Projektowanie Efektywnych Algorytmow

Projekt

22/12/2020

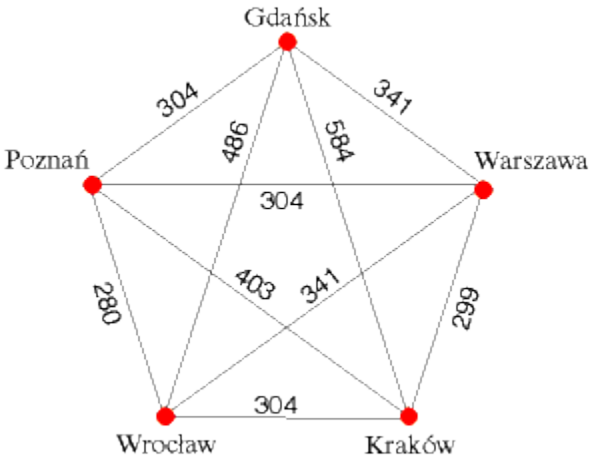
249451 Kewin Warzecha

(4) Symulowane wyżarzanie

1. Sformułowanie Zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu symulowanego wyżarzania (Simulated annealing) rozwiązującego problem komiwojażera. Algorytm należy przetestować na minimum 5 instancjach oraz znaleźć maksymalną wielkość problemu rozwiązywalnego na danym sprzęcie.

Rozwiązaniem problemu jest znalezienie cyklu Hamiltona o najmniejszym koszcie przejścia przez wszystkie wierzchołki w grafie pełnym. Cykl Hamiltona to taki cykl w grafie, w którym każdy wierzchołek grafu odwiedzany jest dokładnie raz. Rozwiązanie wydaje się trywialne, jednak w rzeczywistości jest zupełnie odwrotnie. Problem należy do klasy problemów NP-trudnych. Jest to jeden ze starszych problemów, dla których nie jest znany żaden algorytm dający gwarancję rozwiązania dokładnego w czasie co najwyżej wielomianowym.



Rysunek 1 przykładowy graf o pięciu wierzchołkach, źródło: mini.pw.edu.pl

2. Metoda

Zadanie zostało rozwiązane za pomocą jednej z technik projektowania algorytmów heurystycznych – metody symulowanego wyżarzania. Jest to metoda iteracyjna, której cechą charakterystyczną jest możliwość opuszczania ekstremów lokalnych – stosowana jest funkcja akceptacji dopuszczająca przyjmowanie gorszych rozwiązań, zwłaszcza w początkowej fazie działania. Najważniejszym parametrem jest temperatura, od której zależy skłonność do zmiany sąsiedztwa, im wyższą wartość ma ten parametr, tym większa skłonność. W trakcie przebiegu algorytm zmierza w kierunku minimalnej temperatury(wychłodzenia). Na początku działania algorytmu temperatura jest wysoka, dzięki czemu algorytm może bardzo często zmieniać konfigurację rozwiązania, niejednokrotnie wybierając rozwiązanie gorsze. Wraz z kolejnymi iteracjami algorytmu temperatura spada i wybierane są częściej rozwiązania lepsze. Pod koniec pracy algorytmu, temperatura jest na tyle niska, że prawdopodobieństwo wyboru gorszego rozwiązania jest bliskie zeru. Algorytm zachowuje się wówczas, jak typowy algorytm iteracyjny i stara się maksymalnie ulepszyć rozwiązanie. Inspiracją dla algorytmu jest proces wyżarzania ciał stałych.

Metoda symulowanego wyżarzania jest metaheurystyką, nie daje gwarancji znalezienia rozwiązania optymalnego, ale znajduje bardzo dobre rozwiązania w stosunkowo dobrym czasie (w porównaniu do algorytmów stosowanych w poprzednich zadaniach projektowych).

Algorytm cechuje się następującymi parametrami :

- początkową temperaturą T_0,
- końcową temperaturą T_{min},
- funkcją prawdopodobieństwa P,
- funkcją chłodzenia G.

Kluczowy dla wyników działania algorytmu jest dobór tych parametrów. Wartości temperatury początkowej i końcowej zostały dobrane metodą prób i błędów, tak aby zapewnić optymalne działanie algorytmu. Postać funkcji prawdopodobieństwa jest zaczerpnięta z literatury (rozkład Boltzmanna) i ma postać:

$$P = \begin{cases} 1 & dla\ f(X') > f(X) \\ e^{-\left|\frac{f(X\prime)-f(X)}{T}\right|} & dla\ f(X') < f(X) \end{cases}$$

Gdzie f(X) to koszt najlepszego znalezionego rozwiązania a f(X') to kosz rozważanego rozwiązania.

Podczas badań sprawdzono również kilka typów funkcji chłodzenia:

- Cauchy'ego (liniowy)

$$T_{p+1} = T_p - \frac{T_{min} - T}{N}$$

- Geometryczny

$$T_{p+1} = T_p * \left(\frac{T_{min}}{T_0}\right)^{\frac{1}{N}}$$

- Boltzmanna (logarytmiczny)

$$T_{p+1} = \frac{T_p}{1 + T_p * \left(\frac{T_0 - T_{min}}{N * T_0 * T_{min}}\right)}$$

Gdzie:

- T_p - aktualna temperatura
- N – maksymalna liczba iteracji

Aby zbadać czas wykonania poszczególnych instancji użyto funkcji *default_timer()* z biblioteki *timeit* dostępnej w środowisku *python*.

