Projektowanie Efektywnych Algorytmów

Projekt
20.12.2022

254307 Paul Paczyński

(4) Algorytm Symulowanego Wyżarzania

1. Sformułowanie zadania

Zadane polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu symulowanego wyżarzania rozwiązującego problem komiwojażera dla instancji o różnych wielkościach. Dodatkowo należy zaimplementować dwie różne metody chłodzenia, wyboru sąsiedztwa, oraz sprawdzić jakość otrzymanych wyników.

2. Metoda

Metoda użyta do rozwiązania problemu komiwojażera to symulowane wyżarzanie. Nazwa metody jak i sposób działania jest powiązany ze zjawiskiem stygnięcia metali. W momencie kiedy temperatura rozgrzanego metalu spada wystarczająco wolno, to jego cząsteczki tworzą równomierną strukturę.

Algorytm ten bazuje na metodzie iteracyjnej która zakłada, że ciągle próbujemy ulepszać nasze rozwiązanie, aż nie będziemy w stanie już go bardziej poprawić. W praktyce natomiast dojście do najlepszego rozwiązania nie musi być warunkiem stopu, gdyż sama metoda może być bardzo czasochłonna, dlatego możemy przestać szukać najlepszego rozwiązania w momencie przekroczenia określonego czasu.

Przejście z jednego rozwiązania do drugiego polega na znalezieniu lepszego sąsiedniego rozwiązania, otrzymywanego za pomocą różnych metod, oraz zapamiętaniu go. W symulowanym wyżarzaniu natomiast istnieje możliwość wyboru gorszego rozwiązania, aby na nim dalej wykonywać obliczenia. Jest to sposób na wyjście z ekstremum lokalnego, w celu znalezienia ekstremum globalnego. Wybór taki jest zależny od temperatury, która maleje wraz z czasem działania algorytmu. Temperatura użyta jest we wzorze na prawdopodobieństwo akceptacji gorszego wyniku.

3. Algorytm

Główną strukturą algorytmu była lista, gdyż większość operacji wykonywane było na pojedynczej ścieżce. W liście tej przechowywałem trasę między wierzchołkami zaczynając od wierzchołka początkowego i na nim kończąc. Nie przetrzymywałem w tej strukturze kosztu drogi jaką trzeba było przebyć z wierzchołka a do b, gdyż informacje te brałem wyszukując poszczególne wierzchołki w macierzy zawierającej informacje o problemie.

3.1 Wybór pierwszej ścieżki

Do wyboru ścieżki początkowej postanowiłem zaimplementować algorytm zachłanny. Powodem była jego prosta implementacja, oraz większa szansa na otrzymanie przydatnego wyniku już w pierwszym kroku. Algorytm zachłanny polega na wyborze najlepszej możliwej drogi z danego punktu do punktu następnego. Nie bierze on jednak pod uwagę kolejnych kroków.

3.2 Obliczanie temperatury początkowej

Temperatura początkowa obliczana jest po znalezieniu początkowej ścieżki przez algorytm zachłanny i obliczenia jej kosztu, który jest potem mnożony przez wielkość instancji problemu.

3.3 Chłodzenie geometryczne

Chłodzenie geometryczne sprowadza się do obliczenia nowej temperatury, powstałej poprzez pomnożenie starej wartości temperatury i wybranego współczynnika alfa.

3.4 Chłodzenie logarytmiczne

Chłodzenie logarytmiczne otrzymywane jest poprzez podzielenie starej wartości temperatury, przez logarytm z całkowitej liczby wykonanych kroków (nie mylić z epokami). Kroki możemy opisać jako pojedyncze wykonanie algorytmu przeglądu sąsiedztwa.

3.5 Metoda przeglądu sąsiedztwa typu swap

Metoda polega na zamianie dwóch losowych punktów miejscami, otrzymując nową ścieżkę.

```
    def swap_solution(self,solution):
        index = range(len(solution))
        i1,i2 = random.sample(index,2)
        solution[i1],solution[i2] = solution[i2], solution[i1]
        return solution
```

3.6 Metoda przeglądu sąsiedztwa typu inverse

Metoda polega na wybraniu dwóch losowych punktów, które zostaną początkiem i końcem podciągu. Taki ciąg zamieniamy kolejnością, dzięki czemu otrzymujemy rozwiązanie sąsiednie.

```
    def inverse_solution(self, solution):
        node_one = random.choice(solution)
        new_list = list(filter(lambda ver: ver != node_one, solution))
        node_two = random.choice(new_list)
        solution[min(node_one, node_two):max(node_one, node_two)] =
solution[min(
            node_one, node_two):max(node_one, node_two)][::-1]

        return solution
```

