Projektowanie Efektywnych Algorytmów

Zadanie projektowe nr 3

Implementacja i analiza efektywności algorytmu genetycznego dla asymetrycznego problemu komiwojażera (ATSP).

Wtorek, 15:15 TNP

Autor:

Michał Lewandowski #264458

1. Wstęp teoretyczny

1.1. Podstawowe założenia

Podczas realizacji zadania należy przyjąć następujące założenia:

- używane struktury danych powinny być alokowane dynamicznie
- program powinien umożliwić wczytanie danych testowych z plików: ftv47.atsp, ftv170.atsp, rgb403.atsp
- program musi umożliwiać wprowadzenia kryterium stopu jako czasu wykonania podawanego w sekundach,
- implementacje algorytmów należy dokonać zgodnie z obiektowym paradygmatem programowania,
- kod źródłowy powinien być komentowany

1.2. Opis algorytmu genetycznego

Algorytm Genetyczny, będący rodzajem heurystyki, wchodzi w skład rodziny algorytmów ewolucyjnych. Jego metodologia opiera się na analizie alternatywnych rozwiązań, w porównaniu do aktualnie posiadanych, celem identyfikacji optymalnych opcji. Pomysłodawca tej metody, John Henry Holland, zaczerpnął inspirację z biologii, co znajduje odzwierciedlenie w nazwie i strukturze algorytmu, odnoszącej się do biologicznej ewolucji.

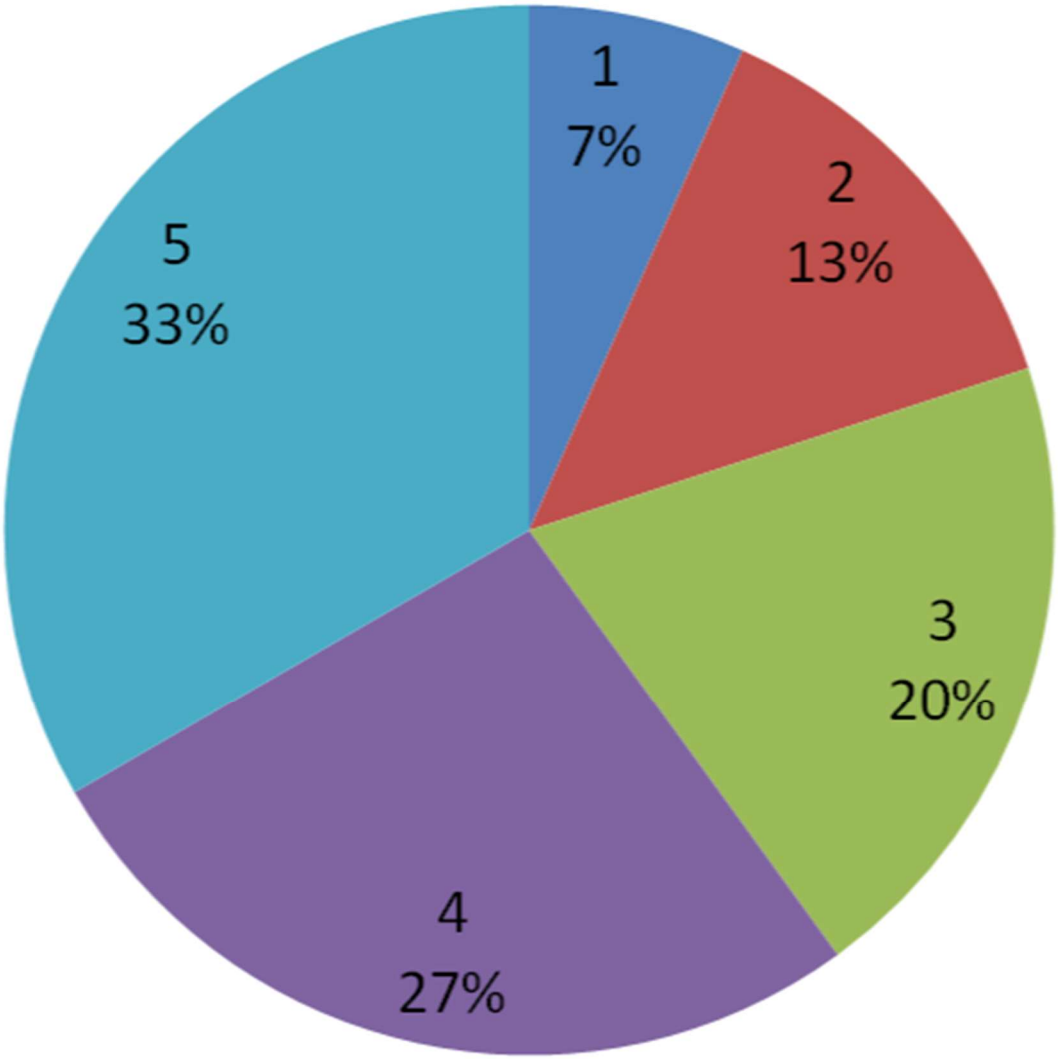
Terminologia związana z Algorytmem Genetycznym czerpie z biologii. Przykładowo, zbiór rozwiązań w danej iteracji określany jest jako populacja, a każdy element tej populacji posiada genotyp składający się z chromosomów. To właśnie chromosomy, będące kluczowymi komponentami osobnika, przechowują informacje o rozwiązaniu. Z kolei chromosomy są zbudowane z pojedynczych genów.