Do zbadania złożoności pamięciowej użyto modułu *Memory Profiler* dostępnego w środowisku Python. Program pozwala zbadać wielkość pamięci użytej przez każde wywołanie danej funkcji, udostępnia również interfejs do tworzenia wykresów za pomocą biblioteki matplotlib.

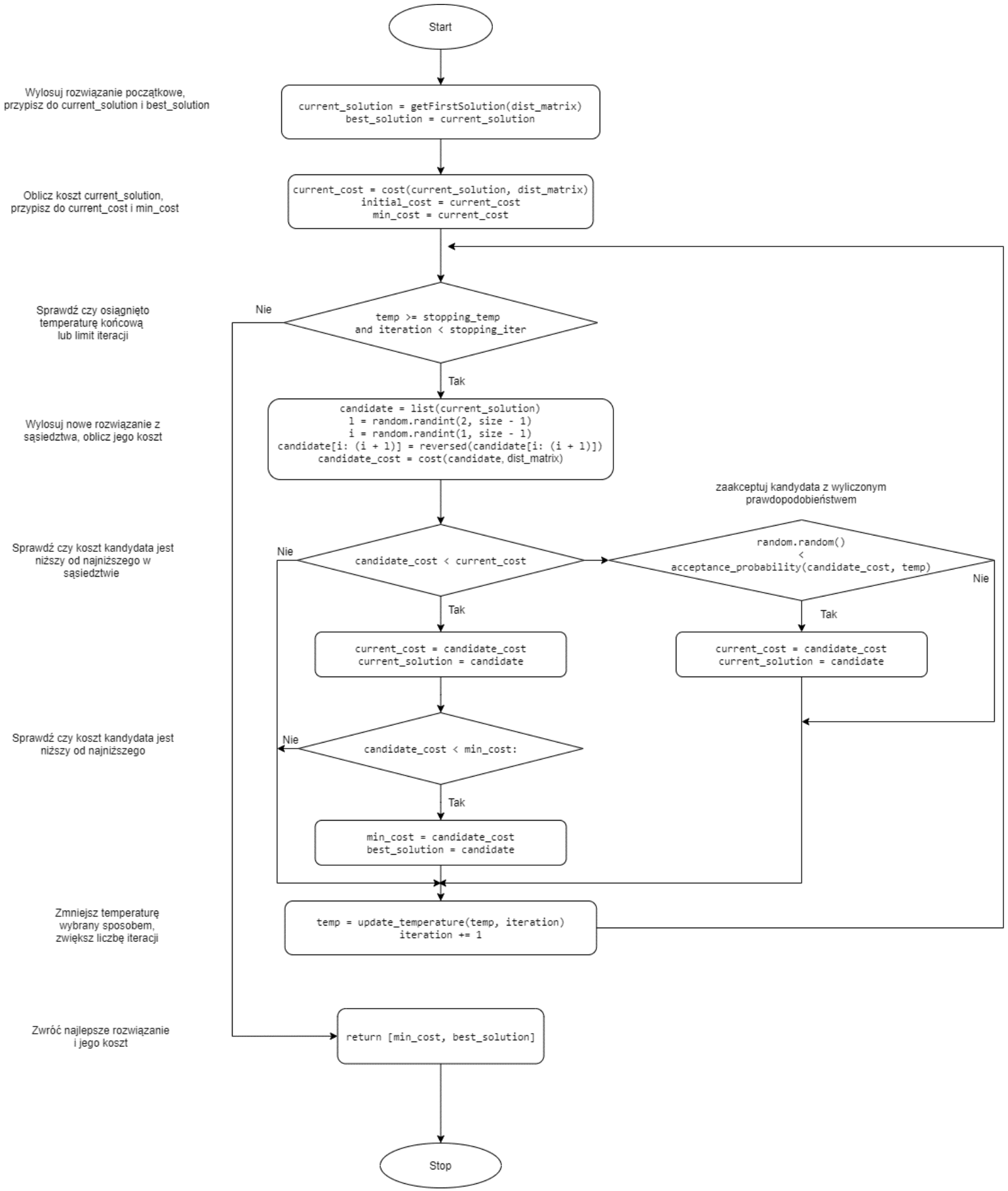
3. Algorytm

Szukanie rozwiązania następuje za pomocą funkcji *solve(matrix, temp, stopping_temp, stopping_iter, cooling_type, neighbourhood_type).*

Funkcja przyjmuje parametry:

- Matrix – macierz z kosztami przejść pomiędzy poszczególnymi wierzchołkami
- temp – temperatura początkowa
- stopping_temp – temperatura końcowa
- stopping_iter – graniczna liczba iteracji
- anneal_type - typ chłodzenia
- neighbourhood_type – typ doboru sąsiedztwa

