Podstawy efektywnych algorytmów

Prowadzący zajęcia: dr inż. Jarosław Mierzwa
Termin zajęć: Piątek, godz. 11:15

Autor projektu: Mateusz Gawłowski 264463	**Rodzaj grupy:** Późniejsza
Etap projektu: 3	**Data oddania sprawozdania:** 19.01.2024

Spis treści

1. Wstęp teoretyczny

Genetic Algorithm

1. Generacja początkowej populacji za pomocą metody zachłannej.
2. Z obecnej populacji generacja nowej z wykorzystaniem metody turniejowej. Wybieramy kilku osobników z aktualnej populacji i wybieramy spośród nich najlepszego.
3. Krzyżowanie osobników c razy, gdzie
    - *c = population_size * crossing_rate * mutation_rate*
4. Mutacja osobników m razy, gdzie
    - *m = population_size * mutation_rate * mutation_rate*
5. Kroki od 2 do 5 są powtarzane aż do osiągnięcia zadanego czasu.

Metoda Turniejowa

Z populacji wybieranych jest 5 osobników, a do nowej populacji przechodzi tylko najlepszy. Turniej jest powtarzany x razy, gdzie

- *x = population_size x tournament_size*

Krzyżowanie

Ordered Crossover (OX)

Z obecnej populacji wybieramy 2 osobniki. Na przykład:

Parent_1 = 1 2 3 4 5 6 7 8 9
Parent_2 = 7 4 9 8 5 6 3 2 1

(Zera zostały pominięte ze względu na ich brak udziału w krzyżowaniu – są one elementami stałymi.)

Następnie generujemy początek i koniec. Na przykład:

Begin = 2
End = 7

Parent_1 = 1 2 3 4 5 6 7 8 9

Potomek Descendant_1 zaczyna się tak:

Descendant_1 = [ ] [ ] 3 4 5 6 7 8 [ ]

Puste miejsca są uzupełniane z Parent_2. Przy każdym uzupełnieniu sprawdzamy, czy element nie został już skopiowany z Parent_1. Jeśli został, nie dodajemy go i przechodzimy do kolejnego elementu z Parent_2.

- Parent_2_0 = 7 -> Istnieje już w Parent_1 -> Nie dodajemy do Descendant_1
- Parent_2_1 = 4 -> Istnieje już w Parent_1 -> Nie dodajemy do Descendant_1
- Parent_2_2 = 9 -> Nie istnieje w Parent_1 -> Dodajemy do Descendant_1

Descendant_1 = 9 [ ] 3 4 5 6 7 8 [ ]

- Parent_2_3 = 8 -> Istnieje już w Parent_1 -> Nie dodajemy do Descendant_1
- Parent_2_4 = 5 -> Istnieje już w Parent_1 -> Nie dodajemy do Descendant_1
- Parent_2_6 = 6 -> Istnieje już w Parent_1 -> Nie dodajemy do Descendant_1
- Parent_2_7 = 3 -> Istnieje już w Parent_1 -> Nie dodajemy do Descendant_1
- Parent_2_8 = 2 -> Nie istnieje w Parent_1 -> Dodajemy do Descendant_1

Descendant_1 = 9 2 3 4 5 6 7 8 [ ]

- Parent_2_9 = 1 -> Nie istnieje w Parent_1 -> Dodajemy do Descendant_1

Descendant_1 = 9 2 3 4 5 6 7 8 1

Po zakończeniu dodawania ostatniej cyfry, postępujemy analogicznie dla Descendant_2.

