Projektowanie efektywnych algorytmów

Zadanie 3: Implementacja i analiza efektywności algorytmu genetycznego dla wybranego problemu Komiwojażera

Autor: Michał Sikacki (259152)

Grupa projektowa: czwartek 13:15- 15:00

Prowadzący: mgr inż. Antoni Sterna

1. Wstęp teoretyczny

Algorytm przedstawiony w tym zadaniu ma za zadanie rozwiązać problem Komiwojażera. Podobnie jak w przypadku wcześniej omawianego w drugim zadaniu projektowym symulowanego wyżarzania oraz tabu search, algorytm genetyczny stara się odszukać rozwiązanie, które jest bliskie idealnemu (nie zawsze odszuka najlepsze możliwe rozwiązanie). Jest on wykorzystywany głównie dla większych problemów, dla których takie algorytmy jak B&B lub DP nie znalazłyby rozwiązanie w rozsądnym czasie.

Algorytm genetyczny to rodzaj heurystyki zaliczany do tzn. algorytmów ewolucyjnych. Oznacza to, że stara się on naśladować bądź symulować naturalne procesy ewolucyjne zachodzące w przyrodzie. Pionierem tego typu algorytmów jest John Henry Holland, który po raz pierwszy zaprezentował tego typu algorytm w 1975 r.

Pojęcia jakie wykorzystywane są w tym algorytmie odzwierciedlają mechanizmy ewolucyjne. Są to m.in. selekcja, krzyżowanie i mutacje, które są etapami działania algorytmu. Oprócz tego pojawiają się takie pojęcia jak chromosom, czyli reprezentacja potencjalnego rozwiązania.

Algorytm ten przede wszystkim generuje określone populacje złożone z osobników (rozwiązania problemów), które po każdej iteracji algorytmu ulegają modyfikacji tzn. w miejsce starego pokolenia jest generowane nowe, które przybliża nas do znalezienia lepszego rozwiązania.

Pierwszym etapem w każdej iteracji jest selekcja osobników. Polega ona na wybraniu części osobników z populacji do następnego etapu zwanego krzyżowaniem. Metod selekcji jest wiele i wybór odpowiedniej może się różnić w zależności od problemu, którego chcemy rozwiązać. Metoda jaka została wybrana w tym zadaniu to metoda ruletki. Polega ona w skrócie na tym, że każdemu osobnikowi przydzielamy prawdopodobieństwo, że zostanie wybrany do krzyżowania. Konkretnie, dla każdego osobnika i-tego z populacji, przydziela się mu prawdopodobieństwo selekcji pi = fi / (sigma(f)) gdzie fi to jego przystosowanie, a sigma(f) to suma przystosowań dla całej populacji. Przystosowanie jest tym wyższe, tym lepsze jest rozwiązanie. Następnie losujemy losowy punkt z zakresu od 0 do sumy prawdopodobieństw selekcji dla każdego osobnika. Taka metoda pozwala na przyznaniu większej szansy na wybranie do krzyżowania dla lepszych osobników, dzięki czemu przekazujemy lepsze geny kolejnemu pokoleniu. Z drugiej jednak strony nie odrzuca ona koniecznie gorszych osobników, dzięki czemu algorytm nie zatrzyma się zbyt szybko w lokalnym optimum. Do krzyżowania w zależności od potrzeby może zostać wybrana różna część populacji.

Kolejnym krokiem jest krzyżowanie. Z populacji, która przeszła przez selekcję wybieramy dwóch osobników nazywanych rodzicami, które ze sobą skrzyżujemy w celu otrzymania nowych osobników zwanych potomkami. Krzyżowanie zachodzi z określonym prawdopodobieństwem. Metod dla tego kroku jest wiele. W zadaniu została wybrana metoda Davis's ordered crossover. Polega ona na losowaniu dwóch punktów krzyżowania na chromosomach rodzicielskich i kopiowaniu fragmentów między dwoma tymi punktami do ich potomków. Pozostałe geny dopasowywane do chromosomu potomnego w określony sposób. Załóżmy, że mamy dwóch rodziców. Niech genami tych rodziców będzie kolejnością miast w problemie Komiwojażera. Założymy, że ich genotypy to:

Rodzic A: [10, 2, 7, 4, 8, 6, 1, 3, 5, 9]
Rodzic B: [5, 7, 8, 3, 1, 2, 4, 6, 9, 10]

Wybieramy punkt krzyżowania 4 i 7. Punkty krzyżowania iterujemy od 0. Kopiujemy fragmenty genów od 4 do 7:

Rodzic A: [-, -, -, -, 8, 6, 1, 3, -, -]
Rodzic B: [-, -, -, -, 1, 2, 4, 6, -, -]

Dla potomka A będziemy uzupełniać resztę genów zaczynając w miejscu na prawo od drugiego punktu krzyżowania (7). Teraz należy spojrzeć na rodzica B. Tutaj również ustawiamy wskaźnik w tym samym punkcie co wcześniej. Sprawdzamy czy dany gen, na którym jest ustawiony wskaźnik w rodzicu B już wystąpił w potomku A. Jeśli nie to dodajemy go do potomka A we wcześniej wyznaczonym punkcie i inkrementujemy wskaźniki dla rodzica i potomka. Jeśli tak to inkrementujemy wskaźnik dla rodzica B w celu znalezienia genu, który nie wystąpił w potomku A. Jeśli wszystkie elementy na prawo od większego punktu krzyżowania zostaną wypełnione w potomku, to wypełniamy to, co zostało (czyli wracamy do indeksu 0 potomka). Oto jak będą wyglądali potomkowie po tym procesie:

Rodzic A: [5, 7, 2, 4, 8, 6, 1, 3, 9, 10]
Rodzic B: [10, 7, 8, 3, 1, 2, 4, 6, 5, 9]

Zaletą użycia takiej metody może być m. in. zachowanie kolejności elementów na chromosomach potomnych, co oznacza przeniesienie pewnej cechy rodzica na potomka.

