Jakub Balcerzak

Projektowanie Efektywnych Algorytmów

Termin zajęć: czwartek godz. 7:30

Prowadzący:

Zadanie projektowe 3

- sprawozdanie

0. Cele projektu:

Celem projektu była implementacja oraz analiza efektywności algorytmu genetycznego, zastosowanego do problemu komiwojażera.

1. Wstęp teoretyczny do algorytmu genetycznego:

1.1. Idea algorytmu genetycznego:

Algorytm genetyczny należy do grupy algorytmów ewolucyjnych. Opiera się on na *populacji* składającej się z *osobników*. Każdy osobnik posiada swój *chromosom*, czyli zestaw *genów* - pojedynczych elementów lub inaczej *cech*. Jeżeli osobnik posiada więcej, niż jeden chromosom, wówczas mówimy o jego *genotypie*. Genotyp przedstawiciela populacji uwidacznia się w jego *fenotypie*, który stanowi wartość, bądź zestaw wartości dla danego konkretnego genotypu.

W algorytmie genetycznym mamy do czynienia z populacją początkową. W myśl idei algorytmu z populacji tej wybierane są najlepsze osobniki. Ich oceny dokonuje *środowisko,* sprawdzając ich jakość (dopasowanie do warunków zewnętrznych). Następnie wybrane osobniki rozmnażają się, generując tym samym potomstwo. Potomkowie powstają na drodze dziedziczenia genotypów rodziców, krzyżowania się ich oraz ewentualnych mutacji.

Kolejnym krokiem jest zastąpienie inaczej *sukcesja* pokolenia poprzedniego przez nowe – lepiej przystosowane pokolenie. W wyniku sukcesji poprzedni osobnicy mogą wymrzeć w całości lub mogą zachować się wyłącznie przedstawiciele, którzy byli najlepiej dopasowani lub przetrwanie poszczególnych jednostek może być losowe.

Powyżej opisany proces przebiega nieustannie i nazywany jest *ewolucją*.

1.2. Zastosowanie strategii genetycznej do optymalizacji problemu komiwojażera:

W swej podstawowej wersji algorytm genetyczny zakłada, że osobnik reprezentowany jest przez stały ciąg bitów. Jednakże rozpatrywany przez nas problem komiwojażera nie jest jednostkowy, zatem należy używać tu raczej terminu strategii genetycznej, niż ściśle algorytmu genetycznego.

W problemie komiwojażera osobnik jest wektorem liczb rzeczywistych, które w najprostszym przypadku oznaczają kolejność odwiedzanych miast – reprezentacja ścieżkowa. (Istnieje także reprezentacja porządkowa.) Wynika z tego, że genotyp osobnika jest permutacją zbioru wszystkich miast.

Oceną jednostki zajmuje się funkcja przystosowania (ang. fitness function). Funkcja ta pełni, więc rolę środowiska. Odwzorowuje ona ciąg wartości (permutację miast) w genotypie w liczbę (wartość) należącą do R^+. Może być użyta bezpośrednio, jako funkcja celu.

Dodatkowo, należy jeszcze określić: sposób generacji populacji początkowej, generatory genetyczne, czyli metodę selekcji oraz operatory krzyżowania i mutacji, a także pozostałe parametry algorytmu takie, jak: rozmiar populacji, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji, warunki zakończenia działania i inne.

Kryterium stopu mogą stanowić: czas, liczba iteracji algorytmu, zadana wartość funkcji dopasowania, czy moment, gdy najlepsze dopasowanie nie ulega poprawie.

Operatory genetyczne zostały omówione w podpunktach 1.3., 1.4., 1.5.

1.3. Selekcja:

Wyróżnia się dwie architektury selekcji:

- generacyjną – tworzona jest populacja P' z N najlepiej dopasowanych osobników z populacji P. W procesie generacji P' biorą udział wyłącznie osobniki z populacji pierwotnej.
- iteracyjną – polegającą na wyborze pojedynczego osobnika N razy. Za każdym razem przeprowadzana na wybranym osobniku jest operacja krzyżowania oraz ewentualna mutacja. Następnie w populacji pierwotnej znajdowane jest miejsce dla nowopowstałego osobnika, co oznacza, że pewien osobnik z populacji P został zabity. W tworzeniu P' biorą udział osobniki z populacji P, ale także osobniki nowe.

W każdej z architektur wybór osobników oparty jest o wartość funkcji dopasowania dla każdego z nich. Najczęściej wybieranymi osobnikami będą te najbardziej dopasowane (duża wartość funkcji).

Istnieje kilka rodzajów metod selekcji:

- selekcja ruletkowa – tworzone są przedziały, rozmiar przedziału zależy od ilości osobników posiadających daną wartość funkcji przystosowania. Następnie losowane są kolejne liczby z dostępnego zakresu. Wylosowana liczba będzie znajdować się pewnym przedziale. Za każdym losowaniem wybierany jest nowy osobnik. Najczęściej wybieranymi jednostkami będą te, których dopasowanie leży w większych przedziałach,
- selekcja turniejowa – w turnieju bierze udział pewna liczba osobników. Zwycięża ten, którego wartość funkcji dopasowania jest najlepsza. Zwycięzca staje się reprezentantem generowanej populacji P'. Turniej przeprowadzany jest wielokrotnie tzn. tyle razy, ilu osobników potrzebujemy, by populacja P' osiągnęła rozmiar N,
- selekcja wagowa,
- selekcja wagowa z liniową funkcją prawdopodobieństwa.

Wynikiem selekcji jest powstanie populacji P' – populacji rodziców.

