Sprawozdanie

Projektowanie Efektywnych Algorytmów P

Etap 3

Opracował: Patryk Uzarowski

Informatyka Techniczna, Wydział Informatyki i Telekomunikacji

Termin oddania projektu: 19 styczeń 2022

1. Wstęp

Celem trzeciego zadania projektowego było opracowanie Algorytmu Genetycznego rozwiązującego przybliżenie problem Komiwojażera (Travelling Salesman Problem). Problem ten opiera się na znalezieniu najbardziej optymalnej drogi z i do wierzchołka początkowego poprzez wszystkie pozostałe wierzchołki. Problem polega więc na odnalezieniu cyklu Hamiltona o najmniejszej sumie wag krawędzi.
Założenia dotyczące problemu Komiwojażera:

a) Asymetryczny – drogi docelowe oraz powrotne nie muszą być sobie równe ((i => j) != (j => i)).
b) Droga z miasta (i => i) może być reprezentowana jako 0 / 9999 / -1.
c) Wierzchołek 0 jest punktem startowym.

Założenia dotyczące struktury projektu:

a) Program napisany jest w podejściu Obiektowym wykorzystując język C++.
b) Program wyposażony jest w Meny Główne, w którym można dokonać wyboru algorytmu, wczytać macierz sąsiedztwa rozpatrywanego grafu lub wygenerować ją losowo. W pod menu algorytmów można również dokonać wyboru kryterium stopu (czasu maksymalnego egzekucji algorytmu) oraz wyboru parametrów rozruchu Algorytmu Genetycznego.
c) Algorytm egzekwowany jest przez klasy:
   GeneticAlgorithm – realizacja Algorytmu Genetycznego.
   Dodatkowo do najważniejszych klas w projekcie należy również klasa **Menu**, implementująca prosty interfejs użytkownika w konsoli, **Matrix** reprezentująca strukturę macierzy sąsiedztwa rozważanego grafu oraz **Result**, pomocnicza klasa przechowująca pary: <Koszt_najlepszego_rozwązania, ścieżka_najlepszego_rozwiązania>
d) W przypadku objaśnień teoretycznych algorytmu genetycznego argumentem wejściowym jest macierz sąsiedztwa reprezentująca rozpatrywany graf np. :

wierzchołek	0	1	2	3
0	INF	3	4	7
1	4	INF	6	9
2	2	5	INF	3
3	8	1	6	INF

2. Algorytm Genetyczny

Algorytm Genetyczny opiera swe działanie na generacji populacji elementów rozważanego problemu (W przypadku problemu Komiwojażera są to permutacje wierzchołków rozpatrywanego grafu) oraz odpowiedniej selekcji, modyfikacji oraz uzupełnieniu potomstwa populacji bazując na schemacie opisanym poniżej. Jakość algorytmu jest przede wszystkim określana poprzez założone implementacje wypisanych powyżej elementów składowych, w przypadku zbyt generalnej selekcji możemy nigdy nie znaleźć się w otoczeniu najkorzystniejszych rozwiązań, w

przypadku zbyt ostrej mutacji, rezultaty przeszukiwań mogą być przeniesione do populacji dzieci w formie skrajnie zaburzonej itp. Ogólny schemat algorytmu prezentuje się następująco:

2.1. Na początku algorytmu należy wygenerować populację początkową bazując na wielkości docelowej.
2.2. Kolejnym etapem jest ewaluacje kondycji każdego elementu początkowej populacji, kondycja elementu jest wyznaczana w zależności od problemu (W przypadku Problemu Komiwojażera jest to po prostu koszt danego rozwiązania)
2.3. Generowanie nowej populacji dopóki rozmiar populacji dzieci jest mniejszy od założonego. Na tym etapie kolejno występuję **selekcja elementów, krzyżowanie oraz mutacja** dobranych kandydatów, przeniesienie kandydatów do zbioru populacji dzieci.
2.4. Kolejnym etapem jest przypisanie populacji rodziców wartości populacji dzieci (Przy kolejnej iteracji algorytmu, rozważanym zbiorem staną się dzieci z poprzedniej iteracji)
2.5. W przypadku zaspokojenia kryterium stopu należy tutaj zakończyć algorytm, w przeciwnym wracamy do punktu 2.2.