Algorytm rozpoczyna działanie od wylosowania początkowego rozwiązania, następnie przypisywane jest ono do poszczególnych zmiennych i obliczany jest jego koszt. Po inicjalizacji pozostałych zmiennych odpowiednimi wartościami przechodzi do pętli, która jest wykonywana do momentu kiedy temperatura będzie mniejsza niż temperatura końcowa lub liczba iteracji będzie większa niż maksymalna liczba iteracji. Następnie za pomocą jednej z dwóch metod wybierany jest wierzchołek z sąsiedztwa poprzedniego rozwiązania. Zbadane zostały metody 2-zamiany i insert. Kolejnym krokiem jest sprawdzenie czy rozwiązanie jest lepsze od dotychczasowych (lokalnych i całkowitych), jeśli nie jest lepsze to obliczamy prawdopodobieństwo z jakim przyjmiemy gorsze rozwiązanie – zastosowano przegląd sąsiedztwa typu greedy. Ostatnim etapem jest chłodzenie za pomocą wybranej metody – liniowej (Cauchy'ego), logarytmicznej(Boltzmanna) lub geometrycznej. Algorytm kończy działanie poprzez zwrócenie najlepszej drogi i jej kosztu. Graf przedstawiający rozwiązanie został przedstawiony poniżej, w wersji z wykorzystaniem metody 2-zamiany.



Rysunek 2 schemat blokowy przedstawiający działanie algorytmu

4. Dane testowe

Do sprawdzenia poprawności i efektywności działania algorytmu wybrano następujący zestaw instancji:

1. tsp_6_1.txt
2. tsp_6_2.txt
3. tsp_10.txt
4. tsp_12.txt
5. tsp_13.txt
6. tsp_14.txt
7. tsp_15.txt
8. tsp_17.txt

Źródło: http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

6. gr21.tsp
7. ulysses22.tsp
8. gt24.tsp
9. fri26.tsp

Źródło: http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/pea/instances.zip

10. att48.tsp
11. eil51.tsp
12. pr76.tsp
13. kro100A.tsp
14. kroB150.tsp
15. ch150.tsp
16. pr152.tsp
17. kroB200.tsp
18. a280.tsp
19. pr2392.tsp

Źródło: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. W związku z tym procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym .INI (format pliku: nazwa_instancji liczba_wykonań rozwiązanie_optymalne [ścieżka optymalna]).

Zawartość użytego pliku .INI:

testData/tsp_6_1.txt 10 132 [0 1 2 3 4 5 0]

testData/tsp_6_2.txt 10 80 [0 5 1 2 3 4 0]

testData/tsp_10.txt 10 212 [0 3 4 2 8 7 6 9 1 5 0]

testData/tsp_12.txt 10 264 [0 1 8 4 6 2 11 9 7 5 3 10 0]

testData/tsp_13.txt 10 269 [0 10 3 5 7 9 11 2 6 4 8 1 12 0]

testData/tsp_14.txt 10 282 [0 10 3 5 7 9 13 11 2 6 4 8 1 12 0]

testData/tsp_15.txt 10 291 [0 12 1 14 8 4 6 2 11 13 9 7 5 3 10 0]

testData/tsp_17.txt 10 39 [0 11 13 2 9 10 1 12 15 14 5 6 3 4 7 8 16 0]

testData/tsp_21.txt 1 2707 [0 11 3 10 19 18 16 9 17 12 13 14 20 1 2 8 4 15 5 7 6 0]

testData/tsp_22.txt 1 6929 [0 15 2 1 16 21 17 3 14 4 10 8 9 20 19 18 5 6 13 12 11 7 0]

testData/tsp_24.txt 1 1272 [0 15 10 2 6 5 23 7 20 4 9 16 21 17 18 14 1 19 13 12 8 22 3 11 0]

testData/tsp_26.txt 0 937 [0 0]

output.csv

Każda z instancji wykonywana była określoną liczbę razy, np. tsp_6_1.txt wykonana została 10 razy, natomiast tsp_24.txt tylko raz. Do pliku wyjściowego „output.csv" zapisywane są czasy wykonania i otrzymane rozwiązania dla poszczególny instancji (koszt drogi, różnica od rozwiązania optymalnego i numery kolejnych węzłów). Plik wyjściowy zapisywany jest w formacie csv. Poniżej przedstawiono fragment zawartości pliku wyjściowego:

Tabela 1: przykładowa zawartość pliku wyjściowego

file	repetitions	cost	time	diff	Cooling	neighbour
testData/kroA100.tsp	3	21282			geometric	2opt
		22486.0	3.297	5.657		
		22264.0	3.339	4.615		
		22313.0	3.739	4.846		
testData/kroB150.tsp	3	26130			geometric	2opt
		28532.0	4.932	9.191		
		28700.0	maj.15	9.837		
		28793.0	4.846	10.192		
testData/ch150.tsp	3	6528			geometric	2opt
		7019.0	maj.35	7.519		
		6664.0	5.705	2.083		
		7072.0	4.965	8.331		

testData/pr152.tsp	3	73682			geometric	2opt
		75497.0	5.662	2.463		
		76570.0	5.914	mar.92		
		76179.0	4.863	3.389		
testData/kroB200.tsp	3	29437			geometric	2opt
		31947.0	6.152	8.527		
		32317.0	6.268	9.784		
		32535.0	6.346	10.524		
testData/a280.tsp	3	2579			geometric	2opt
		2867.0	8.312	11.155		
		2842.0	08.mar	10.211		
		2865.0	8.119	11.sty		
testData/pr2392.tsp	3	378032			geometric	2opt
		458856.0	71.254	21.38		
		457413.0	72.76	20.999		
		457777.0	76.464	21.095		
testData/kroA100.tsp	3	21282			geometric	2opt
		22486.0	3.297	5.657		
		22264.0	3.339	4.615		
		22313.0	3.739	4.846		
testData/kroB150.tsp	3	26130			geometric	2opt
		28532.0	4.932	9.191		
		28700.0	maj.15	9.837		
		28793.0	4.846	10.192		
testData/ch150.tsp	3	6528			geometric	2opt
		7019.0	maj.35	7.519		