Ustalając taką nomenklaturę, możemy przejść do opisu operacji wykonywanych na populacji w celu uzyskania optymalnego rozwiązania. Każda iteracja algorytmu ma za zadanie wytworzyć nową populację, która teoretycznie powinna oferować lepsze rozwiązania problemów. W tym celu przeprowadzana jest selekcja osobników, z wykorzystaniem różnych metod, w tym metody koła ruletki. W tej metodzie każdy

osobnik jest oceniany w kontekście przechowywanego przez niego rozwiązania, a jego szanse na selekcję są proporcjonalne do bliskości tego rozwiązania do ideału. Na podstawie tych procentowych szans wybierane są osobniki, które utworzą nowe pokolenie o rozmiarze równym poprzedniemu. Po wyselekcjonowaniu osobników, następuje etap główny, obejmujący operatory krzyżowania i mutacji. Krzyżowanie, które zachodzi jako pierwsze, polega na połączeniu par osobników. Decyzja o krzyżowaniu zależy od zdefiniowanego prawdopodobieństwa; jeśli krzyżowanie ma miejsce, wykorzystuje się do tego określoną metodę, na przykład Partially-mapped Crossover (PMX). Wybierany jest punkt krzyżowania, po czym dochodzi do wymiany genów między osobnikami do tego punktu, a następnie uzupełnienia brakujących genów po punkcie krzyżowania, zgodnie z kolejnością występowania w osobniku potomnym. Kolejnym etapem jest mutacja, której prawdopodobieństwo również jest z góry określone. Operator mutacji, mający na celu dywersyfikację rozwiązań, polega na zamianie miejscami dwóch losowo wybranych genów w obrębie danego osobnika - metoda (swap) lub polega na usunięciu elementu z pierwszego podanego indeksu i wpisaniu go na drugi podany Indeks - metoda (insert).

1.2.1 Metoda selekcji

Zastosowano prosty wariant metody ruletkowej. W myśl strategii, że lepsze rozwiązanie posiada większą miarę dopasowania, następuje losowanie rozwiązań reprezentowanych przez wspomniane miary. W zaimplementowanym algorytmie wartości miary dopasowania zostają znormalizowane, w celu losowania wartości z zakresu od 0 do 1.



