Algorytmy metaheurystyczne
Sprawozdanie 1

Autorzy:

Wojciech Gabryelski 261721

Karol Dzwonkowski 247373

Szczegóły implementacyjne:

Język programowania: Java,

Metoda kompilacji: Kompilacja z poziomu Ide (jetbrain)

Kompilator: JDK17.

Parametry sprzętowe:

System operacyjny: Windows 10,

Model procesora i7-4790K, 4-rdzenie fizyczne, 8-rdzeni logicznych, taktowanie 4.4GHz, pamięć ram 16GB DDR3.

Instancje:

Instancje wykorzystane w badaniach obejmują problemy dla konfiguracji TSP i ATSP z biblioteki TSPLIB, które wyrażone są za pomocą listy współrzędnych wierzchołków miast, macierzy odległości lub macierzy trójkątnych dla problemu TSP. Niektóre instancje zostały wygenerowane losowo, przy użyciu metody nextInt() lub nextDouble() (w przypadku generowania instancji euklidesowych) wbudowanych w klasę Random biblioteki standardowej Javy. Wygenerowane zostały zarówno macierze symetryczne jak i asymetryczne, których wartości znajdujące się na przekątnej są równe zero. Rozmiar poszczególnych instancji (liczba miast) zawiera się w przedziale od 30 do 6000 miast.

Eksperymenty:

1. Badanie wpływu czynników na jakość algorytmów.

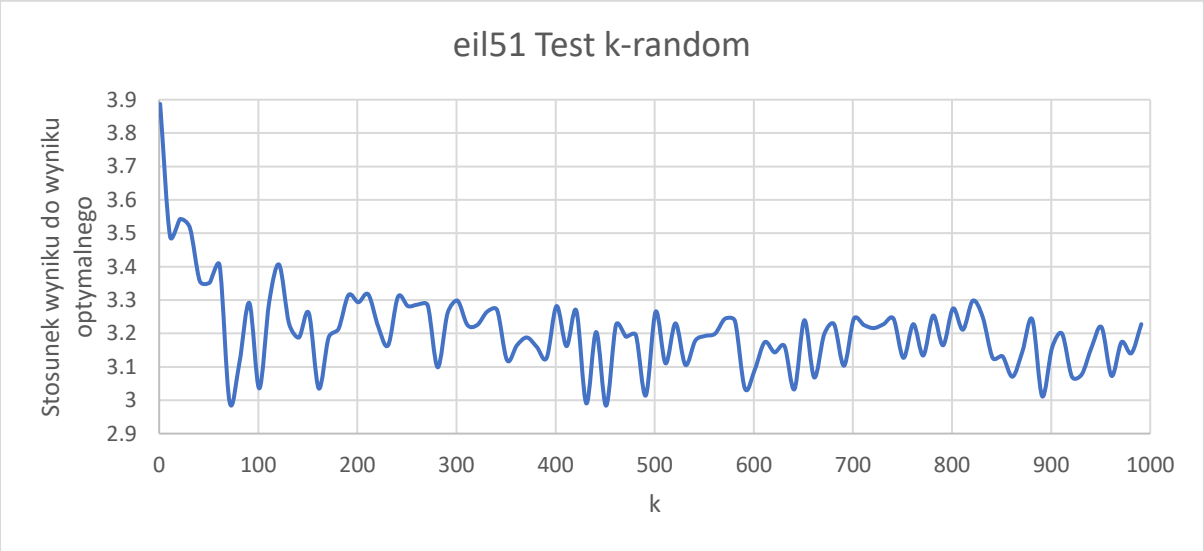
W części eksperymentów odnotowywany jest współczynnik jakości równy stosunkowi otrzymanego wyniku do optymalnego. Optymalne wyniki zostały zaczerpnięte ze strony http://comopt.ifi.uni-heidelberg.de/ .

Wpływ parametru k na jakość algorytmu k-random.

