Mariusz MAKUCHOWSKI
Politechnika Wrocławska

DYNAMICZNY ALGORYTM WSTAWIEŃ ORAZ DYNAMICZNY ALGORYTM POPRAW DLA EUKLIDESOWEGO PROBLEMU KOMIWOJAŻERA

Streszczenie. W pracy proponuje się dwa algorytmy dedykowane euklidesowemu problemowi komiwojażera. Pierwszy z nich to algorytm konstrukcyjny bazujący na metodzie wstawień z elementami optymalizacji opartej na programowaniu dynamicznym. Drugi natomiast jest algorytmem popraw opartym na programowaniu dynamicznym. Proponowane algorytmy porównane są z ich klasycznymi odpowiednikami znanymi z literatury. Praca zawiera wyniki badań eksperymentalnych przeprowadzonych na literaturowych przykładach testowych.

DYNAMIC INSERTION ALGORITHM AND DYNAMIC IMPROVED ALGORITHM FOR EUCLIDEAN TRAVELING SALESMAN PROBLEM

Summary. The work deals with new two algorithms dedicated to the Euclidean traveling salesman problem. The first of them is a construction algorithm based on the insertion method with elements optimization based on dynamic programming. The second is the improvement algorithm based on dynamic programming. The proposed algorithms are compared with their classical equivalent known from the literature. The efficiency of the algorithms is tested on well–known literature benchmarks.

1. Wstęp

Jednym z najwcześniej, a zarazem najczęściej studiowanym problemem kombinatorycznym jest problem komiwojażera (TSP, ang. Traveling Salesman Problem) [1, 5]. Fakt ten wynika z trudności w wyznaczeniu rozwiązania optymalnego przy jednoczesnym bardzo prostym sformułowaniu tego zagadnienia. Problem TSP należy do klasy problemów NP-zupełnych [19].

W literaturze często można spotkać specjalne przypadki TSP, w którym nałożone są pewne naturalne ograniczenia. Często omawianym problemem jest Euklidesowy TSP, w skrócie ETSP. Wariant w którym miasta położone są w przestrzeni k wymiarowej a odległości między nimi wyznaczane są zgodnie z metryką euklidesową. Problem ETSP jest także zagadnieniem należącym do klasy problemów NP-zupełnych nawet dla przestrzeni dwuwymiarowej. Dla ETSP istnieje algorytm typu PTAS [2].

W praktyce do rozwiązywania dużych instancji problemów należących do klasy NP-trudnych stosuje się algorytmy przybliżone, od dedykowanych hueurystyk po metaheurystyki ogólnego zastosowania.

Wszystkie algorytmy przybliżone możemy podzielić na dwa typy: algorytmy konstrukcyjne oraz algorytmy popraw. Pierwsze z nich budują rozwiązanie problemu tylko na podstawie danych instancji. Z kolei algorytmy popraw działają na dostarczonych rozwiązaniach początkowych, modyfikując je w celu uzyskania jeszcze lepszych rozwiązań.

Dla TSP najczęściej opisywane algorytmy konstrukcyjne to metoda najbliższego sąsiada [21], metody wstawień [21], algorytm Christofidesa [4], metody krzywych wypełniających [18, 22], samoorganizujących się sieci neuronowych [15, 16], czy metoda podziału i cięć [17].

Z heurystyk optymalizacyjnych najpopularniejsze to algorytmy oparte na ruchach 2-Opt, 3-Opt [6, 12] oraz sąsiedztwie LK-Opt [13, 9]. Te ostatnie wykazują się największą efektywnością. Na ich bazie tworzone są bardziej rozbudowane metaheurystki takie jak algorytmy genetyczne [3, 8] symulowane wyżarzanie [14, 11], czy poszukiwania z zabronieniami [10, 7],

W pracy prezentuje się jeden algorytm konstrukcyjny DFI oraz jeden algorytm popraw D-Opt.

