamy do czynienia z euklidesowym zadaniem komiwojażera na płaszczyźnie. Większość wyników obliczeń omawianych w niniejszym opracowaniu dotyczy tego właśnie przypadku. Inną często rozważaną odległością jest tzw. metryka miejska (l_1).

Tak postawione zadanie (2) jest problemem NP-trudnym [5]. Uzyskanie rozwiązania optymalnego dla dużych n (rzędu kilkuset) wymaga znacznego na-

kładu obliczeń. Z tego względu duży wysiłek poświęca się konstruowaniu me przybliżonych, które w akceptowalnym czasie znajdują rozwiązania bliskie tymalnym. Szczególnie istotne i pożądane jest opracowanie takich metod, k re można stosować na mikrokomputerach.

Wśród wielu heurystyk zaproponowanych dla rozwiązywania zadania kom jażera można wyróżnić grupę metod składających się z dwóch modułów:
1^{o} modułu generacji rozwiązania startowego (preprocesor);
2^{o} modułu poprawiania rozwiązań (poprocesor).
Z chwilą zakończenia pracy modułu preprocesora znane jest rozwiązanie dop szczalne (obwód Hamiltona), którego długość jest następnie zmniejszana w dule poprocesora. W przypadku zadania komiwojażera na płaszczyźnie stosuj się wiele różnych heurystyk, z których mogą być zestawione oba moduły. (O szerny przegląd tych możliwości zawiera praca [4]). W wielu przypadkach m duł poprocesora jest bardzo rozbudowany i zawiera kilka następujących po s bie algorytmów.

Niniejsza praca prezentuje wyniki badań dotyczących nowej heurystyki zaproponowanej w [1] oraz jej modyfikacji, które mogą być użyte w modułach generacji i poprawiania rozwiązania startowego. Metoda ta jest oparta na bardzo prostej i oryginalnej idei wykorzystującej do konstrukcji rozwiąza startowego krzywą wypełniającą kwadrat.

W pracy przedstawione są kolejno: opis procedury wyznaczania rozwiąz nia startowego (pkt 2), omówienie kilku jej modyfikacji i rozszerzeń (pkt oraz wyniki eksperymentów obliczeniowych (pkt 4). Celem wspomnianych eksp rymentów było zweryfikowanie użyteczności metod omówionych w pktach 2 i 3 w połączeniu ze znaną heurystyką typu 2-opt [8].

2. Podstawowy algorytm generowania rozwiązania startowego przy użyciu krzywej wypełniającej kwadrat

Zadaniem preprocesora w dwumodułowej heurystyce jest wygenerowanie startowego obwodu Hamiltona. Zwykle w tym kroku stosowane są metody stopn wo budujące taki obwód poprzez włączanie kolejnych punktów zgodnie z rozm itymi regułami heurystycznymi. W [1] zaproponowana została zupełnie odmien metoda konstrukcji rozwiązania początkowego.

Krzywą wypełniającą kwadrat (inaczej: krzywą Peano, zob. np. [2]) n zywa się ciągłe odwzorowanie $\varphi: I_o - S_o$ takie, że $\varphi(I_o)=S_o$, gdzie $I_o=[0,1]$, $S_o=[0,1]\times[0,1]$. W metodzie [1] wykorzystano krzywą zdefiniowaną przez Sier pińskiego [7], mającą dodatkowo tę własność, że $\varphi(0)=\varphi(1)$.

Załóżmy, że $A \subset S_o$. Podstawowa idea konstrukcji startowego obwodu Ham tona podana w [1] polega na przyjęciu kolejności punktów w obwodzie zgodnej z kolejnością ich występowania na krzywej Sierpińskiego. Algorytm postępow nia jest następujący (będziemy go nazywać krótko algorytmem SFC od angiel skiego "space filling curve"):

ALGORYTM SFC

1^0 Dla każdego i=1,...,n wybierz liczbę $\Theta_i \in \varphi^{-1}(a_i)$.