1.4. Krzyżowanie:

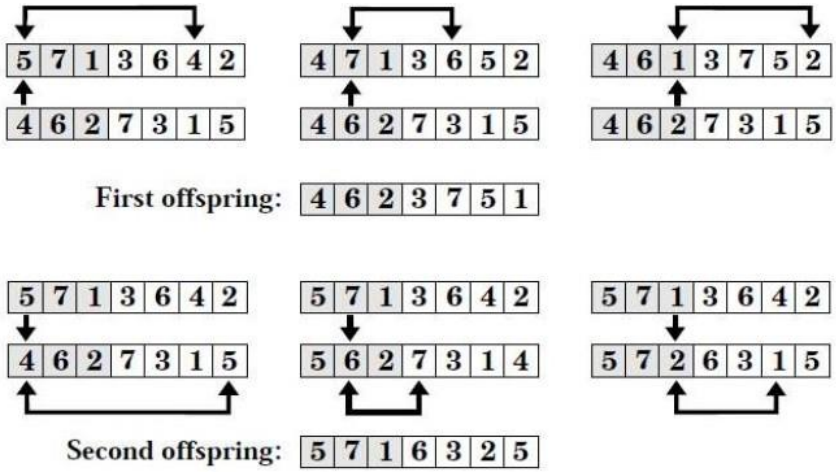
Z utworzonej na drodze selekcji populacji P' wybierana jest za każdym razem losowo para rodziców. Z pary tej pochodzi para dzieci dziedziczących po rodzicach ich genotypy. Z prawdopodobieństwem p następuje u dzieci krzyżowanie się genów.

Stosując strategię genetyczną dla problemu komiwojażera oraz przy wyborze reprezentacji ścieżkowej niezbędne jest zdefiniowanie specjalnych operatorów krzyżowania. Dostępnymi operatorami są m.in..:

- PMX,
- POS,
- EX,
- OX,

- SMX,
- ER,
- AP,
- VR i inne.

Poniższy rysunek prezentuje jedną z metod krzyżowania – PMX. Metoda ta została wykorzystana także w implementacji projektu.



W metodzie tej wybierane są dwa punkty krzyżowania. Następnie iteracyjnie pomiędzy tymi punktami, idąc od lewego punktu, wykonujemy następującą procedurę: sprawdź w genotypie dziecka c1, na której pozycji znajduje się wartość z genotypu rodzica p2 na aktualnie przetwarzanym indeksie. Zamień w c1 pozycje wskazywaną przez bieżący indeks z odnalezioną pozycją, a następnie zwiększ indeks o jeden i powtórz procedurę, aż do osiągnięcia prawego punktu krzyżowania. Następnie procedurę tę przeprowadzamy dla pary c2, p1.

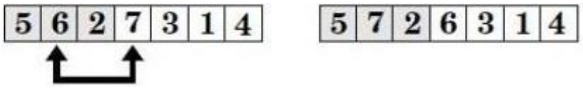
1.5. Mutacje:

Mutacja w genotypie dziecka odbywa się z pewnym prawdopodobieństwem p.

Podobnie, jak w przypadku krzyżowania, operacja mutacji wymaga zdefiniowania specjalnych operatorów mutacji. Są nimi np.:

- inversion
- insertion
- displacemant/swap
- transposition

W projekcie implementowany był operator mutacji – swap. Jego działanie polega na wybraniu dwóch losowych punktów w genotypie, a następnie zamianie genów znajdujących się na wybranych pozycjach. Działanie to prezentuje poniższy rysunek:



1.6. Ogólny schemat działania:

Powyżej opisany algorytm genetyczny można w ogólnej postaci wyrazić poniższym pseudokodem:

```
1.    wybierz populację początkową
2.    oceń przystosowanie osobników
3.    while(warunek zatrzymania)
4.          dokonaj selekcji – utwórz populację rodziców
5.          wykonaj próbę krzyżowania chromosomów
6.          wykonaj próbę mutacji
7.          oceń przystosowanie nowych osobników
8.          utwórz nową populację
9.    zwróć najlepszego osobnika
```

2. Opis implementacji:

W celu wprowadzenia kryterium stopu, jako czasu wykonywania się algorytmu zaimplementowano klasę Timer. Obiekt tej klasy po wywołaniu metody startTime() rozpoczyna odliczanie (analogicznie, jak stoper). Metoda getTime() pozwala uzyskać dokładny czas wyrażony w sekundach. Metoda getIntTime() jest szczególnym przypadkiem powyższej metody, zwraca ona czas również wyrażony w sekundach, lecz bez dodatkowych miejsc po przecinku. Funkcja ta została użyta przy wyświetlaniu wyników bieżących, co sekundę, w trakcie wykonywania się algorytmu genetycznego.

Algorytm genetyczny dla problemu komiwojażera jest zaimplementowany w klasie GeneticAlgorithmTSP. Użytkownik ma dostęp do następujących metod publicznych:

- readGraphFromFile(filename) – funkcja wczytująca graf z pliku o nazwie filename,
- startGA() – funkcja realizująca algorytm genetyczny,
- getBestRoute(d)/getBestCurrentRoute(d) – funkcje zwracające odpowiednio najlepsze znalezione rozwiązanie oraz rozwiązanie bieżące

Zmienne prywatne zawarte wewnątrz klasy potrzebne do prawidłowego działania algorytmu to:

- bestOrder – wyznaczone najlepsze rozwiązanie,
- currentOrder – najlepsze bieżące rozwiązanie,
- population/parents/children – struktury przechowujące populacje osobników,
- numOfKnights – rozmiar turnieju (liczba osobników biorących udział),
- numOfSwaps – liczba mutowanych genów,
- mutationProbability/crossoverProbability – odpowiednio prawdopodobieństwo wystąpienia mutacji oraz krzyżowana

Metody prywatne wykorzystywane w algorytmie genetycznym:

- generateInitialPopulation() – funkcja generująca populację startową składającą się z losowych permutacji miast,
- generateParentsPopulation() – funkcja generująca populację rodziców z populacji pierwotnej (population → parents),
- generateChildrenPopulation() – funkcja generująca populację dzieci z populacji rodziców (parents → children),
- setNewPopulation() – funkcja ustawiająca wygenerowaną populację dzieci, jako populację pierwotną (children → population),
- countRouteLength(order) – funkcja obliczająca długość ścieżki komiwojażera dla permutacji order,
- getRandomProbability() – funkcja zwracająca wartość z przedziału [0,1].

Pozostałe niewymienione wyżej zmienne oraz metody są odpowiednio okomentowane w kodzie źródłowym programu.

3. Pomiary:

Pomiary przeprowadzono dla trzech plików: tsp_17.txt, tsp_56.txt, tsp_171.txt. Cyfra w nazwie wskazuje na rozmiar instancji problemu. Dla każdego z plików obserwowano, jak pomiary będą różnić się w zależności od wielkości populacji. Pomiary dla każdego pliku były przeprowadzane dla trzech różnych rozmiarów populacji: 30, 300, 3000. Tabele zawarte w kolejnych punktach zestawiają te różnice prezentując wartości zwracane przez zaimplementowany algorytm, a także błąd względny liczony w kontekście znanego rozwiązania optymalnego. Wykresy ilustrują zmiany błędu względnego (wyrażonego w %) w zależności od czasu.