Schemat selekcji elementów – może zostać zaimplementowany na wiele sposobów, deklaruje on metodę wybierania kandydatów z populacji rodzica do rozważań (modyfikacji) na dalszym etapie algorytmu. Przykładowym modelem selekcji może być dobór na podstawie kondycji elementów:

Z obecnie rozważanej populacji dobieramy dwa losowe elementy, porównujemy ich kondycje, a następnie do zbioru rozważań dodajemy korzystniejszy. (**Selekcja turniejowa, zaimplementowana w projekcie**)

Innym sposobem selekcji elementów może być chociażby selekcja bazująca na schemacie „Koła z ruletką”:

Z obecnie rozważanej populacji sumujemy kondycje wszystkich elementów otrzymując sumę – SF. Generujemy liczbę pseudolosową z przedziału <0, SF>. Iterujemy przez populację sumując kondycje elementów, jeżeli sumowana wartość wyższa od wygenerowanej liczby pseudolosowej, zatrzymujemy iterację i zwracamy kandydata, na których „zatrzymała się” iteracja.

Schemat Krzyżowania elementów – strategia polegająca na modyfikacji dwóch dobranych na podstawie selekcji kandydatów uwzględniająca pewne prawdopodobieństwo zajścia krzyżowania. Może zostać zaimplementowana na wiele sposobów. W przypadku zaimplementowanego rozwiązania jest to krzyżowanie OX ze stałym punktem (indeksem) przecięcia. Polega ono na przepisaniu elementów z pierwszego kandydata do określonego indeksu, reszta elementów zostaje wypełniona wierzchołkami z kandydata 2, które jeszcze nie znalazły się w tablicy wynikowej. Przykładowo dla kandydatów kandydat 1 - <0,1,2,3,4,5>, kandydat 2 - <4,0,5,3,1,2>,

dzieckiem skrzyżowanym o punkcie przecięcia [2] będzie element wynikowy - <0, 1, 2, 4, 5, 3>.

Schemat Mutacji elementów – strategia polegająca na modyfikacji pojedynczych elementów uwzględniająca pewne prawdopodobieństwo zajścia mutacji. W przypadku zaimplementowanego projektu zdecydowano się na mutację Swap oraz Insert:

Mutacja Swap:
Najprostszy typ mutacji generowany poprzez zamianę dwóch wierzchołków rozważanej ścieżki, przykładowo dla cyklu {1,3,5,7,9} => swap(1,7) modyfikuje element do postaci => {7,3,5,1,9}
Mutacja Insert
Modyfikacja elementu polega na wstawieniu wierzchołka i w miejsce wierzchołka j oraz adekwatnego przesunięcia reszty cyklu, przykładowo dla cyklu {1,3,5,7,9} => Insert(1,7) modyfikuje element do postaci {3,5,1,7,9}.