Partially Mapped Crossover (PX)

Z obecnej populacji wybieramy 2 osobniki. Na przykład:

Parent_1 = 1 2 3 4 5 6 7 8 9
Parent_2 = 7 4 9 8 5 6 3 2 1

Następnie generujemy początek i koniec. Na przykład:

Begin = 2
End = 7

Parent_1 = 1 2 3 4 5 6 7 8 9
Parent_2 = 7 4 9 8 5 6 3 2 1

Tworzymy mapę, która wygląda następująco:

3 - 9
4 - 8
5 - 5
6 - 6
7 - 3
8 - 2

Potomek Descendant_1 zaczyna się tak:

Descaentant_1 = [ ] [ ] 3 4 5 6 7 8 [ ]

Puste miejsca są uzupełniane z Parent_2. Przy każdym uzupełnieniu sprawdzamy, czy element nie został już skopiowany z Parent_1. Jeśli został, patrzymy na utworzoną mapę.

- Parent_2_0 = 7 -> Istnieje już w Parent_1. Odczytujemy z mapy: 7 - 3 -> 3 Istnieje już w Parent_1
- Parent_2_0 = 3 -> Istnieje już w Parent_1. Odczytujemy z mapy: 3 - 9 -> 9 nie istnieje w Parent_1

Descendant_1 = 9 [ ] 3 4 5 6 7 8 [ ]

- Parent_2_1 = 4 -> Istnieje już w Parent_1. Odczytujemy z mapy: 4 - 8 -> 8 Istnieje już w Parent_1
- Parent_2_1 = 8 -> Istnieje już w Parent_1. Odczytujemy z mapy: 8 - 2 -> 2 nie istnieje w Parent_1

Descendant_1 = 9 2 3 4 5 6 7 8 [ ]

- Indeks z Parent_2 w (przeciwieństwie do OX) odpowiada temu z Descendant_1. W związku z tym będzie on równy 8.

- Parent_2_8 = 1 -> Nie istnieje w Parent_1 -> Przypisujemy do Descendant_1.

Descendant_1 = 9 7 8 4 5 6 7 2 1

Po zakończeniu dodawania ostatniej cyfry, postępujemy analogicznie dla Descendant_2.

Mutacja

Mutacja polega na losowaniu osobnika i zamianie losowo dwóch jego elementów.

2. Plan eksperymentu

Analiza efektywności działania algorytmu została przeprowadzona dla trzech instancji problemu komiwojażera o różnych rozmiarach. Dla każdego z zadanego pliku, konieczne było obliczenie błędu dostarczonego przez algorytm w zależności od rozmiaru populacji oraz schematu krzyżowania.

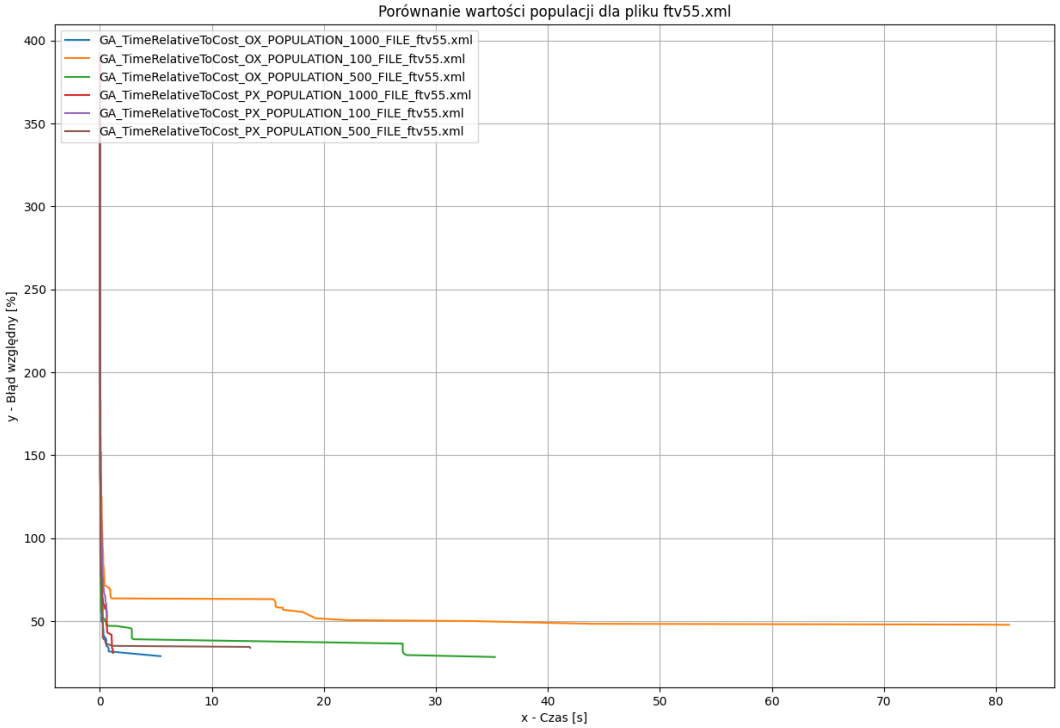
Proces eksperymentu został podzielony na trzy etapy:

- Badanie rozmiaru populacji
- Badanie współczynnika mutacji
- Porównanie z algorytmem Simulated Annealing

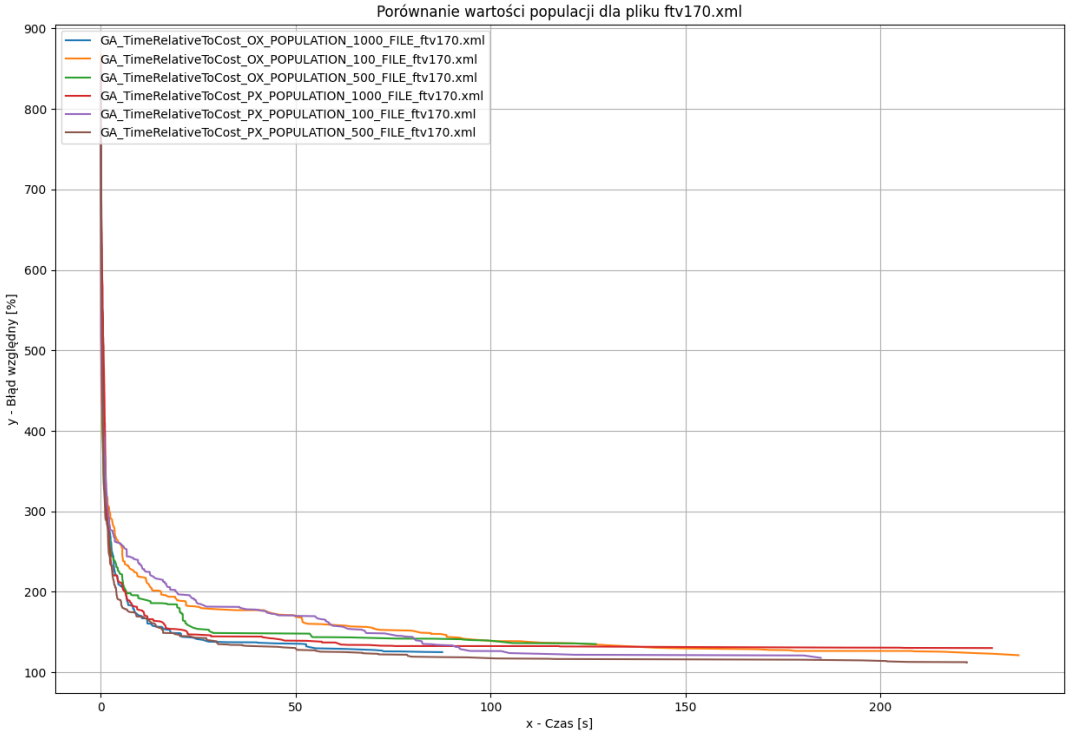
3. Zestawienie pomiarów z eksperymentu.

Poniżej zestawione zostały wyniki przeprowadzonych analiz dla każdego etapu eksperymentu.