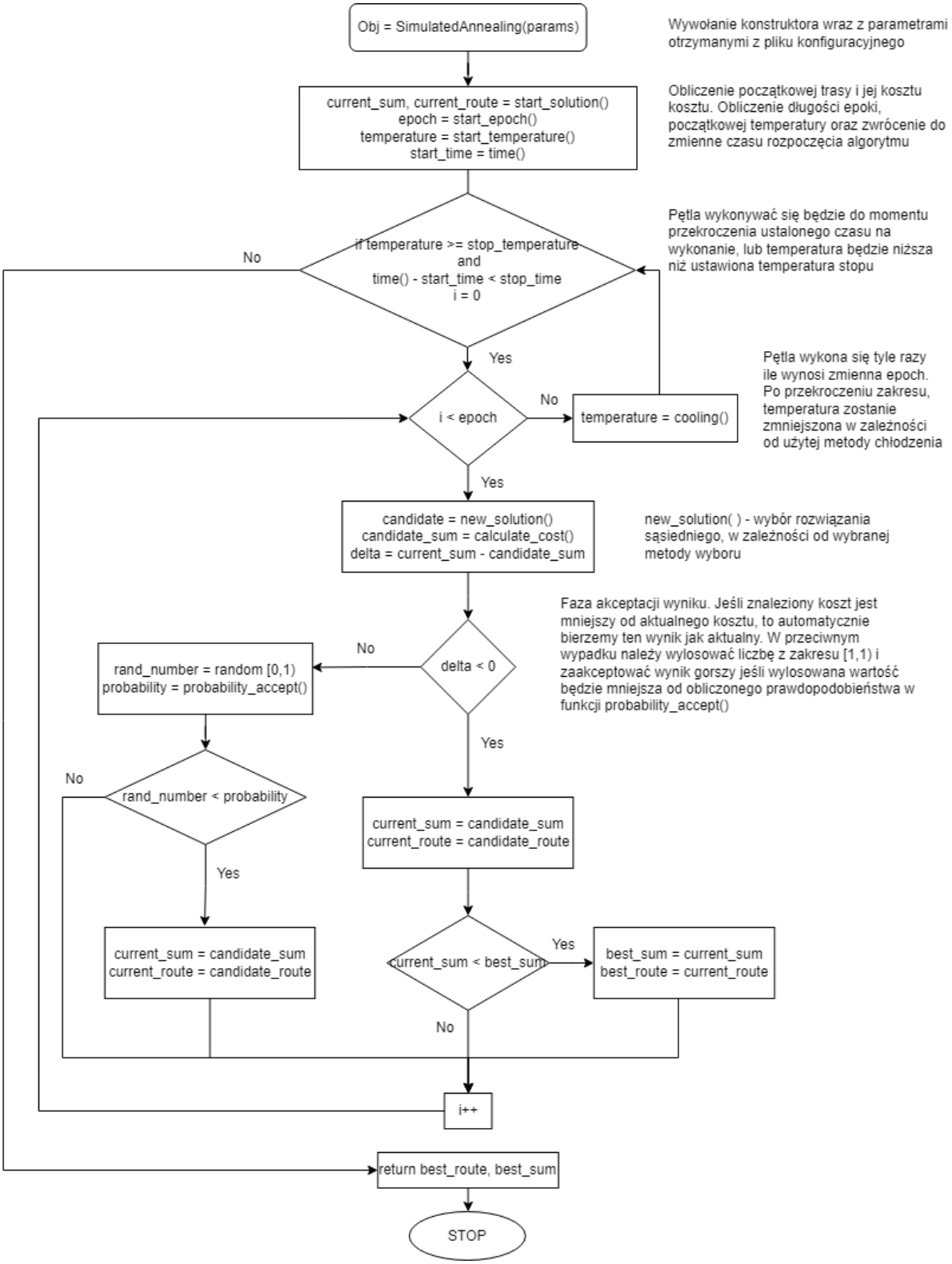
3.7 Akceptacja gorszego wyniku

W algorytmie istnieje możliwość akceptacji gorszego wyniku podczas przeszukiwania sąsiedztwa. Dzieje się tak wtedy, gdy wylosowana wartość z zakresu [0,1) będzie mniejsza od wyliczonej wartości prawdopodobieństwa. Oblicza się ją w następujący sposób, gdzie candidate_sum to koszt akceptowanej wartości.

```
    def probability_accept(self, candidate_sum):
        delta = candidate_sum - self.current_sum
        return math.exp(-(delta) / self.temperature)
```

Po otrzymaniu wartości zwrotnej algorytm porównuje wartość losową i obliczoną, a następnie odrzuca lub akceptuje w zależności od wyniku porównania.

3.8 Diagram blokowy



4. Dane testowe

6_1.txt
10.txt
12.txt
13.txt
14.txt
15.txt
17.txt
gr21.tsp
gr24.tsp
ftv33.atsp
ftv38.atsp
ftv44.atsp
ftv55.atsp
ftv64.atsp
ftv70.atsp
gr96.tsp
kroA100.tsp
kro124p.atsp
kroB150.tsp
pr152.tsp
ftv170.atsp
kroB200.tsp
rbg323.atsp

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. Przy implementacji metody symulowanego wyżarzania musieliśmy jeszcze pamiętać o parametrach początkowych potrzebnych do wykonania eksperymentu. Te parametry to:
- Temperatura początkowa
- Zmienna potrzebna do chłodzenia geometrycznego
- Długość ery

W moim programie postanowiłem użyć jeszcze trzech innych zmiennych użytych w pliku inicjacyjnym, mianowicie:
- Maksymalny czas na wykonanie algorytmu
- Punkt początkowy
- Ilość powtórzeń danej instancji

Rozpoczęcie procedury badawczej polegało więc na edycji pliku konfiguracyjnego .ini i uruchomieniu programu.

Struktura pliku config.ini:

<Liczba instancji>
<Sposób chłodzenia> <Metoda wyboru sąsiada> // <geo/log> <swap/inverse>
<Nazwa Pliku> <α > <t> <e> <s> <p> <r>
.
.
#<nazwa pliku wyjściowego> // Nazwa poprzedzona znakiem „#"

Gdzie
α - Współczynnik alfa potrzebny do chłodzenia geometrycznego. Ustawić jakąkolwiek wartość w przypadku chłodzenia logarytmicznego.
t – Zmienna odpowiadająca za temperaturę przy osiągnięciu której program się zatrzyma: 10^{-t}
e – Liczba epok obliczana jest z równania: e * N, gdzie N to wielkość instancji.
s – Liczba minut po których program zakończy obliczenia dla danej instancji.
p – Wierzchołek początkowy
r – Ilość powtórzeń algorytmu dla danej instancji

Treść przykładowego pliku ini

23
log inverse
6_1.txt 0.99 15 1500 5 0 5
10.txt 0.99 15 1500 5 0 5
12.txt 0.99 15 1500 5 0 5
13.txt 0.99 15 1500 5 0 5
14.txt 0.99 15 1500 5 0 5
15.txt 0.99 15 1500 5 0 5
17.txt 0.99 15 1500 5 0 5
gr21.tsp 0.99 15 1500 5 0 5
gr24.tsp 0.99 15 1500 5 0 5
ftv33.atsp 0.99 15 1500 5 0 3
ftv38.atsp 0.99 15 1500 5 0 3
ftv44.atsp 0.99 15 1500 5 0 3
ftv55.atsp 0.99 15 1500 5 0 3
ftv64.atsp 0.99 15 1500 5 0 3
ftv70.atsp 0.99 15 1500 5 0 3
gr96.tsp 0.99 15 1500 5 0 3
kroA100.tsp 0.99 15 1500 5 0 3
kro124p.atsp 0.99 15 1500 5 0 3
kroB150.tsp 0.99 15 1500 5 0 3
pr152.tsp 0.99 15 1500 5 0 3
ftv170.atsp 0.99 15 1500 5 0 3
kroB200.tsp 0.99 15 1500 5 0 3
rbg323.atsp 0.99 15 1500 5 0 3
#wyniki_log_inverse

Wyniki zostały zapisane do pliku wyjściowego w formacie csv. Zapisywane w nim dane przedstawiane są w postaci: Plik; Czas; Koszt; Ścieżka.

6. Wyniki

6.1 Sprzęt użyty do badania

Procesor: Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz   3.50 GHz
RAM: 8 GB
System: Windows 64bit, procesor x64

6.2 Sposób pomiaru czasu

Do pomiaru czasu wykorzystuję bibliotekę time dla języka python. Tworzone są dwie zmienne, start_time i end_time, które przyjmują wartość zwróconą z funkcji time.time(), zwracającej stan zegara. Umiejscowienie algorytmu pomiędzy dwoma zmiennymi, a następnie odjęciu start_time od end_time daje nam czas wykonania algorytmu w sekundach.

Ilość powtórzeń danej instancji zależała od jej wielkości. 5 razy dla instancji do 33 wierzchołków, 3 dla reszty.