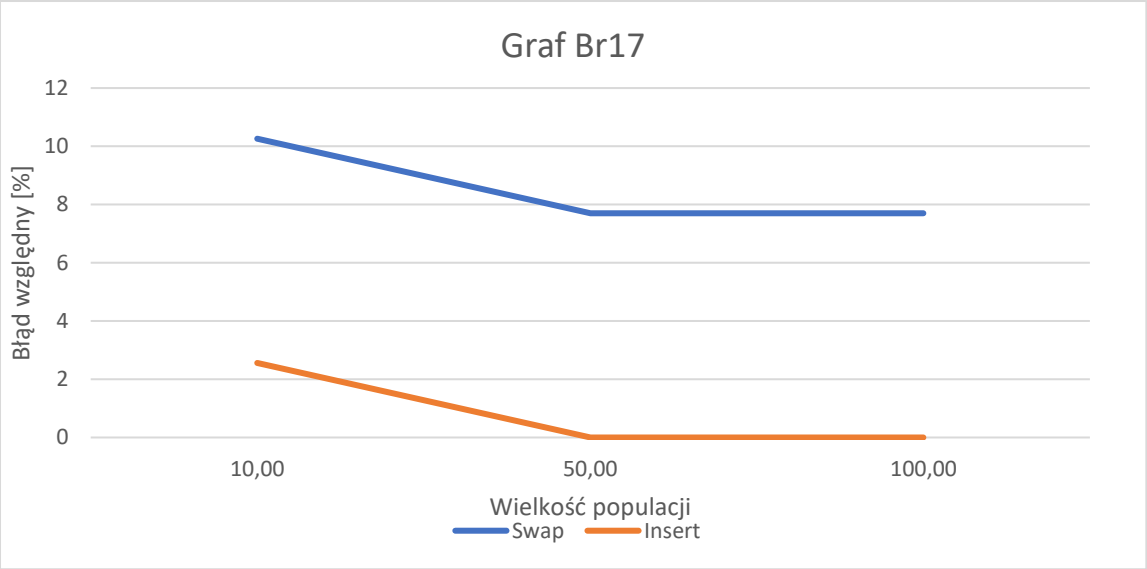
3. Plan eksperymentu

Zgodnie z założeniami projektowymi testy zostały przeprowadzone dla trzech różnych grafów: Br17, Ftv55, Ftv170. Dobrane wielkości populacji do eksperymentów wynosiły odpowiednio: 10, 50, 100. Czasy pomiarów różnią się w zależności od wielkości instancji rozważanego problemu. Testy rozmiarów populacji odbywały się na zasugerowanych w dokumentacji współczynnikach: krzyżowanie – 0.8 oraz mutacji – 0.01.

4. Badania wpływu wielkości populacji

Graf BR17:
Najlepszy znany wynik: 39
Czas algorytmu: 30s

	Mutacja Swap		Mutacja Insert	
Rozmiar populacji	Wynik	Błąd względny [%]	Wynik	Błąd względny [%]
10	43	10,26	40	2,56
50	42	7,70	39	0,00
100	42	7,70	39	0,00

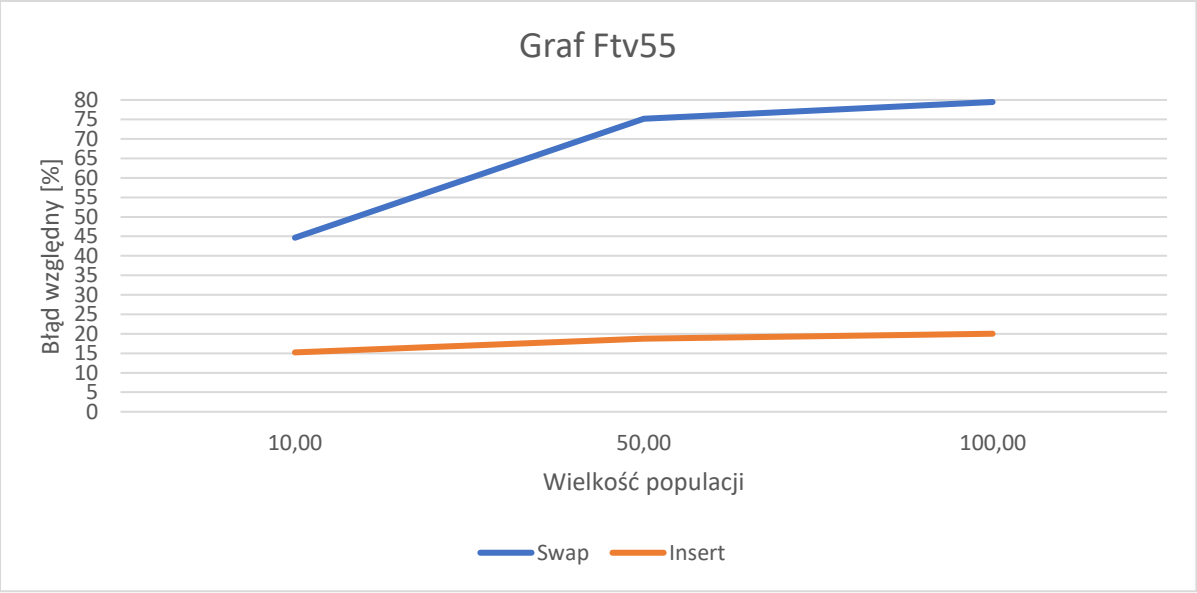


Graf Ftv55:

Najlepszy znany wynik: 1608

Czas algorytmu: 30s

	Mutacja Swap		Mutacja Insert	
Rozmiar populacji	Wynik	Błąd względny [%]	Wynik	Błąd względny [%]
10	2326	44,65	1853	15,23
50	2817	75,18	1910	18,78
100	2886	79,47	1930	20,02

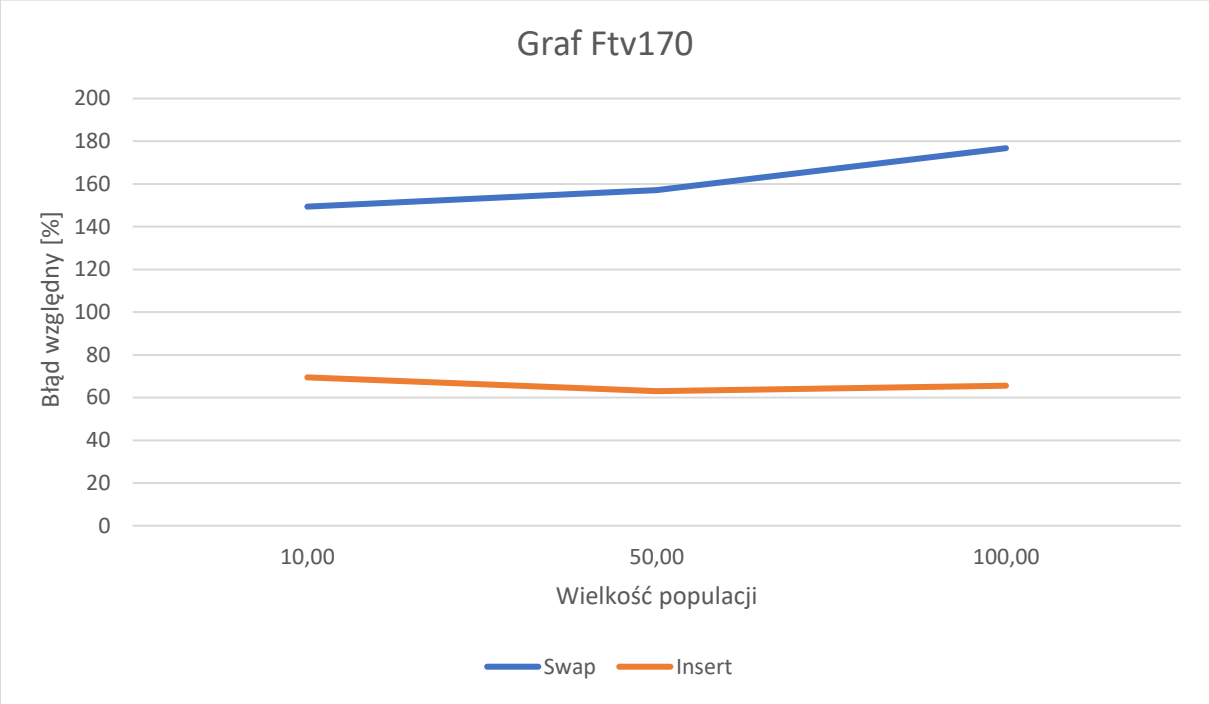


Graf Ftv170:

Najlepszy znany wynik: 2755

Czas algorytmu: 60s

	Mutacja Swap		Mutacja Insert	
Rozmiar populacji	Wynik	Błąd względny [%]	Wynik	Błąd względny [%]
10	6871	149,4	4669	69,47
50	7084	157,13	4491	63,01
100	7625	176,76	4560	65,51



Wnioski:

Rozmiar populacji ma duże znaczenie w przypadku mutacji Swap, zauważono, że w przypadku średnich i dużych instancji problemów wielkości populacji z przedziału 10-50 dają najlepsze wyniki. W przypadku średnich instancji problemów mutacja typu Swap zaczyna diametralnie odstawać od Insert, co mocno rzutuje na jakoś wyników końcowych. W przypadku problemu Ftv170 mutacja Swap generowała wyniki z błędem nawet 150%. Rodzaj mutacji mocno wpływa na rezultaty końcowe i tak jak w przypadku poprzedniego etapu projektowego, sąsiedztwo Invert nie było optymalne dla klas większych klas problemów, tak tutaj mutacje Swap bardzo mocno wpływają na wyniki końcowe.