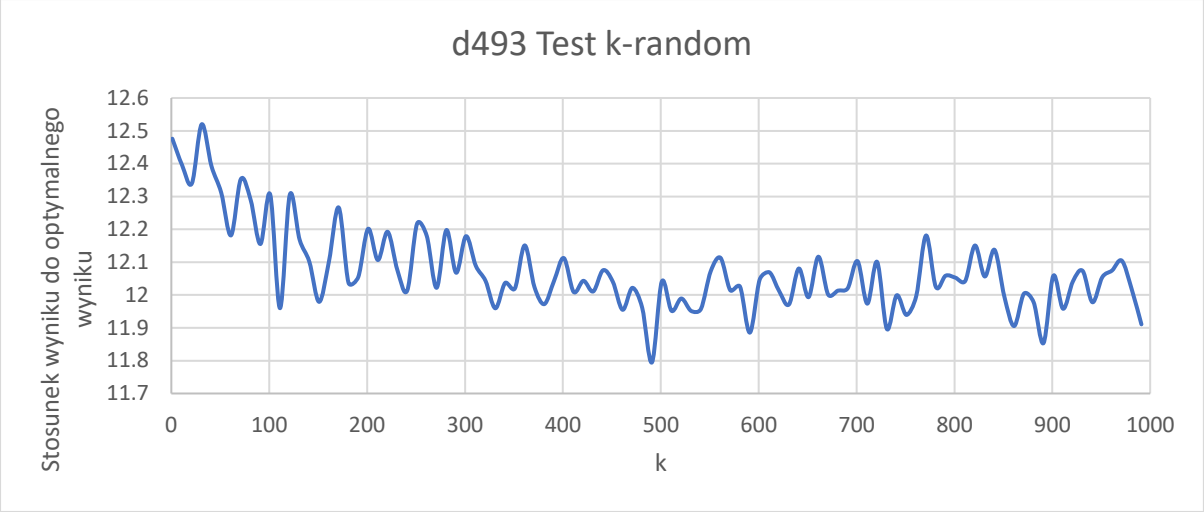
Celem badania jest sprawdzenie jak wpływa współczynnik k na jakość algorytmu dla różnej wielkości instancji.

Dla algorytmu nie ma znaczenia czy instancje są symetryczne czy asymetryczne.

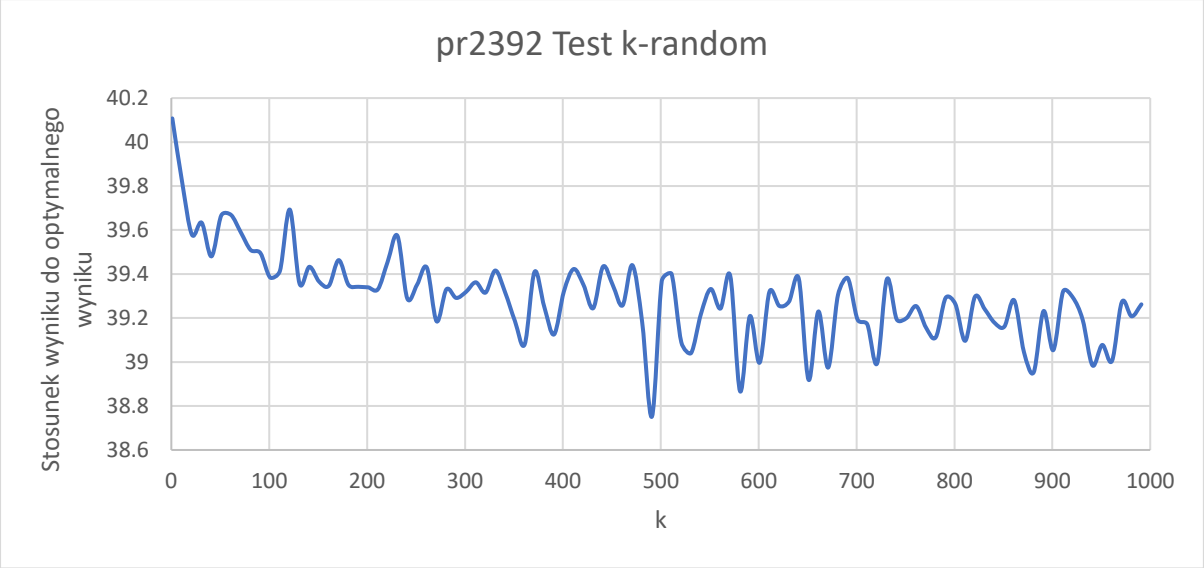
Można zauważyć, że wraz ze wzrostem parametru k otrzymujemy coraz bardziej zbliżone wartości dystansu zwróconego przez algorytm w porównaniu do optymalnych rozwiązań, jednak wartości nie ulegają dużej zmianie. Przeprowadzając badania na coraz większych instancjach widać wyraźnie, że algorytm zwraca coraz słabsze rozwiązania. Na wykresach oś pozioma opisuje zmianę parametru k zaś oś pionowa stosunek otrzymanego wyniku do optymalnego rozwiązania.