2. Problem komiwojażera - model matematyczny

Dany jest zbiór $C = \{1, 2, \ldots, n\}$ reprezentujący n miast oraz kwadratowa macierz d o rozmiarze $n \times n$. Wartość $d_{ij} \in \mathbb{N}^+$ definiuje odległość z miasta i do miasta j. Niech π oznacza permutację miast C. Zbiór wszystkich permutacji oznaczać będziemy przez Π. Permutacja π utożsamiana jest z cykliczną trasą kolejno odwiedzanych miast. Dla każdej trasy π możemy wyznaczyć jej długość, oznaczanej przez $D(\pi)$;

$$D(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(0)}. \tag{1}$$

Problem polega na wyznaczeniu permutacji π^* minimalizującej długość cyklu,

$$\pi^* \in \arg\min_{\pi \in \Pi} D(\pi). \tag{2}$$

3. Dynamiczny algorytm wstawień: DFI

Algorytm DFI (ang. Dynamic Farest Insert) bazuje na algorytmie FI (ang. Farest Insert) obecnie najbardziej efektywnym algorytmie konstrukcyjnym dedykowanym problemowi TSP, [21, 23].

Algorytm FI jest algorytmem, który sekwencyjnie rozbudowuje cykl o kolejne miasta. Cykl początkowy może składać się z np. pary najbardziej oddalonych od siebie miast lub losowym cyklem składającym się z jednego lub większej liczby miast. Następnie algorytm wybiera miasto najbardziej oddalone względem najbliższego miasta z cyklu. Po wybraniu miasta, następnie szukane jest dla niego najlepsze miejsce w cyklu czyli takie, które spowoduje najmniejszy przyrost długości trasy. Algorytm FI kończy swoje działanie po włączeniu wszystkim miast w cykl.

Algorytm DFI działa podobnie jak algorytm FI, jednak po dołączeniu kolejnego miasta następuje optymalizacja części cyklu. Optymalizacji podana jest ścieżka, o ustalonej długości p, będąca fragmentem cyklu w której dołączone miasto znajduje się na

środkowej pozycji. Długość ścieżki p ustalana jest na wartości 11 miast. Nowa wyznaczona ścieżka zwana DynaPath nie zmienia pierwszego i ostatniego miasta, przechodzi przez te same miasta co optymalizowany fragment cyklu i jest najkrótszą możliwą z takich ścieżek.

Ścieżka DynaPath wyznaczona jest przy pomocy metody programowania dynamicznego opisanej poniżej. Metoda wyznacza kolejno wartości dynamicznej tablicy $X(S,k)$ o rozmiarze $2^{p-1} \times (p-1)$ zawierającej długość optymalnej ścieżki z miasta 1 do miasta k, przechodzącej przez wszystkie miasta z podzbioru $S \cup \{1\}$. Przez miasto i należy rozumieć miasto znajdujące się na i-tej pozycji w optymalizowanej ścieżce. Wartości $X(S,k)$ wyznacza się dla przypadków, w których $\{k\} \in S$ według zależności:

$$X(S,k) = \begin{cases} d_{1,k} & \text{dla } S = \{k\} \\ min_{i \in S} X(S \setminus \{i\}) + d_{i,k} & \text{w przeciwnym przypadku} \end{cases}. \tag{3}$$

Na podstawie wyznaczonych wartości X optymalnych ścieżek odtworzona jest kolejność miast w optymalnej ścieżce DynaPath.

Złożoność obliczeniowa opisanej metody DynaPath wynosi $O(2^p p^2)$ a złożoność pamięciowa $O(2^p p)$, gdzie p jest długością optymalizowanej ścieżki. Metodę można stosować dla długość optymalizowanych fragmentów o rozmiarze $p = 20$. Prezentowana metoda DynaPath jest szczególnym przypadkiem ruchu k-opt, $k = p - 1$, w którym wszystkie usuwane krawędzie, w optymalizowanym rozwiązaniu, występują po sobie. DynaPath jest co prawda tylko szczególnym ruchem k-opt ale można go wykonywać dla stosunkowo dużego k. W literaturze pełne sąsiedztwo ruchów k-opt jest wykorzystywane dla $k \in \{2,3\}$ oraz tylko pewne ruchy k-opt, z zmiennym k w sąsiedztwie LK.

Złożoność proponowanego algorytmu DFI wynosi: $O(n^2 + n(2^p p^2))$ co przy ustalonym p równoważne jest $O(n^2)$.

4. Dynamiczny algorytm popraw: D-opt

Proponowany algorytm D-opt (ang. Dynamic Optimization) jest metodą lokalnych popraw. Wyznacza on w bieżącym rozwiązaniu kolejne fragmenty cyklu będące optymalizowane metodą DynaPath (opisaną powyżej).

W algorytmie należy podać przepis wyznaczania fragmentów poddawanych optymalizacji. Proponuje się ustalić długość tych fragmentów na $p = 11$ miast (czyli 10 krawędzi) oraz przesuwać optymalizowany fragment o $s = (p-1)/2$ miast.