Sumarycznie, sąsiedztwo Insert rozpoczynając od średnich klas problemów dawały dużo lepsze wyniki, ich miarodajną oraz optymalna wielkość populacji wynosiła 50-100 dla dużych klas problemów oraz od 10-50 (z większym naciskiem na tę większą

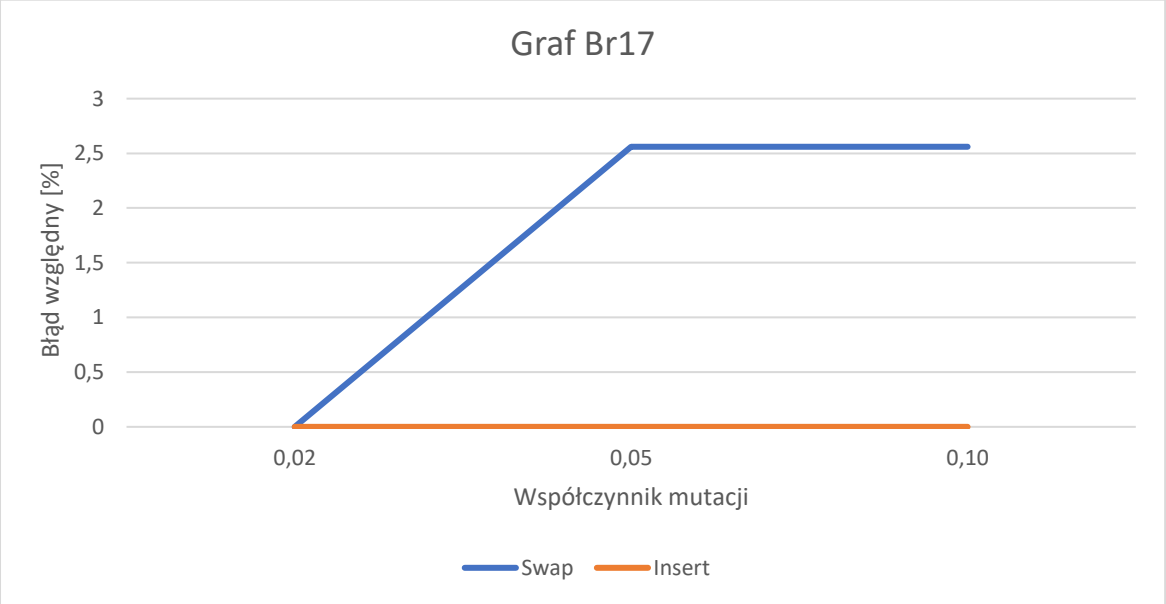
wartość) dla instancji mniejszych. Drobne przekrzywienie wyników na stronę populacji 100 dla instancji Br17 wynikała głównie z szybkiego odnalezienia rozwiązań optymalnych w tablicy kondycji aktualnych rozważań, jednak dla większych klas problemów, zwiększenie populacji miało już większy sens.

5. Badania wpływu współczynnika mutacji

Kolejnym etapem badań było sprawdzenie wpływu współczynnika mutacji na jakość końcowych wyników**. Współczynniki mutacji** miały wartość odpowiednio: **0.02, 0.05 oraz 0.10**. Przy obydwu mutacjach zdecydowano się na **wielkość populacji równą 50** jednostek.