Jeszcze innym zagadnieniem jest mutacja. Jej celem jest wprowadzenie większej różnorodności genetycznej. Tutaj również ustawiamy parametr prawdopodobieństwa zajścia mutacji. Gdyby ten etap zostałby pominięty w algorytmie, to niestety zwiększamy szanse na zatrzymanie się w lokalnym optimum. Wyróżniamy różne metody implementacji mutacji. W tym zadaniu projektowym zostały wybrane dwa: transpozycja i inwersja. Transpozycja polega na zamianie dwóch losowych genów w osobniku. W problemie TSP zamieniamy ze sobą dwa miasta w sekwencji miast. Inwersja wybiera dwa różne punkty w sekwencji i odwraca kolejność występowania miast pomiędzy nimi. Poniżej zaprezentowane zostały przykładowe mutacje:

Transpozycja:

Przed mutacją: [5, 7, 2, 4, 8, 6, 1, 3, 9, 10]
Po mutacji: [5, 7, 3, 4, 8, 6, 1, 2, 9, 10]

Inwersja:

Przed mutacją: Przed mutacją: [5, 7, 2, 4, 8, 6, 1, 3, 9, 10]
Po mutacji: Przed mutacją: [5, 7, 2, 1, 6, 8, 4, 3, 9, 10]

Po wykonaniu wszystkich operacji, uzyskane nowe pokolenie można zmniejszyć usuwając najgorszych osobników po każdej iteracji.

Złożoność tego algorytmu nie jest łatwo wyznaczyć. Będzie ona jednak w skrócie wyglądała następująco:

$O(P * G * O(\text{selekcja}) * ((P_c * O(\text{krzyżowanie})) + (P_m * O(\text{mutacja}))))$

Gdzie:

P – wielkość populacji,
G – liczba pokoleń,
P_c – prawdopodobieństwo krzyżowania
P_m – prawdopodobieństwo mutacji

Algorytm nie jest stosunkowo trudny w implementacji. Nie jest jednak proste dobranie odpowiednich parametrów w celu osiągnięcia jak najlepszych rozwiązań.

Skrócony schemat działania algorytmu:

1. wybór populacji początkowej chromosomów (losowy)
2. ocena przystosowania chromosomów
3. sprawdzanie warunku zatrzymania
    a. selekcja chromosomów - wybór populacji macierzystej (ang. mating pool)
    b. krzyżowanie chromosomów z populacji rodzicielskiej
    c. mutacja - może być również wykonana przed krzyżowaniem
    d. ocena przystosowania chromosomów
    e. utworzenie nowej populacji
4. wyprowadzenie ,,najlepszego'' rozwiązania

2. Opis implementacji algorytmu

Klasa w projekcie, w której zostały napisane wszystkie funkcje i procedury potrzebne dla algorytmu symulowanego wyżarzania znajduje się w plikach *Genetic.cpp* oraz *Genetic.hpp*. Główną metodą klasy *Genetic* jest *algorithm()*, która wywołuje kolejne etapy algorytmu genetycznego. W funkcji *setParameters()* można określić przede wszystkim warunek stopu (czas podany w sekundach), wielkość populacji dla każdego pokolenia (każde pokolenie będzie miało tą samą liczbę osobników), prawdopodobieństwo mutacji i krzyżowania oraz typ wybranej mutacji. Metoda *rouletteSelection()*, dokonuje selekcji metodą ruletki. Metoda *populationInit()* wybiera populację początkową. Za pomocą *davisCrossover()* krzyżujemy ze sobą osobniki w populacji podaną we wstępie teoretycznym metodą. Do mutacji wykorzystujemy funkcje *mutate()*, a dla utworzenia nowego pokolenia metodę *elite()*, która usuwa dotychczasowo najgorsze osobniki z populacji. Metodą *calculateCurrentValue()* obliczamy drogę dla danej sekwencji wierzchołków. Funkcja *showResult()* wyświetla wyniki działania algorytmu. Do pomiaru czasu wykonywania algorytmu została wykorzystana biblioteka <chrono>. W trakcie wykonywania algorytmu w podanym przez użytkownika czasie sprawdzamy, jakie rozwiązanie jest na danym etapie.

3. Plan eksperymentu.

W wybranych punktach pomiarowych (czasowych) będziemy zapisywali dotychczasową drogę jaką wyznaczył nasz algorytm. Dla każdego punktu pomiar wykonujemy 10 razy i obliczamy średnią arytmetyczną w celu poprawienia jakości uzyskanych wyników testów. Należy ją porównać z rozwiązaniem idealnym. W eksperymencie tym zostały wybrane więc

trzy różne pliki zawierające graf tzn. br17.txt, ftv55.txt oraz ftv170.txt zawierające kolejno 17, 56 oraz 171 wierzchołków, dla których rozwiązanie optymalne jest znane. Porównanie to zostanie przedstawione jako błąd względny. Zatem podstawowe wykresy przedstawiać będą błąd względny w funkcji czasu wykonywania algorytmu.

Najlepsze znane rozwiązania dla plików:

- br17.txt – 39
- ftv55.txt – 1608
- ftv170.txt – 2755

Punkty pomiarowe (w sekundach) zostały zdefiniowane następująco:

- dla pliku br17.txt: [0.03125, 0.0625, 0.09375, 0.125, 0.15625, 0.1875, 0.21875, 0.25, 0.28125, 0.3125, 0.34375, 0.375, 0.40625, 0.4375, 0.46875, 0.5],
- dla pliku ftv55.txt: [0.9375, 1.875, 2.8125, 3.75, 4.6875, 5.625, 6.5625, 7.5, 8.4375, 9.375, 10.3125, 11.25, 12.1875, 13.125, 14.0625, 15]
- dla pliku ftv170.txt: [3.75, 7.5, 11.25, 15, 18.75, 22.5, 26.25, 30, 33.75, 37.5, 41.25, 45, 48.75, 52.5, 56.25, 60]

Wybór różnych punktów pomiarowych dla plików wynikał z różnych czasów znajdowania optimum w zależności od ilości wierzchołków grafu.

Testy zostały wykonane dla domyślnego prawdopodobieństwa mutacji wynoszącego 0.01 oraz prawdopodobieństwa krzyżowania 0.8. Wielkości populacji, które ze sobą porównujemy wynoszą kolejno 100, 250, 500. W dodatkowym polu tabeli przedstawionych poniżej znajduje się pole *Najlepszy wynik (%)*, który podaje jaki najmniejszy błąd względny dla danego pomiaru uzyskano ze wszystkich dokonanych 10 pomiarów branych do średniej. Średni błąd jest pokazany w kolumnie *Błąd względny (%)*. Średnio uzyskana droga jest ukazana w kolumnie *Znaleziona droga*. Porównanie dwóch różnych zastosowanych typów mutacji (transpozycja i inwersja) zostały ukazane na tym samym wykresie.