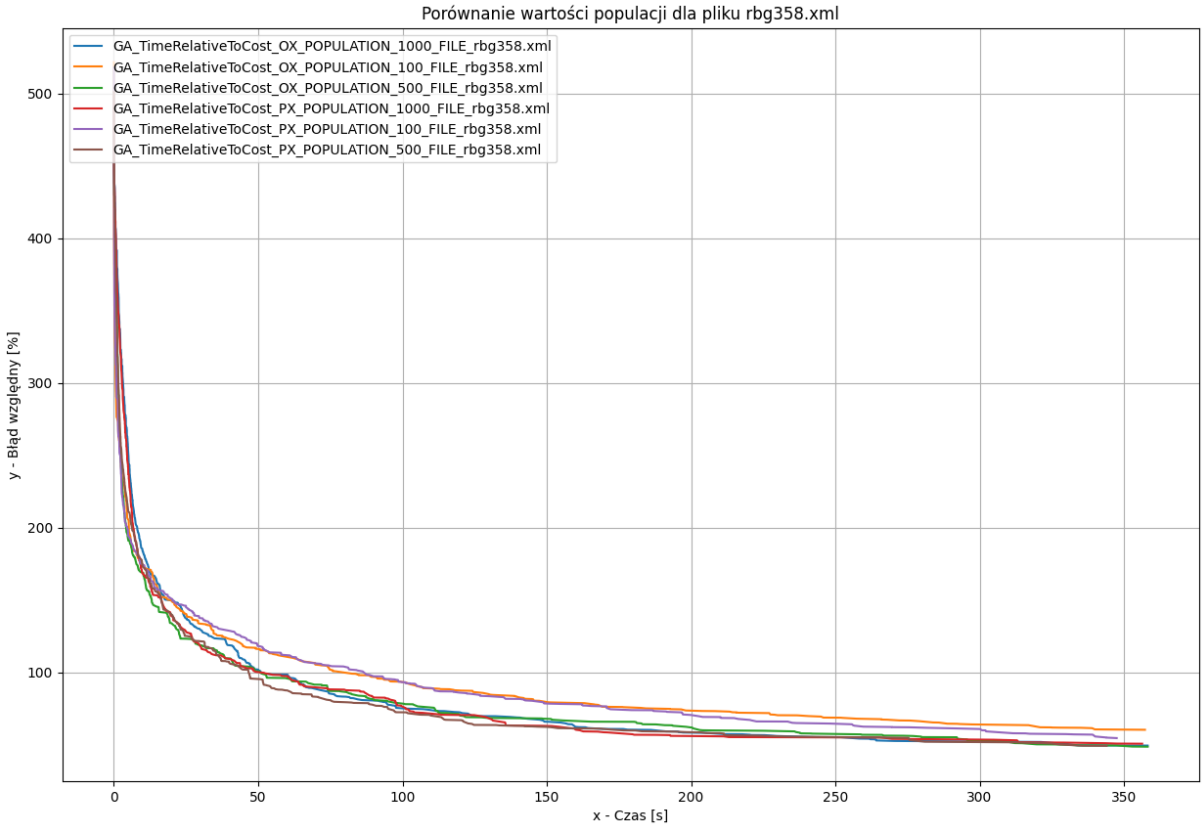
Badanie rozmiaru populacji



Rysunek 1 – Porównanie wartości populacji dla pliku ftv55.xml

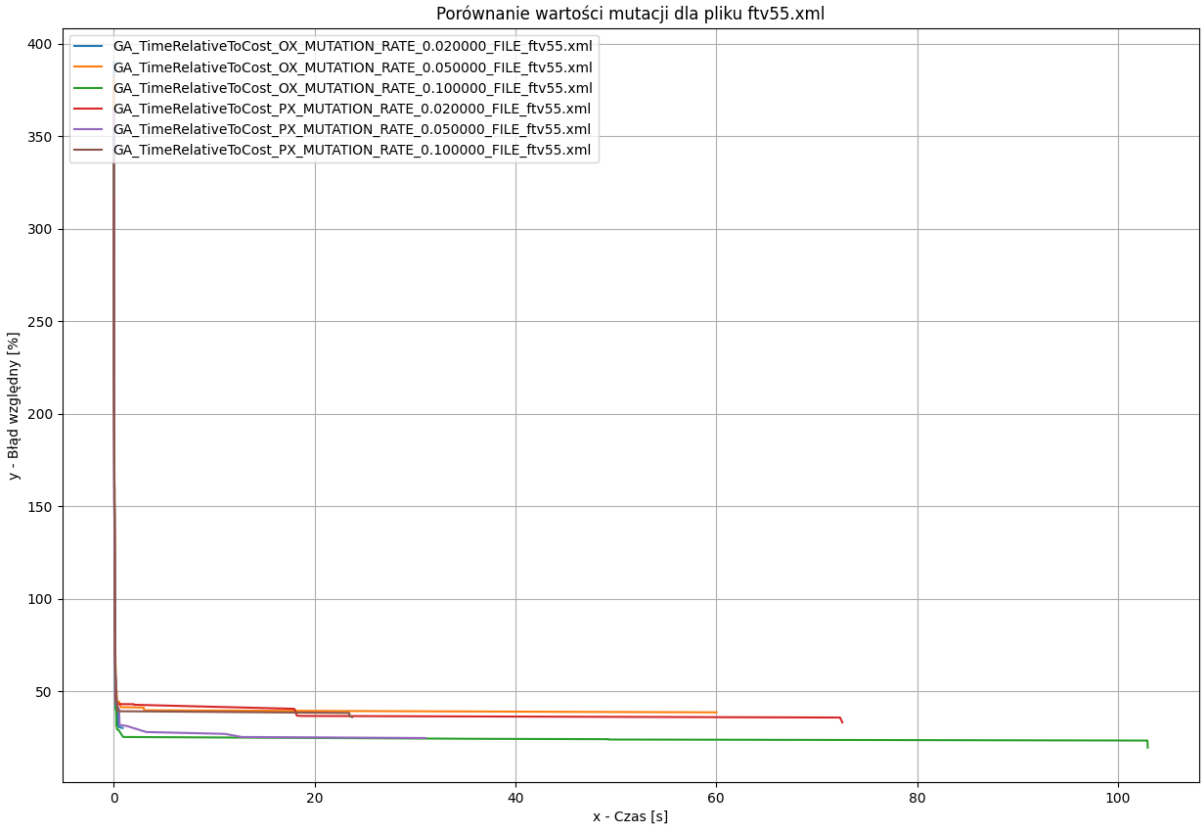


Rysunek 2 – Porównanie wartości populacji dla pliku ftv170.xml

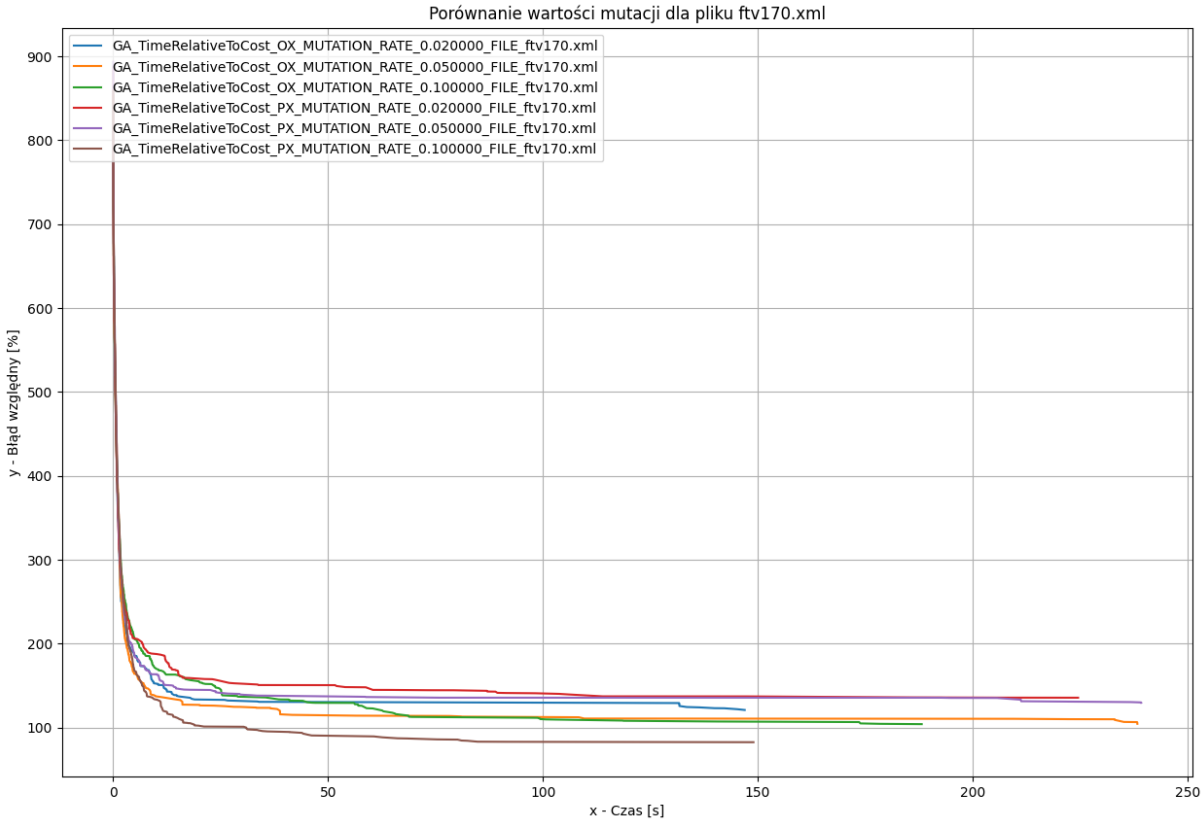


Rysunek 3 – Porównanie wartości populacji dla pliku rbg358.xml

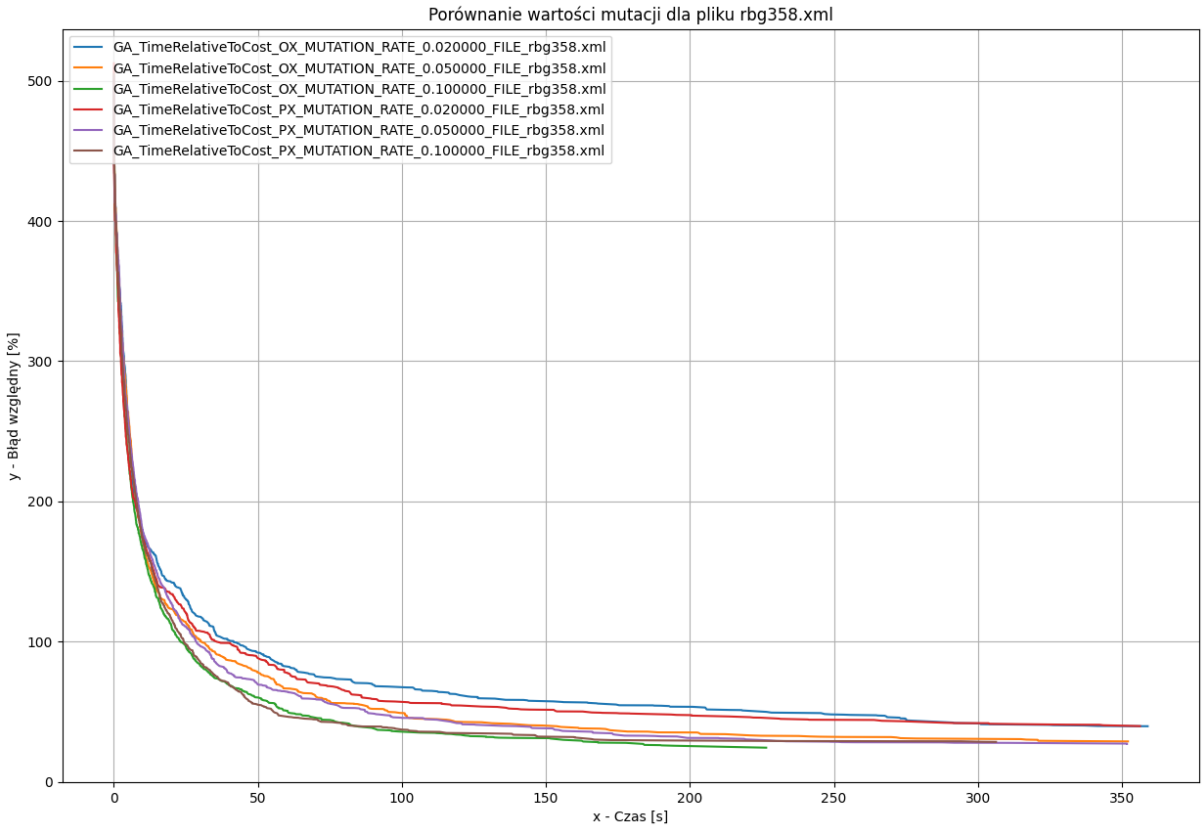
Badanie współczynnika mutacji



Rysunek 4 – Porównanie wartości mutacji dla pliku ftv55.xml

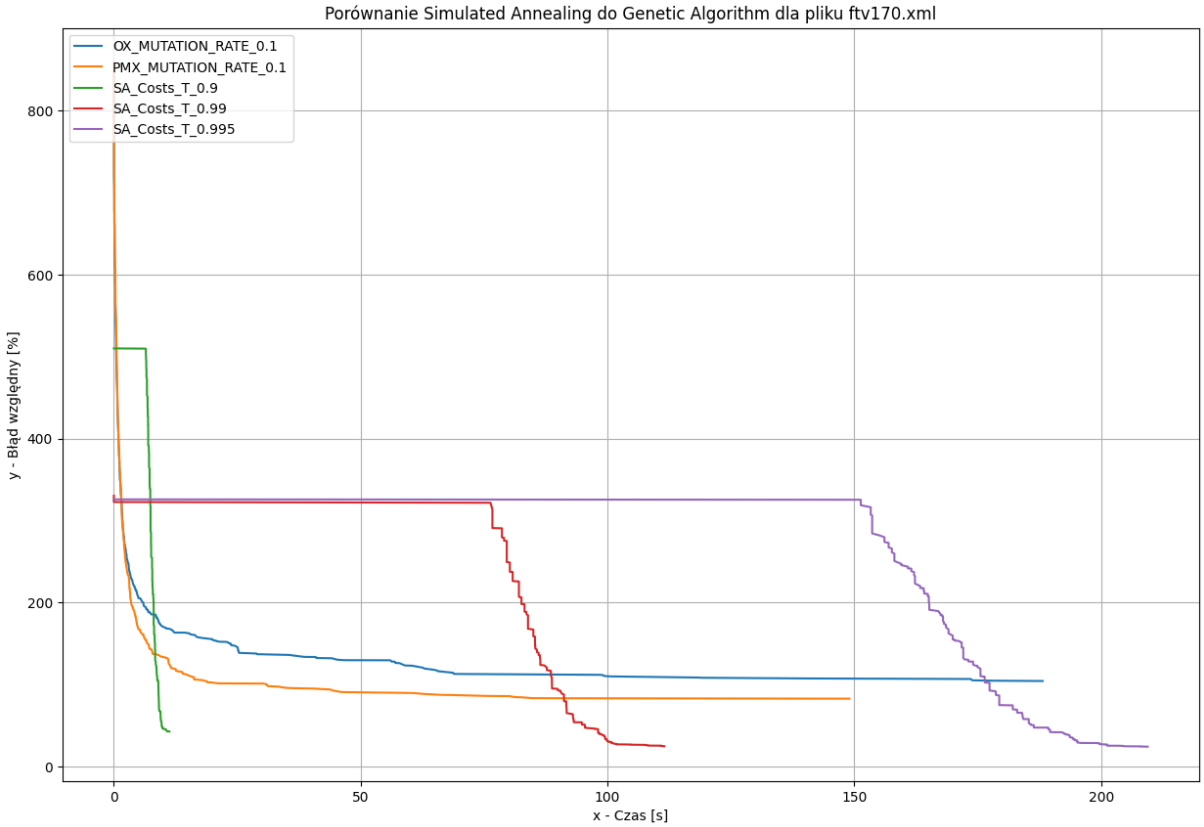


Rysunek 5 – Porównanie wartości mutacji dla pliku ftv170.xml