2^0 Wyznacz permutację $\bar{p} \in \Pi(n)$ taką, że $\Theta_{\bar{p}(1)} \leqslant \Theta_{\bar{p}(2)} \leqslant \ldots \leqslant \Theta_{\bar{p}(n)}$.

Wybór startowej permutacji $\bar{p}$ polega więc na znalezieniu liczb Θ takich, że $\varphi(\Theta)=a$, dla każdego punktu $a \in A$. Następnie, gdy obliczone są wartości Θ_i dla i=1,...,n, można je posortować konstruując tym samym rozwiązanie startowe $\bar{p}$ w O(n log n) krokach.

W [1] podano, że wartość $D_A(\bar{p})$ rozwiązania wyznaczonego algorytmem SFC spełnia nierówność $D_A(\bar{p}) \leqslant 2\sqrt{n}$. Stwierdzono tam również, na podstawie wyników eksperymentów, że błąd względny algorytmu SFC jest, przy n dążącym do ∞, w przybliżeniu równy 25%. (Wartość ta została potwierdzona w eksperymentach relacjonowanych w pkcie 4).

Numeryczna realizacja algorytmu SFC wymaga określenia sposobu obliczania wartości Θ_i. Krzywa wypełniająca kwadrat jest definiowana poprzez wskazanie ciągu funkcji ciągłych $\{\varphi_k\}_{k=0}^{\infty}$, $\varphi_k : I_0 - S_0$, jednostajnie zbieżnego do φ. W przypadku krzywej Sierpińskiego kolejne elementy tego ciągu mogą być zadane za pomocą prostych zależności rekurencyjnych (zob. [1]).

Dla k=0,1,2,... zdefiniujmy następujące podzbiory odcinka I_0 i kwadratu S_0: $I_0(k)=\{i \cdot 2^{-k} : i=0,1,\ldots,2^k\}$, $S_0(k)=I_0(k) \times I_0(k)$, $P_0(k)=I_0 \times I_0(k) \cup I_0(k) \times I_0$, $V_0 = \{0,1\} \times \{0,1\}$.

Funkcje φ_k (k=0,1,...) określające krzywą Sierpińskiego mają następujące własności istotne dla algorytmu SFC i jego modyfikacji:

$$\varphi_k(0) = \varphi_k(1) = (0,0) \tag{3}$$

$$\varphi_k(I_0) = P_0(k) \tag{4}$$

$$\#\varphi_k^{-1}(a) = \begin{cases} 4, \text{jeśli } a \in S_0(k) \cap \operatorname{int} S_0 \\ 2, \text{jeśli } a \in S_0(k) \setminus (\operatorname{int} S_0 \cup V_0) \\ 1, \text{jeśli } a \in V_0 \end{cases} \tag{5}$$

(symbol # oznacza moc zbioru),

$$\text{jeśli } a \in S_0(k), \text{ to } \varphi_k^{-1}(a) = \varphi_m^{-1}(a) \quad \text{dla} \quad m \geqslant k \tag{6}$$

Sformułujemy wersję algorytmu SFC gotową do implementacji w postaci programu komputerowego. W istocie służyła ona za punkt wyjścia do opisywanych w dalszych punktach pracy modyfikacji algorytmu SFC.

Przyjmijmy, że ustalona jest liczba naturalna k. Można ją interpretować jako liczbę uwzględnianych w obliczeniach znaczących cyfr dwójkowych (bitów) w binarnym rozwinięciu współrzędnych każdego punktu $a \in A$.

ALGORYTM SFC_k

1^0 dla i=1,...,n, wyznacz punkt $a_k^i \in S_0(k)$ aproksymujący punkt a_i,

2^0 dla i=1,...,n wybierz liczbę $\Theta_i \in \varphi_k^{-1}(a_k^i)$;

3^0 wyznacz permutację $\bar{p} \in \Pi(n)$ taką, że $\Theta_{\bar{p}(1)} \leqslant \Theta_{\bar{p}(2)} \leqslant \ldots \leqslant \Theta_{\bar{p}(n)}$.