Algorytm wykonuje kolejne optymalizacje, aż okno w którym znajduje się optymalizowana ścieżka, przesunie się po całej trasie w wybranym kierunku. Po każdej optymalizacji następuje przesunięcie okna o s miast w przypadku gdy optymalizacja była nieskuteczna, oraz cofnięcie okna o s miast w przypadku poprawy trasy. Cofnięcia te mają na celu dalszą optymalizację nowo powstałych fragmentów trasy.

Przy doborze parametrów p i s należy pamiętać, iż wydłużenie optymalizowanego fragmentu o 1 miasto skutkuje ponad dwukrotnym zwiększeniem czasu obliczeń natomiast s wpływa odwrotnie proporcjonalnie na ten czas.

Złożoność obliczeniowa proponowanego algorytmu wynosi $O(k \cdot 2^p p^2 n/s)$ gdzie k jest liczbą wykonanych optymalizacji. Ponieważ jednak trudno jest oszacować największą liczbę k, złożoność algorytmu D-opt jest problemem otwartym.

5. Badania numeryczne

Wszystkie eksperymenty numeryczne zrealizowano na komputerze MacBook Pro z procesorem Intel i5 2.5GHz pod systemem OSX 10.9.5. Zastosowane instancje testowe są literaturowymi benchmarkami pochodzącymi z biblioteki TSPLIB [20].

Dla wszystkich instancji typu euklidesowego dwu wymiarowego biblioteka TSPLIB zawiera także rozwiązania optymalne. Dla każdego rozwiązania π możemy więc wyznaczyć, liczony w stosunku do rozwiązania optymalnego π^*, błąd względny długości trasy:

$$\rho(\pi) = \frac{D(\pi) - D(\pi^*)}{D(\pi^*)} \cdot 100\%. \tag{4}$$

Ponieważ w algorytmach FI, DFI, FI+D-opt uruchamiano budowę rozwiązań z losowego miasta, wyniki kolejnych uruchomień nie były powtarzalne. Z powyższego powodu, daną instancję problemu rozwiązywano dziesięciokrotnie każdym badanym algorytmem. Na podstawie otrzymanej serii rozwiązań wyznaczono odpowiednio: wartość średnią błędu względnego ρ_{av}, odchylenie standardowe błędu względnego ρ_{dev} oraz całkowity czas działania 10 uruchomień algorytmu t_{sum}.

W pierwszym eksperymencie porównamy jakość proponowanego algorytmu DFI względem klasycznego odpowiednika FI. Aby algorytm DFI w przybliżeniu wykorzystywał 50% czasu działania na klasyczne operacje i 50% czasu na operacje związane z dynamiczną optymalizacją należy przyjąć parametr $p = 11$. W bieżącym badaniu przetestowany zostanie ponadto wpływ zmiany tego parametru. Ostatecznie algorytm DFI został uruchomiany w dwuch wersjach z $p \in \{7, 11\}$. Algorytm DFI z ustaloną wartość parametru p oznaczany jest przez DFI^p.

Niestety nie istnieje, możliwość bezpośredniego porównania algorytmu DFI i FI działającego dokładnie w tym samym czasie. Jest tak dlatego, gdyż algorytm DFI wykonuje wszystkie etapy algorytmu FI oraz dodatkowe elementy dynamicznej optymalizacji. Czas działania DFI jest więc zawsze większy niż w przypadku pojedynczego algorytmu FI. Zdecydowano się zatem stworzyć algorytm 2×FI polegający na dwukrotnym uruchomieniu algorytmu FI zwracający lepsze z dwóch otrzymanych rozwiązań.

Ze względu na dużą liczbę instancji testowych, szczegółowe wartości średniego błędu ρ_{av}, jego odchylenia standardowego ρ_{dev} oraz czas działania algorytmów zaprezentowane są dla co piątej instancji testowej. Wyniki omówionego badania zawarte są w Tabeli 1.

Proponowany algorytm konstrukcyjny DFI, jest znacznie lepszy niż najlepszy algorytm konstrukcyjny znany z literatury FI. Porównując DFI^7 z FI, wynika iż, kosztem 10% zwiększenia czasu, (59s dla FI i 64s dla DFI) średni błąd generowanych rozwiązań jest blisko o 1/3 mniejszy (12.65% dla FI i 9.10% dla DFI). Ponadto zwiększając dokładność kroku optymalizacji algorytm DFI^{11} przy czasie obliczeń 101s czyli 70% dłuższym niż czas FI dostarcza rozwiązań z średnim błędem na (7.18%) to jest blisko o połowę mniejszym niż wzorcowy algorytm FI.