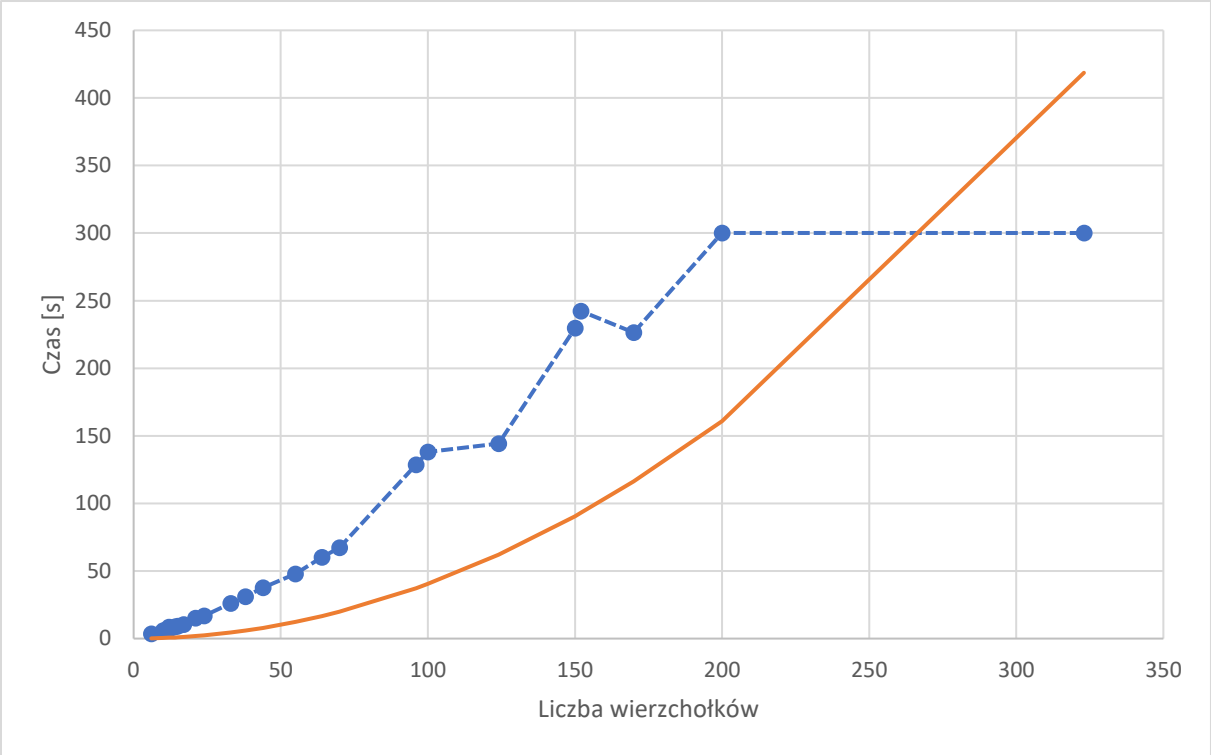
6.3 Prezentacja wyników

6.3.1 Chłodzenie: geometryczne. Wybór rozwiązań w sąsiedztwie: Invert

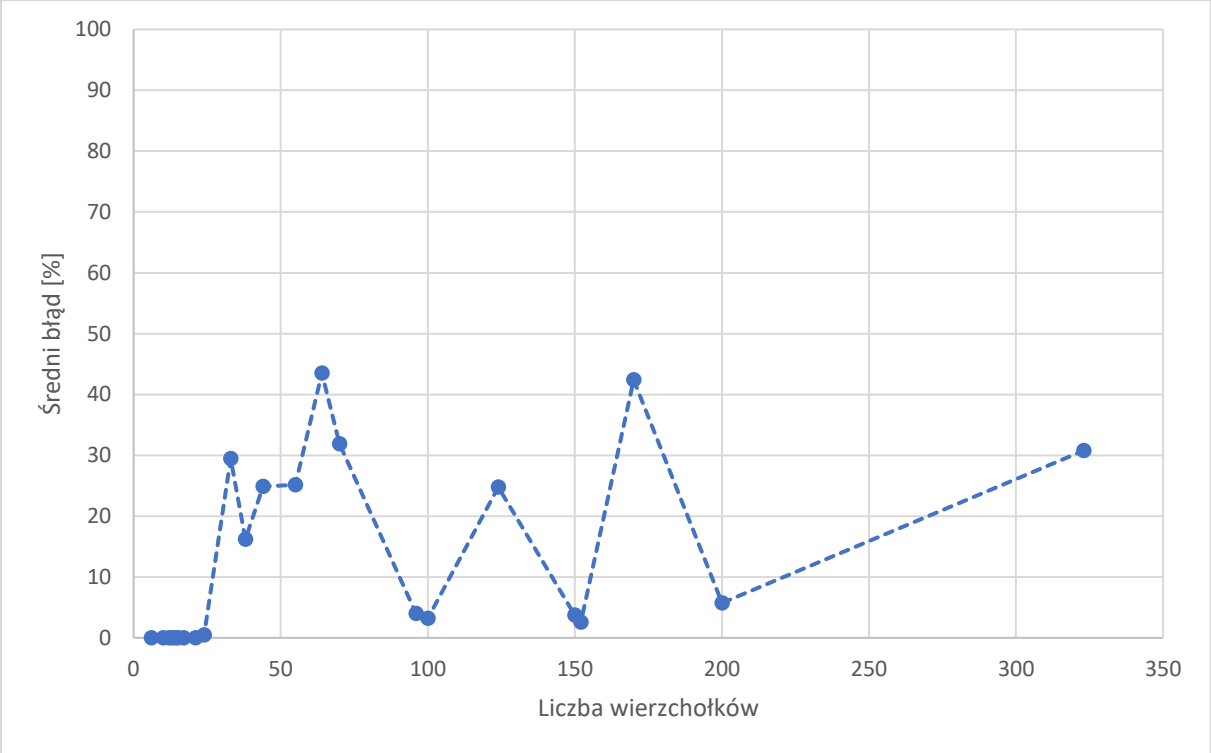
Parametry startowe:
α = 0.999, t = 10, e = 7, s = 5, p = 0, r = 5

Tabela 1: Wyniki dla chłodzenia geometrycznego i sąsiedztwa invert

Plik	Optymalny koszt	Średni koszt	Średni błąd [%]	Średni czas [s]
tsp_6_1.txt	132	132	0,00	3,5685
tsp_10.txt	212	212	0,00	6,0526
tsp_12.txt	264	264	0,00	8,6195
tsp_13.txt	269	269	0,00	8,1277
tsp_14.txt	282	282	0,00	8,7747
tsp_15.txt	291	291	0,00	9,3568
tsp_17.txt	39	39	0,00	10,4413
gr21.tsp	2707	2707	0,00	15,0896
gr24.tsp	1272	1278	0,46	16,9074
ftv33.atsp	1286	1665	29,47	25,9803
ftv38.atsp	1530	1778	16,21	31,0331
ftv44.atsp	1613	2014	24,86	37,7646
ftv55.atsp	1608	2012	25,12	47,8874
ftv64.atsp	1839	2639	43,50	60,2650
ftv70.atsp	1950	2571	31,85	67,2856
gr96.tsp	55209	57421	4,01	128,5912
kroA100.tsp	21282	21965	3,21	138,2092
kro124p.atsp	36230	45207	24,78	144,3371
kroB150.tsp	26130	27107	3,74	229,8591
pr152.tsp	73682	75573	2,57	242,2878
ftv170.atsp	2755	3923	42,40	226,3490
kroB200.tsp	29437	31115	5,70	300,0378
rbg323.atsp	1326	1734	30,77	300,0712