4. Wyniki eksperymentu.

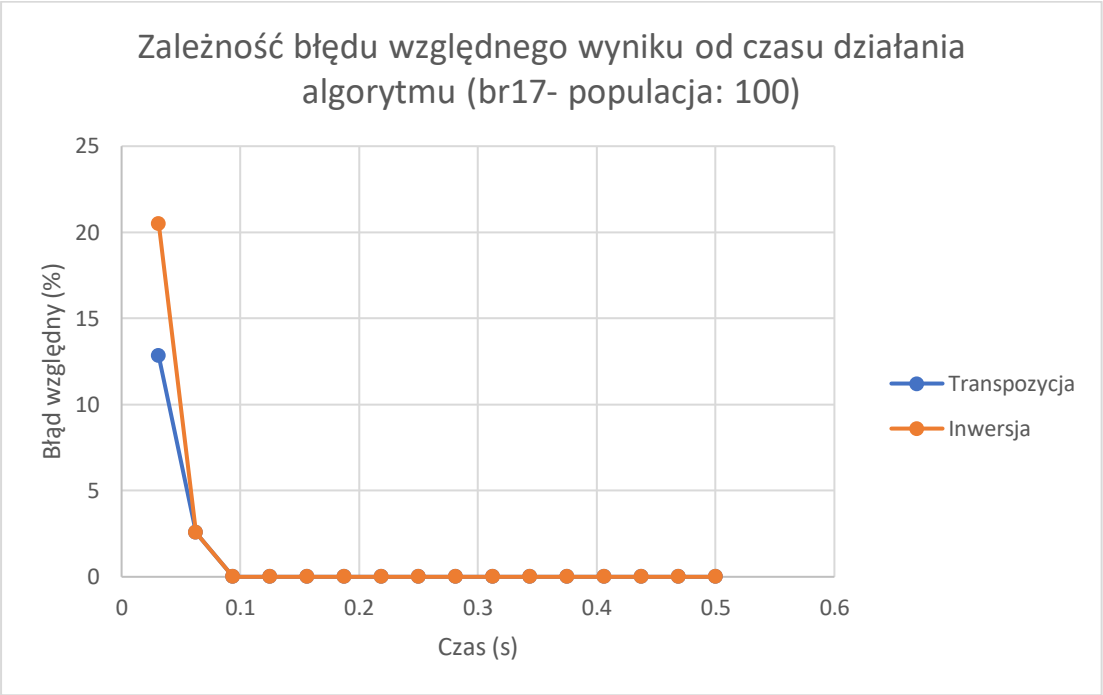
a) br17.txt

Tabela 1. Wyniki dla pliku testowego br17.txt (populacja: 100, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.03125	44	12.82051282	0
0.0625	40	2.564102564	
0.09375	39	0	
0.125	39	0	
0.15625	39	0	
0.1875	39	0	
0.21875	39	0	
0.25	39	0	
0.28125	39	0	
0.3125	39	0	
0.34375	39	0	
0.375	39	0	
0.40625	39	0	
0.4375	39	0	
0.46875	39	0	
0.5	39	0	

Tabela 2. Wyniki dla pliku testowego br17.txt (populacja: 100, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.03125	47	20.51282051	0
0.0625	40	2.564102564	
0.09375	39	0	
0.125	39	0	
0.15625	39	0	
0.1875	39	0	
0.21875	39	0	
0.25	39	0	
0.28125	39	0	
0.3125	39	0	
0.3437	39	0	
0.375	39	0	
0.40625	39	0	
0.4375	39	0	
0.46875	39	0	
0.5	39	0	



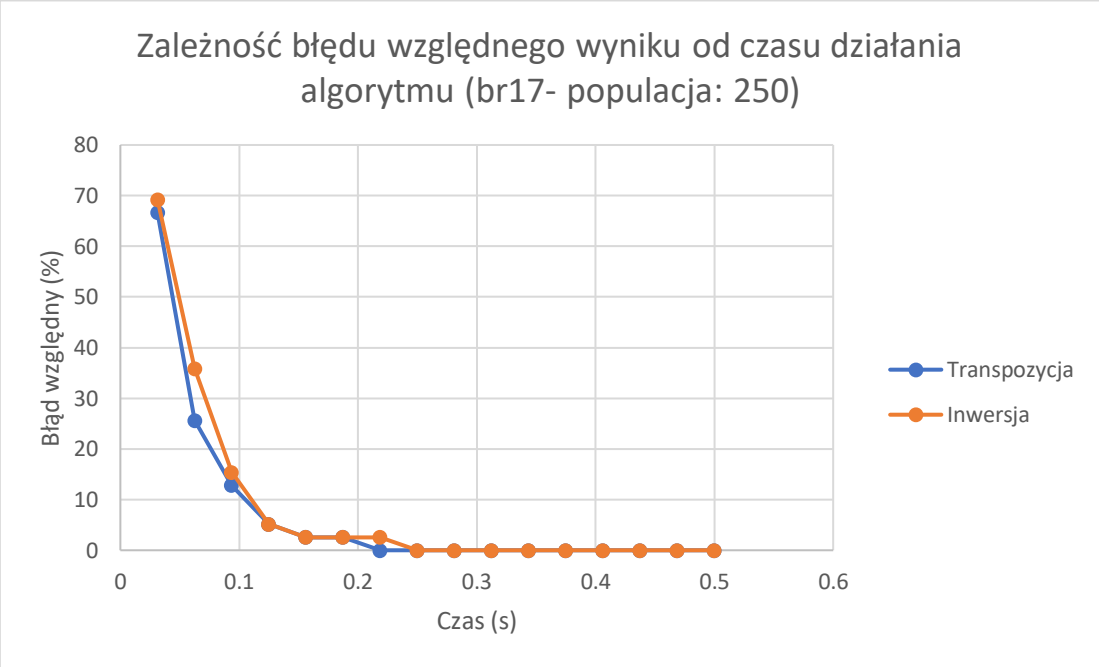
Rys. 1 Wykres dla pliku testowego br17.txt (populacja: 100)

Tabela 3. Wyniki dla pliku testowego br17.txt (populacja: 250, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.03125	65	66.66666667	0
0.0625	49	25.64102564	
0.09375	44	12.82051282	
0.125	41	5.128205128	
0.15625	40	2.564102564	
0.1875	40	2.564102564	
0.21875	39	0	
0.25	39	0	
0.28125	39	0	
0.3125	39	0	
0.34375	39	0	
0.375	39	0	
0.40625	39	0	
0.4375	39	0	
0.46875	39	0	
0.5	39	0	