Rysunek 1: Metoda ruletki. 1-5 to miary dopasowania elementu populacji

1.2.2 Metoda krzyżowania

Partially Mapped Crossover (PMX) - podstawą tego krzyżowania jest dziedziczenie przez potomków odpowiadających sobie fragmentów rodziców, następnie przebiega proces uzupełniania ich o brakujące elementy

Przykład:

Po określeniu losowej sekcji dopasowania, dla każdego z potomków tworzona jest tablica odwzorowań elementów

R1=(123456789)

R2=(536781294)

Tablica odwzorowań dla pierwszego potomka: [3 => 6][4 => 7][5 => 8][6 => 1]
Następnie dodawane zostają brakujące elementy z drugiego rodzica które nie powodują konfliktów

P1=(_3456 29_)

Dodawanie elementów powodujących konflikty odbywa się na podstawie tablicy odwzorowań. Elementami powodującymi konflikt jest 4, 5 oraz 3.

- 4 przechodzi na 7

P1=(_3456297)

- 5 przechodzi na 8

P1=(8_3456297)

- 3 przechodzi na 6 i 6 przechodzi na 1

P1=(813456297)

Drugi potomek jest generowany w analogiczny sposób.

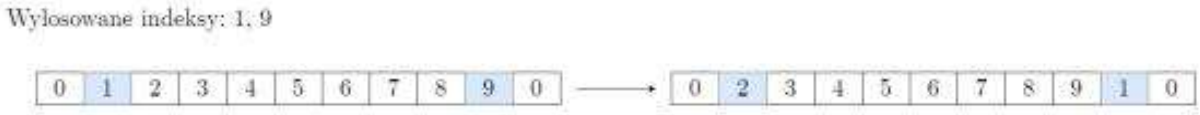
1.2.3 Metody mutacji

Zastosowano metody swap oraz insert. Mutacja elementu populacji jest wykonywana dla każdego elementu na losowych, różnych indeksach, z pewnym prawdopodobieństwem, za po-mocą wybranej metody.

Metoda swap zamienia miejscami miasta o podanych różnych indeksach w podanej ścieżce.

Rysunek 2: Przykładowy przebieg metody swap

Metoda insert usuwa element z pierwszego podanego indeksu i wpisuje go na drugi podany indeks w podanej ścieżce.



Rysunek 3: Przykładowy przebieg metody insert

2. Opis implementacji algorytmu

Klasa GeneticAlgorithm składa się z 8 zmienncyh:

std::vector<std::vector<int>> graph: macierz dynamiczna przechowująca połączenia między miastami w formie grafu.

std::vector<std::vector<int>>::size_type graph_size: tablica dynamiczna przechowująca wszystkie wierzchołki grafu.

int* localShortestRoute: wskaźnik na zmienną typu int służący do przechowywania obecnie najlepszej znalezionej trasy przez algorytm.

int localShortestRouteValue: zmienna typu int przechowująca długość (koszt) znalezionej trasy przez **localShortestRoute**.

int startingNode: wierzchołek początkowy dla problemu TSP.

std::mt19937 rng: instancja generatora liczb losowych.

int theShortestRouteValue: zmienna typu int przechowująca globalnie najlepszy znaleziony koszt trasy przez algorytm.

std::vector<int> theShortestRoute: dynamicznie alokowana tablica do przechowywania globalnie najlepszej znalezionej trasy przez algorytm.