Badania przeprowadzono na komputerze ASUS VivoBook Pro, którego specyfikację przedstawiono poniżej:

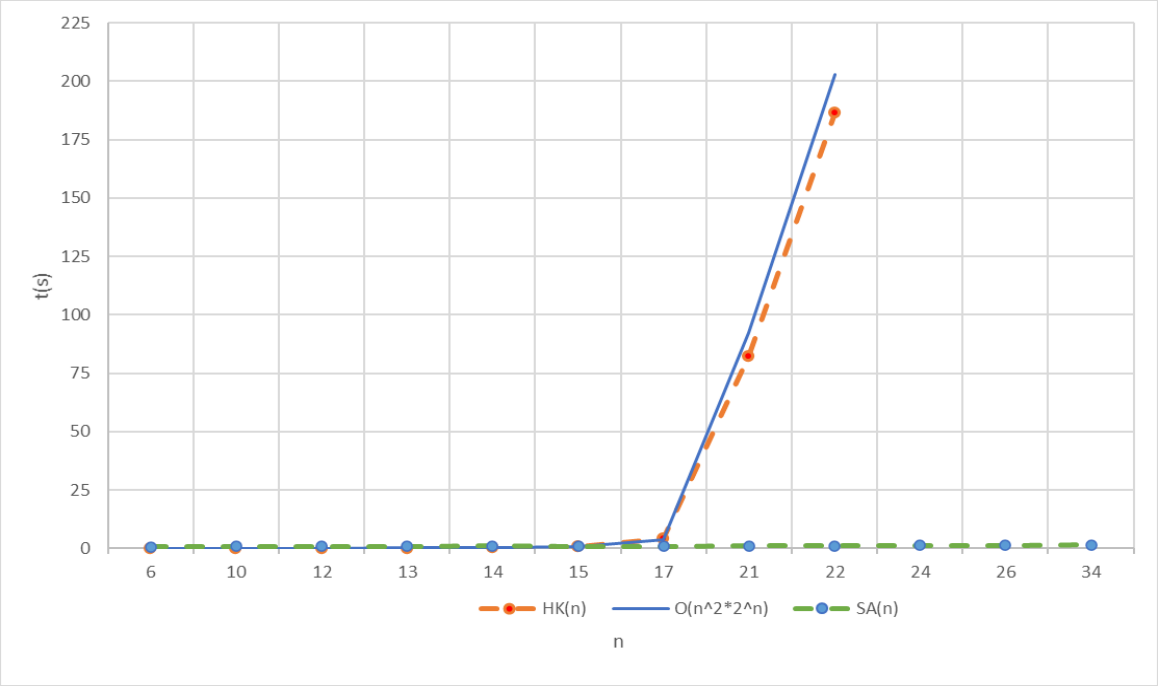
- **procesor**: Intel Core I7 7700HQ
- **pamięć RAM:** 2x 8GB DDR4-2400 (1200MHz)
- **system**: Windows 10 Edu

6. Wyniki

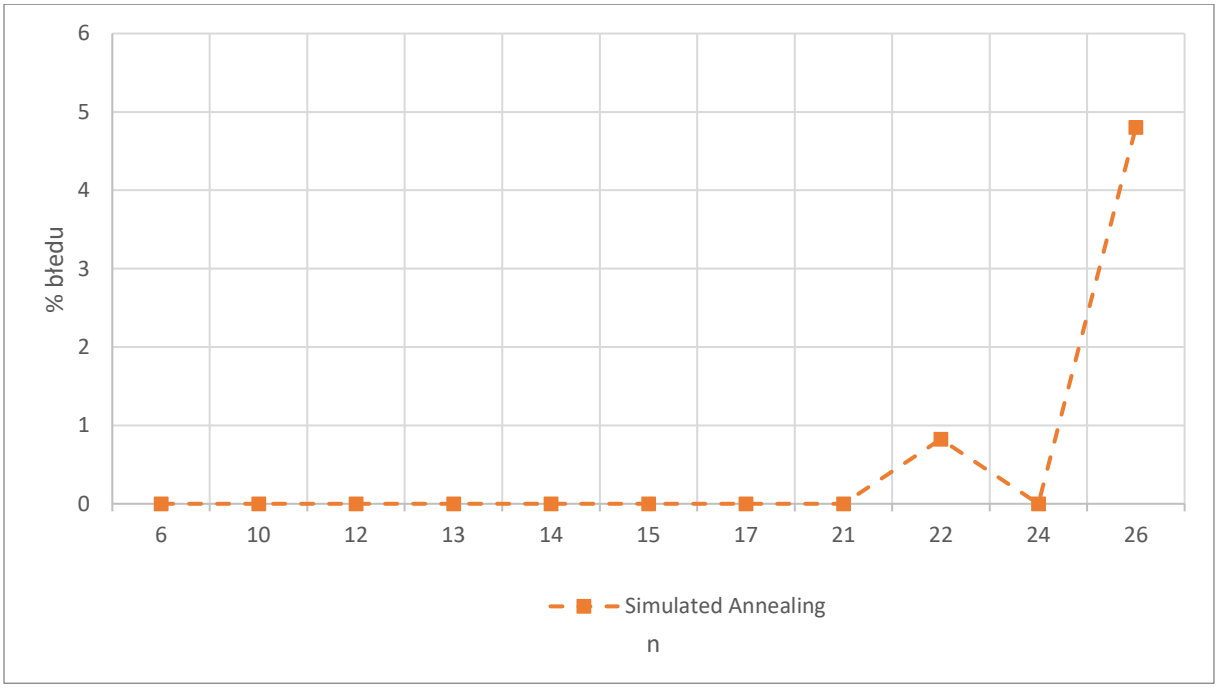
Wyniki zostały zapisane w pliku: output.csv i dołączone do raportu. Znajdują się na dysku Google pod adresem:

https://drive.google.com/drive/folders/15ZRZzR9IBbWHzvQIEqLhfHgzUY7a4_II?usp=sharing

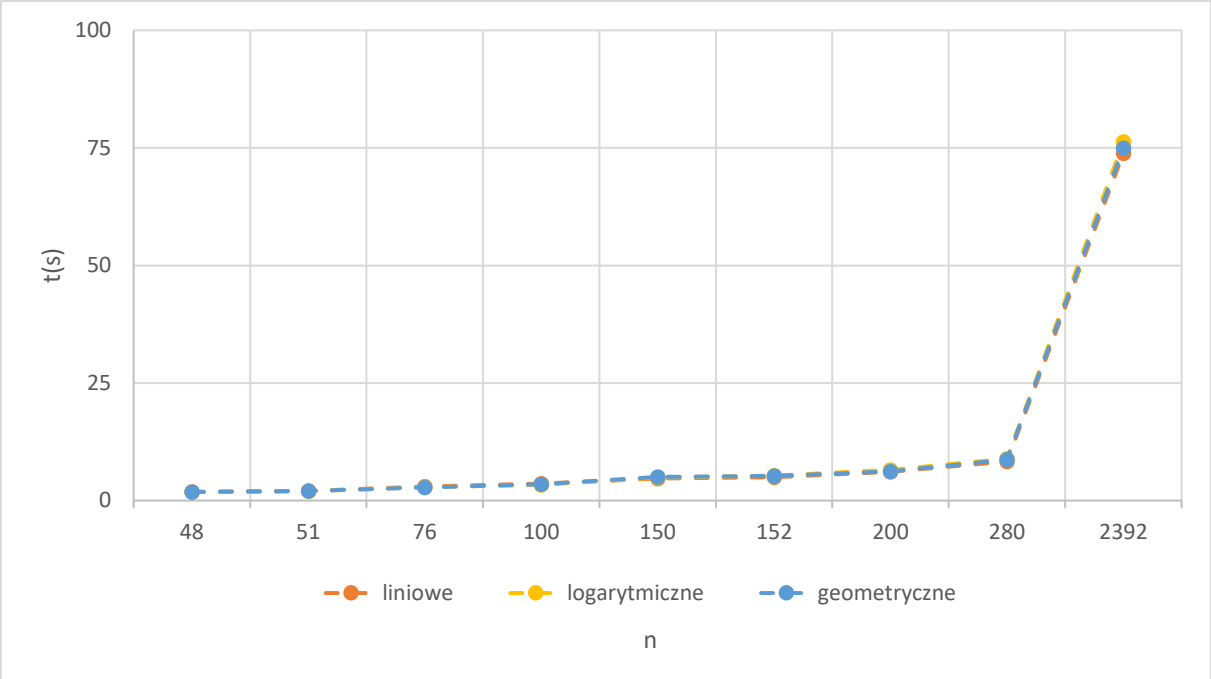
Wyniki przedstawione zostały w postaci poniższych wykresów:



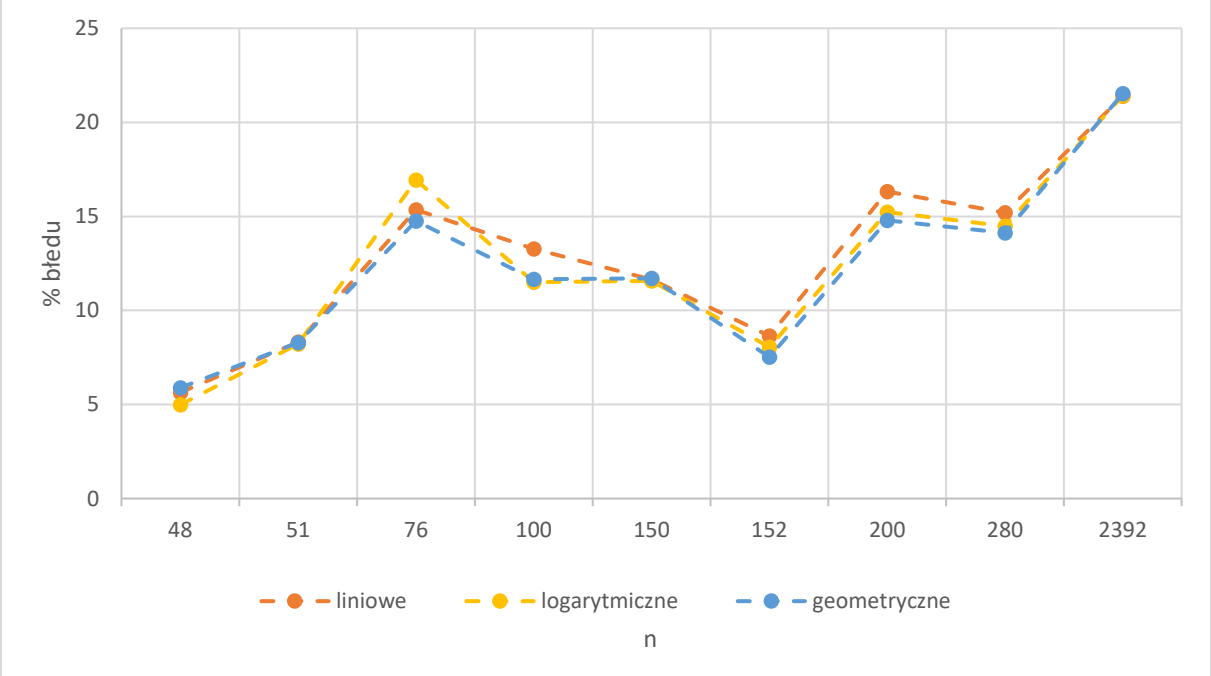
Wykres 1 porównanie algorytmu Helda-Karpa i Symulowanego Wyżarzania



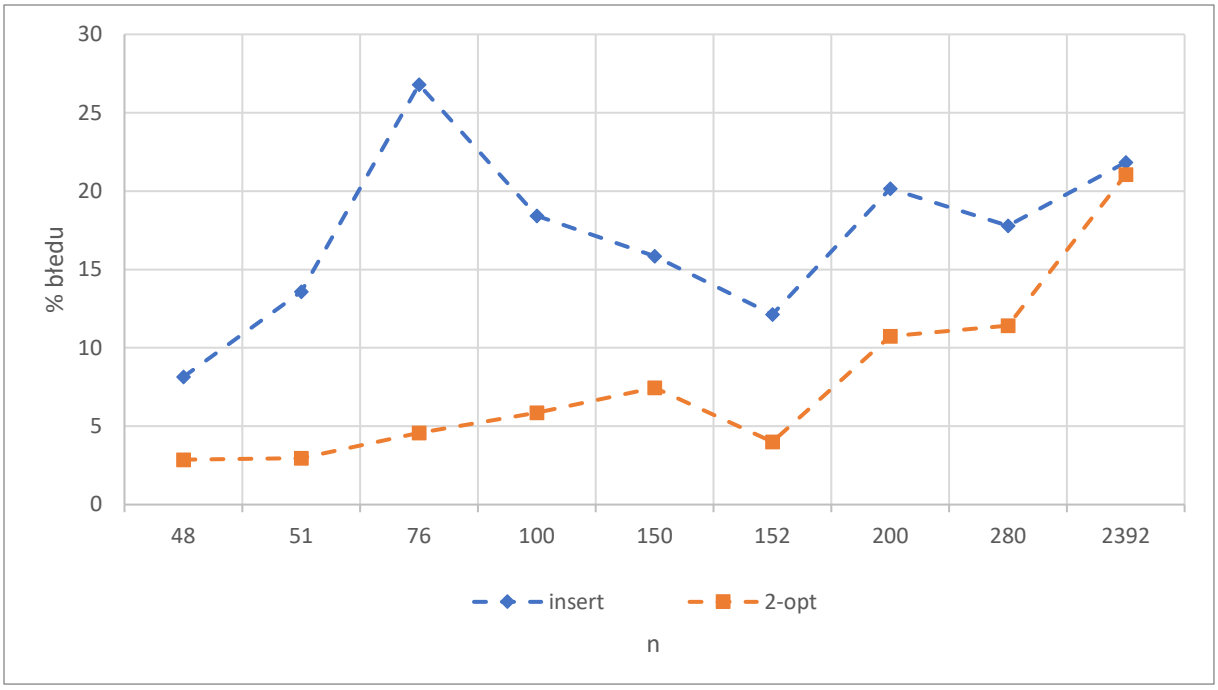
Wykres 2 Wielkość błędu dla małych instancji