3. Modyfikacje i rozszerzenia algorytmu podstawowego

3.1. Nie wszystkie elementy algorytmu SFC_k zostały w podanym schemacie ustalone jednoznacznie. Po pierwsze, aproksymacja a_i^k punktu a_i nie musi być punktem zbioru $S_o(k)$ najbliższym a_i. Nie ma żadnych powodów dyskwalifikujących inne punkty w $S_o(k)$, "bliskie" a_i. W szczególności mogą nimi być wierzchołki najmniejszego kwadratu o bokach zawartych w $P_o(k)$, do którego należy a_i.

Po drugie, jak zauważyliśmy w (5), dla punktów $a \in S_o(k)$ zbiór $\varphi^{-1}(a)$ zawiera na ogół - z wyjątkiem punktów brzegowych kwadratu - cztery liczby. Rodzina wszystkich n-tek $(\Theta_1,\ldots,\Theta_n)$ takich, że $\Theta_i \in \varphi^{-1}(a_i^k)$, określa wiele permutacji $\bar{p} \in \Pi(n)$, zgodnych z warunkiem w kroku 3^o algorytmu SFC_k. Oczywiście pożądane byłoby wybranie spośród nich permutacji minimalizującej wartość kryterium D_A.

Dla punktu $a \in S_o$ przez $T_k(a)$ oznaczmy zbiór wartości parametru Θ, które związemy z punktem a przy ustalonej liczbie uwzględnianych bitów k. Do zbioru $T_k(a)$ należą wszystkie lub niektóre z elementów zbiorów $\varphi^{-1}(a^k)$, gdzie a^k przebiega pewien określony zbiór punktów sąsiednich dla a w $S_o(k)$. Nie będziemy tutaj bliżej precyzować zawartości $T_k(a)$, pozostawiając ten problem do rozstrzygnięcia przy tworzeniu konkretnego programu komputerowego.

Najprostszy algorytm, który można zaproponować jako modyfikację algorytmu SFC_k wykorzystującą wielość wartości parametru Θ, polega na wzbogaceniu poprzedniego schematu o losowy przegląd rodziny osiągalnych permutacji.

ALGORYTM SFC_k-rand

(A) Ustal liczbę g generowanych permutacji.

(B) Dla m=1,...,g wykonaj:

1^o wybierz losowo $\Theta_i^m \in T_k(a_i)$, i=1,..,n;

2^o wyznacz permutację $p_m \in \Pi(n)$ taką, że $\Theta^m_{p_m(1)} \leqslant \ldots \leqslant \Theta^m_{p_m(n)}$

(C) Przyjmij jako rozwiązanie permutację

$\bar{p} = \{p_1,\ldots,p_q\}$ taką, że $D_A(\bar{p}) = \min\{D_A(p_m): m=1,\ldots,q\}$.

Inny algorytm można oprzeć na idei lokalnego przeglądu zbioru rozwiązań dopuszczalnych (zob. np. [6]). Przeglądowi jest przy tym poddana nie cała rodzina $\Pi(n)$, ale jej podrodzina, której elementy są wyznaczone przez przenumerowanie w porządku rosnącym wszystkich n-tek $(\Theta_1,\ldots,\Theta_n) \in T_k(a_1) \times \ldots \times T_k(a_n)$. Dla permutacji p odpowiadającej n-tce $(\Theta_1,..,\Theta_n)$ za sąsiednią jest uważana każda permutacja p' odpowiadająca n-tce $(\Theta'_1,\ldots,\Theta'_n)$ takiej, że dla pewnego i, $1 \leqslant i \leqslant n$ zachodzi $\Theta'_i \neq \Theta_i$ oraz $\Theta'_j = \Theta_j$ dla $j \neq i$. Algorytm przybiera zatem następującą postać (ls jest skrótem słów "local search").

ALGORYTM SFC_k-ls

(A) Ustal listę $L \in \Pi(n)$ wyznaczającą kolejność, według której będą analizowane punkty zbioru A, zgodnie z regułą:

dia $T_k(a_{L(1)}) \geqslant$ dia $T_k(a_{L(2)}) \geqslant \ldots \geqslant$ dia $T_k(a_{L(n)})$
gdzie dia oznacza maksymalną różnicę par liczb należących do zbioru.