Rysunek 1: Czas wykonania programu geo/invert



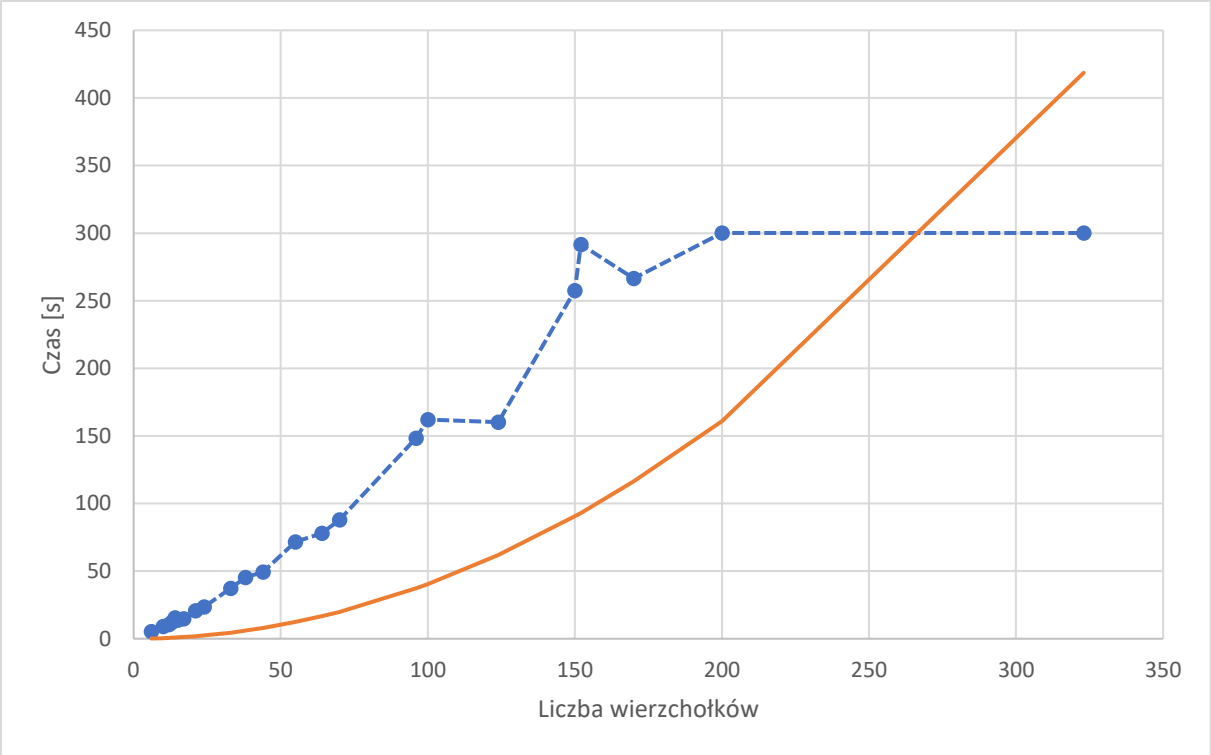
Rysunek 2: Błąd procentowy geo/invert

6.3.2 Chłodzenie: geometryczne. Wybór rozwiązań w sąsiedztwie: Swap

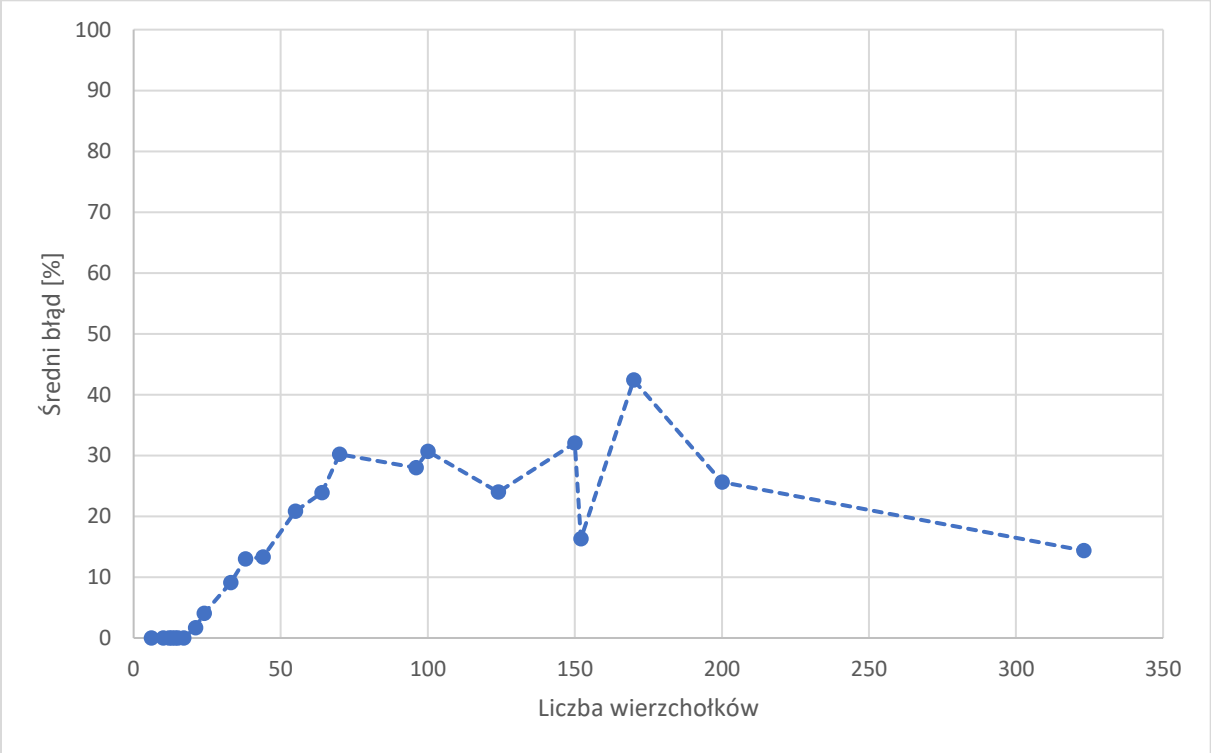
Parametry startowe:
α = 0.999, t = 10, e = 7, s = 5, p = 0, r = 5