Tabela 4. Wyniki dla pliku testowego br17.txt (populacja: 250, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.03125	66	69.23076923	0
0.0625	53	35.8974359	
0.09375	45	15.38461538	
0.125	41	5.128205128	
0.15625	40	2.564102564	
0.1875	40	2.564102564	
0.21875	40	2.564102564	
0.25	39	0	
0.28125	39	0	
0.3125	39	0	
0.3437	39	0	
0.375	39	0	
0.40625	39	0	
0.4375	39	0	
0.46875	39	0	
0.5	39	0	



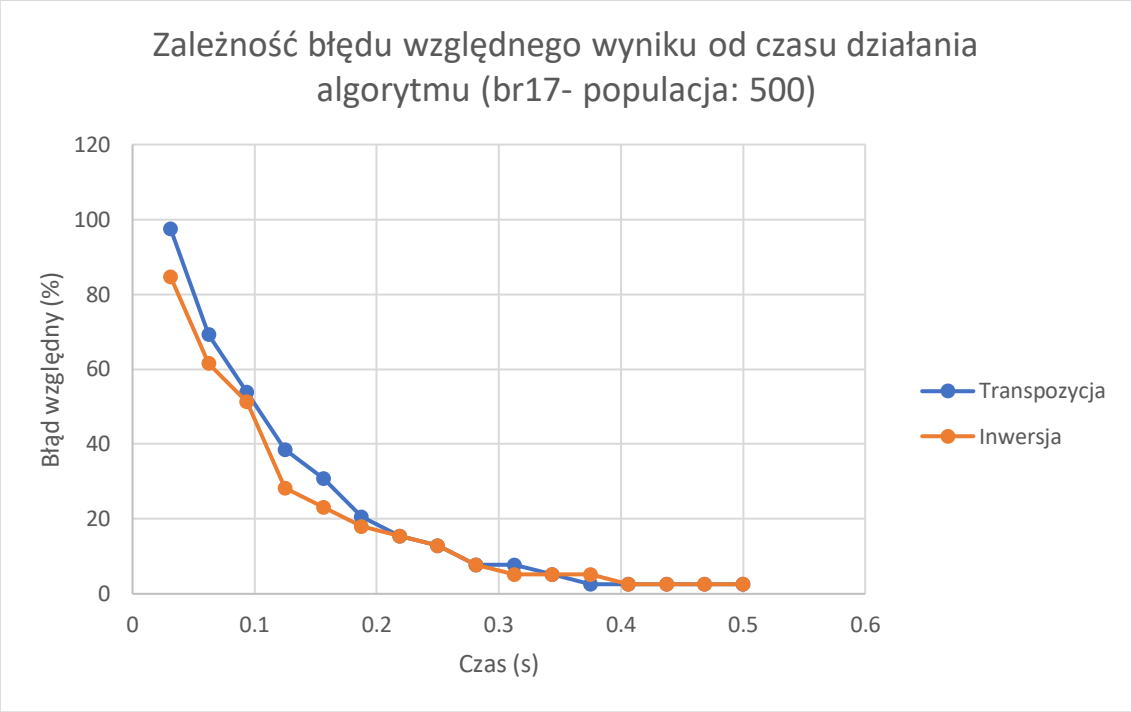
Rys. 2 Wykres dla pliku testowego br17.txt (populacja: 250)

Tabela 5. Wyniki dla pliku testowego br17.txt (populacja: 500, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.03125	77	97.43589744	0
0.0625	66	69.23076923	
0.09375	60	53.84615385	
0.125	54	38.46153846	
0.15625	51	30.76923077	
0.1875	47	20.51282051	
0.21875	45	15.38461538	
0.25	44	12.82051282	
0.28125	42	7.692307692	
0.3125	42	7.692307692	
0.34375	41	5.128205128	
0.375	40	2.564102564	
0.40625	40	2.564102564	
0.4375	40	2.564102564	
0.46875	40	2.564102564	
0.5	40	2.564102564	

Tabela 6. Wyniki dla pliku testowego br17.txt (populacja: 500, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.03125	72	84.61538462	0
0.0625	63	61.53846154	
0.09375	59	51.28205128	
0.125	50	28.20512821	
0.15625	48	23.07692308	
0.1875	46	17.94871795	
0.21875	45	15.38461538	
0.25	44	12.82051282	
0.28125	42	7.692307692	
0.3125	41	5.128205128	
0.3437	41	5.128205128	
0.375	41	5.128205128	
0.40625	40	2.564102564	
0.4375	40	2.564102564	
0.46875	40	2.564102564	
0.5	40	2.564102564	



Rys. 3 Wykres dla pliku testowego br17.txt (populacja: 500)

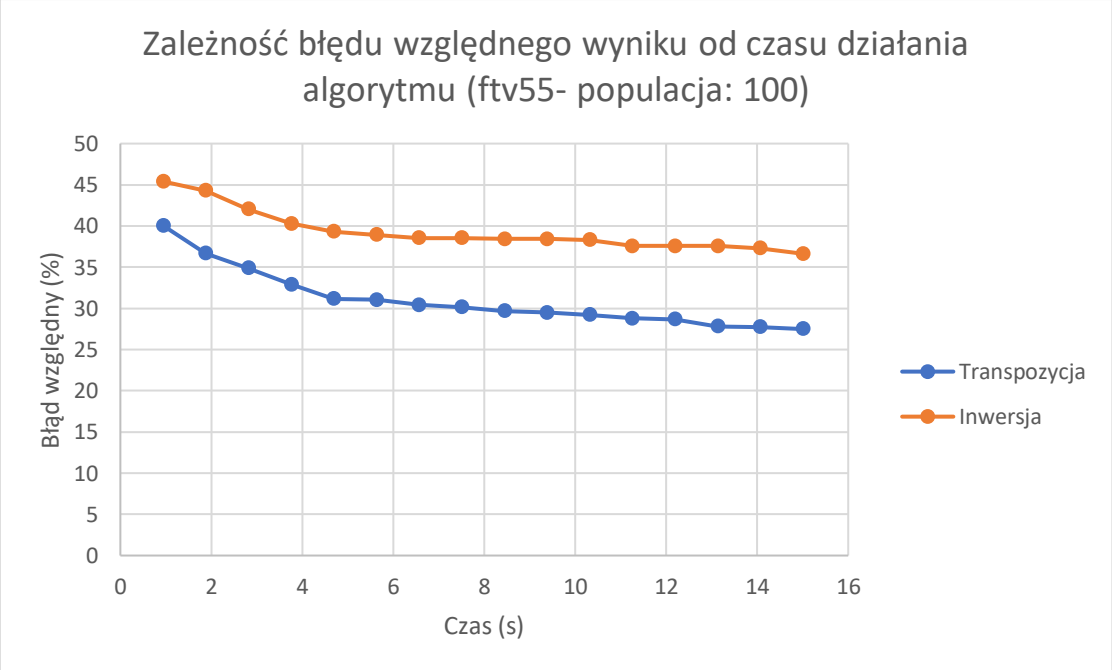
a) ftv55.txt

Tabela 7. Wyniki dla pliku testowego ftv55.txt (populacja: 100, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.9375	2252	40.04975124	12.06467662
1.875	2198	36.69154229	
2.8125	2169	34.8880597	
3.75	2137	32.89800995	
4.6875	2109	31.15671642	
5.625	2107	31.03233831	
6.5625	2097	30.41044776	
7.5	2093	30.16169154	
8.4375	2085	29.6641791	
9.375	2082	29.47761194	
10.3125	2078	29.22885572	
11.25	2071	28.79353234	
12.1875	2069	28.66915423	
13.125	2055	27.79850746	
14.0625	2054	27.73631841	
15	2050	27.48756219	