(B) Wyznacz startową permutację $\bar{p}$ zgodnie z algorytmem SFC_k. Niech $\Theta_1,\ldots,\Theta_n$ będą liczbami $\Theta_i \in T_k(a_i)$, determinującymi $\bar{p}$.

(C) Dla j=1,...,n wykonaj:
sprawdź, czy istnieje $\Theta \in T_k(a_{L(j)})$ takie, że zastąpienie aktualnego $\Theta_{L(j)}$ przez Θ i odpowiednia zmiana permutacji spowodują skrócenie obwodu Hamiltona. Jeśli tak, to wybierz Θ' dające największą poprawę i podstaw $\Theta_{L(j)} := \Theta'$, $\bar{p} := p'$, gdzie p' jest odpowiednio skorygowaną permutacją.

(D) Jeśli w kroku (C) nie uzyskano poprawy, to STOP.
W przeciwnym przypadku powtórz (C).

3.2. Do tej pory rozpatrywaliśmy algorytmy typu SFC opierające się na konstrukcji krzywej Sierpińskiego w kwadracie S_0. To podejście oczywiście może być zastosowane do dowolnego obszaru prostokątnego na płaszczyźnie, przy czym ze względów numerycznych należy ograniczyć się do prostokątów, których boki są równoległe do osi współrzędnych.

Powyższe spostrzeżenie, chociaż jest trywialne i oczywiste z teoretycznego punktu widzenia, w odniesieniu do omawianych algorytmów pociąga za sobą kolejne niejednoznaczności, a więc również nowe możliwości ich modyfikacji. Dla ustalonego zbioru $A=\{a_1,\ldots,a_n\}$ w $\mathbb{R}^2$ istnieje nieskończenie wiele prostokątów zawierających go. Zastosowanie dowolnego z opisanych algorytmów do różnych prostokątów daje na ogół różne obwody Hamiltona.

3.3. Na zakończenie przeglądu modyfikacji algorytmu SFC, których przydatność była weryfikowana eksperymentalnie, zauważmy, że długości obwodów Hamiltona w zbiorze A nie zależą od izometrycznych przekształceń płaszczyzny, w szczególności od obrotów. Przez h_α oznaczmy obrót płaszczyzny o kąt α. Algorytm SFC_k i wszystkie omówione modyfikacje można stosować do zbioru $h_\alpha(A)$. Permutacje $\bar{p} \in \Pi(n)$ otrzymane dla różnych α są w ogólności różne.

4. Eksperymenty obliczeniowe

4.1. Algorytm SFC_k oraz wszystkie modyfikacje omówione w poprzednim punkcie zostały opracowane w postaci programów komputerowych. Za ich pomocą przeprowadzono eksperymenty obliczeniowe. Miały one dwa zasadnicze cele: (i) porównanie efektywności wyróżnionych wariantów algorytmu i wskazanie najlepszego wśród nich, (ii) sprawdzenie przydatności wybranego wariantu algorytmu jako poprocesora (lub jednego ze składników poprocesora).

Wszystkie testowane programy były napisane w języku Turbo Pascal 3.0 z przeznaczeniem na komputer IBM PC/XT/AT.

Większość rozwiązywanych zadań testowych dotyczyła układów punktów ge-

nerowanych losowo (z rozkładem równomiernym) w prostokącie. W części (i) eksperymentu brano pod uwagę również układy puktów generowane losowo w kol W części (ii) eksperymentu skoncentrowano się na układach punktów wybieranych losowo w kwadracie o boku 1000. Liczba punktów w tych zadaniach zmieniała się od n=10 do n=2000. Rozwiązywano również standardowe problemy zna ne z literatury.

Za podstawowy wariant algorytmu przyjęto SFC_{10} (tj. dla k=10) i wyniki uzyskiwane przy jego zastosowaniu stanowiły punkt odniesienia dla badania innych wariantów. Ogólnie można stwierdzić, że algorytm SFC_k daje lepsze rozwiązanie dla większych wartości k, chociaż okupione jest to nieco dłuższym czasem obliczeń. W praktyce jednak zwiększenie k ponad 10 nie daje istotnej poprawy uzyskiwanych rozwiązań. (Wartość k=10 przyjęta była również w [1]).