Klasa składa się z przeciążonego konstruktora oraz osiemnastu metod

- GeneticAlgorithm::GeneticAlgorithm: Konstruktor, który inicjuje algorytm genetyczny przy użyciu określonego grafu (macierzy odległości) oraz generatora liczb losowych.
- GeneticAlgorithm::~GeneticAlgorithm: Destruktor, który usuwa dynamicznie alokowaną pamięć, gdy obiekt jest niszczony.
- GeneticAlgorithm::initRoute: Inicjuje początkową trasę przy użyciu zachłannego podejścia.
- GeneticAlgorithm::calculateRouteLength: Oblicza długość (koszt) danej trasy.
- GeneticAlgorithm::setRandomRoute: Generuje losową trasę.
- GeneticAlgorithm::gradeRoute: Oblicza ocenę (zdolność) danej trasy.
- GeneticAlgorithm::normalizeGrades: Normalizuje oceny tras, aby upewnić się, że są większe niż zero.
- GeneticAlgorithm::copyPopulation i GeneticAlgorithm::copyPopulationIntact: Funkcje do kopiowania jednej populacji do drugiej.
- GeneticAlgorithm::selectByRoulette: Wybiera rodziców na podstawie metody koła ruletki, używając ocen.
- GeneticAlgorithm::mutate: Modyfikuje trasę poprzez zamianę dwóch różnych punktów.
- GeneticAlgorithm::swapElements: Zamienia ze sobą dwa elementy w tablicy.
- GeneticAlgorithm::insertElements: Usuwa element z pierwszego podanego indeksu z podanej tablicy i wpisuje go na drugi podany indeks w tablicy
- GeneticAlgorithm::crossOnePoint: Wykonuje krzyżowanie jednopunktowe (PMX) między dwiema trasami.
- GeneticAlgorithm::findInArray: Znajduje podaną liczbę w tablicy i zwraca jej pozycję.
- GeneticAlgorithm::copyRoute: Kopiuje jedną trasę do drugiej.
- GeneticAlgorithm::getShortestRoute: Zwraca najkrótszą trasę jako ciąg znaków.
- GeneticAlgorithm::getShortestRouteValue: Zwraca długość najkrótszej trasy.
- GeneticAlgorithm::solve: Główna funkcja, która stosuje algorytm genetyczny w celu znalezienia najkrótszej trasy. Inicjuje populację, wybiera rodziców, wykonuje krzyżowanie i mutację, aktualizuje najlepsze znalezione rozwiązanie i powtarza ten proces przez określoną liczbę iteracji lub sekund.

3. Eksperymenty

3.1 Plan eksperymentu

Pomiary wykonywane były dla problemów o rozmiarze 47 i 403 miast dla dwóch metod mutacji (swap i insert) oraz operatorze krzyżowania PMX. Pomiary zostały wykonane dla różnych rozmiarów populacji i ich rezultaty porównane na wykresach. Dla pliku ftv170.atsp moja implementacja z niewiadomych dla mnie przyczyn nie działała poprawnie w wyniku czego zdecydowałem się na nie dodawanie do sprawozdania rezultatów pomiarów działania algorytmu na tym pliku ponieważ zawierały błąd gruby który wpłynąłby znacząco na wnioski.

Dla każdego czasu dla danej populacji metody mutacji i pliku były wykonywane 10 pomiarów a z ich rezultatów została liczona średnia która umieszczona jest w tabeli rozwiązanie.

Pomiary czasu zostały wykonane przy użyciu biblioteki std::chrono

Podczas testów przyjęto parametry:

Rozmiar populacji = 10/100/1000,

Liczba iteracji = -1

Współczynnik mutacji = 0.01,

Współczynnik krzyżowania = 0.8,

Kryterium stopu = 30 sekund,

Limit czasowy = 1/2/3/5/6/7/8/9/10

Metoda mutacji = 0/1.

3.2 Analiza danych pomiarowych

Tabela 1. Przedstawiająca wyniki pomiarów działania algorytmu o populacji = 10 genetycznego dla pliku ftv47.atsp