Wykres 1.1 Zależność stosunku otrzymanego rozwiązania do rozwiązania optymalnego od liczby losowań w algorytmie k-random dla instancji o rozmiarze 51



Wykres 1.2 Zależność stosunku otrzymanego rozwiązania do rozwiązania optymalnego od liczby losowań w algorytmie k-random dla instancji o rozmiarze 493



Wykres 1.3 Zależność stosunku otrzymanego rozwiązania do rozwiązania optymalnego od liczby losowań w algorytmie k-random dla instancji o rozmiarze 2392

Wpływ parametru k na jakość algorytmu k-opt.

Celem badania jest zbadanie wpływu parametru k na jakość algorytmu k-opt. Badania zostaną przeprowadzone dla różnej wielkości instancji, zachowując tą samą permutację początkową, przy czym wyróżnimy instancje symetryczne i asymetryczne. Zbadane zostaną k nie większe od 5 gdyż złożoność obliczeniowa algorytmu byłaby zbyt duża.

Parametr k nie determinuje jakości algorytmu, ponieważ najkrótsze znalezione trasy mogą być najkrótsze w swoim otoczeniu, jednak nieoptymalne dla całej instancji. Niemniej jednak, można zauważyć tendencję do polepszania się algorytmu wraz ze wzrostem k (dla badanego przedziału $k \in \{2,3,4,5\}$). Ponadto czas działania algorytmu k-opt rośnie wraz ze wzrostem wartości parametru k. Analizując wyniki znaleźć można anomalie charakteryzujące się słabą jakością rozwiązania. Wynikają one z niefortunnej permutacji początkowej, co może doprowadzić do znalezienia dużo gorszego rozwiązania końcowego.

Instancja	k	Jakość rozwiązania	Czas [ms]
Instancje asymetryczne			
Ftv33	2	1.440	26
	3	1.045	163
	4	1.075	2370
	5	1.056	56985
P43	2	1.0005	57
	3	1.0007	684
	4	1	12657
	5	1	397928
Ft53	2	1.860	41
	3	1.064	747
	4	1.074	23473
	5	1.014	1446012
Ftv170	2	4.013	455
	3	1.202	69807
Instancje symetryczne			
Bays29	2	1.006	11
	3	1.012	50
	4	1.006	403
	5	1	6338
Berlin52	2	1.055	22
	3	1.031	302
	4	1.005	9247
	5	1.038	319121
Ch130	2	1,091	161
	3	1,013	9007

Tabela 1.4 Wyniki badań algorytmu k-opt dla różnych instancji, sprawdzamy czas działania algorytmu oraz stosunek rozwiązania do rozwiązania optymalnego.

Wpływ permutacji początkowej na algorytm 2-opt'a

Celem badania jest zbadanie wpływu początkowej instancji na jakość algorytmu 2-opt.

Instancja	Losowa permutacja			Wynik algorytmu najbliższego sąsiada		
	Czas	Odległość	Jakość	Czas	Odległość	Jakość
Instancje Symetryczne						
Berlin52	12	8172	1.083	21	7696	1.02
D198	78	16608	1.052	92	15952	1.01
P654	2460	35977	1.038	1171	35873	1,035
Pr2392	209978	421012	1.113	43102	397960	1,052
Instancje Asymetryczne						
Ftv33	13	2168	1.685	16	1507	1,171
Ftv55	22	3425	2.129	21	1948	1,211
Ftv170	130	10215	3.707	123	3536	1,283
Rbg443	1279	5625	2.068	339	3849	1,415

Tabelka 1.5 Wyniki działania algorytmu 2-opt na różnych instancjach początkowych.

Na podstawie powyższej tabelki możemy wywnioskować, że czas działania algorytmu k-opt oraz odległość będąca wynikiem tego algorytmu, dla początkowego rozwiązania będącego wynikiem

zwróconym przez algorytm najbliższego sąsiada, są zwykle mniejsze (szczególnie dla instancji o większym rozmiarze) niż czas działania i wynik algorytmu k-opt zainicjowanego losową permutacją miast.