Badane były rozmaite sposoby określania zbiorów $T_k(a)$, z których wybie ra się w algorytmach SFC_k-rand i SFC_k-ls parametry Θ. W żadnym z testowanych wariantów nie zawierały one więcej niż 5 elementów.

Wyniki eksperymentów dotyczących (a) wyboru różnych prostokątów zawierających zbiór A, (b) algorytmu SFC_k-rand oraz (c) wpływu obrotów płaszczyzny na wyznaczane rozwiązanie były przedstawione w raporcie [3]. Tutaj ograniczymy się do sformułowania najważniejszych wniosków.

Testowanie wpływu wyboru różnych prostokątów potwierdziło przewidywania: najlepsze wyniki osiągane były dla prostokąta o najmniejszej powierzchni wśród prostokątów zawierających zbiór A (prostokąt "opisany" na zbiorze A). Nie dotyczyło to jednak wszystkich przypadków. Wyjątek stanowiły zadanie, dla których stosunek boków tego prostokąta był mniejszy od 0.5. Wówczas korzystniejszy był wybór kwadratu o najmniejszej powierzchni wśród kwadratów zawierających A.

Zarówno algorytm SFC_k-rand, jak algorytm SFC_k powtarzany dla rozmaitych kątów obrotu płaszczyzny, dawały średnio kilkuprocentową poprawę wyznaczanego rozwiązania. Jej wielkość w przypadku obrotów silnie zależała jednak od kształtu obszaru, w którym losowano punkty zbioru A: największa była dla koła, znikoma dla prostokątów. W przypadku algorytmu zrandomizowanego osiągana poprawa podlegała nieregularnym wahaniom w przedziale od 0 do 8%, nawet przy zwiększanej liczbie badanych permutacji.

Kolejnym algorytmem opracowanym i testowanym przez autorów był SFC_k-ls. Uzyskiwane wyniki były bardziej zachęcające niż w przypadku pozostałych modyfikacji algorytmu SFC. Dlatego na nim oparto następne etapy eksperymentów obliczeniowych. Zamieszczone dalej tabele zawierają między innymi podsumowanie wyników otrzymywanych za pomocą tego algorytmu oraz jego porównanie z SFC_k.

4.2. Algorytm SFC_k daje rozwiązanie dopuszczalne zadania komiwojażera, a

zatem może być użyty jako preprocesor w dwuczłonowej heurystyce opisanej w punkcie 1. Jest przy tym preprocesorem o wyjątkowo niskiej złożoności obliczeniowej. Można się spodziewać, że rozwiązanie początkowe wygenerowane przez ten algorytm będzie istotnie poprawione przez dołączenie drugiego członu heurystyki, czyli poprocesora. W istocie opisany w punkcie 3 algorytm SFC_k-ls może być potraktowany jako taka dwuczłonowa heurystyka, w której SFC_k jest preprocesorem, a rolę poprocesora pełni przegląd (kroki (C), (D) algorytmu).

W badaniach różnych poprocesorów dla SFC_k skoncentrowano się na algorytmach wykorzystujących znaną heurystykę 2-opt w wersji opisanej w [8]. Badano następujące algorytmy:

(i) SFC_k (bez poprocesora)
(ii) SFC_k-2opt (poprocesor: 2opt)
(iii) SFC_k-ls (poprocesor: przegląd lokalny)
(iv) SFC_k-ls-2opt (poprocesor: przegląd i 2opt)
(v) SFC_k-2opt-3opt (poprocesor: 2opt, 3opt [7])
(vi) 2opt (x4) (czterokrotnie powtórzony algorytm 2opt dla losowo wybranych permutacji startowych).

Większość zadań testowych stanowiły zadania generowane losowo. Rozwiązano również kilka problemów opisywanych w literaturze.