Tabela 8. Wyniki dla pliku testowego ftv55.txt (populacja: 100, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.9375	2338	45.39800995	21.82835821
1.875	2320	44.27860697	
2.8125	2284	42.039801	
3.75	2256	40.29850746	
4.6875	2240	39.30348259	
5.625	2234	38.93034826	
6.5625	2228	38.55721393	
7.5	2228	38.55721393	
8.4375	2226	38.43283582	
9.375	2226	38.43283582	
10.3125	2224	38.30845771	
11.25	2212	37.56218905	
12.1875	2212	37.56218905	
13.125	2212	37.56218905	
14.0625	2208	37.31343284	
15	2197	36.62935323	



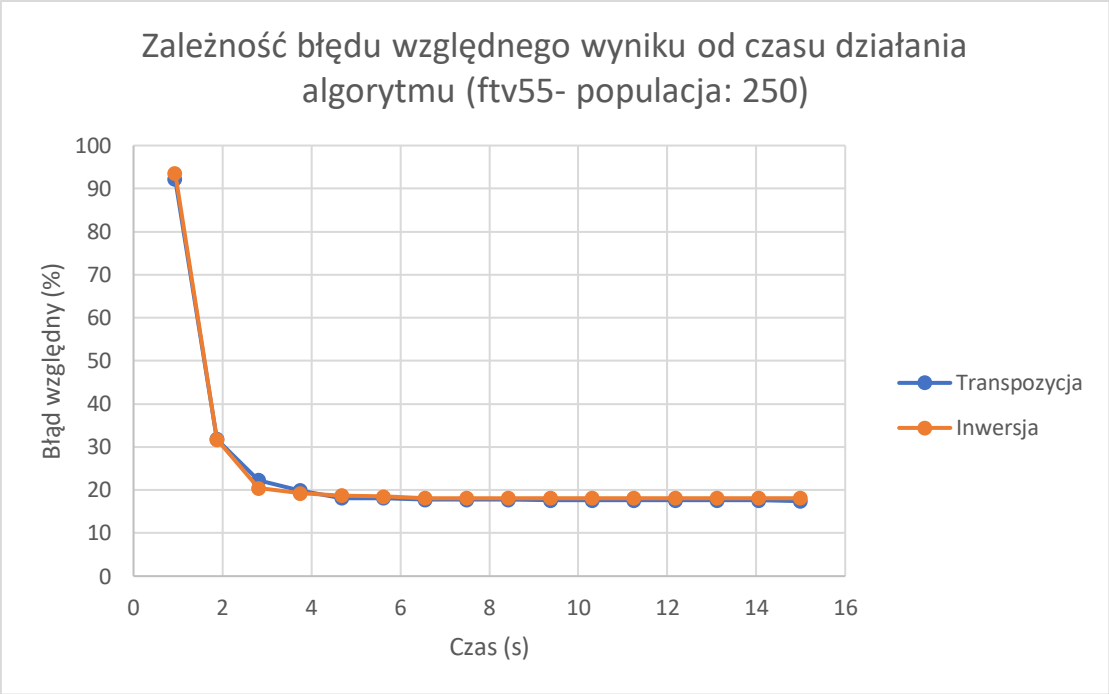
Rys. 4 Wykres dla pliku testowego ftv55.txt (populacja: 100)

Tabela 9. Wyniki dla pliku testowego ftv55.txt (populacja: 250, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.9375	3090	92.1641791	11.31840796
1.875	2120	31.84079602	
2.8125	1966	22.26368159	
3.75	1928	19.90049751	
4.6875	1899	18.09701493	
5.625	1899	18.09701493	
6.5625	1894	17.78606965	
7.5	1894	17.78606965	
8.4375	1893	17.7238806	
9.375	1891	17.59950249	
10.3125	1891	17.59950249	
11.25	1891	17.59950249	
12.1875	1891	17.59950249	
13.125	1891	17.59950249	
14.0625	1891	17.59950249	
15	1888	17.41293532	

Tabela 10. Wyniki dla pliku testowego ftv55.txt (populacja: 250, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.9375	3113	93.59452736	8.02238806
1.875	2118	31.71641791	
2.8125	1937	20.460199	
3.75	1917	19.21641791	
4.6875	1909	18.71890547	
5.625	1905	18.47014925	
6.5625	1900	18.15920398	
7.5	1900	18.15920398	
8.4375	1900	18.15920398	
9.375	1899	18.09701493	
10.3125	1899	18.09701493	
11.25	1899	18.09701493	
12.1875	1899	18.09701493	
13.125	1899	18.09701493	
14.0625	1899	18.09701493	
15	1899	18.09701493	