2. Porównanie algorytmów między sobą.

Celem badania jest porównanie algorytmów parami między sobą. Do przenalizowania testów użyto testu Wilcoxona. Porównywane są ze sobą po kolei algorytmy:

1) 2-opt & k-random,
2) 2-opt & algorytm najbliższego sąsiada,
3) Algorytm najbliższego sąsiada & k-random.

Pierwszy test został przeprowadzony dla 18-tu instancji problemu asymetrycznego, natomiast drugi dla 10-ciu instancji problemu symetrycznego. Algorytm k-random był uruchamiany dla parametru k=100.

Numer badania	Wynik testu Wilcoxona
Instancje asymetryczne	
1	0.03125
2	0.03125
3	0.03125
Instancje symetryczne	
1	$7.629394 * 10^{-6}$
2	$1.045227 * 10^{-3}$
3	$7.629394 * 10^{-6}$

Tabela 2.1 Wyniki analizy przeprowadzonej za pomocą testu Wilcoxona.

Wartości z powyższej tabeli są p-wartościami dla testu Wilcoxona i określają one, jakie jest prawdopodobieństwo tego, że testowana zależność zaszła przypadkowo (w tym przypadku jest to zależność mówiąca o tym, że testowane algorytmy różnią się istotnie, a dokładniej mediany ich rezultatów). Widzimy, że powyższe wartości są małe, zatem z dużym prawdopodobieństwem możemy stwierdzić, że wszystkie testowane algorytmy są różne.

Instancja	k-random ($k = 5 * 10\hat{}4$)			Najbliższy sąsiad			2-opt		
	Czas	Odległość	Jakość	Czas	Odległość	Jakość	Czas	Odległość	Jakość
Instancje Symetryczne									
Berlin52	61	23176	3.072	11	8164	1.082	12	8172	1.083
D198	149	155594	9.86	66	17662	1.119	78	16608	1.052
P654	632	1841770	53.164	669	42960	1.24	2460	35977	1.038
Pr2392	4069	$1.467 * 10^7$	38.807	21809	458264	1.2012	209978	421012	1.113
Instancje Asymetryczne									
Ftv33	68	3101	2.411	7	1590	1.236	13	2168	1.685
Ftv55	69	5812	3.614	13	1948	1.211	22	3425	2.129
Ftv170	150	23127	8.394	67	3582	1.3	130	10215	3.707
Rbg443	359	7689	2.826	304	3858	1.418	1279	5625	2.068

Tabela 2.2 Wyniki algorytmów wraz z czasem ich działania i stosunkiem wyniku otrzymanego do optymalnego

Z tabelki możemy wywnioskować, że we wszystkich instancjach algorytm k-random wypadł najgorzej. Dla instancji symetrycznych najdokładniejszy jest algorytm 2-opt, jednakże działał on dłużej aniżeli algorytm najbliższego sąsiada. Dla instancji asymetrycznych zdecydowanie najlepsze wyniki (najdokładniejsze i w najkrótszym czasie) dla większości instancji zwrócił algorytm

najbliższego sąsiada. Podsumowując najgorzej w badaniach wypadł algorytm k-random, zaś 2-opt oraz algorytm najbliższego sąsiada są silnie uzależnione od badanej instancji. W tabelce można zauważyć anomalię taką jak badana instancja Rbg443, która zaburza trendy widniejące w tabelce takie jak spadek jakości algorytmu k-random dla coraz liczniejszych instancji, spowodowane jest to charakterystyką tej instancji. Mianowicie pojawiają się w niej miasta o podobnej odległości.

Analiza algorytmów

Algorytm k-opt

Przeanalizujmy złożoność obliczeniową algorytmu k-opt. Algorytm bierze pod uwagę wszystkie te przypadki, w których przerywanych jest k niesąsiadujących ze sobą krawędzi. Wyznaczmy liczbę wszystkich układów takich krawędzi. Ponumerujmy kolejne krawędzie w początkowym rozwiązaniu numerami od 1 do n. W celu uniknięcia powtórzeń przyjmijmy, że po przerwaniu i-tej krawędzi możemy przerwać tylko krawędzie o numerach większych od i. Każda przerwana krawędź musi mieć numer o co najmniej 2 większy od poprzedniej usuniętej krawędzi. Niech a_l oznacza liczbę takich układów przerywanych krawędzi, w których krawędź o największym numerze ma numer równy l. Wówczas dla $j < n$ zachodzi