Algorytm 2×FI działa 116s i dostarcza rozwiązania z średnim błędem 11.69%. Proponowany w pracy algorytmy DFI^7 i DFI^{11} dostarczają rozwiązania z mniejszym średnim błędem w krótszym czasie niż algorytm 2×FI.

Znane są w literaturze algorytmy dające lepsze rozwiązania niż algorytm DFI, jednakże są to algorytmy popraw, które modyfikują rozwiązania otrzymane na swoim

Tabela 1

Porównanie efektywności algorytmu FI i DFI

instancja	FI			2×FI			DFI^{7}			DFI^{11}		
	ρ_{avg}	ρ_{dev}	t_{sum}	ρ_{avg}	ρ_{dev}	t_{sum}	ρ_{avg}	ρ_{dev}	t_{sum}	ρ_{avg}	ρ_{dev}	t_{sum}
eil51	7.28	2.79	0.1	5.66	1.46	0.1	3.64	1.07	0.1	2.68	1.07	0.1
rat99	10.55	1.57	0.1	8.67	2.26	0.1	7.16	1.80	0.1	5.01	1.68	0.2
kroE100	5.80	1.57	0.1	5.80	1.57	0.1	4.48	0.75	0.1	2.85	1.01	0.2
pr124	7.87	2.70	0.1	5.27	1.66	0.1	5.03	2.47	0.1	2.79	0.52	0.2
ch150	9.69	2.77	0.1	8.48	2.66	0.1	5.98	0.92	0.1	5.33	1.88	0.3
rat195	12.69	1.86	0.1	10.96	1.42	0.1	10.31	1.66	0.1	7.55	0.83	0.3
tsp225	9.84	2.00	0.1	8.97	0.91	0.1	7.96	0.84	0.1	6.68	1.16	0.3
lin318	8.67	1.47	0.1	8.28	0.95	0.1	8.21	1.95	0.1	5.90	0.90	0.5
d493	9.81	1.50	0.2	8.77	1.44	0.2	7.05	0.87	0.2	5.48	0.87	0.7
u724	12.45	1.17	0.2	12.20	1.17	0.2	9.57	1.02	0.2	7.14	0.91	1.0
pcb1173	16.18	0.74	0.3	15.66	0.82	0.5	13.80	0.83	0.3	11.36	0.92	1.6
fl1400	6.60	1.48	0.4	5.40	0.38	0.6	4.27	0.97	0.4	2.52	0.53	2.0
u1817	37.25	0.94	0.5	36.36	0.89	0.9	17.85	0.65	0.6	15.00	0.47	2.7
pr2392	14.13	0.84	0.9	13.66	0.49	1.6	11.71	0.74	1.0	9.72	0.67	3.7
rl5934	21.28	0.48	5.1	20.78	0.71	9.9	18.84	0.48	5.6	16.26	0.66	12.7
d18512	12.25	0.12	50.6	12.21	0.12	100.6	9.83	0.18	54.8	8.56	0.13	75.4
wszystko	12.65	1.50	59.2	11.69	1.18	115.5	9.10	1.08	64.2	7.18	0.89	101.7

wejściu. Wejściowe rozwiązanie jest najczęściej rozwiązaniem losowym lub wygenerowane algorytmem konstrukcyjnym, np. FI. Autorowi wydaje się, że zastosowanie algorytmu DFI jako algorytmu generującego rozwiązanie startowe jest bardzo obiecujące i może podnieś ogólną efektywność takich duetów.

W drugim eksperymencie porównano metodę lokalnej optymalizacji D-opt z najpopularniejszą literaturową metodą lokalnej optymalizacji 2-opt. Proponowane algorytmy D-opt^{11} i D-opt^{13} dostarczają w krótszym czasie lepsze rozwiązania niż algorytm 2-opt.

Jako całość algorytm FI+D-opt wypada słabo na tle metaheurystyk takich jak poszukiwanie z zabronieniami, symulowane wyżarzanie w szczególności tych bazujących na sąsiedztwie zaproponowanym przez Lin i Kernighan'a. W szczególności pakiet Concorde'a autorstwa Cook'a rozwiązuje wszystkie testowane instancje w sposób optymalny [1].