Wyniki testów dla zadań losowych podane są w Tabelach 1,2. W Tabelach 3,4 przedstawiono rezultaty otrzymane dla dwóch wybranych problemów z literatury: zadania 24/100 Krolaka i zadania 318 Lina (patrz np. [4]). Wszystkie zadania losowe generowano wybierając n punktów z rozkładem równomiernym z kwadratu o boku 1000. Dla n=10,20,30,40,50,80,200 generowano po 10 zadań, natomiast dla n=200...2000 po 5 zadań.

W tabelach przytoczone są następujące wielkości dla opisanych wyżej algorytmów:

t - średni czas obliczeń danym algorytmem w sekundach na jedno zadanie dla IBM PC/AT z koprocesorem numerycznym;

λ - procentowe zmniejszenie średniej wartości rozwiązania w wyniku dołączenia poprocesora do algorytmu SFC_k obliczone następująco: $\lambda = 100\% \, (l_{SFC_k} - l)/l_{SFC_k}$, gdzie l, l_{SFC_k} są wartościami średnimi rozwiązań uzyskanych danym algorytmem i algorytmem SFC_k;

τ - zwiększenie czasu obliczeń w wyniku dodania poprocesora: $\tau = t/t_{SFC_k}$, gdzie t_{SFC_k} jest średnim czasem obliczeń dla algorytmu SFC_k.

W żadnym z generowanych losowo zadań nie dysponowano wartościami optymalnymi rozwiązań. Dla większych zadań pewnym punktem odniesienia może być asymptotyczna wartość średnia długości optymalnego obwodu Hamiltona. Jeśli punkty zbioru A są generowane losowo z rozkładem równomiernym z prostokąta o polu R, wówczas wartość średnia długości najkrótszego obwodu Hamiltona $L(n,R)$ jest dana zależnością [4]: $L(n,R) = B\sqrt{nR}$.

Tabela 1.

Porównanie algorytmów dla n=10.. 200

n		10	20	30	40	50	80	200
SFC_k	t	0.27	0.55	0.82	1.10	1.39	2.24	5.83
SFC_k-2opt	t	0.39	1.31	3.22	6.94	12.57	51.04	694.66
	λ	4.3	6.0	7.9	10.1	9.8	12.9	11.1
	τ	1.4	2.4	3.9	6.2	9.0	22.8	119.2
SFC_k-ls	t	0.76	1.57	2.35	3.15	4.06	6.70	37.68
	λ	2.4	4.0	6.2	6.9	8.2	9.9	10.3
	τ	2.8	2.8	2.9	2.9	2.3	3.0	6.5
SFC_k-ls-2opt	t	0.86	2.09	3.79	6.77	8.50	22.82	265.1
	λ	4.4	6.7	8.7	11.5	10.4	12.8	12.8
	τ	3.2	3.8	4.6	6.1	6.1	10.2	45.5
2opt(x4)	t	1.07	9.11	31.28	79.93	159.61	517.8	-
	λ	4.5	7.6	10.8	13.8	13.6	14.7	-
	τ	4.0	16.6	38.2	72.7	114.8	231.2	-

Empirycznie oszacowano (patrz [4]), że B mieści się w przedziale $[0.765, 0.765 + \frac{4}{n}]$. W tabeli 2 w rubryce β podano wartość obliczoną według zależności $\beta = 1/(1000\sqrt{n})$ umożliwiającą oszacowania średniego błędu algorytmu przybliżonego na podstawie stosunku β/B.

Dla zadań o rozmiarach 10...200 badane były wszystkie algorytmy z wyjątkiem heurystyki SFC_k-2opt-3opt. W przypadku tej heurystyki nie prowadzono systematycznych testów ze względu na znaczny wzrost czasu obliczeń spowodowany wzbogaceniem poprocesora o algorytm 3opt. Dla 10-elementowej próbki zadań dla n=50 (największy rozmiar zadania, dla którego stosowano tę heurystykę) otrzymano w wyniku dołączenia 3opt do SFC_k-2opt średnią poprawę wartości rozwiązania o 0.7% kosztem blisko 23-krotnego wzrostu czasu obliczeń.