$$a_l = [x^l]\sum_{i=1}^{\infty} x^i \left(\sum_{j=2}^{\infty} x^j\right)^{k-1} = [x^l]\left(\sum_{i=1}^{\infty} x^i\right)\left(\sum_{i=2}^{\infty} x^i\right)^{k-1} = [x^l]\left(\left(\sum_{i=2}^{\infty} x^i\right)^{k} - x\left(\sum_{i=2}^{\infty} x^i\right)^{k-1}\right),$$

gdyż na początek wybieramy krawędź o dowolnym numerze, a następnie wybieramy $k-1$ krawędzi, z których każda kolejna ma numer o co najmniej 2 większy od numeru poprzedniej krawędzi ($[x^l]$ oznacza współczynnik przy x^l w podanym wielomianie). Zauważmy, że w przypadku, gdy numer ostatniej krawędzi wynosi n, pierwsza wybrana krawędź musi mieć numer większy niż 1, gdyż krawędź o numerze 1 sąsiaduje z krawędzią o numerze n. Wobec tego

$$a_n = [x^n]\sum_{i=2}^{\infty} x^i \left(\sum_{j=2}^{\infty} x^j\right)^{k-1} = [x^n]\left(\sum_{i=2}^{\infty} x^i\right)\left(\sum_{i=2}^{\infty} x^i\right)^{k-1} = [x^n]\left(\sum_{i=2}^{\infty} x^i\right)^{k}.$$

Ostatnia krawędź musi mieć numer nie większy od n. W związku z tym, korzystając z własności splotu oraz z własności współczynnika dwumianowego Newtona $\binom{-a}{b} = (-1)^b\binom{a+b-1}{b}$, łączna liczba układów niesąsiadujących krawędzi jest równa

$$\sum_{j=1}^{n} a_j = \sum_{j=1}^{n-1}[x^j]\left(\left(\sum_{i=2}^{\infty} x^i\right)^{k} - x\left(\sum_{i=2}^{\infty} x^i\right)^{k-1}\right) + [x^n]\left(\sum_{i=2}^{\infty} x^i\right)^{k} =$$

$$= \sum_{j=1}^{n}[x^j]\left(\sum_{i=2}^{\infty} x^i\right)^{k} - \sum_{j=1}^{n-1}[x^j]x\left(\sum_{i=2}^{\infty} x^i\right)^{k-1} =$$

$$= [x^n]\left(\sum_{i=2}^{\infty} x^i\right)^{k}\left(\sum_{i=0}^{\infty} x^i\right) - [x^{n-1}]x\left(\sum_{i=2}^{\infty} x^i\right)^{k-1}\left(\sum_{i=0}^{\infty} x^i\right) =$$

$$= [x^n]\left(\frac{x^2}{1-x}\right)^{k}\left(\frac{1}{1-x}\right) - [x^{n-1}]x\left(\frac{x^2}{1-x}\right)^{k-1}\left(\frac{1}{1-x}\right) =$$

$$= [x^n]\frac{x^{2k}}{(1-x)^{k+1}} - [x^{n-1}]\frac{x^{2k-1}}{(1-x)^k} = [x^{n-2k}](1-x)^{-(k+1)} - [x^{n-2k}](1-x)^{-k} =$$
$$= (-1)^{n-2k}\binom{-(k+1)}{n-2k} + (-1)^{n-2k}\binom{-k}{n-2k} =$$
$$= (-1)^{n-2k}(-1)^{n-2k}\binom{k+1+n-2k-1}{n-2k} + (-1)^{n-2k}(-1)^{n-2k}\binom{-k+n-2k-1}{n-2k} =$$
$$= \binom{n-k}{n-2k} + \binom{n-k-1}{n-2k} = \binom{n-k}{n-2k}\left(1+\frac{k}{n-k}\right) = \frac{n}{n-k}\binom{n-k}{k}.$$