Tabela 2: Wyniki dla chłodzenia geometrycznego i sąsiedztwa swap

Plik	Optymalny koszt	Średni koszt	Średni błąd [%]	Średni czas [s]
tsp_6_1.txt	132	132	0,00	5,2699
tsp_10.txt	212	212	0,00	9,0696
tsp_12.txt	264	264	0,00	10,5455
tsp_13.txt	269	269	0,00	12,1691
tsp_14.txt	282	282	0,00	15,4292
tsp_15.txt	291	291	0,00	13,8498
tsp_17.txt	39	39	0,00	14,8377
gr21.tsp	2707	2753	1,68	20,7579
gr24.tsp	1272	1324	4,07	23,6087
ftv33.atsp	1286	1403	9,07	37,3070
ftv38.atsp	1530	1729	13,01	45,4080
ftv44.atsp	1613	1827	13,29	49,2626
ftv55.atsp	1608	1943	20,83	71,6010
ftv64.atsp	1839	2278	23,87	78,0174
ftv70.atsp	1950	2539	30,22	87,9077
gr96.tsp	55209	70671	28,01	148,3914
kroA100.tsp	21282	27807	30,66	162,1287
kro124p.atsp	36230	44923	23,99	160,1993
kroB150.tsp	26130	34499	32,03	257,5330
pr152.tsp	73682	85699	16,31	291,5493
ftv170.atsp	2755	3923	42,40	266,5429
kroB200.tsp	29437	36980	25,62	300,0460
rbg323.atsp	1326	1516	14,35	300,1070



Rysunek 3: Czas wykonania programu geo/swap



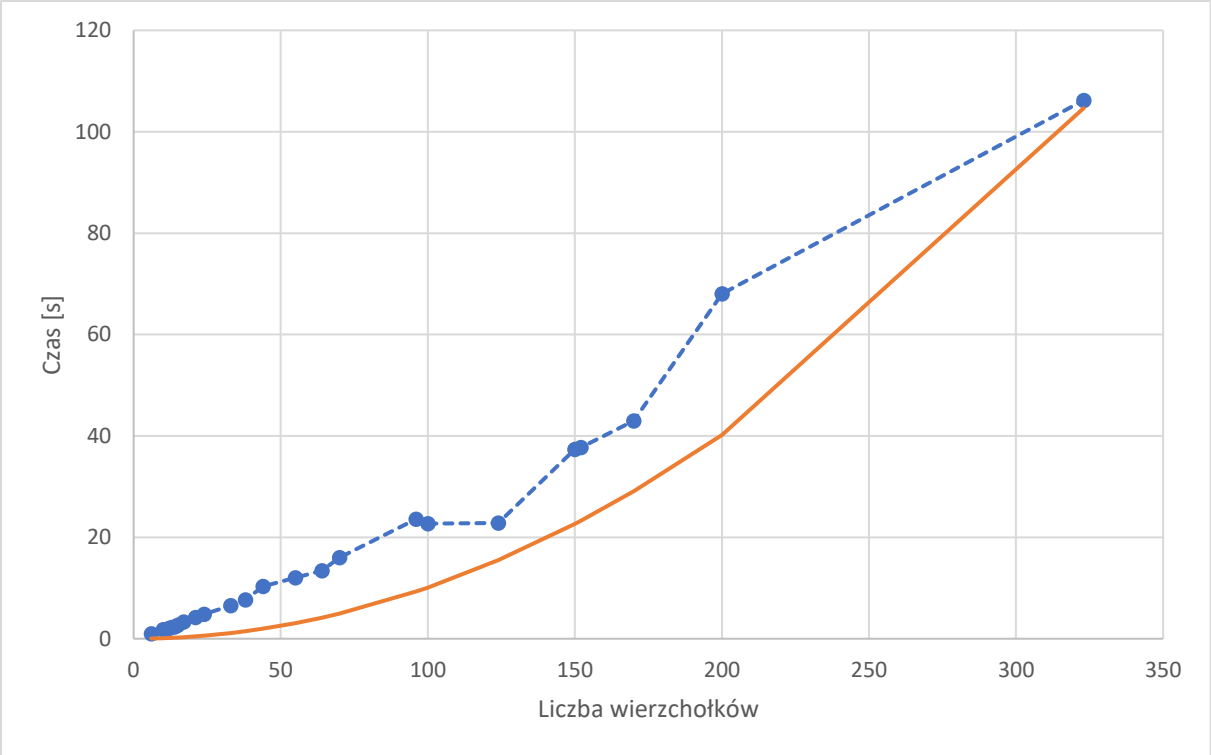
Rysunek 4: Błąd procentowy geo/swap

6.3.3 Chłodzenie: logarytmiczne. Wybór rozwiązań w sąsiedztwie: Swap

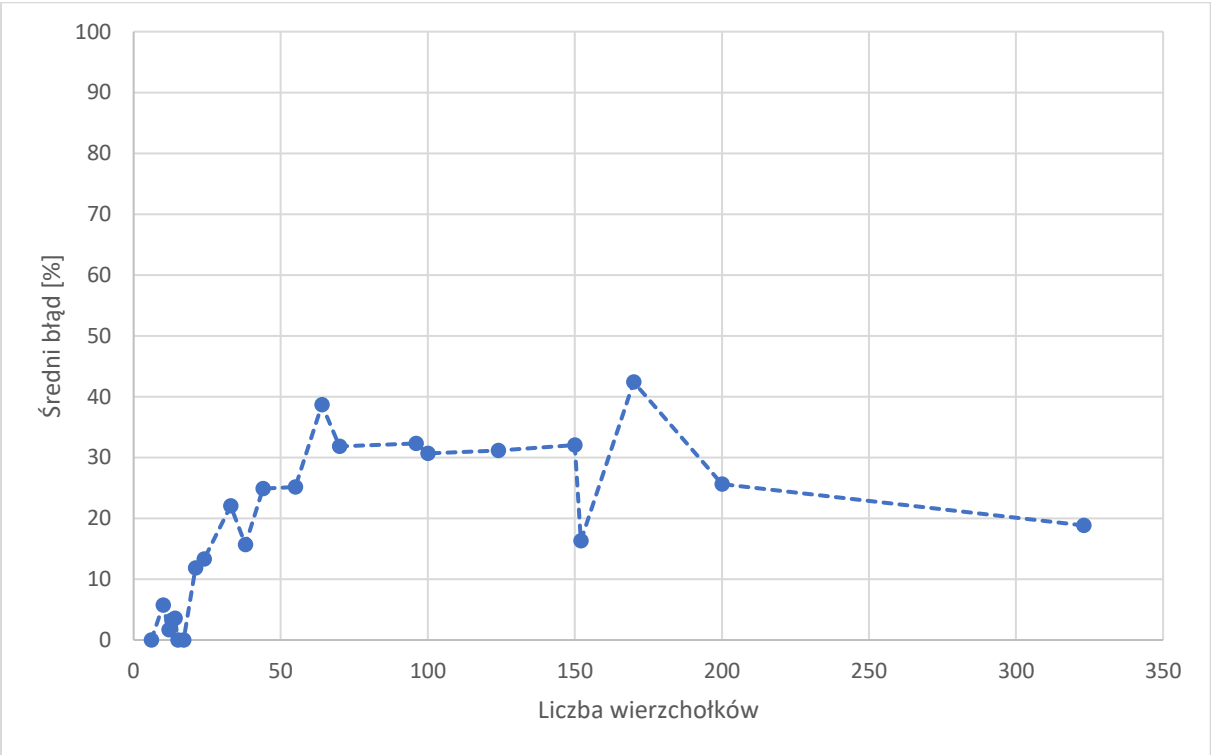
Parametry startowe:
t = 17, e = 200, s = 5, p = 0, r = 5