Graf BR17:
Najlepszy znany wynik: 39
Czas algorytmu: 30s
Rozmiar populacji: 50

	Mutacja Swap		Mutacja Insert	
Współczynnik mutacji	Wynik	Błąd względny [%]	Wynik	Błąd względny [%]
0,02	39	0,00	39	0
0,05	40	2,56	39	0
0,10	40	2,56	39	0



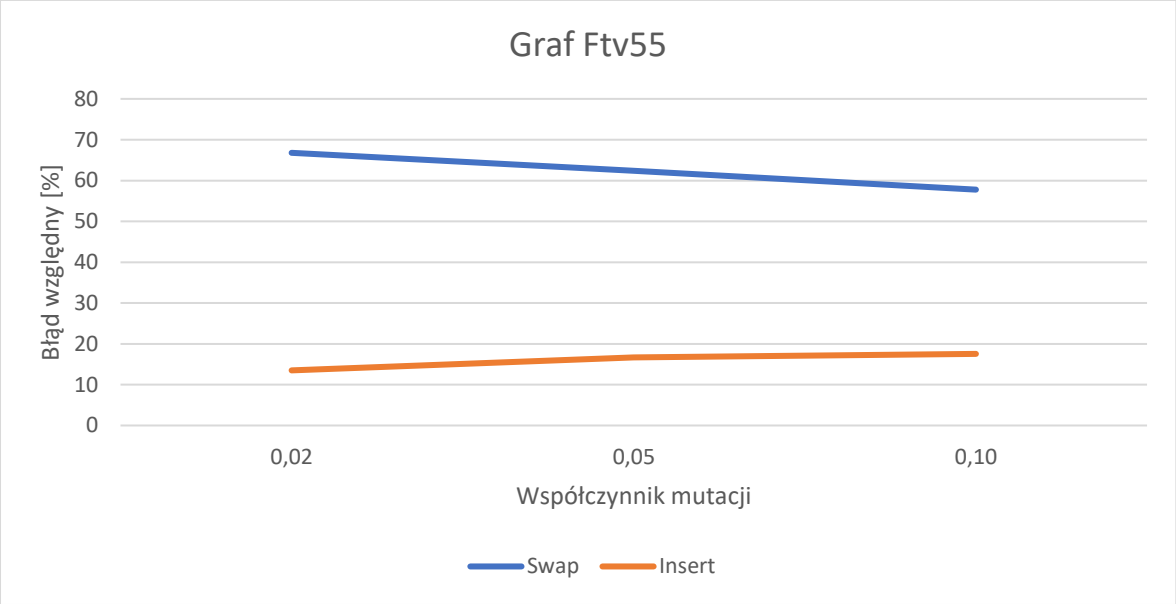
Graf Ftv55:

Najlepszy znany wynik: 1608

Czas algorytmu: 60s

Rozmiar populacji: 50

	Mutacja Swap		Mutacja Insert	
Współczynnik mutacji	Wynik	Błąd względny [%]	Wynik	Błąd względny [%]
0,02	2682	66,79	1825	13,50
0,05	2612	62,43	1876	16,67
0,10	2537	57,77	1890	17,54