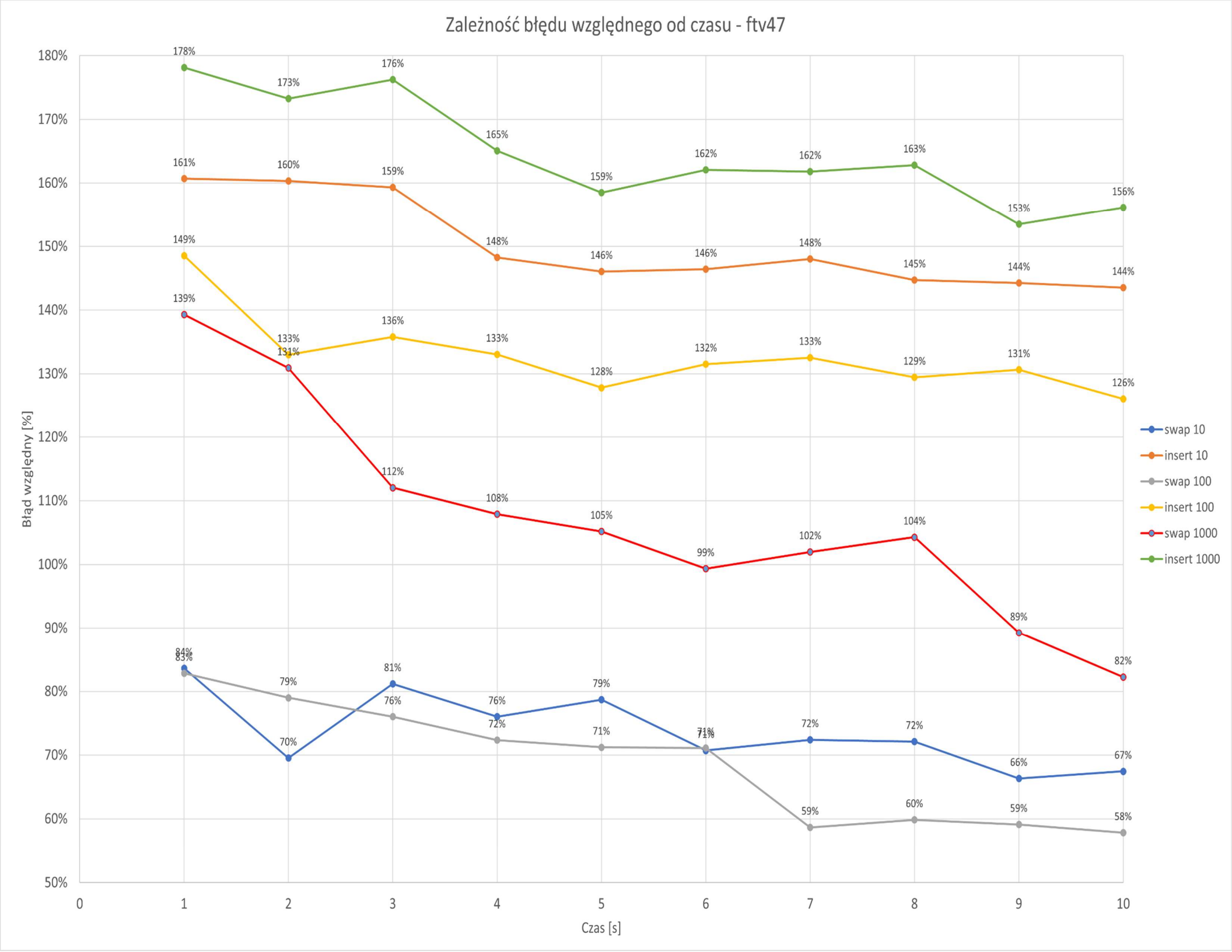
POPULATION = 10				
ftv47				
czas [s]	swap		insert	
	rozwiązanie	błąd względny	rozwiązanie	błąd względny
1	3261	84%	4630	161%
2	3011	70%	4624	160%
3	3218	81%	4606	159%
4	3126	76%	4409	148%
5	3174	79%	4370	146%
6	3032	71%	4376	146%
7	3062	72%	4405	148%
8	3057	72%	4346	145%
9	2954	66%	4338	144%
10	2974	67%	4325	144%

Tabela 2. Przedstawiająca wyniki pomiarów działania algorytmu genetycznego o populacji = 100 dla pliku ftv47.atsp