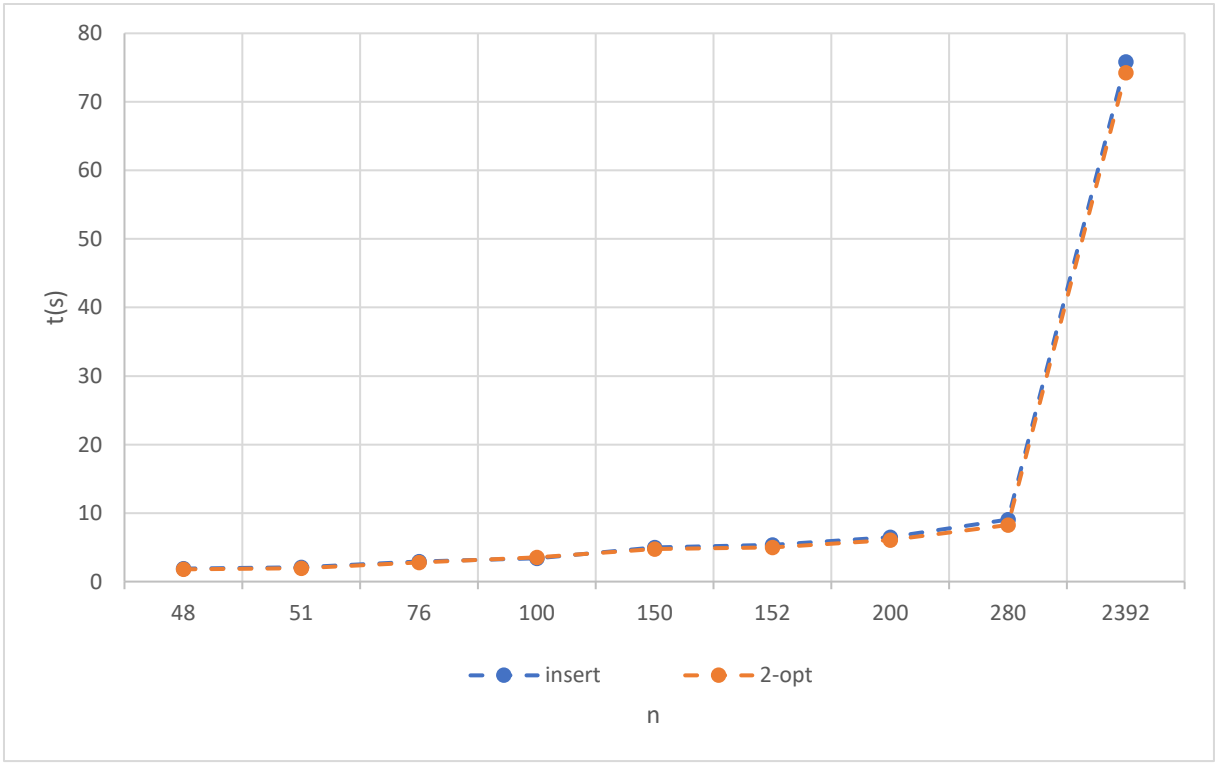
Wykres 3 Czas działania algorytmu w zależności od schematu chłodzenia



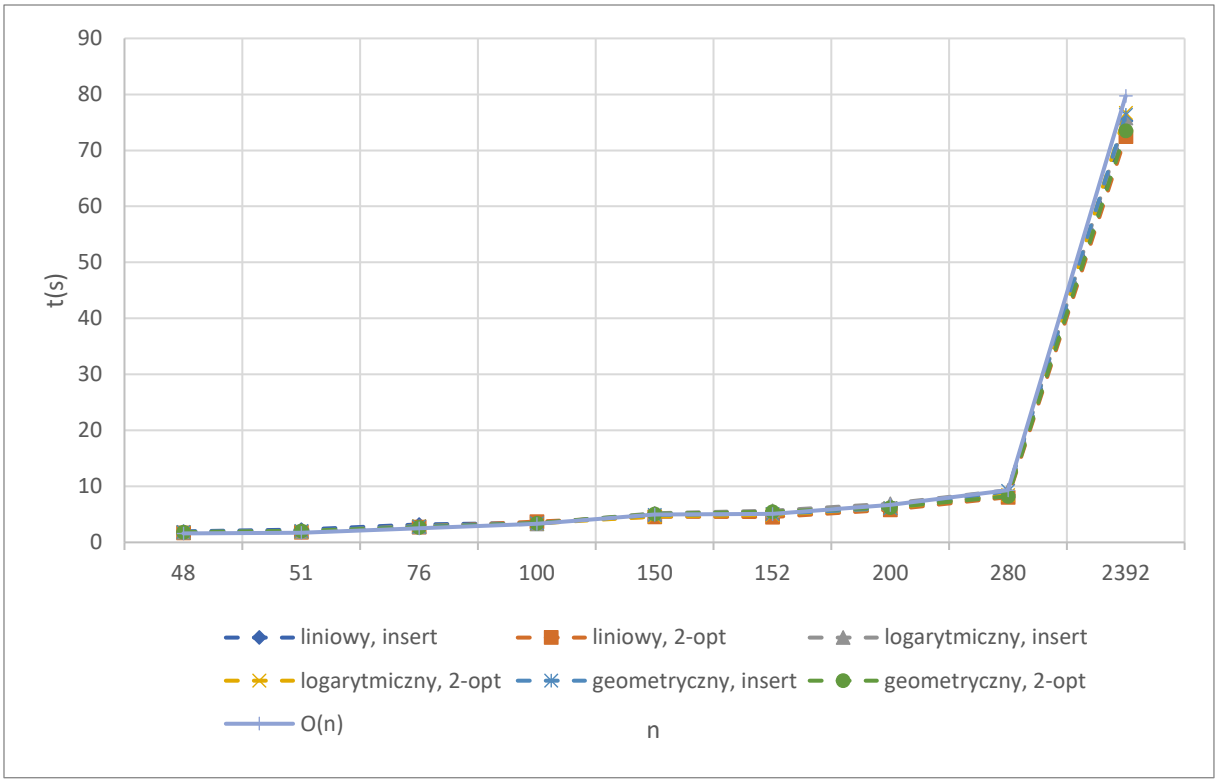
Wykres 4 Wielkość błędu w wynikach algorytmu w zależności od schematu chłodzenia