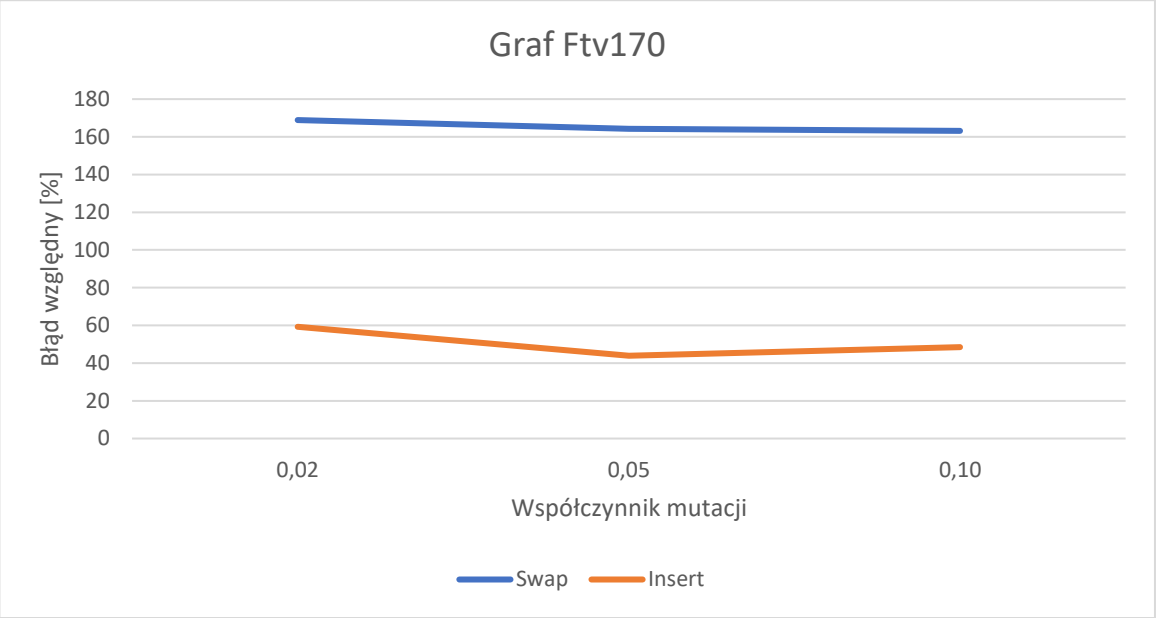
Graf Ftv170:

Najlepszy znany wynik: 2755

Czas algorytmu: 60s

Rozmiar populacji: 50

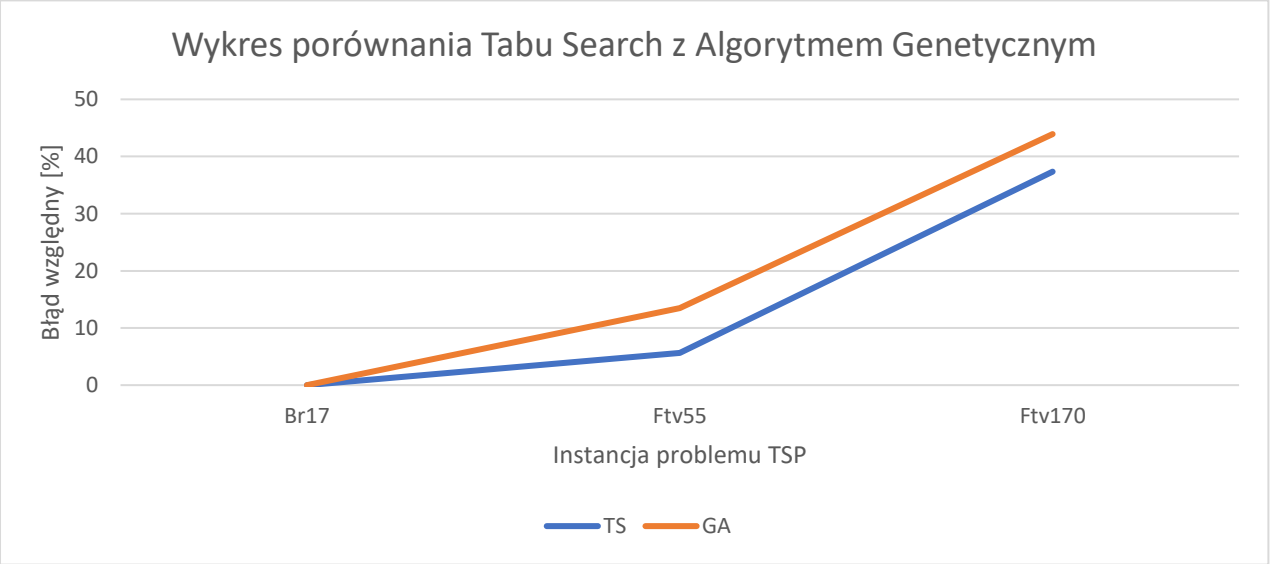
	Mutacja Swap		Mutacja Insert	
Współczynnik mutacji	Wynik	Błąd względny [%]	Wynik	Błąd względny [%]
0,02	7408	168,89	4388	59,27
0,05	7282	164,32	3965	43.92
0,10	7250	163,16	4092	48,53