Dla zadań o rozmiarach 200...2000, dla których rezultaty zamieszczono w Tabeli 2, badano tylko dwa algorytmy: SFC_k i SFC_k-ls.

W przypadku zadania 24/100/Krolak w Tabeli 3 podano rezultaty dla tych samych algorytmów, co w przypadku zadań z n=10.. 200. Tabela zawiera dodatkowo wartość procentowego błędu względnego w stosunku do rozwiązania optymalnego l_{opt}, tzn. $\Delta = (l-l_{opt})/l_{opt} \cdot 100\%$.

Dla zadania 318/Lin w Tabeli 4 zamieszczono rezultaty dla tych samych algorytmów, co w przypadku zadania 24/100/Krolak z wyjątkiem heurystyki 2opt(x4).

Tabela 2.

Porównanie algorytmów dla n=200.. 2000

n		200	400	1000	2000
SFC_k	t	5.7	11.5	29.2	59.2
	β	0.995	0.965	0.958	0.952
SFC_k-1s	t	39.6	94.8	434.9	1262.6
	λ	11.8	12.8	11.2	7.9
	τ	7.0	8.2	14.9	21.3
	β	0.842	0.841	0.851	0.876

Tabela 3.

Porównanie algorytmów dla zadania 24/100/Krolak (patrz [4]) n=100, l_{opt} = 21282.0

	l	t	λ	τ	Δ
SFC_k	29996.0	2.85	0	1	40.9
SFC_k-2opt	22520.1	147.85	24.9	51.9	5.8
SFC_k-1s	25602.1	14.08	14.6	4.9	20.3
SFC_k-1s-2opt	22613.8	82.02	24.6	28.8	6.3
2opt x 4	22431.1	1512.50	25.2	530.7	5.4

Tabela 4.

Porównanie algorytmów dla zadania 318/Lin (patrz [4]); n=318, l_{opt}= 43864.7

	l	t	λ	τ	Δ
SFC_k	57700.6	9.12	0	1	31.5
SFC_k-2opt	48094.7	4312.3	16.6	472.8	9.6
SFC_k-1s	49781.8	83.2	13.7	9.1	13.5
SFC_k-1s-2opt	47372.4	1466.7	17.9	160.8	8.0

4.3. Przeprowadzone eksperymenty obliczeniowe były nastawione głównie na badanie różnych poprocesorów dla heurystyki SFC_k. Ze względu na to, że nie prowadzono badań porównawczych z innymi algorytmami generującymi rozwiązanie początkowe, trudno ocenić wartość algorytmu SFC_k jako preprocesora. Tym niemniej można stwierdzić, że jest to algorytm bardzo szybki (dla n=2000 rozwiązanie uzyskuje się po 60 s, dla n=200 średni czas obliczeń nie przekracza 6s), dający jednak stosunkowo duży błąd względny. Dla dużych n błąd ten może być szacowany na ok. 25%, co jest wartością dużą w zestawieniu z innymi preprocesorami opisywanymi w literaturze, dla których średnie błędy nie przekraczają kilkunastu procent (patrz np. [4]). Szczególnie duże błędy zaobserwowano dla zadań 24/100/Krolak i 318/Lin - 41% i 32%.

Dodanie przeglądu lokalnego, tzn. użycie algorytmu SFC_k-1s, daje stosunkowo niewielki wzrost czasu obliczeń w stosunku do algorytmu SFC_k

(dla małych zadań około trzykrotny, dla n=200-sześciokrotny, dla n=1000, 2000 - kilkunastokrotny) i względną poprawę rozwiązania od kilku procent dla małych zadań do kilkunastu procent (10-13%) dla większych zadań.

Dołączenie poprocesora 2opt daje znacznie większy wzrost czasu obliczeń (dla n=200 około 120-krotny), ale względna procentowa poprawa jest b większa niż w przypadku poprocesora ls (o około 2-3%).

Użycie poprocesora ls-2opt daje niemal identyczną poprawę jak w przypadku poprocesora 2opt w krótszym czasie (dla większych zadań czas oblicz jest około dwukrotnie krótszy).