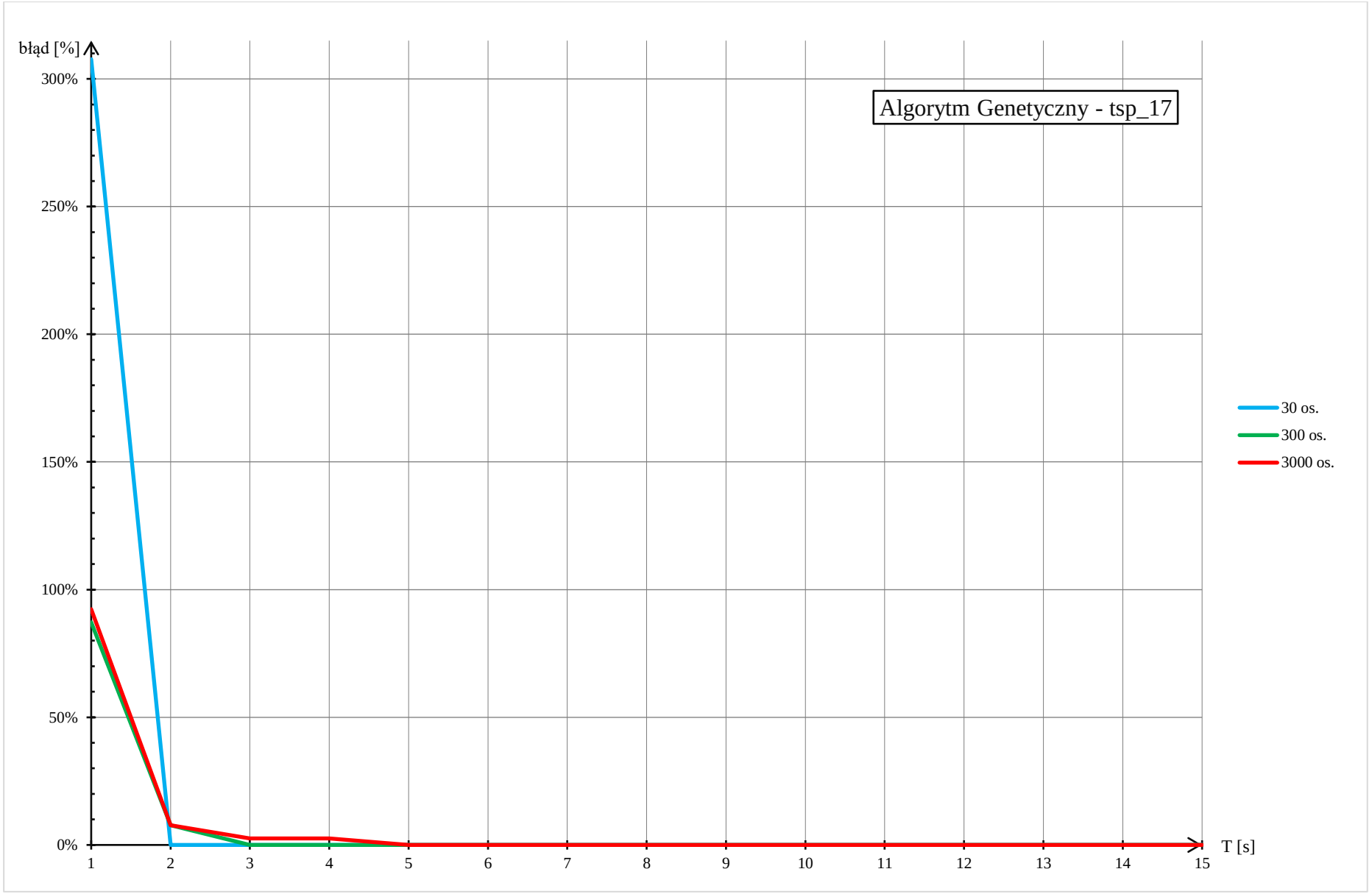
Dodatkowo, w celu porównania algorytmu genetycznego z algorytmem symulowanego wyżarzania dodano wykresy z pomiarami dla trzech rozpatrywanych plików z zadania projektowego 2, gdzie był on implementowany.

