Sprawozdanie
Projektowanie Efektywnych Algorytmów

Temat:
Implementacja i analiza efektywności algorytmu genetycznego (ewolucyjnego) dla ATSP

Politechnika Wrocławska
Dawid Szeląg 264008
Prowadzący: dr. inż. Jarosław Mierzwa
18.01.2024r.

Politechnika
Wrocławska

Spis treści

1 Algorytm genetyczny

1.1 Krótki opis algorytmu

Algorytmy genetyczne odzwierciedlają procesy ewolucyjne, których celem jest jak najlepsze dopasowanie osobników w populacji, do konkretnych warunków życia. Korzystają one z zasady, że najlepsze osobniki, najlepiej przystosowane, przeżywają. Cały ciąg algorytmu jest następujący:

1. Utworzenie początkowej populacji (w tym przypadku jest to rozwiązane w sposób losowy)
2. Selekcja do puli macierzystej
3. Operacje genetyczne na puli macierzystej (krzyżowanie oraz mutacja)
4. Sukcesja oraz utworzenie nowej populacji
5. Ocena nowo utworzonej populacji
6. Powtórz krok 2-5, dopóki nie zostanie spełniony warunek zatrzymania (w tym przypadku czas)

1.2 Metoda selekcji

W projekcie zastosowano **metodę turniejową**. Polega ona na wylosowaniu z populacji **k** osobników, gdzie **k** oznacza wielkość turnieju, a następnie z wybranej części populacji, wygrywa najlepiej przystosowany, który później trafia do puli macierzystej.
W projekcie **k** jest stałą, zależną od wielkości populacji w następujący sposób: $k = 0.3 * N$, gdzie N oznacza wielkość populacji.

1.3 Operacje genetyczne

Po selekcji osobników do puli macierzystej, następuję wykonanie operacji genetycznych, z pewnym prawdopodobieństwem.

1.3.1 Krzyżowanie

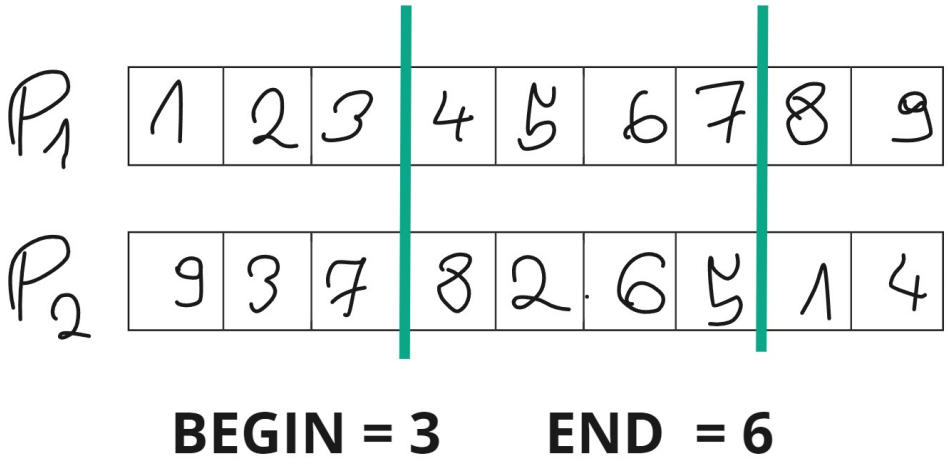
Krzyżowanie polega na wymianie informacji, pomiędzy 2 osobnikami, z których następnie powstaje 2 potomków, gdzie każdy z nich ma część informacji od jednego rodzica, a część od drugiego.

UWAGA - Przedstawione przykłady pokazują ścieżki bez początkowego i końcowego miasta!

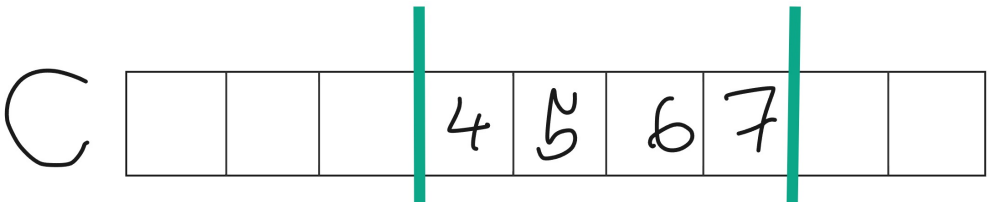
W projekcie zastosowano krzyżowanie **PMX** oraz **OX**.