Po przerwaniu k krawędzi otrzymujemy k niezłączonych ze sobą dróg (zbiorów krawędzi łączących kolejne miasta). Każdą drogę utożsamiamy z miastami, przez które ta droga przechodzi. Ustalmy początkowe miasto, od którego komiwojażer zaczyna swoją podróż. Podróżując drogą, do której należy początkowe miasto, komiwojażer musi w końcu dojść do miasta, które jest połączone krawędzią z tylko jednym miastem, gdyż druga krawędź wychodząca z tego miasta została przerwana. Wówczas to miasto musi być połączone z jednym z dwóch krańcowych miast innej drogi, aby komiwojażer mógł podróżować dalej. Dla każdej z k dróg mamy więc dwie możliwości wybrania krańcowego miasta, do którego komiwojażer uda się bezpośrednio wychodząc z miasta należącego do innej drogi. Wobec tego, dla każdego układu k przerwanych krawędzi, mamy 2^k możliwych opcji wybrania k miast, do których komiwojażer uda się bezpośrednio z miasta należącego do innej drogi. W przypadku symetrycznego problemu komiwojażera nie ma znaczenia, w którą stronę uda się komiwojażer, zatem w tej sytuacji możemy się ograniczyć do 2^{k-1} możliwości.

Na koniec pozostaje wybór kolejności dróg, przez które będzie przechodził komiwojażer (możemy założyć, że pierwszą drogą, przez którą przechodzi komiwojażer jest ta, do której należy początkowe miasto). Pierwsza droga jest utożsamiana z początkowym miastem, więc jest ona ustalona. Pozostaje wybór $k-1$ dróg. Istnieje dokładnie $(k-1)!$ różnych permutacji tych dróg. Wobec tego dla każdego układu przerywanych krawędzi i każdego układu początkowo odwiedzanych miast każdej drogi istnieje $(k-1)!$ możliwości wybrania kolejności dróg.

Dla każdego układu kolejno odwiedzanych miast czas obliczania w sporządzonym algorytmie długości drogi, którą pokona komiwojażer, jest stały (niezależny od parametrów k i n), a czas kopiowania elementów tablic jest $O(k)$. Kopiowanie tablic znajduje się we wszystkich etapach (wyznaczanie przerywanych krawędzi, wyznaczanie początkowo odwiedzonych miast każdej drogi oraz wyznaczanie permutacji dróg), zatem złożoność obliczeniowa tego algorytmu wynosi

$$O\left(\frac{n}{n-k}\binom{n-k}{k}\left(2^k\left((k-1)!\,k+k\right)+k\right)\right) = O\left(\frac{n}{n-k}\binom{n-k}{k}2^k k!\right),$$

co po ustaleniu parametru k odpowiada złożoności $O(n^k)$.

Powyższy algorytm jest wywoływany, dopóki wartość funkcji celu ulega poprawie, zatem rzeczywista złożoność algorytmu może być większa.

Złożoność pamięciowa algorytmu k-opt jest równa $O(n^2)$, gdyż struktura, w której są przechowywane odległości, jest macierzą o rozmiarze $n \times n$, a wysokość drzewa rekursji dla każdej rekurencji użytej w algorytmie jest nie większa od k i w każdym węźle drzewa złożoność alokowanej pamięci jest równa $O(1)$.

Algorytm k-random

Złożoność obliczeniowa algorytmu k-random wynosi $O(k * n)$, gdyż wyznaczanie losowego rozwiązania ma złożoność $O(n)$. Losowanie zaś generuje k takich permutacji, stąd powyższa

złożoność. Złożoność pamięciowa tego algorytmu jest równa $O(n^2)$, gdyż struktura, w której są przechowywane odległości, jest macierzą o rozmiarze $n \times n$, a w algorytmie nie są wykorzystywane struktury o większym rozmiarze.

Algorytm najbliższego sąsiada

Złożoność algorytmu najbliższego sąsiada, w którym uwzględniamy tylko jedno najbliższe miasto i w którym uwzględniamy wszystkie możliwe punkty startowe, jest równa $O(n^3)$, ponieważ dla każdego kolejnego miasta (oprócz ostatniego) przeszukujemy listę dotąd nienapotkanych miast w poszukiwaniu najbliższego, zatem dla danego miasta początkowego złożoność wyznaczenia rozwiązania algorytmem najbliższego sąsiada wynosi $O(\sum_{i=1}^{n}(n-i)) = O\left(\frac{n(n-1)}{2}\right) = O(n^2)$, a algorytm jest wykonywany dla n różnych początkowych miast. Złożoność pamięciowa tego algorytmu wynosi $O(n^2)$ z podobnych przyczyn, dla których k-random ma tę samą złożoność pamięciową.