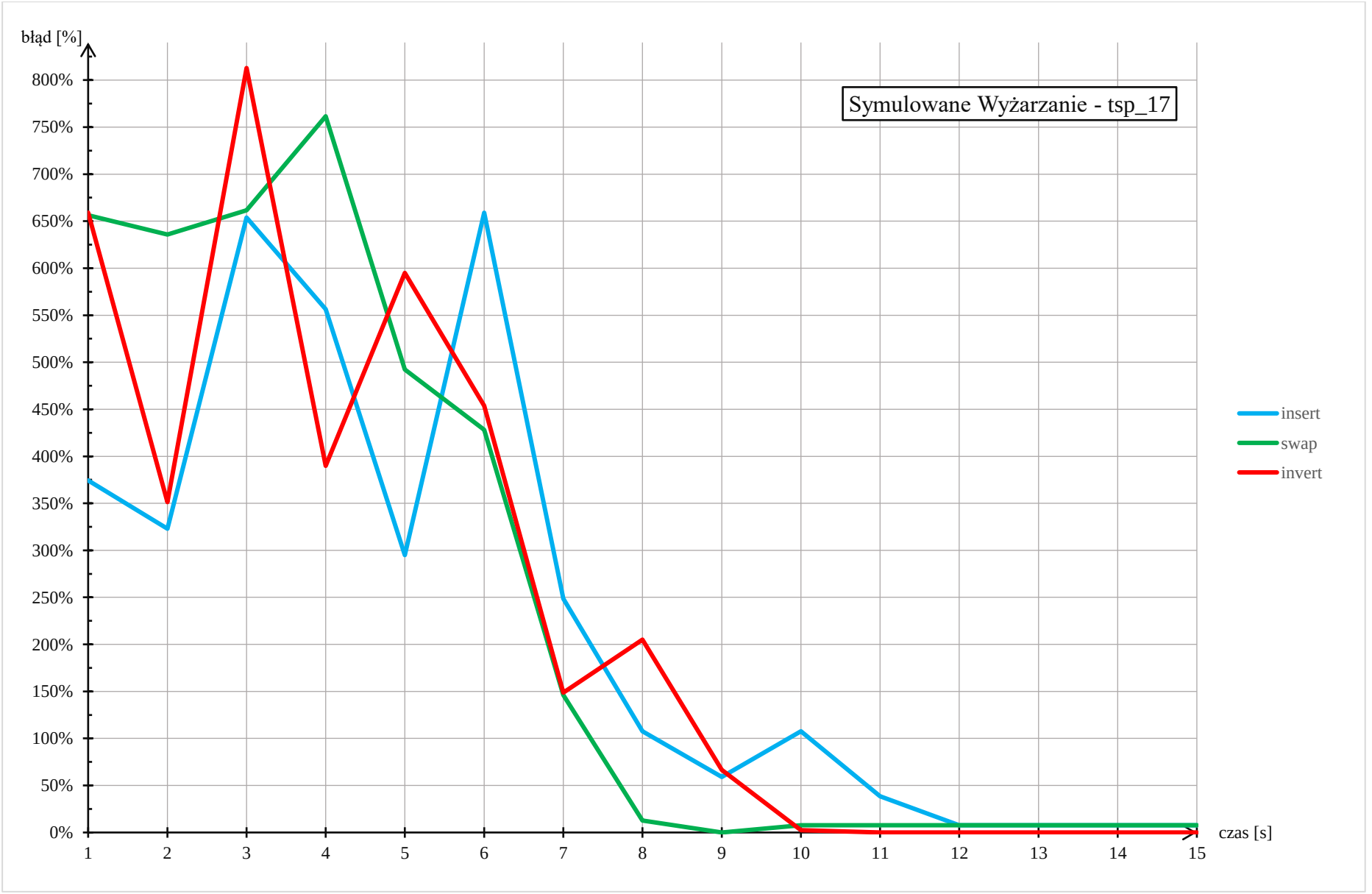
Czasy ustawiane dla algorytmu genetycznego były identyczny z czasami ustawianymi dla symulowanego wyżarzania w zależności od rozmiaru instancji problemu.

3.1. tsp_17.txt:

- T = 15 s
- rozw. optymalne = 39

	30 osobników		300 osobników		3000 osobników	
czas [s]	koszt	błąd [%]	koszt	błąd [%]	koszt	błąd [%]
1	159	307,69%	73	87,18%	75	92,31%
2	39	0,00%	42	7,69%	42	7,69%
3	39	0,00%	39	0,00%	40	2,56%
4	39	0,00%	39	0,00%	40	2,56%
5	39	0,00%	39	0,00%	39	0,00%
6	39	0,00%	39	0,00%	39	0,00%
7	39	0,00%	39	0,00%	39	0,00%
8	39	0,00%	39	0,00%	39	0,00%
9	39	0,00%	39	0,00%	39	0,00%
10	39	0,00%	39	0,00%	39	0,00%
11	39	0,00%	39	0,00%	39	0,00%
12	39	0,00%	39	0,00%	39	0,00%
13	39	0,00%	39	0,00%	39	0,00%
14	39	0,00%	39	0,00%	39	0,00%
15	39	0,00%	39	0,00%	39	0,00%


błąd [%]
300%
250%
200%
150%
100%
50%
0%
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
T [s]
Algorytm Genetyczny - tsp_17
30 os.
300 os.
3000 os.