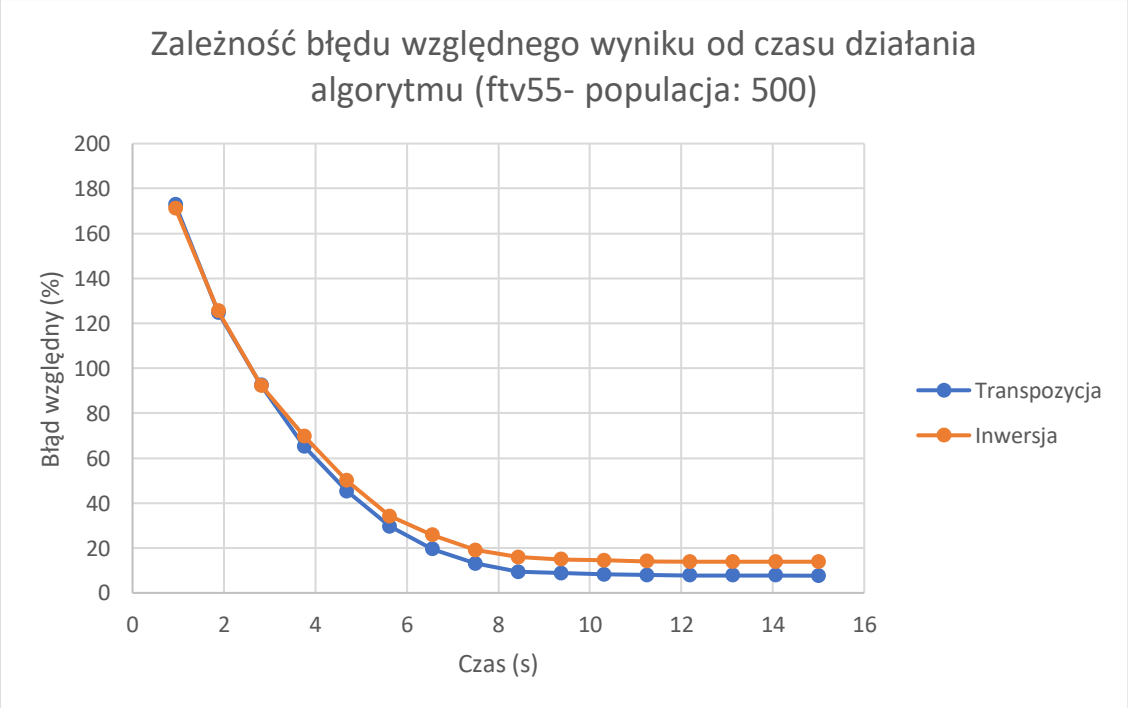
Rys. 5 Wykres dla pliku testowego ftv55.txt (populacja: 250)

Tabela 11. Wyniki dla pliku testowego ftv55.txt (populacja: 500, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.9375	4388	172.8855721	2.549751244
1.875	3614	124.7512438	
2.8125	3095	92.47512438	
3.75	2657	65.23631841	
4.6875	2335	45.21144279	
5.625	2084	29.60199005	
6.5625	1920	19.40298507	
7.5	1818	13.05970149	
8.4375	1758	9.328358209	
9.375	1748	8.706467662	
10.3125	1740	8.208955224	
11.25	1735	7.89800995	
12.1875	1733	7.773631841	
13.125	1732	7.711442786	
14.0625	1732	7.711442786	
15	1731	7.649253731	

Tabela 12. Wyniki dla pliku testowego ftv55.txt (populacja: 500, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
0.9375	4361	171.2064677	9.577114428
1.875	3625	125.4353234	
2.8125	3091	92.22636816	
3.75	2730	69.7761194	
4.6875	2411	49.93781095	
5.625	2157	34.14179104	
6.5625	2022	25.74626866	
7.5	1915	19.0920398	
8.4375	1863	15.85820896	
9.375	1847	14.86318408	
10.3125	1841	14.49004975	
11.25	1834	14.05472637	
12.1875	1831	13.8681592	
13.125	1831	13.8681592	
14.0625	1831	13.8681592	
15	1831	13.8681592	



Rys. 6 Wykres dla pliku testowego ftv55.txt (populacja: 500)

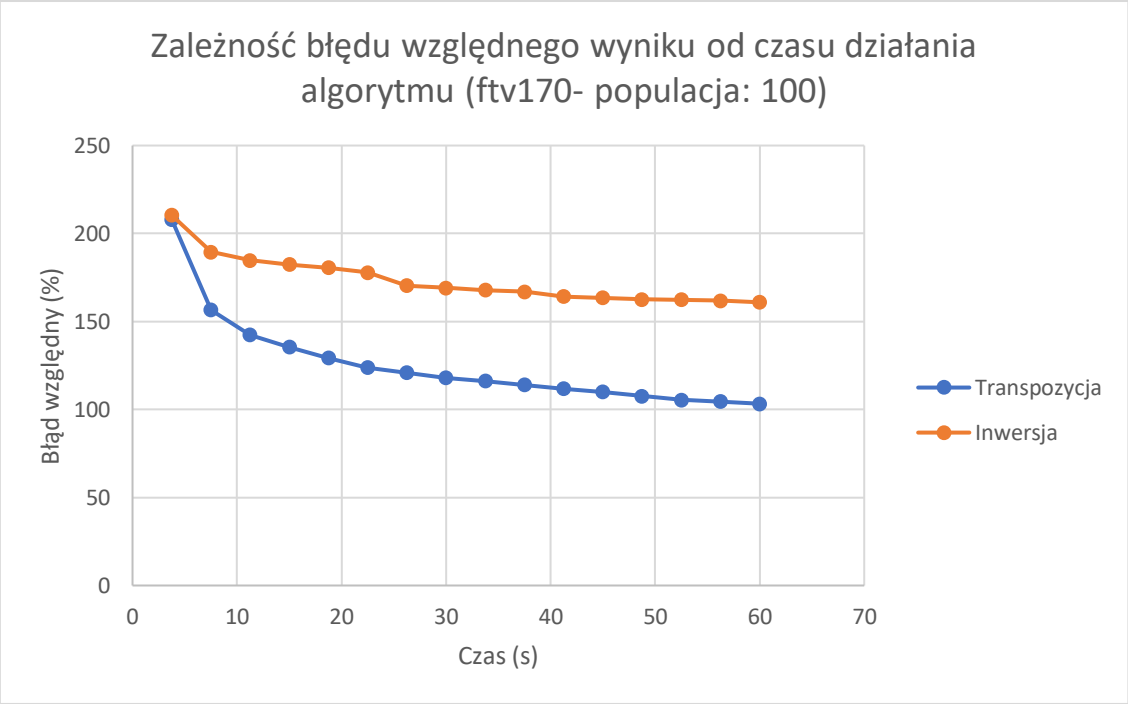
a) ftv170.txt

Tabela 13. Wyniki dla pliku testowego ftv170.txt (populacja: 100, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
3.75	8485	207.9854809	73.97459165
7.5	7070	156.6243194	
11.25	6676	142.323049	
15	6488	135.4990926	
18.75	6317	129.292196	
22.5	6166	123.8112523	
26.25	6086	120.907441	
30	6006	118.0036298	
33.75	5958	116.261343	
37.5	5896	114.0108893	
41.25	5837	111.8693285	
45	5786	110.0181488	
48.75	5720	107.6225045	
52.5	5662	105.5172414	
56.25	5636	104.5735027	
60	5601	103.3030853	