Rysunek 6 – Porównanie wartości mutacji dla pliku rbg358.xml

Porównanie z algorytmem Simulated Annealing



Rysunek 7 – Porównanie Simulated Annealing do Genetic Algorithm dla pliku rbg358.xml

4. Wnioski

Podczas eksperymentów projektowych skoncentrowano się na analizie różnych aspektów algorytmu genetycznego w kontekście rozwiązania konkretnego problemu optymalizacyjnego. Poniżej przedstawiono podsumowanie wniosków z poszczególnych eksperymentów:

- Badanie Wielkości Populacji:

Analiza wyników wykazała, że populacja równa 1000 osobników przynosi najbardziej optymalne rezultaty dla rozwiązywanego zadania. Zarówno krzyżowanie OX_POPULATION_1000, jak i PX_POPULATION_1000, wydają się efektywnie dążyć do optymalnego rozwiązania. Stąd wniosek, że rozmiar populacji stanowi kluczowy czynnik wpływający na skuteczność algorytmu genetycznego w tym konkretnym kontekście.

- Badanie Współczynnika Mutacji:

Wyniki analizy wskazują, że współczynnik mutacji równy 0.1 jest najbardziej optymalny dla rozpatrywanego problemu optymalizacyjnego. Zarówno krzyżowanie OX_MUTATION_0.1, jak i PX_MUTATION_0.1, uzyskują lepsze wyniki lub korygują szybciej. Stąd wniosek, że