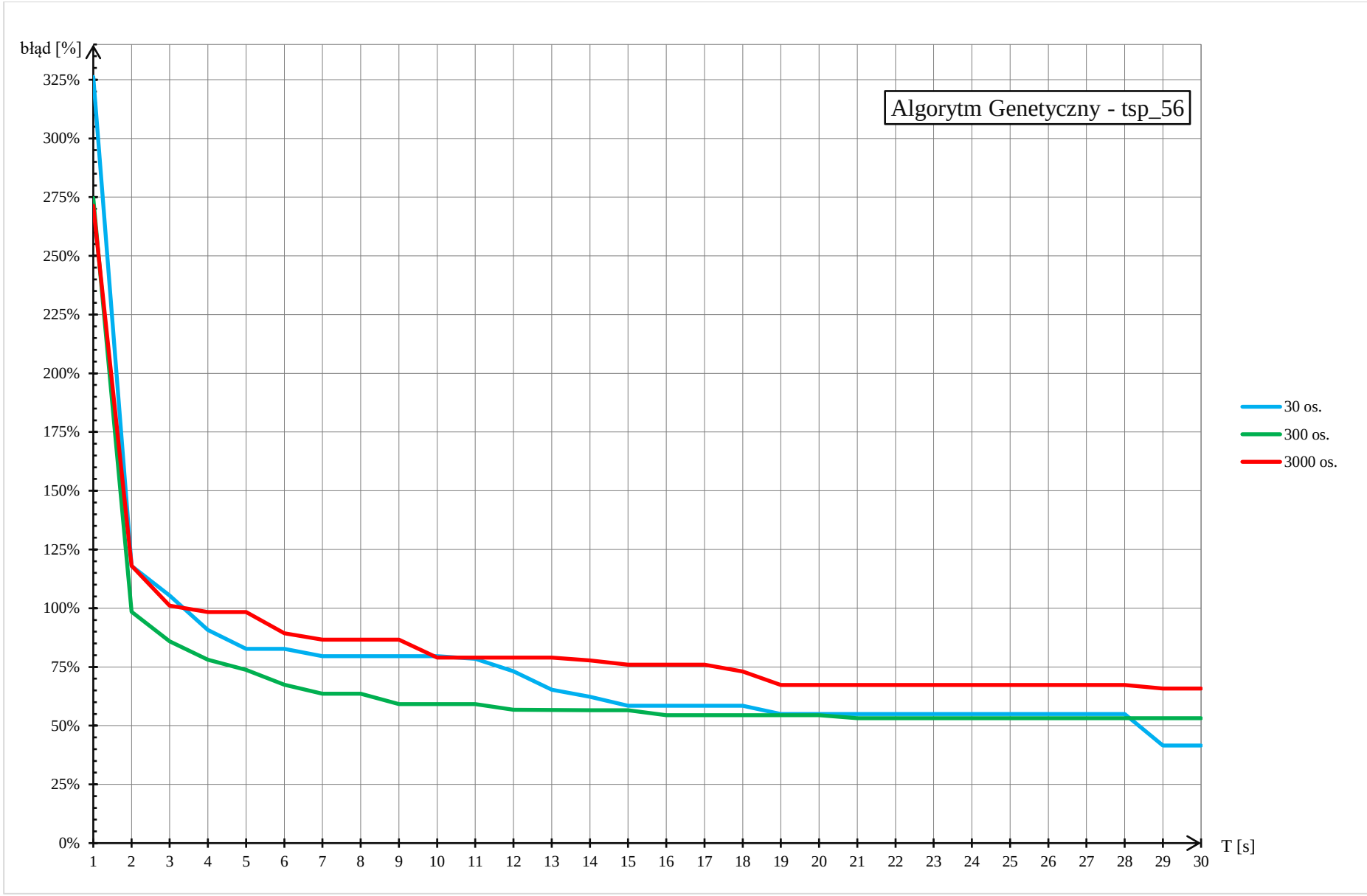

błąd [%]
800%
750%
700%
650%
600%
550%
500%
450%
400%
350%
300%
250%
200%
150%
100%
50%
0%
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
czas [s]
Symulowane Wyżarzanie - tsp_17
insert
swap
invert

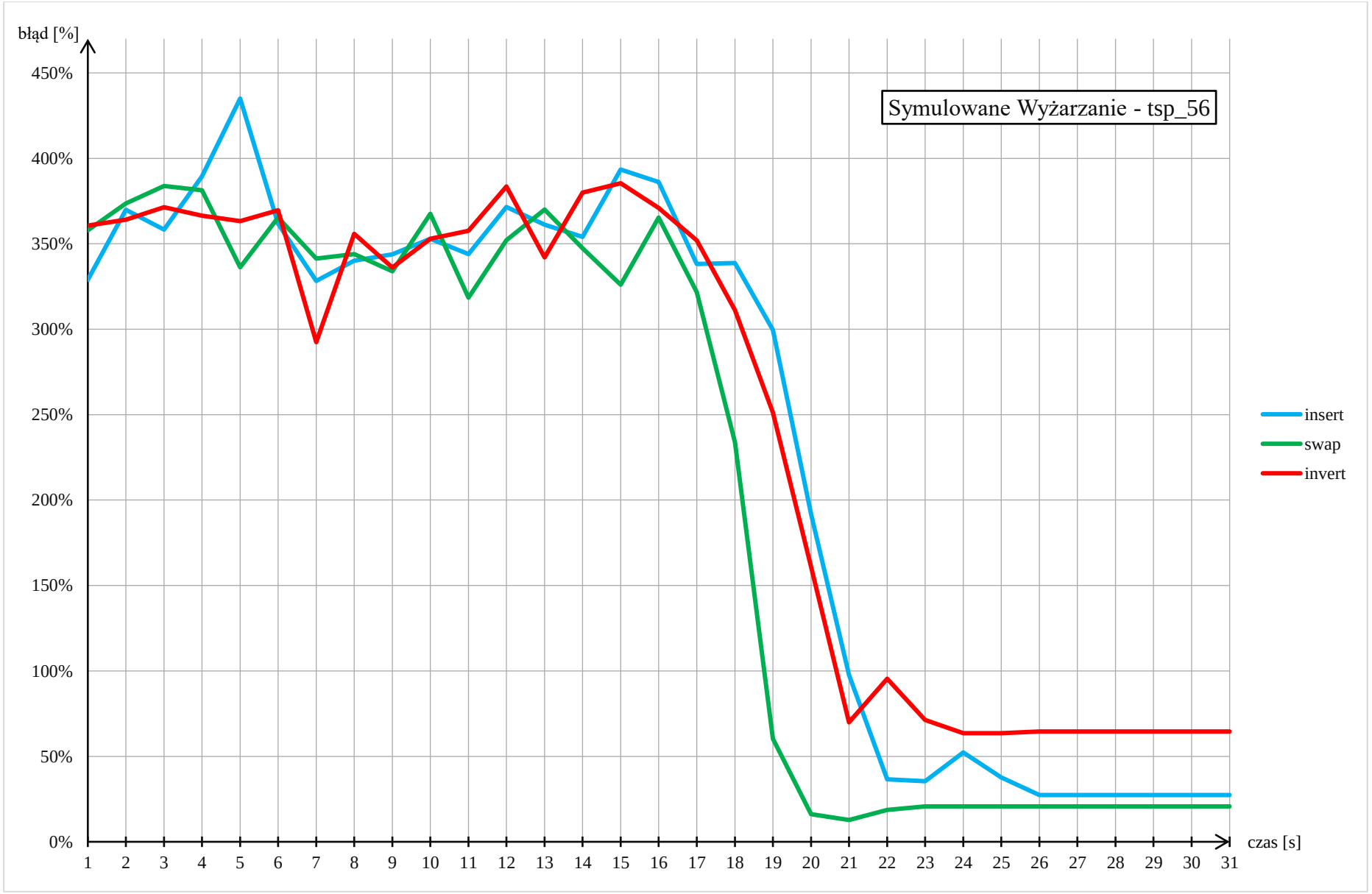
3.2. tsp_56.txt:

- T = 30 s
- rozw. optymalne = 1608

	30 osobników		300 osobników		3000 osobników	
czas [s]	koszt	błąd [%]	koszt	błąd [%]	koszt	błąd [%]
1	6853	326,18%	6024	274,63%	5971	271,33%
2	3505	117,97%	3191	98,45%	3506	118,03%
3	3303	105,41%	2990	85,95%	3234	101,12%
4	3067	90,73%	2863	78,05%	3190	98,38%
5	2938	82,71%	2794	73,76%	3190	98,38%
6	2938	82,71%	2692	67,41%	3045	89,37%
7	2887	79,54%	2631	63,62%	3000	86,57%
8	2887	79,54%	2631	63,62%	3000	86,57%
9	2887	79,54%	2560	59,20%	3000	86,57%
10	2887	79,54%	2560	59,20%	2878	78,98%
11	2870	78,48%	2560	59,20%	2878	78,98%
12	2784	73,13%	2521	56,78%	2878	78,98%
13	2658	65,30%	2519	56,65%	2878	78,98%
14	2609	62,25%	2517	56,53%	2859	77,80%
15	2548	58,46%	2517	56,53%	2829	75,93%
16	2548	58,46%	2483	54,42%	2829	75,93%
17	2548	58,46%	2483	54,42%	2829	75,93%
18	2548	58,46%	2483	54,42%	2783	73,07%
19	2491	54,91%	2483	54,42%	2691	67,35%
20	2491	54,91%	2483	54,42%	2691	67,35%
21	2491	54,91%	2463	53,17%	2691	67,35%
22	2491	54,91%	2463	53,17%	2691	67,35%
23	2491	54,91%	2463	53,17%	2691	67,35%
24	2491	54,91%	2463	53,17%	2691	67,35%
25	2491	54,91%	2463	53,17%	2691	67,35%