Pomimo iż, algorytm FI + D-opt jest słabszy nawet niż DFI, samą metodę optymalizacji D-opt można uznać za przydaną i stosować ją np. dodatkowo po zakończonych krokach optymalizacyjnych literaturowych algorytmów. Zabieg taki może potencjalnie zwiększyć ich efektywność rozumianą przez skrócenie czasu działania przy generowaniu tak samo dobrych rozwiązań, lub poprzez otrzymanie lepszych rozwiązań w podobnym czasie. Przykładowo dodając procedurę D-opt^{11} do duetu algorytmów FI + 2-opt, kosztem niewielkiego zwiększenia czasu około 5% uzyskuje się zauważalnie lepsze rozwiązania; ze średnim błędem 8.60% zamiast 9.75%.

Tabela 2

Porównanie efektywności algorytmu 2-opt i D-opt

instancja	FI+2-opt			FI+D-opt^{11}			FI+D-opt^{13}			FI+2-opt + D-opt^{11}		
	ρ_{avg}	ρ_{dev}	t_{sum}	ρ_{avg}	ρ_{dev}	t_{sum}	ρ_{avg}	ρ_{dev}	t_{sum}	ρ_{avg}	ρ_{dev}	t_{sum}
eil51	5.49	1.25	0.1	3.92	0.66	0.1	3.31	0.94	0.2	3.92	0.66	0.1
rat99	8.02	1.91	0.2	6.78	1.73	0.2	5.72	1.95	0.3	6.44	2.00	0.1
kroE100	4.72	1.56	0.1	3.92	1.50	0.1	3.75	1.37	0.3	4.05	1.47	0.1
pr124	7.12	3.01	0.1	6.90	2.57	0.1	6.42	2.62	0.3	6.52	2.63	0.1
ch150	8.69	2.79	0.1	7.26	2.35	0.2	6.57	2.28	0.4	7.02	2.19	0.2
rat195	11.21	1.21	0.1	10.03	1.44	0.2	9.41	1.40	0.5	9.48	1.25	0.2
tsp225	9.12	2.09	0.2	7.52	1.57	0.2	7.09	1.50	0.5	7.54	1.83	0.2
lin318	7.69	1.09	0.2	7.27	1.29	0.2	6.97	1.25	0.7	7.05	1.01	0.2
d493	7.81	1.07	0.2	7.45	1.34	0.3	7.16	1.16	1.2	6.87	0.91	0.4
u724	9.74	0.79	0.3	9.24	0.87	0.4	8.90	0.96	1.5	8.59	0.81	0.5
pcb1173	14.14	0.92	0.7	14.01	0.73	0.7	13.59	0.60	2.4	13.02	0.78	1.1
fl1400	4.90	1.06	1.0	4.82	1.33	0.8	4.57	1.31	2.6	4.12	1.04	1.4
u1817	17.59	0.63	1.7	17.87	0.75	1.5	17.30	0.62	5.8	16.25	0.66	2.3
pr2392	12.20	0.57	2.7	12.02	0.92	1.7	11.55	0.96	5.2	11.07	0.62	3.5
rl5934	17.28	0.84	21.3	19.67	0.44	6.7	19.23	0.51	13.8	16.59	0.81	22.9
d18512	10.30	0.14	194.5	9.59	0.10	56.2	9.20	0.12	84.6	9.02	0.14	200.1
wszystko	9.75	1.31	223.7	9.27	1.22	69.7	8.80	1.22	120.3	8.60	1.18	233.4
czas bez FI	–	–	164.5	–	–	10.5	–	–	61.1	–	–	174.5

6. Podsumowanie

Głównym wynikiem pracy jest opracowany algorytm konstrukcyjny DFI. Jest on wydajniejszy niż FI, najlepszy znany algorytm konstrukcyjnym dedykowanych problemowi TSP. Dostarcza on rozwiązań z średnim błędem blisko dwukrotnie mniejszym niż, algorytm FI. Czas działania proponowanego algorytmu jest tylko nieznacznie większy niż dla FI. Algorytm DFI jest szybszy i dający lepsze rozwiązania niż dwukrotne uruchomienie FI (dostarczającego lepsze z dwóch różnych wygenerowanych rozwiązań).