Tabela 3: Wyniki dla chłodzenia logarytmicznego i sąsiedztwa swap

Plik	Optymalny koszt	Średni koszt	Średni błąd [%]	Średni czas [s]
tsp_6_1.txt	132	132	0	1,0031
tsp_10.txt	212	224	5,75	1,8087
tsp_12.txt	264	268	1,67	2,0395
tsp_13.txt	269	278	3,27	2,2446
tsp_14.txt	282	292	3,55	2,3554
tsp_15.txt	291	291	0	2,6973
tsp_17.txt	39	39	0	3,3056
gr21.tsp	2707	3027	11,81	4,2013
gr24.tsp	1272	1441	13,3	4,8182
ftv33.atsp	1286	1570	22,06	6,5340
ftv38.atsp	1530	1770	15,66	7,6426
ftv44.atsp	1613	2014	24,86	10,3375
ftv55.atsp	1608	2012	25,12	12,0273
ftv64.atsp	1839	2550	38,68	13,4375
ftv70.atsp	1950	2571	31,85	15,9863
gr96.tsp	55209	73039	32,3	23,5945
kroA100.tsp	21282	27807	30,66	22,7261
kro124p.atsp	36230	47506	31,12	22,8475
kroB150.tsp	26130	34499	32,03	37,3455
pr152.tsp	73682	85699	16,31	37,7205
ftv170.atsp	2755	3923	42,4	42,9412
kroB200.tsp	29437	36980	25,62	68,0196
rbg323.atsp	1326	1576	18,83	106,1902



Rysunek 5: Czas wykonania programu log/swap



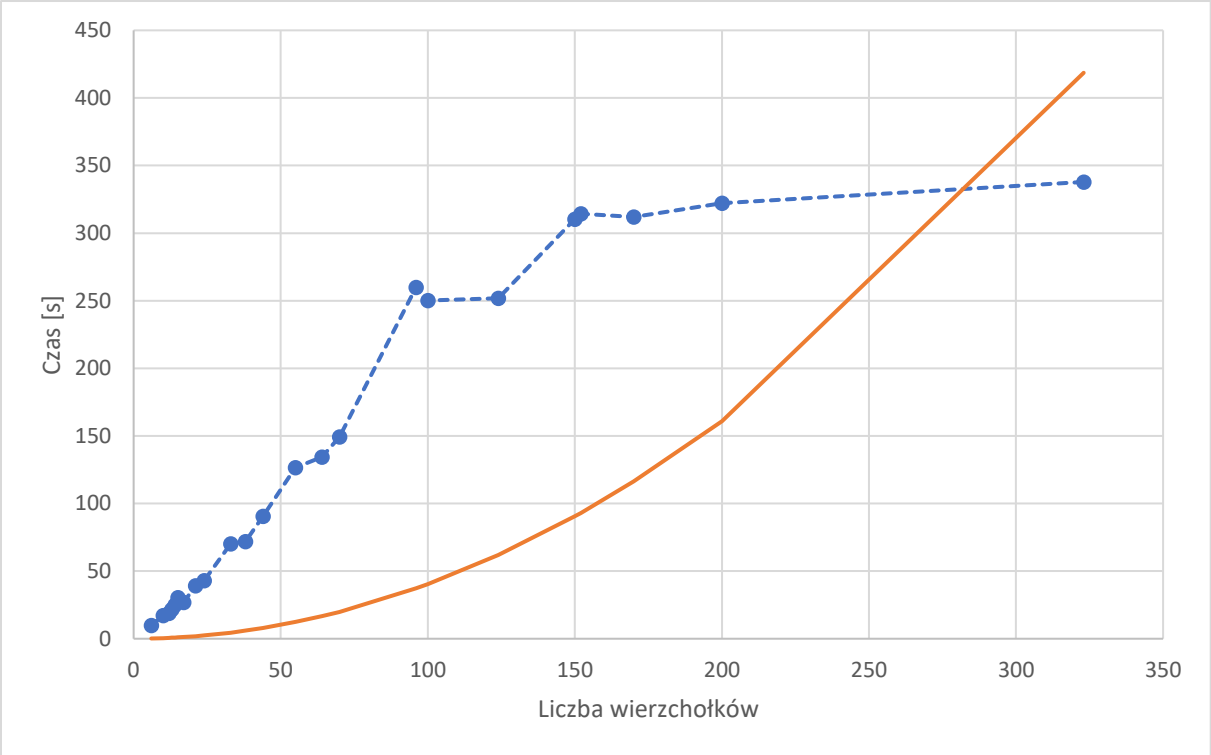
Rysunek 6: Błąd procentowy log/swap

6.3.4 Chłodzenie: logarytmiczne. Wybór rozwiązań w sąsiedztwie: Inverse

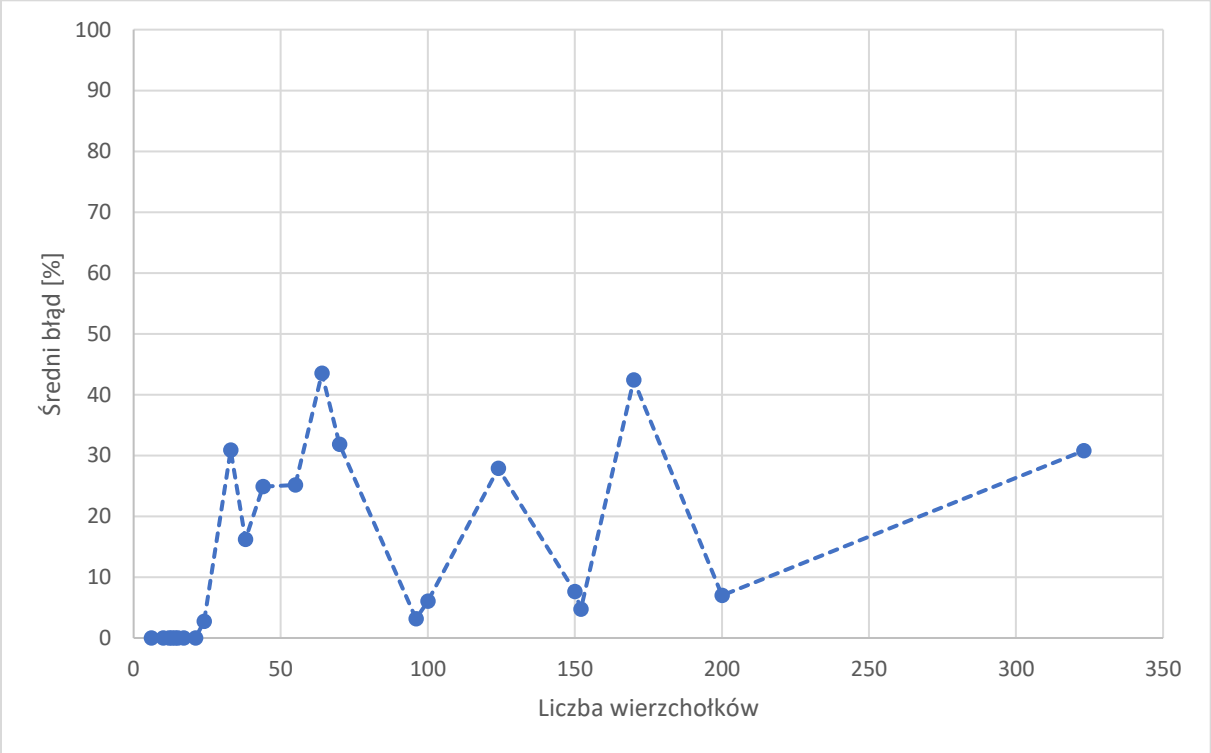
Parametry startowe:
t = 15, e = 1500, s = 5, p = 0, r = 5