POPULATION = 100				
ftv47				
czas [s]	swap		insert	
	rozwiązanie	błąd względny	rozwiązanie	błąd względny
1	3247	83%	4414	149%
2	3179	79%	4138	133%
3	3126	76%	4188	136%
4	3061	72%	4139	133%
5	3041	71%	4046	128%
6	3039	71%	4112	132%
7	2818	59%	4130	133%
8	2839	60%	4075	129%
9	2826	59%	4096	131%
10	2803	58%	4015	126%

Tabela 3. Przedstawiająca wyniki pomiarów działania algorytmu genetycznego o populacji = 1000 dla pliku ftv47.atsp

POPULATION = 1000				
ftv47				
czas [s]	swap		insert	
	rozwiązanie	błąd względny	rozwiązanie	błąd względny
1	4250	139%	4940	178%
2	4101	131%	4853	173%
3	3766	112%	4906	176%
4	3692	108%	4708	165%
5	3644	105%	4591	159%
6	3540	99%	4655	162%
7	3587	102%	4650	162%
8	3628	104%	4668	163%
9	3361	89%	4502	153%
10	3237	82%	4549	156%


Zależność błędu względnego od czasu - ftv47
Błąd względny [%]
Czas [s]
swap 10
insert 10
swap 100
insert 100
swap 1000
insert 1000
180%
170%
160%
150%
140%
130%
120%
110%
100%
90%
80%
70%
60%
50%
0
1
2
3
4
5
6
7
8
9
10
178%
173%
176%
165%
159%
162%
162%
163%
153%
156%
161%
160%
159%
148%
146%
146%
148%
145%
144%
144%
149%
133%
131%
136%
133%
128%
132%
133%
129%
131%
126%
139%
112%
108%
105%
99%
102%
104%
89%
82%
84%
83%
79%
76%
81%
76%
72%
79%
71%
71%
70%
72%
72%
66%
67%
59%
60%
59%
58%

Na podstawie wykresu 1 można zaobserwować, że wraz ze wzrostem liczebności populacji błąd względny ma tendencję do zmniejszania się. Widać to wyraźnie po spadku błędu względnego w każdej wybranej populacji (10, 100, 1000) istnieje również ogólna tendencja do poprawy jakości rozwiązań w czasie. Metoda mutacji swap dale lepsze wyniki dla każdego rozmiaru populacji.

Tabela 4. Przedstawiająca wyniki pomiarów działania algorytmu o populacji = 10 genetycznego dla pliku rbg403.atsp