PMX Metoda Partially Matched Crossover (PMX), polega na:

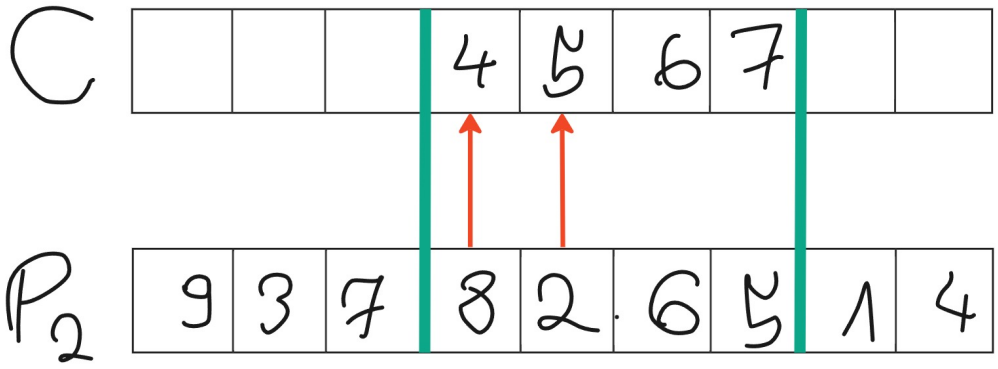
1. Wybraniu losowo 2 punktów krzyżowania (liczymy od 0),



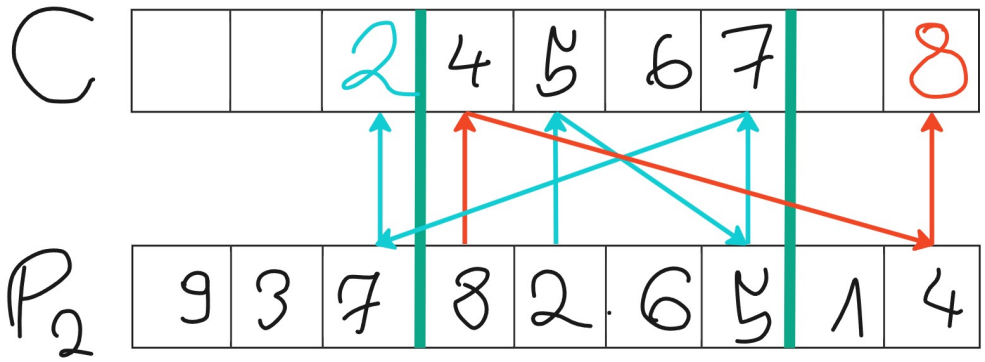
2. Przepisaniu informacji zawartych pomiędzy tymi 2 punktami u pierwszego rodzica do potomka



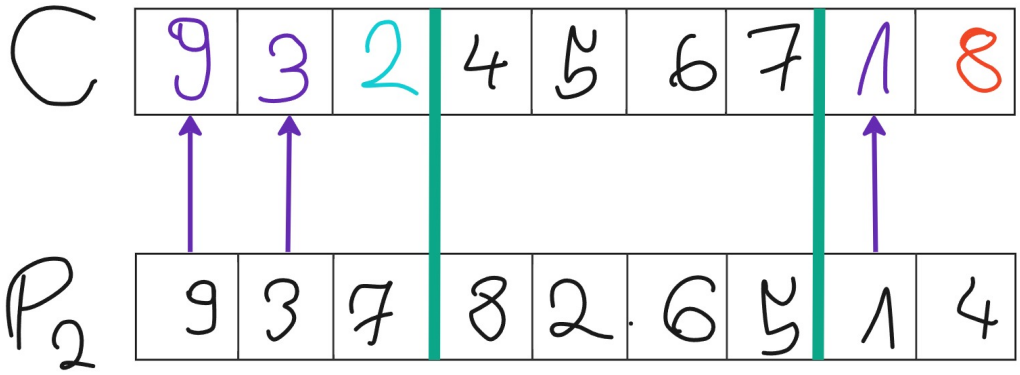
3. Wyznaczeniu par pomiędzy drugim rodzicem, a potomkiem, dla liczb które znajdują się u drugiego rodzica, lecz **NIE** znajduja się u potomka (porównanie odnosi się do sekcji dopasowania)



4. Zgodnie z przyporządkowaniem w parach, następuje przepisanie liczb od drugiego rodzica do potomka, na pozycji od drugiego rodzica, która posiada liczbę wskazującą liczbe potomka. Jeśli pozycja znajduje się w sekcji doposowania, następuje stworzenie kolejnej pary.