kontrolowanie wartości mutacji ma kluczowe znaczenie dla efektywności algorytmu genetycznego w tym przypadku.

- Porównanie z Algorytmem Simulated Annealing:

Porównanie z algorytmem Symulowanego Wyżarzania (SA) wykazało, że SA osiąga lepsze wyniki dla każdej instancji problemu. Mimo początkowej przewagi algorytmu genetycznego dla wyższych współczynników chłodzenia w SA, to SA zdaje się lepiej radzić sobie po etapie eksploracji. W związku z tym, dla badanego zestawu problemów, wywnioskowano, że implementacja algorytmu SA jest bardziej efektywna niż algorytm genetyczny.

Podsumowując, wyniki eksperymentów wskazują, że optymalna konfiguracja algorytmu genetycznego dla rozwiązania danego problemu obejmuje populację równą 1000 osobników i współczynnik mutacji na poziomie 0.1. Jednakże, w porównaniu z SA, algorytm genetyczny może być mniej efektywny dla tego konkretnego zestawu problemów. Wartość parametrów algorytmu genetycznego jest kluczowa, a ich optymalizacja może zależeć od specyfiki badanego zadania optymalizacyjnego.

5. Bibliografia

- http://aragorn.pb.bialystok.pl/~wkwedlo/EA5.pdf
- http://www.imio.polsl.pl/Dopobrania/Cw%20MH%2007%20(TSP).pdf