26	2491	54,91%	2463	53,17%	2691	67,35%
27	2491	54,91%	2463	53,17%	2691	67,35%
28	2491	54,91%	2463	53,17%	2691	67,35%
29	2276	41,54%	2463	53,17%	2666	65,80%
30	2276	41,54%	2463	53,17%	2666	65,80%


błąd [%]
325%
300%
275%
250%
225%
200%
175%
150%
125%
100%
75%
50%
25%
0%
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
T [s]
Algorytm Genetyczny - tsp_56
30 os.
300 os.
3000 os.


błąd [%]
450%
400%
350%
300%
250%
200%
150%
100%
50%
0%
Symulowane Wyżarzanie - tsp_56
insert
swap
invert
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
czas [s]

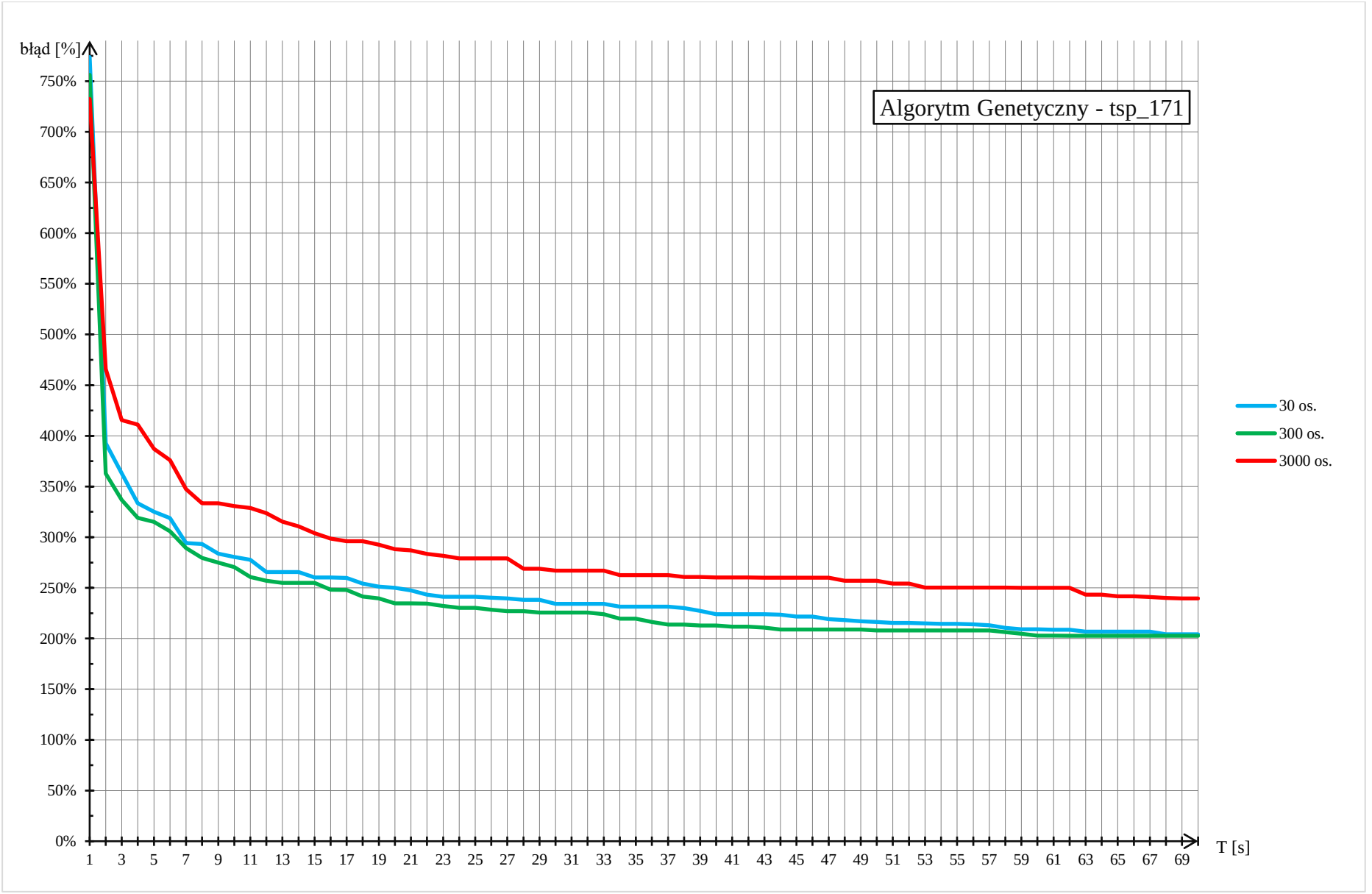
3.3. tsp_171.txt:

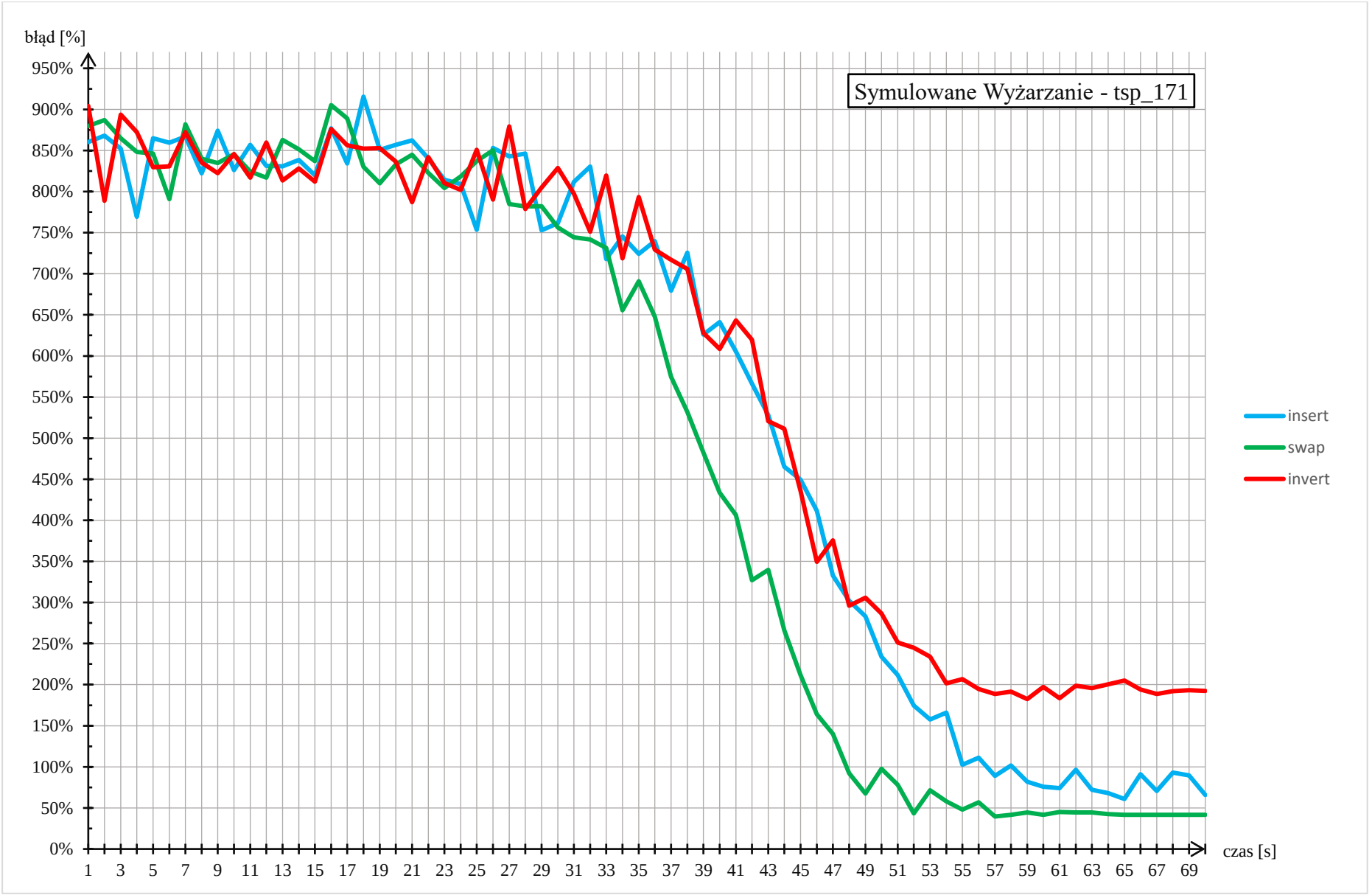
- T = 70 s
- rozw. optymalne = 2755

	30 osobników		300 osobników		3000 osobników	
czas [s]	koszt	błąd [%]	koszt	błąd [%]	koszt	błąd [%]
1	24073	773,79%	23589	756,23%	22932	732,38%
2	13573	392,67%	12746	362,65%	15595	466,06%
3	12754	362,94%	12030	336,66%	14201	415,46%
4	11945	333,58%	11543	318,98%	14082	411,14%
5	11709	325,01%	11438	315,17%	13420	387,11%
6	11541	318,91%	11178	305,74%	13111	375,90%
7	10857	294,08%	10725	289,29%	12324	347,33%
8	10833	293,21%	10455	279,49%	11944	333,54%
9	10570	283,67%	10329	274,92%	11944	333,54%
10	10481	280,44%	10205	270,42%	11864	330,64%
11	10407	277,75%	9938	260,73%	11814	328,82%
12	10075	265,70%	9834	256,95%	11671	323,63%
13	10075	265,70%	9781	255,03%	11445	315,43%
14	10075	265,70%	9781	255,03%	11313	310,64%
15	9923	260,18%	9781	255,03%	11126	303,85%
16	9923	260,18%	9592	248,17%	10984	298,69%
17	9910	259,71%	9589	248,06%	10913	296,12%
18	9759	254,23%	9406	241,42%	10913	296,12%
19	9675	251,18%	9357	239,64%	10817	292,63%
20	9643	250,02%	9222	234,74%	10694	288,17%
21	9571	247,40%	9222	234,74%	10661	286,97%
22	9459	243,34%	9215	234,48%	10564	283,45%
23	9399	241,16%	9151	232,16%	10515	281,67%
24	9399	241,16%	9101	230,34%	10443	279,06%
25	9399	241,16%	9097	230,20%	10443	279,06%

26	9374	240,25%	9047	228,38%	10443	279,06%
27	9353	239,49%	9012	227,11%	10443	279,06%
28	9320	238,29%	9012	227,11%	10162	268,86%
29	9315	238,11%	8972	225,66%	10162	268,86%
30	9209	234,26%	8972	225,66%	10113	267,08%
31	9209	234,26%	8972	225,66%	10113	267,08%
32	9209	234,26%	8972	225,66%	10113	267,08%
33	9209	234,26%	8929	224,10%	10113	267,08%
34	9130	231,40%	8806	219,64%	9989	262,58%
35	9130	231,40%	8806	219,64%	9989	262,58%
36	9130	231,40%	8715	216,33%	9989	262,58%
37	9130	231,40%	8644	213,76%	9989	262,58%
38	9096	230,16%	8644	213,76%	9938	260,73%
39	9018	227,33%	8620	212,89%	9938	260,73%
40	8926	223,99%	8620	212,89%	9924	260,22%
41	8926	223,99%	8590	211,80%	9924	260,22%
42	8926	223,99%	8590	211,80%	9924	260,22%
43	8926	223,99%	8561	210,74%	9920	260,07%
44	8914	223,56%	8511	208,93%	9920	260,07%
45	8862	221,67%	8511	208,93%	9920	260,07%
46	8862	221,67%	8511	208,93%	9920	260,07%
47	8789	219,02%	8511	208,93%	9920	260,07%
48	8767	218,22%	8511	208,93%	9834	256,95%
49	8736	217,10%	8511	208,93%	9834	256,95%
50	8717	216,41%	8483	207,91%	9834	256,95%
51	8688	215,35%	8483	207,91%	9758	254,19%
52	8688	215,35%	8483	207,91%	9758	254,19%
53	8674	214,85%	8483	207,91%	9647	250,16%
54	8663	214,45%	8483	207,91%	9647	250,16%
55	8663	214,45%	8483	207,91%	9647	250,16%