Wersja algorytmu 2opt(×4) daje zwykle nieco lepsze rozwiązanie niż algorytmy SFC_k-2opt, SFC_k-ls-2opt (zwykle poprawa rozwiązania jest lepsza o ok. 2%) w znacznie dłuższym czasie (dla n=100 odpowiednie czasy obliczeń są równe ok. 148 s dla SFC_k-2opt, ok. 82 s dla SFC_k-ls-2opt i ok. 1512 s dla 2opt(×4).

Użycie poprocesora 2opt-3opt (dla $n \leqslant 50$) daje niewielką poprawę rozwiązania (średnio ok. 1%) w stosunku do poprocesora 2opt przy znacznym wzroście czasu obliczeń.

LITERATURA

[1] Bartholdi III J.J., Platzman L.K.: An O(n log n) planar travelling salesman heuristic based on spacefilling curves, Opns Res. Letters 1 (1982) 121-125.

[2] Engelking R., Sieklucki K.: Geometria i topologia. Cz. II: Topologia PWN, Warszawa, 1980.

[3] Kryński S.: Uwagi o algorytmie SFC dla zadania komiwojażera na płaszczyźnie i testach komputerowych. Opr. wewn. IBS PAN ZPM-28-56/86, 198

[4] Lawler E.L., Lenstra J.K., Rinnoy Kan A.H.G., Shmoys D.B.: The Traveling Salesman Problem. John Wiley & Sons Ltd, 1985.

[5] Papadimitriou C.H.: The Euclidean travelling salesman problem is NP-complete, Theoret. Comput. Sci., 4 (1977) 237-244.

[6] Papadimitrion C.H., Steiglitz K.: Combinatorial Optimization. Algorithms and Complexity. Prentice Hall, Englewood Cliffs, NJ., 1982

[7] Sierpiński W.: Sur une courbe continué qui remplit toute une aire pl Biuletyn PAU, Kraków 1912, 462-478.

[8] Sysło M.M., Deo N., Kowalik J.S.: Discrete Optimization Algorithms wi Pascal Programs. Prentice Hall, Englewood Cliffs, NJ., 1983.

Recenzent:Doc.dr hab.inż.M.Zaborowski

Wpłynęło do Redakcji do 1988-04-30.

ПРИБЛИЖЁННЫЕ АЛГОРИТМЫ ДЛЯ ЗАДАЧИ КОММИВОЯЖЁРА НА ПЛОСКОСТИ С ИСПОЛЬЗОВАНИЕМ КРИВОЙ ЗАПОЛНЯЮЩЕЙ КВАДРАТ

Р е з ю м е

В работе рассмотрены приближённые алгоритмы для задачи коммивояжёра на плоскости , основанные на подходе предложенным в [1] , который состоит в употреблении кривой заполняющей квадрат для построения пути коммивояжёра и отличается низкой вычислительной сложностью , В работе дана общая идея этого подхода и предложены его модификации. Приведены результаты вычислительного эксперимента с двухэтапными приближёнными алгоритмами, в которых начальное приближение находится с употреблением кривой заполняющей квадрат с дальнейшим его улучшением с помощью разных алгоритмов , основанных на идеи локального поиска.

HEURISTIC ALGORITHMS FOR PLANAR TRAVELLING SALESMAN PROBLEM BASED ON THE SPACEFILLING CURVE CONSTRUCTION

S u m m a r y

A new heuristic problem of complexity O(n log n) for the planar travelling salesman has been proposed in [1]. The order of points in the travelling salesman tour produced by this heuristic is determined by the order of their appearonce on the Sierpiński spacefilling curve. Several simple modifications of this approach based on randomization, plane rotation and local search are discussed in the paper. Results of numerical experiments with there modifications are described. Various two step heuristics composed of initial tour construction step /preprocessor/ and tour improvement step /postprocessor/ are also investigated. In all of them the initial solution is constructed by the spacefilling curve heuristic. As postprocessor different combinations of local search algorithms are used. Numerical experiments with randomly generated Euclidean travelling salesman problems of size n=10 ... 2000 and several standard test problems are reported.