W stosunku do najlepszych znanych algorytmów popraw algorytm DFI daje słabe rozwiązania. Ponieważ jednak algorytmy popraw na wejściu dostają pewne rozwiązanie, które następnie sukcesywnie jest poprawiane, algorytm DFI można zastosować do generacji startowego rozwiązania (lub rozwiązań) w algorytmach tego rodzaju.

Drugi prezentowany algorytm optymalizacyjny D-opt jest szybszy i generuje lepsze rozwiązania niż algorytm 2-opt. Zastosowanie optymalizacji D-opt po optymalizacji 2-opt przynosi również znaczną poprawę rozwiązania.

Jednakże w stosunku do najlepszych istniejących algorytmów popraw daje słabe rozwiązania. Jego właściwe zastosowanie to dodatkowa optymalizacja wykonywana razem z optymalizacją 2-opt w algorytmach popraw wykorzystujących tę metodę.

LITERATURA

1. Applegate D., Bixby R. Chvatal V., Cook W.: The Traveling Salesman Problem. A Computational Study, Princeston Series in Applied Mayhematics, 2006.
2. Arora S.: Polynomial-time approximation schemes for Euclidean traveling salesman and other geometric problems, Journal of the ACM 45, 1998, p. 753-–782.
3. Chatterjee S., Carrera C., and Lynch L.: Genetic algorithms and traveling salesman problems, European Journal of Operational Research, 93, 1996, p. 490-510.
4. Christofides N.: Worst-case analysis of a new heuristic for the travelling salesman problem, Report No. 388, GSIA, Carnegie-Mellon University, Pittsburgh, PA, 1976.
5. Cook W.: In Pursuit of the Traveling Salesman Mathematics at the Limits of Computation, Princeton University Press, 2014.
6. Croes A.: A method for solving traveling salesman problems, Operations Res. 6, 1958, p. 791–812.
7. Fiechter C.N.: A parallel tabu search algorithm for large traveling salesman problems, Disc. Applied Math. 51, 1994, p. 243–267.
8. Grefenstette J.J., Gopal R., Rosmaita B., Van Gucht D.: Genetic algorithms for the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of an International Conference on Genetic Algorithms and Their Applications. Carnegie-Mellon University, 1985, p. 160–168.
9. Helsgaun K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, European Journal of Operational Research 126 (1), 2000, p. 106-–130.
10. Knox J.: Tabu search performance on the symmetric traveling salesman problem, Computers & Ops. Res. 21, 1994, p. 867–876.
11. Lee J., Choi M. Y.: Optimization by multicanonical annealing and the traveling salesman problem, Physical Review E 50, 1994, p. 651–654.
12. Lin S.: Computer solutions of the traveling salesman problem, Bell Syst. Tech. J. 44, 1965, p. 2245–2269.
13. Lin S., Kernighan B.: An Effective Heuristic Algorithm for the Traveling-Salesman Problem, Operations Research 21 (2), 1973, p. 498-–516.
14. Malek, M., Guruswamy, M., Pandya, M., Owens H.: Serial and Parallel simulated annealing and tabu search algorithms for the traveling salesman problem, Annals of Operations Research, 21, 1989, p. 59–84.
15. Matsuyama, Y.: Self-organization neural networks and various Euclidean traveling salesman problems, Systems and Computers in Japan, 23(2), 1992, p. 101–112.
16. Modares, A., Somhom, S., Enkawa, T.: A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems, International Transactions in Operational Research, 6, 1999, p. 591–606.

17. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the solution of large-scale traveling salesman problems, SIAM Review, 33, 1991, p. 60–100.
18. Platzman L.K. and Bartholdi J.J. III.: Spacefilling curves and the planar travelling salesman problem. Journal of the ACM, Vol.36, 1989, p. 719–737.
19. Reinelt G.: The Traveling Salesman: Computational Solutions for TSP Applications, Lectures Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 1994.
20. Reinhelt, G.: Set of TSP instances: TSPLIB,
Internet: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.
21. Rosenkrantz D.J., Stearns R.E., Lewis P.M. II,: An Analysis of Several Heuristics for the Traveling Salesman Problem, SIAM J. Comput. 6, 3, 1977, p. 563–581.
22. Steele J.M.: Efficacy of spacefilling heuristics in euclidean combinatorial optimization. Operations Reserch Letters, Vol.8, 1989, p. 237–239.
23. Sysło M.M., Deo N., Kowalik J.: Algorytmy dyskretnej optymalizacji. PWN, Warszawa 1993.