Wnioski

Analizując tablice oraz wykresy eksperymentu, można zauważyć jak stopniowe zwiększanie współczynnika mutacji wpływa na zachowanie algorytmu przy obydwu podejściach mutacji elementów. W przypadku Mutacji Insert można zauważyć, że dla dużych instancji problemów (Ftv170), zwiększanie współczynnika wpłynęło pozytywnie na zachowanie algorytmu. Najlepsza wartość znajduje się w przedziale <0,05-0,1> (Z naciskiem na dolną granicę przedziału). Udało się usprawnić algorytm nawet to 43,92 [%] błędu względnego, co jest satysfakcjonującym wynikiem. W przypadku małych oraz średnich instancji problemu (W przypadku wyników badania możemy się wzorować głównie na grafie Ftv55), zwiększanie współczynnika mutacji nie miało znaczącego wpływu na jakość rozwiązania. Co prawda poprzez zwiększenie współczynnika do wartości 0,10, zauważono spadek jakości do 17,54 [%], jednak nie jest to różnica wykluczająca wyszczególniony współczynnik. Najlepsze wyniki dawał jednak współczynnik mutacji rzędu <0,02-0,05> (Ponownie z naciskiem na dolną granicę przedziału). Podobnie jak w przypadku drugiego zadania projektowego, zwiększenie współczynnika mutacji przy relatywnie dobrym sposobie generowania zmutowanych elementów wpływa na zwiększenie eksploracji przedziału rozwiązań, co tłumaczyło by zachowanie algorytmu przy większym prawdopodobieństwie zajścia zjawiska. W przypadku zbyt dużego prawdopodobieństwa zaczynamy jednak tracić jakość rozwiązania przez zbyt duże promowanie eksploracji względem eksploatacji przestrzeni obecnie rozważanej. Podobne zjawisko występuje w przypadku mutacji Swap, gdzie przy średnich oraz dużych instancjach problemu, wyniki poprawiają się o parę / parenaście punktów procentowych, nie wpływa to jednak wystarczająco na fakt jak słabo mutacja Swap radzi sobie z dużymi instancjami problemów wciąż bardzo mocno odstając od mutacji Insert.

6. Porównanie Algorytmu Genetycznego z Przeszukiwaniem Tabu



7. Wnioski ogólne

Algorytm genetycznym jest kolejnym z możliwych metod oszacowania rozwiązania problemu Komiwojażera. Do elementów kluczowych, które miały największy wpływ na działanie algorytmu były metody selekcji oraz mutacji elementów (Z większym naciskiem na metody selekcji kandydatów). Wpływ sposobu mutacji można w prosty sposób dostrzec na którymkolwiek z przeprowadzonych eksperymentów, podobnie jak w przypadku algorytmu Tabu Search czy Symulowanego Wyżarzania, mutacje (sąsiedztwo) bazujące na generacji Insert dawały skrajnie lepsze wyniki od generacji Swap.

W przypadku dalszych rozważań nad algorytmem, należałoby się zastanowić nad doborem metody krzyżowania oraz selekcji, w projekcie zostało zaimplementowane proste krzyżowanie OX ze stałym indeksem, a selekcja została zaimplementowana na bazie metody turniejowej. Zmiana lub rozszerzenie tych metod mogło być mieć pozytywny wpływ na zachowanie algorytmu.

Porównanie Algorytmu Genetycznego z Tabu Search pokazuje, że dla średnich oraz większych klas problemów choć nieznacznie, to jednak metoda przeszukiwań z tablicą ruchów zakazanych dawała lepsze wyniki. Wpływ na wyniki obydwu tych metod może mieć wiele czynników od implementacji do doboru parametrów rozruchu algorytmu.

Na podstawie powyższych badań określono relatywnie najlepsze współczynniki rozruchu algorytmu:

Typ mutacji: Mutacja Insert

Współczynnik krzyżowania: 0,08

Współczynnik mutacji: 0,05

Kryterium stopu: Dostosować do instancji problemu, w sprawozdaniu większe oraz średnie instancje problemu testowano dla kryterium stopu – 60 s.