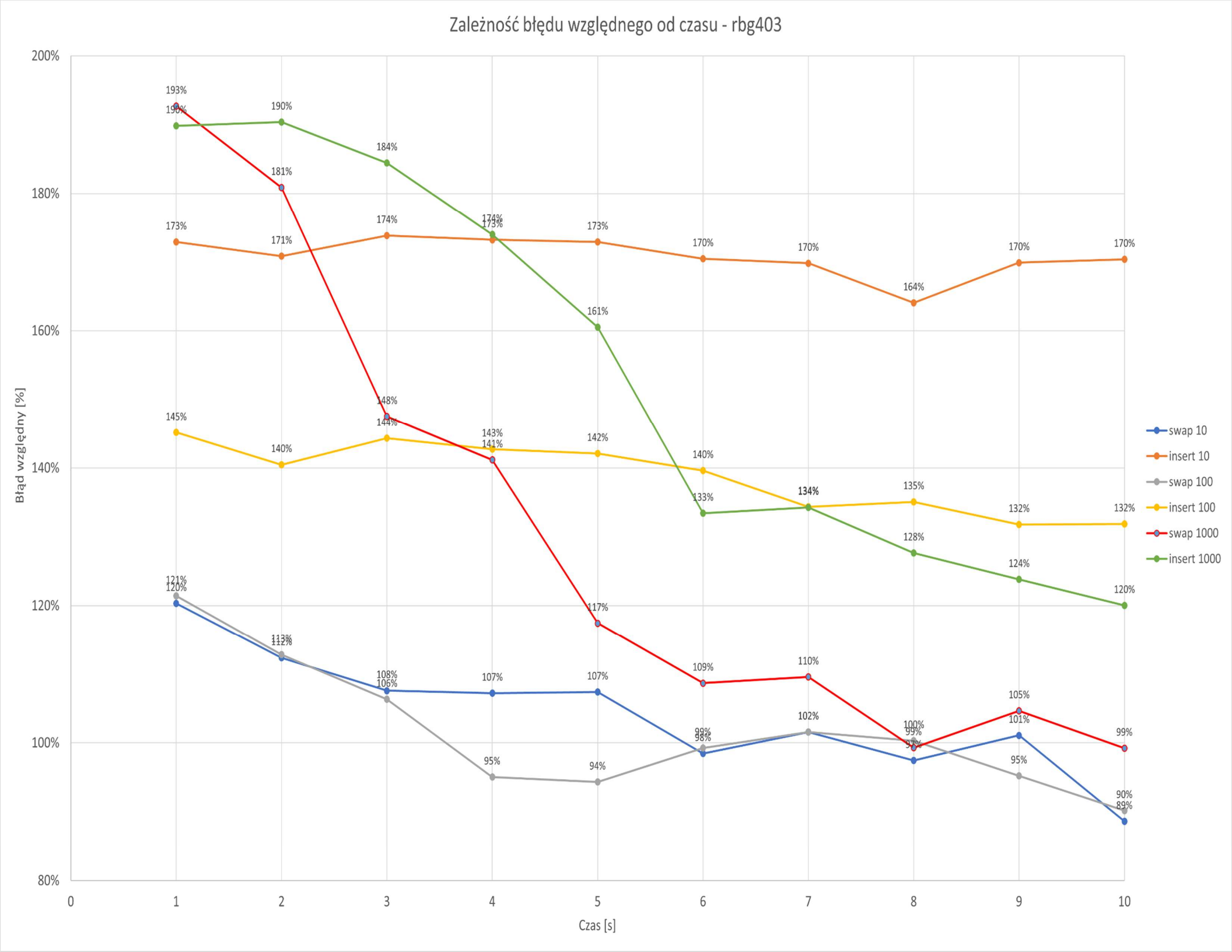
POPULATION = 10				
rbg403				
czas [s]	swap		insert	
	rozwiązanie	błąd względny	rozwiązanie	błąd względny
1	5431	120%	6727	173%
2	5235	112%	6676	171%
3	5117	108%	6750	174%
4	5108	107%	6735	173%
5	5112	107%	6727	173%
6	4892	98%	6667	170%
7	4969	102%	6651	170%
8	4867	97%	6509	164%
9	4957	101%	6653	170%
10	4649	89%	6665	170%

Tabela 5. Przedstawiająca wyniki pomiarów działania algorytmu o populacji = 100 genetycznego dla pliku rbg403.atsp

POPULATION = 100				
rbg403				
czas [s]	swap		insert	
	rozwiązanie	błąd względny	rozwiązanie	błąd względny
1	5458	121%	6044	145%
2	5246	113%	5928	140%
3	5086	106%	6023	144%
4	4808	95%	5984	143%
5	4790	94%	5968	142%
6	4912	99%	5907	140%
7	4969	102%	5777	134%
8	4938	100%	5795	135%
9	4812	95%	5714	132%
10	4688	90%	5716	132%

Tabela 6. Przedstawiająca wyniki pomiarów działania algorytmu o populacji = 1000 genetycznego dla pliku rbg403.atsp