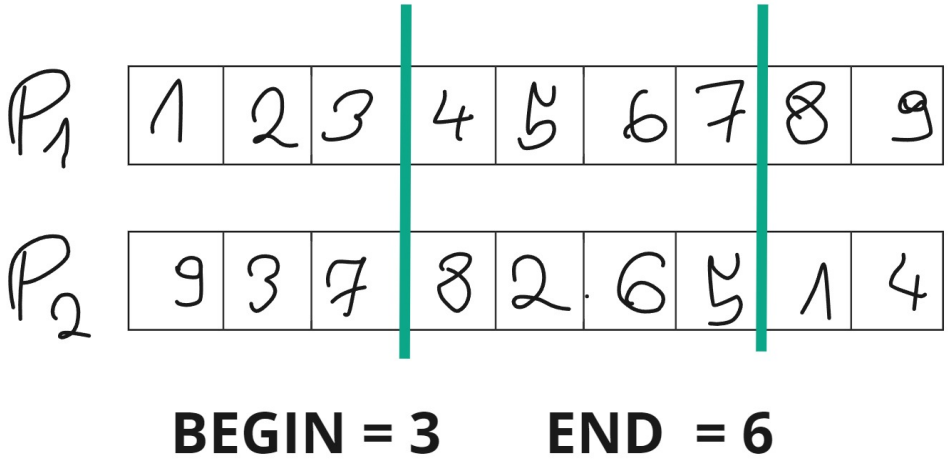
5. Pozostałe liczby przepisujemy



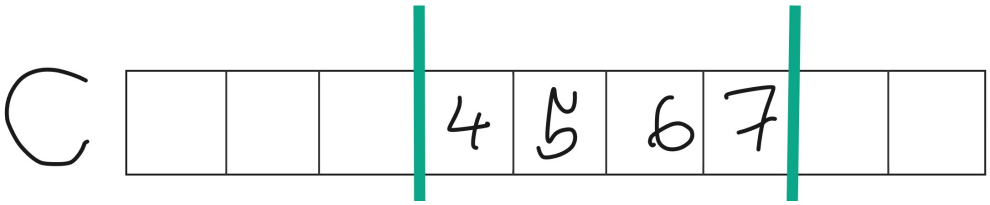
. Dla utworzenia drugiego potomka, należy odwrócić kolejność rodziców. Jeśli krzyżowanie nie zachodzi, do nowej populacji przechodzą rodzice.

OX Metoda Order Crossover (OX), polega na:

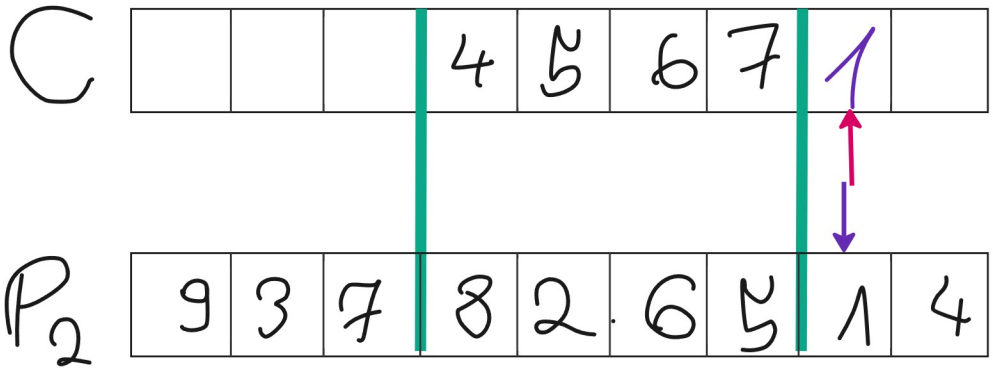
1. Wybraniu losowo 2 punktów krzyżowania (liczymy od 0),