Tabela 4: Wyniki dla chłodzenia logarytmicznego i sąsiedztwa invert

Plik	Optymalny koszt	Średni koszt	Średni błąd [%]	Średni czas [s]
tsp_6_1.txt	132	132	0,00	9,7805
tsp_10.txt	212	212	0,00	17,2706
tsp_12.txt	264	264	0,00	18,8171
tsp_13.txt	269	269	0,00	21,7705
tsp_14.txt	282	282	0,00	24,9615
tsp_15.txt	291	291	0,00	30,3320
tsp_17.txt	39	39	0,00	26,8196
gr21.tsp	2707	2707	0,00	39,2185
gr24.tsp	1272	1307	2,74	42,9595
ftv33.atsp	1286	1683	30,87	70,2074
ftv38.atsp	1530	1778	16,21	71,9507
ftv44.atsp	1613	2014	24,86	90,6629
ftv55.atsp	1608	2012	25,12	126,5651
ftv64.atsp	1839	2639	43,50	134,4414
ftv70.atsp	1950	2571	31,85	149,3528
gr96.tsp	55209	56952	3,16	259,8203
kroA100.tsp	21282	22566	6,03	250,2267
kro124p.atsp	36230	46338	27,90	251,8263
kroB150.tsp	26130	28118	7,61	310,2471
pr152.tsp	73682	77163	4,72	314,4220
ftv170.atsp	2755	3923	42,40	311,9461
kroB200.tsp	29437	31501	7,01	322,0824
rbg323.atsp	1326	1734	30,77	337,8628