Tabela 14. Wyniki dla pliku testowego ftv170.txt (populacja: 100, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
3.75	8551	210.3811252	116.1887477
7.5	7978	189.5825771	
11.25	7847	184.8275862	
15	7778	182.323049	
18.75	7728	180.508167	
22.5	7656	177.8947368	
26.25	7452	170.4900181	
30	7413	169.0744102	
33.75	7378	167.8039927	
37.5	7353	166.8965517	
41.25	7279	164.2105263	
45	7258	163.4482759	
48.75	7234	162.5771325	
52.5	7228	162.3593466	
56.25	7216	161.923775	
60	7188	160.907441	



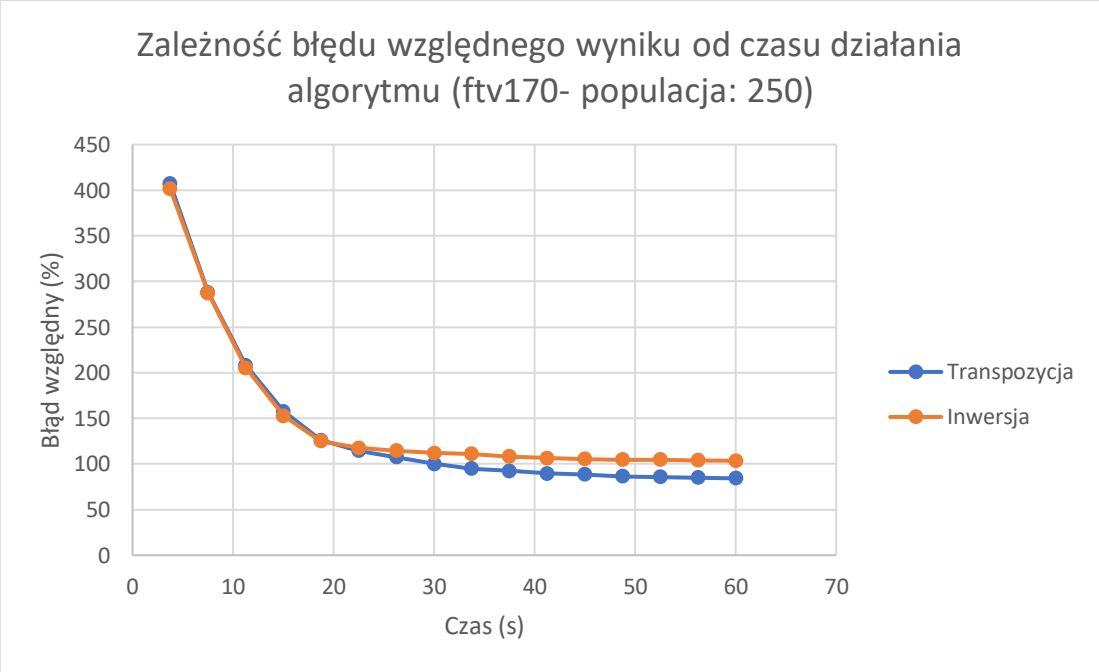
Rys. 7 Wykres dla pliku testowego ftv170.txt (populacja: 100)

Tabela 15. Wyniki dla pliku testowego ftv170.txt (populacja: 250, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
3.75	13976	407.2958258	68.31215971
7.5	10691	288.0580762	
11.25	8489	208.1306715	
15	7098	157.6406534	
18.75	6231	126.1705989	
22.5	5907	114.4101633	
26.25	5714	107.4047187	
30	5516	100.2177858	
33.75	5368	94.84573503	
37.5	5301	92.4137931	
41.25	5221	89.50998185	
45	5189	88.34845735	
48.75	5134	86.35208711	
52.5	5117	85.73502722	
56.25	5093	84.86388385	
60	5075	84.21052632	

Tabela 16. Wyniki dla pliku testowego ftv170.txt (populacja: 250, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
3.75	13813	401.3793103	86.67876588
7.5	10674	287.4410163	
11.25	8414	205.4083485	
15	6961	152.6678766	
18.75	6198	124.9727768	
22.5	5993	117.5317604	
26.25	5912	114.5916515	
30	5842	112.0508167	
33.75	5813	110.9981851	
37.5	5731	108.0217786	
41.25	5689	106.4972777	
45	5661	105.4809437	
48.75	5643	104.8275862	
52.5	5636	104.5735027	
56.25	5618	103.9201452	
60	5608	103.5571688	



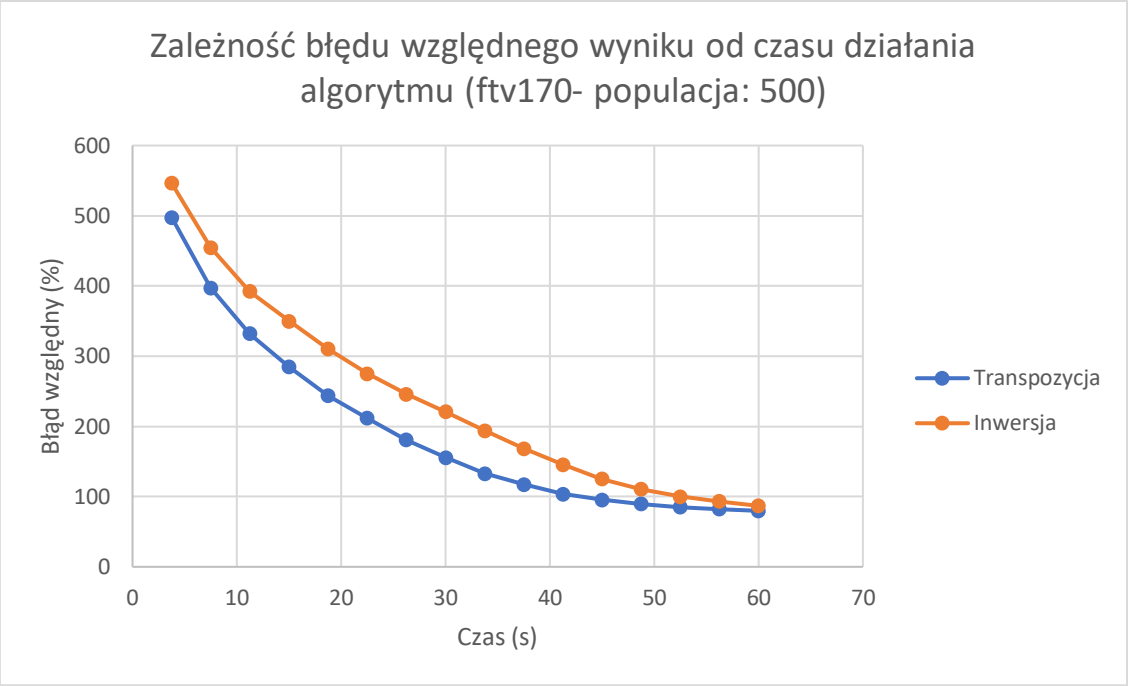
Rys. 8 Wykres dla pliku testowego ftv170.txt (populacja: 250)