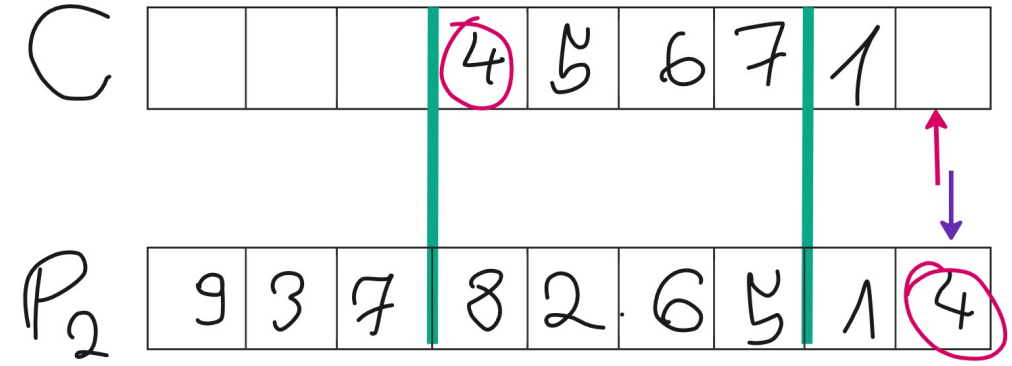
2. Przepisaniu informacji zawartych pomiędzy tymi 2 punktami u pierwszego rodzica do potomka

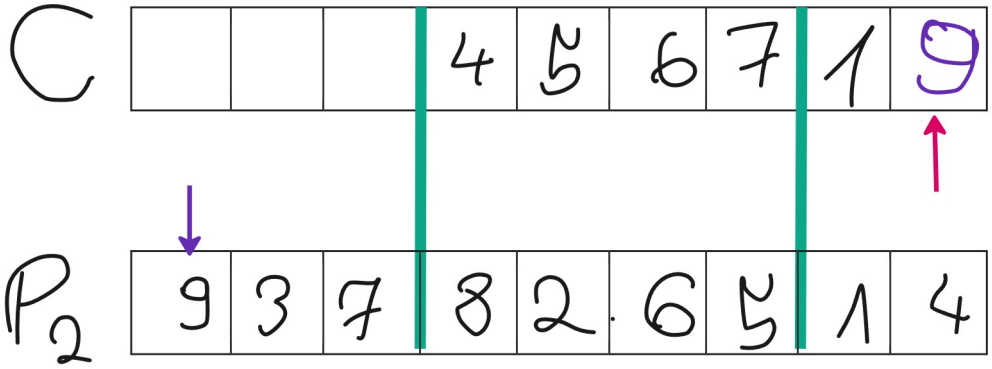


3. Następnie rozpoczyna się przepisywanie liczb od pozycji **end + 1**, od rodzica drugiego do potomka

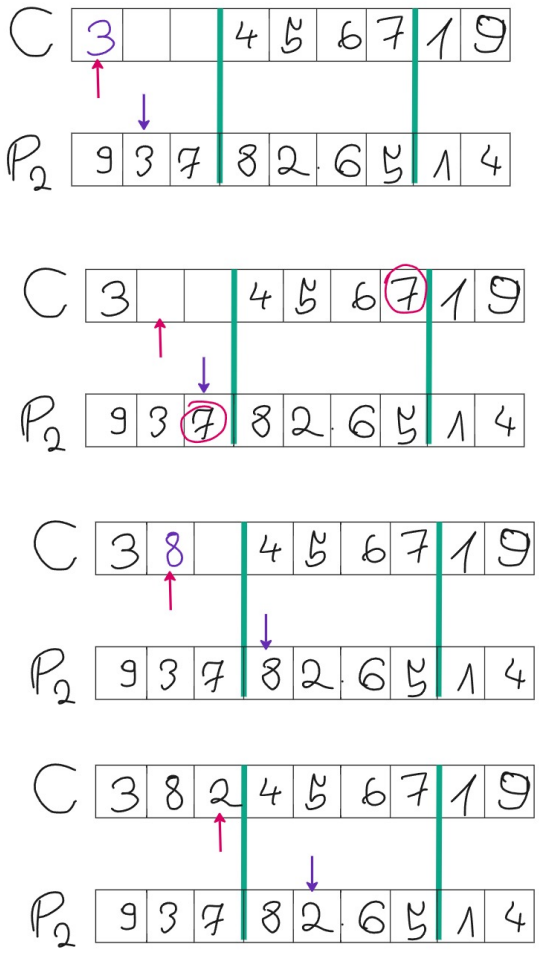


4. Jeśli liczba już znajduje się w potomku, należy przejść do kolejnej liczby (w tym przypadku na początek). **UWAGA!** Aktualna pozycja na potomku **NIE** zmienia się





5. Następnie kontynuujemy procedure, aż nie zapełnimy całego potomka.



Dla utworzenia drugiego potomka, należy odwrócić kolejność rodziców. Jeśli krzyżowanie nie zachodzi, do nowej populacji przechodzą rodzice.

1.3.2 Mutacje

Mutacje polegają na losowych zmianach w danym osobniku. Obarczone jest prawdopodobieństwem, tak samo jak krzyżowanie. Zazwyczaj występuje bardzo rzadko.