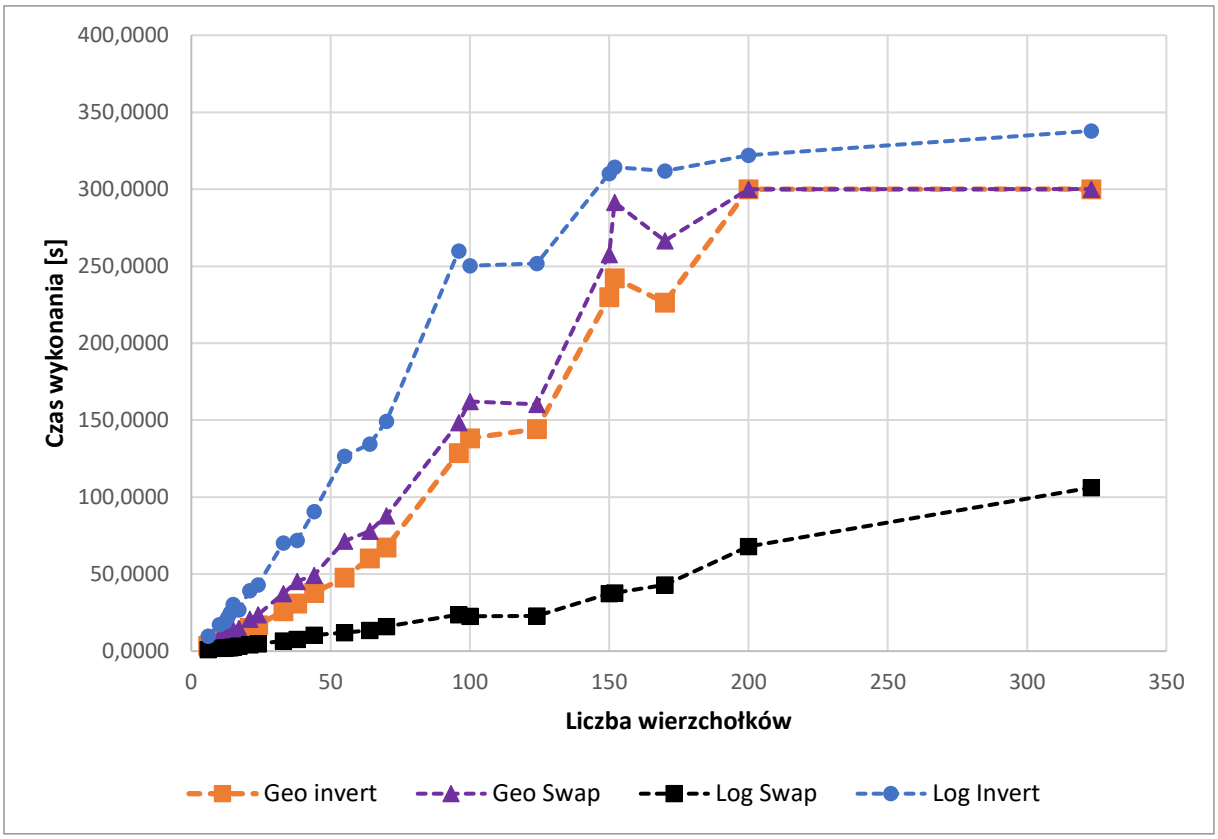


Rysunek 7: Czas wykonania programu log/invert

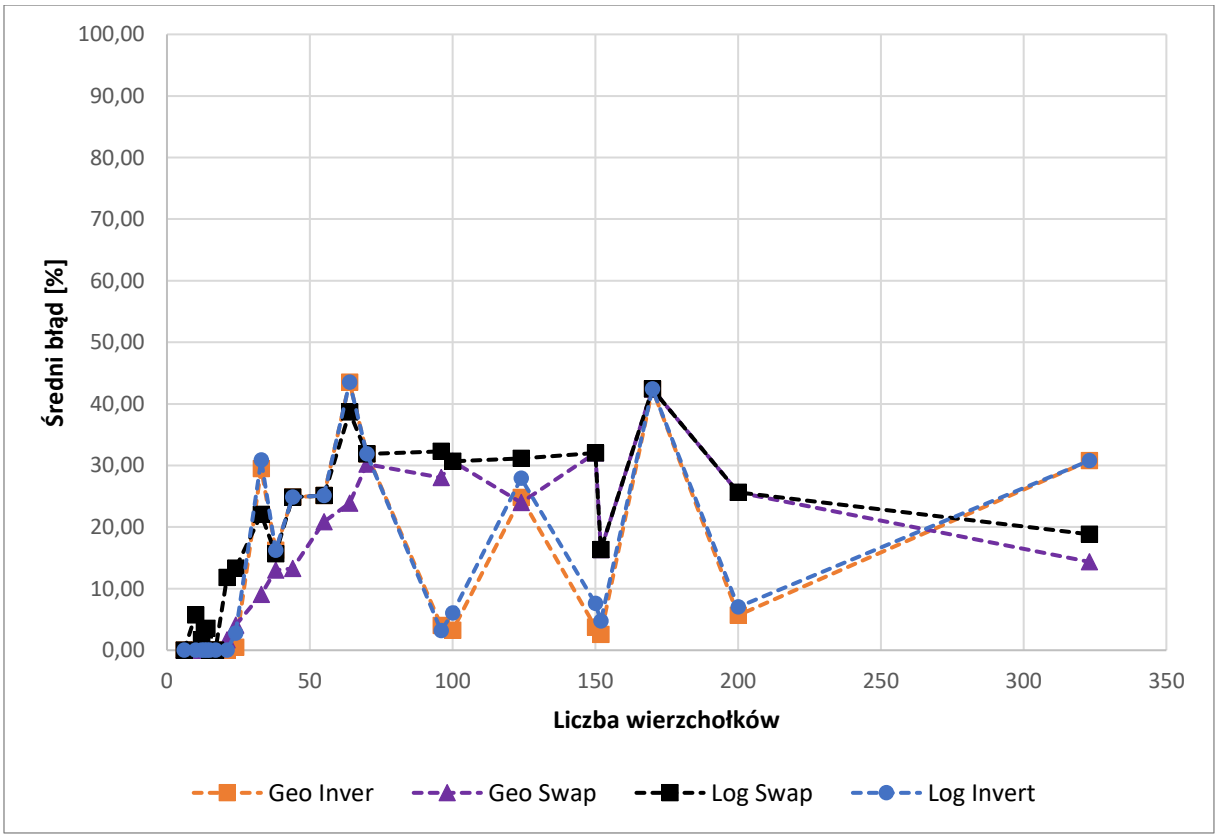


Rysunek 8: Błąd procentowy log/invert

6.3.5 Porównanie czasu wykonania wszystkich kombinacji algorytmu



Rysunek 9: Porównanie czasu wykonania wariacji algorytmu

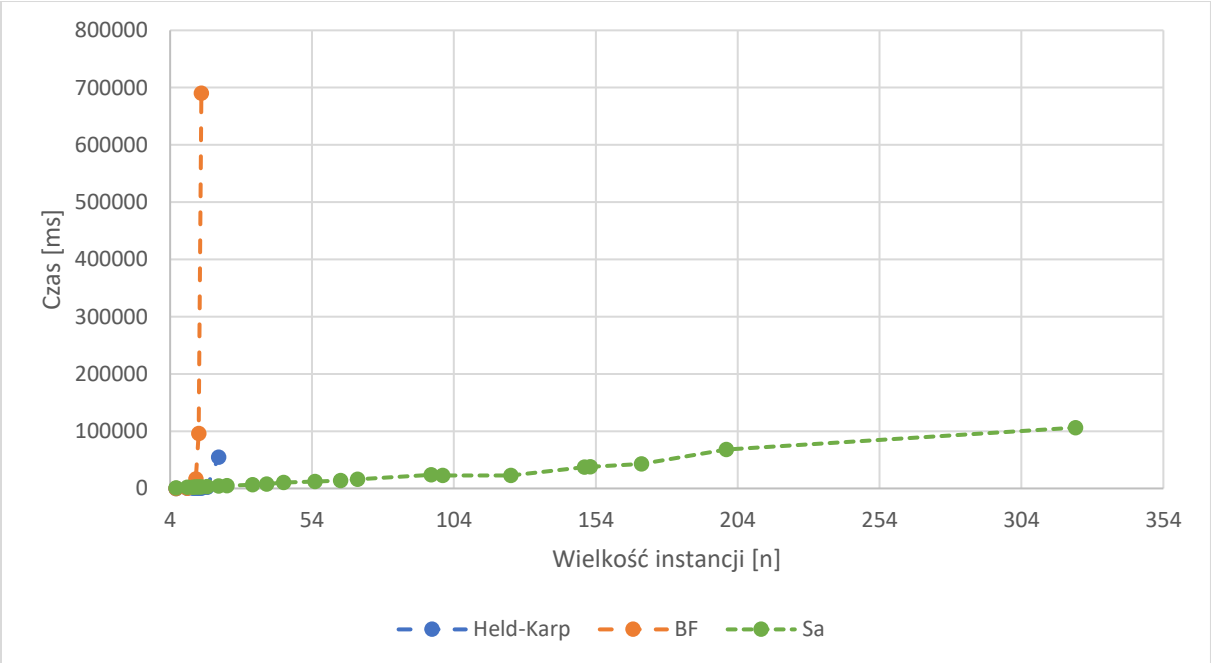


Rysunek 10: Porównanie błędów procentowych wariacji algorytmu

7. Wnioski

Największą trudność w wykonaniu tego zadania sprawiło mi dobranie parametrów początkowych do poszczególnych wariacji algorytmu symulowanego wyżarzania, które znacząco wpływały na jakość działania programu. Jest to na pewno algorytm, w którym znajomość problemu pozwala na znacznie lepsze jego wykonanie. Nie starczyło mi niestety czasu na wyszukanie najbardziej optymalnych parametrów dla poszczególnych instancji, ze względu na długi czas wykonywania niektórych dużych plików, które i tak ograniczałem czasem pięciu minut na wykonanie. Dlatego w każdym teście który przeprowadziłem, parametry między poszczególnymi instancjami w danej wariacji algorytmu różniły się tak naprawdę tylko ilością wykonania poszczególnej instancji, a wartości parametrów początkowych były wybierane albo metodą prób i błędów, albo z informacji zaczerpniętych z literatury.

W zaimplementowanej metodzie głównym wyznacznikiem stopu jest tak naprawdę czas wykonywania programu, oraz średni błąd jaki otrzymuje, idąc zasadą, że lepiej jest mieć rozwiązanie jakiekolwiek, niż żadne. Stworzony algorytm potrafił obliczyć ścieżkę dla instancji powyżej 300 wierzchołków, co jest znacznie lepszym wynikiem, niż dwa poprzednie algorytmy, brute force i Helda-Karpa. Minusem oczywiście jest to, że nie zawsze otrzymywaliśmy wartość najbardziej optymalną. Dodatkowo zużycie pamięci podczas działania programu jest naprawdę niewielkie, gdyż przetrzymujemy w pamięci tylko kilka struktur potrzebnych do wykonywania pracy.



Rysunek 11: Porównanie czasu wykonania dotychczasowo zaimplementowanych algorytmów