56	8649	213,94%	8483	207,91%	9647	250,16%
57	8623	212,99%	8483	207,91%	9647	250,16%
58	8555	210,53%	8443	206,46%	9647	250,16%
59	8518	209,18%	8393	204,65%	9645	250,09%
60	8514	209,04%	8344	202,87%	9645	250,09%
61	8502	208,60%	8344	202,87%	9645	250,09%
62	8502	208,60%	8340	202,72%	9645	250,09%
63	8452	206,79%	8340	202,72%	9455	243,19%
64	8452	206,79%	8340	202,72%	9455	243,19%
65	8452	206,79%	8340	202,72%	9416	241,78%
66	8452	206,79%	8340	202,72%	9416	241,78%
67	8452	206,79%	8340	202,72%	9394	240,98%
68	8381	204,21%	8340	202,72%	9368	240,04%
69	8381	204,21%	8340	202,72%	9355	239,56%
70	8381	204,21%	8340	202,72%	9355	239,56%


błąd [%]
750%
700%
650%
600%
550%
500%
450%
400%
350%
300%
250%
200%
150%
100%
50%
0%
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
T [s]
Algorytm Genetyczny - tsp_171
30 os.
300 os.
3000 os.


błąd [%]
Symulowane Wyżarzanie - tsp_171
950%
900%
850%
800%
750%
700%
650%
600%
550%
500%
450%
400%
350%
300%
250%
200%
150%
100%
50%
0%
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
czas [s]
insert
swap
invert

4. Wnioski:

Z każdego z trzech prezentowanych wykresów wynika, że wraz z upływem czasu rozwiązania ulegały polepszeniu. Najlepiej obrazują to wykresy dla większych rozmiarów instancji tj. 56 i 171.

Widoczne na nich jest, że rozwiązania wyznaczane przez procedurę setInitialPopulation() w każdym przypadku charakteryzowały się bardzo dużym błędem. Z kolei największy jego spadek przypadał na początkową fazę algorytmu, co jest ściśle związane z faktem przejścia ze stanu eksploracji (duża różnorodność genotypów osobników) do stanu eksploatacji (gdy populacje posiadały już coraz to mniej bardzo złych jednostek, a osobniki coraz bardziej upodabniały się do siebie).

Dla przypadku tsp_17 wartość błędu spadła o 307%. Dla rozmiaru 56-u miast stosunek błędu początkowego do końcowego wyniósł 7,85, a dla 171-u miast 3,6. Same rozwiązania uległy polepszeniu odpowiednio: 4-o, 3, 2,87-o krotnie.

Co więcej, na podstawie wykresów dla tsp_56, tsp_171 można wnioskować, że duże populacje negatywnie wpływały na optymalizację. W obu przypadkach, gdy populacja miała rozmiar 3-ech tysięcy osobników, algorytm zwracał najgorsze wyniki (w stosunku do pozostałych dwóch liczności populacji). Najlepsze wyniki pozwalały osiągnąć populacje małe – 300 os. oraz bardzo małe – 30 os.

Ponadto, podczas testowania algorytmu oraz przeprowadzania serii pomiarów obserwowane było „utykanie" algorytmu w lokalnych minimach. Zabezpieczenie przed tym stanowić powinny mutacje. Mutacja 'swap' nie spełniała jednak swojego zadania. Algorytm, co widać po coraz bardziej spłaszczających się liniach wykresu, ma tendencje do 'nasycania się' i traci wraz z tym swoją intensywność. Rozwiązania polepszają się coraz rzadziej, o coraz mniejsze wartości. Wydłużenie czasu działania oraz zwiększenie liczby iteracji nie wpływały znacząco na uzyskiwane rezultaty.

Metoda krzyżowania PMX oraz metoda mutacji 'swap' należą do najłatwiejszych w implementacji operatorów genetycznych. Zastosowanie bardziej złożonych metod mogłoby skutkować uzyskaniem lepszych wyników. Sposób przeprowadzania selekcji także ma duże znaczenie. W prezentowanej implementacji algorytmu genetycznego stara populacja zostaje w całości zastąpiona przez nową. Dodatkowo, populacja rodziców wybierana jest generacyjnie. Powstają więc pytania, czy lepsza jest architektura iteracyjna? Czy powinno pozostawiać się przy życiu część starej populacji np. najlepsze osobniki? Każda podjęta decyzja dot. sposobu implementacji poszczególnych operacji i operatorów ma wpływ na ostateczną jakość działania algorytmu genetycznego.

Na koniec, porównując działanie zaimplementowanego algorytmu genetycznego z implementowanym w 2. zadaniu projektowym algorytmem symulowanego wyżarzania dokonano zestawienia wartości najlepszych otrzymanych rozwiązań dla obu algorytmów. Prócz tego wyliczono odpowiednie stosunki wskazujące na jakość otrzymywanych wyników. W tabeli uwzględniono pomiary dla plików: tsp_56, tsp_171.

	SA*		GA**			
	Koszt	Błąd [%]	Koszt	Błąd [%]	K_{GA}/K_{SA}	B_{GA}/B_{SA}
tsp_56	1731	7,65	2276	41,54	1,32	5,43
tsp_171	3618	31,32	8340	202,72	2,31	6,47

* SA (ang. simmulated annealing) – symulowane wyżarzanie

** GA (ang. genetic algorithm) – algorytm genetyczny

Jak wynika z powyższej tabeli w przypadku rozmiaru problemu 171 algorytm symulowanego wyżarzania otrzymał rozwiązanie, którego koszt był ponad dwukrotnie mniejszy od kosztu otrzymanego w tym samym czasie przez algorytm genetyczny. Stosunek błędów wyniósł 6,47.