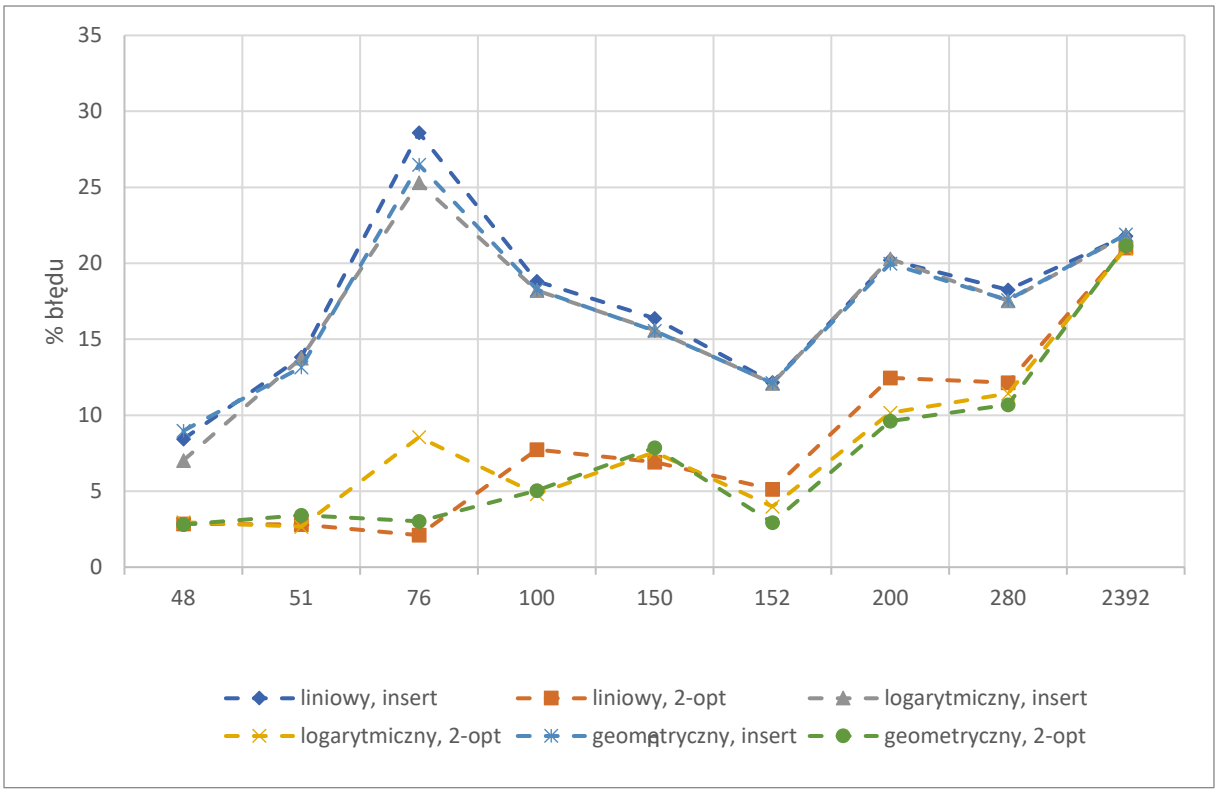
Wykres 5 Wielkość błędu w zależności od sposobu wyboru sąsiedztwa



Wykres 6 Czas działania algorytmu w zależności od sposobu wyboru sąsiedztwa



Wykres 7 czas wykonania instancji dla poszczególnych kombinacji



Wykres 8 wielkość błędu dla poszczególnych kombinacji

7. Analiza wyników i wnioski

Na wykresie 1 zostały porównane algorytmy Helda-Karpa i Symulowanego wyżarzania. Jak widać algorytm SA oferuje znaczny przyrost wydajności, która pozwala na bardzo szybkie rozwiązywanie problemów, których nie można było rozwiązać algorytmem HK. Wadą tego rozwiązania jest to, że nie zawsze algorytm zwraca rozwiązanie optymalne. Jednak jak widać na rysunku 2 dla małych rozmiarów instancji (< 24), jest duża szansa, że znalezione rozwiązanie będzie optymalne lub bardzo bliskie optymalnego.

Na wykresie 3 i 4 przestawiono czas działania i % błędu dla algorytmu w zależności od schematu chłodzenia. Ze względu na dobór parametrów, różnice w wydajności dla poszczególnych rozwiązań nie są duże. Różnice są zauważalne jeśli chodzi o wielkość błędu dla różnych sposobów wyboru sąsiedztwa – w tym przypadku zdecydowanie korzystniej wypada metoda 2-opt.

Na wykresie 7 przedstawiono czasy działania poszczególnych kombinacji. Można zauważyć, że punkty prawie się na siebie nakładają, a linie je łączące prawie idealnie odzwierciedlają krzywą O(n).