POPULATION = 1000				
rbg403				
czas [s]	swap		insert	
	rozwiązanie	błąd względny	rozwiązanie	błąd względny
1	7215	193%	7145	190%
2	6924	181%	7158	190%
3	6102	148%	7012	184%
4	5945	141%	6754	174%
5	5360	117%	6422	161%
6	5144	109%	5754	133%
7	5166	110%	5775	134%
8	4913	99%	5612	128%
9	5045	105%	5517	124%
10	4911	99%	5424	120%


Zależność błędu względnego od czasu - rbg403
Błąd względny [%]
Czas [s]
200%
180%
160%
140%
120%
100%
80%
0
1
2
3
4
5
6
7
8
9
10
swap 10
insert 10
swap 100
insert 100
swap 1000
insert 1000

Na podstawie wykresu 2 można również zaobserwować, że wraz z wzrostem populacji zmniejsza się błąd względny jednakże większe populację potrzebują więcej czasu na znajdywanie ścieżek bliższych optymalnej, również dla tego pliku metoda mutacji swap przewyższa w znajdowaniu mniej kosztownych ścieżek nad metodą insert. Tak jak na wykresie 1 wyraźnie widać tendencje do znajdywania lepszych ścieżek dla każdej metody i populacji wraz z upływem dłuższej ilości czasu.

Tabela 7. Przedstawiająca najlepsze wyniki uzyskane dla symulowanego wyżarzania

Symulowane wyżarzanie				
Rozmiar	Optymalne	Znalezione	Czas [s]	Błąd względny
47	1776	2178	7,31	22%
170	2755	3617	14.75	31%
403	2465	2792	116,72	13%

Tabela 8. Przedstawiająca najlepsze wyniki uzyskane dla algorytmu genetycznego

Algorytm genetyczny				
Rozmiar	Optymalne	Znalezione	Czas [s]	Błąd względny
47	1776	2803	10	58%
170	2755	X	X	X
403	2465	4648	10	89%

4. Wnioski

Wyniki przedstawione na obu wykresach wskazują, że większa liczba osobników w populacji sprzyja efektywniejszej eksploracji przestrzeni rozwiązań, co skutkuje osiągnięciem lepszych wyników

Metoda mutacji typu 'swap' systematycznie wykazuje wyższą efektywność niż metoda mutacji typu 'insert' w różnorodnych rozmiarach populacji i wariantach rozpatrywanych problemów. To podkreśla, że w przypadku rozważanych instancji problemu ATSP, mutacja 'swap' jest bardziej skuteczna.

Zauważono również, że jakość rozwiązania generalnie rośnie wraz z upływem czasu, jednak tempo tej poprawy maleje w miarę dalszego działania algorytmu. Wskazuje to na obniżający się stopień przyrostu jakości rozwiązania wraz z czasem, przy czym największe ulepszenia są obserwowane w początkowych iteracjach. Jest to przesłanka do optymalizacji kryteriów zakończenia procesu obliczeniowego, co pozwoliłoby na zrównoważenie nakładu obliczeniowego i jakości uzyskiwanych rozwiązań.

Wysokie początkowe błędy procentowe, zwłaszcza w przypadku instancji rbg403, sugerują, że inicjalna populacja generowana przez algorytm zachłanny może nie być optymalnym wyborem. Jest to prawdopodobnie jedną z przyczyn, dla której w przypadku instancji ftv170.atsp, koszty uzyskanych ścieżek były na tyle wysokie, że zdecydowałem się nie uwzględniać ich w sprawozdaniu.

Porównanie algorytmu najlepszych wyników algorytmu genetycznego z wynikami symulowanego wyżarzania które są bliższe optymalnych sugeruję, że jego implementacja jest lepiej dostrojona do problemu komiwojażera.

5. Spis źródeł

1. Wykłady oraz prezentacje dr inż. Tomasza Kapłona z kursu Projektowania Efektywnych Algorytmów
2. Algorytmy genetyczne i ich zastosowanie - Goldberg David E.
3. Algorytmy genetyczne+struktury danych=programy ewolucyjne - Zbigniew Michalewicz