Tabela 17. Wyniki dla pliku testowego ftv170.txt (populacja: 500, mutacja: transpozycja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
3.75	16468	497.7495463	65.98911071
7.5	13703	397.3865699	
11.25	11908	332.2323049	
15	10602	284.8275862	
18.75	9465	243.5571688	
22.5	8591	211.8330309	
26.25	7734	180.7259528	
30	7047	155.7894737	
33.75	6413	132.7767695	
37.5	5987	117.3139746	
41.25	5611	103.6660617	
45	5384	95.42649728	
48.75	5218	89.40108893	
52.5	5102	85.19056261	
56.25	5018	82.1415608	
60	4952	79.74591652	

Tabela 18. Wyniki dla pliku testowego ftv170.txt (populacja: 500, mutacja: inwersja)

Czas [s]	Znaleziona droga	Błąd względny (%)	Najlepszy wynik (%)
3.75	17807	546.3520871	72.95825771
7.5	15281	454.6642468	
11.25	13562	392.2686025	
15	12394	349.8729583	
18.75	11301	310.199637	
22.5	10331	274.9909256	
26.25	9534	246.061706	
30	8837	220.7622505	
33.75	8104	194.1560799	
37.5	7391	168.2758621	
41.25	6769	145.6987296	
45	6205	125.2268603	
48.75	5801	110.5626134	
52.5	5515	100.1814882	
56.25	5326	93.32123412	
60	5148	86.86025408	



Rys. 9 Wykres dla pliku testowego ftv170.txt (populacja: 500)

5. Wnioski

Na podstawie wykresów i tabel można dojść do wniosku, że zbyt mała wielkość populacji może prowadzić do uzyskania gorszych rozwiązań. Trzeba mieć jednak na uwadze, że o ile większa populacja rzeczywiście może poprawić jakość rozwiązań, tak też może je pogorszyć. Niestety tym większą populację wybierzemy, tym dłużej algorytm będzie ją przetwarzał. W istocie oznacza to, że dla większej wielkości populacji, liczba pokoleń w danym czasie będzie mniejsza. Jak wiemy, zbyt mała ilość wygenerowanych pokoleń może doprowadzić do pogorszenia jakości rozwiązań (zwłaszcza dla większych problemów). To jaką populację wybierzemy zależy od naszego celu. Jeśli interesuje nas dokładniejsze rozwiązanie- to wielkość populacji zwiększymy. Jeśli jednak zależy nam na czasie a niekoniecznie jakości rozwiązań, to liczbę osobników w populacji zredukujemy. Na wykresach można dostrzec, że przy większej populacji algorytmowi zdecydowanie trudniej jest wpaść w lokalne optimum (wykres tak szybko się nie wypłaszcza).

Poza przedstawionymi wykresami został jeszcze przeprowadzony mały test mający na celu zobaczyć czy wydłużenie czasu trwania algorytmu pomoże w uzyskaniu w jeszcze lepszych wyników. Użyto pliku ftv170.txt i sprawdzono populacje 250 i 500. Dla populacji 250 po dodatkowych 30s działania (łącznie 90s) znaleziono przykładowe rozwiązanie 5015 (błąd 82%), natomiast dla populacji 500 było to już 4297 (błąd 55,9%).

Zastosowanie mutacji okazało się kluczowe dla odpowiedniego działania algorytmu. Dzięki nim zmniejszyliśmy szansę na zbyt szybkie wpadnięcie w lokalne optimum. Można dojść do wniosku, że lepsze wyniki działania algorytmu wychodzą dla mutacji typu „transpozycja". Najprawdopodobniej może wynikać to z tego iż ma ona nieco lepszą złożoność obliczeniową.

Możemy także porównać algorytm genetyczny z wcześniej zaimplementowanymi tabu search oraz simulated annealing. Jakość rozwiązań dla pliku b17.txt nie różni się znacząco. Wszystkie algorytmy potrafią w rozsądnym czasie znaleźć optimum globalne wynoszące 39. Algorytm genetyczny wyznaczył zdecydowanie lepsze wartości dla pliku ftv55.txt. Średni wynik dla SA

wynosił 23.13%, dla tabu search z dywersyfikacją- 46.45% natomiast dla algorytmu genetycznego- 7.65%. Dla pliku ftv170.txt średni wynik dla SA wynosił 63.42%, dla TS- 161.07%, natomiast dla algorytmu genetycznego- 79.75%. Trzeba mieć jednak na uwadze, że średni wynik dla algorytmu genetycznego po 60s nie był najlepszy i można było uzyskać jeszcze lepszy rezultat. Jak to już zostało wcześniej wspomniane, przykładowy wynik dla algorytmu genetycznego po 90s wynosił 55,9%, najlepszy znaleziony wynik dla SA- 53,1%, zatem wyniki potrafią wyjść podobne. Wyniki dla TS nie są najlepsze, jednak implementacja tego algorytmu nie została najlepiej wykonana w drugim zadaniu projektowym. Co prawda SA jest szybszy w działaniu niż zastosowany algorytm genetyczny, wszystko jednak może zależeć tutaj od implementacji. Zasadniczo SA jest prostszy w implementacji niż algorytm genetyczny, dlatego też łatwiej zwiększyć szybkość jego działania. Zauważono jednak, że algorytm SA ma większe tendencje do wpadania w lokalne minimum. Najprawdopodobniej podobna sytuacja jest w przypadku TS. Dlatego można dojść do wniosku, że lepszą jakość rozwiązań gwarantuje raczej algorytm genetyczny. Trzeba także pamiętać o tym, że w algorytmie genetycznym zdecydowanie trudniej jest ustawić poszczególne parametry. W tym zadaniu projektowym zostały pokazane testy tylko dla jednych wybranych prawdopodobieństw mutacji. Nie ma zatem pewności, że algorytm ten nie może działać lepiej.

Podsumowując wyniki testów można uznać za pozytywne. Algorytm genetyczny pozwolił na poznanie zupełnie nowego, nieznanego wcześniej i ciekawego podejścia ewolucyjnego.