W projekcie zastosowano mutacje **INSERT** oraz **SWAP**.

INSERT Bierzemy dwa elementy **xi** i **xj**, **xj** wstawiamy przed **xi**.

SWAP Bierzemy dwa elementy **xi** i **xj**, zamieniamy je ze sobą miejscami.

1.4 Sukcesja oraz nowa populacja

Sukcesja to nic innego jak proces zmiany populacji w kolejnych pokoleniach. W projekcie został wybrany wariant reprodukcji częciowej według stopnia przystosowania.
Mianowicie: **10% najlepszych osobników starej populacji** trafia do nowej. Reszta nowej populacji jest zapełniana poprzez selekcje oraz operatory genetyczne.

2 Implementacja

2.1 Klasa Tour

Klasa odpowiada za przechowywanie ścieżki oraz jej kosztu. Zawiera w sobie:

- int fitness;
- std::vector¡int¿ nodes;

2.2 Klasa Population

Klasa odpowiada za przechowywanie obiektów Tour, które są całą populacją. Zawiera w sobie:

- std::vector<Tour>tours;
- int sizePopulation;
- AdjacencyMatrix* matrix;
- oraz metode *Tour getBestTour()* do zwracania najlepszej trasy

2.3 Klasa CrossoverStrategy

Klasa abstrakcyjna, po której dziedziczą wszystkie strategie krzyżowania. Zawiera w sobie:

- Generator generator; - odpowiada za generowanie liczb
- static AdjacencyMatrix* matrix;

- virtual void doCrossover(Tour &parent1, Tour &parent2) = 0; - metoda która wykonuje krzyżowanie

Dziedziczą po niej klasy **PMXCrossover** oraz **OXCrossover**.

2.4 Klasa MutationStrategy

Klasa abstrakcyjna, po której dziedziczą wszystkie strategie mutacji. Zawiera w sobie:

- Generator generator; - odpowiada za generowanie liczb
- static AdjacencyMatrix* matrix;
- virtual void doMutation(Tour &tour) = 0; - metoda która wykonuje mutacje

Dziedziczą po niej klasy **InsertMutation** oraz **SwapMutation**.

2.5 Klasa GeneticAlgorithm

Zmienne/kolekcje wykorzystane w klasie:

```
AdjacencyMatrix* matrix; //macierz sąsiedztwa
double mutationRate; //prawdopodobieństwo mutacji
double crossRate; //prawdopodobieństwo krzyżowania
int tournamentSize; //rozmiar turnieju
int populationSize; //rozmiar populacji
double stopTime;
CrossoverStrategy* crossoverStrategy;
MutationStrategy* mutationStrategy;
int bestCost;
int bestKnownResult;
Tour bestTour;
Population population;
std::vector<std::vector<double>> shortestPathChanging;
double bestValueTime;
double error;
```

Najważniejsze metody:

```
void startAlgorithm(); //rozpoczęcie algorytmu
void checkBestTour(double time); //sprawdzenie najlepszej
↪  drogi
void selection(Population &parentalPopulation); //selekcja
↪  turniejowa
void crossoverAndMutation(Population &parentalPopulation,
↪  Population &newPopulation); //operacje genetyczne
static bool compareTour(Tour &t1, Tour &t2); //metoda na
↪  potrzeby sortowania

void GeneticAlgorithm::startAlgorithm() {
    Timer timer;

    timer.run();
    while (timer.readS() < stopTime)
    {
        Population newPopulation(matrix);
        Population parentalPopulation(matrix);

        ///10% BEST TO NEW
        //sortowanie populacji
        sort(population.getTours()->begin(),
        ↪  population.getTours()->end(), compareTour);
        //10% populacji (najlepszych tras) przechodzi do
        ↪  nowej
        for (int i = 0; i < populationSize / 10; ++i) {

            ↪  newPopulation.addTour(population.getTours()->at(i));
        }

        ///SELEKCJA DO PULI MACIERZYSTEJ
        selection(parentalPopulation);

        ///KRZYŻOWANIE I MUTACJA W PULI MACIERZYSTEJ ORAZ
        ↪  DODANIE GENEOTYPÓW DO NOWEJ POPULACJI
        crossoverAndMutation(parentalPopulation,
        ↪  newPopulation);

        ///PRZEPISANIE
        population = newPopulation;
```

```
        /// OCENA
        checkBestTour(timer.readMs());
    }
}


void GeneticAlgorithm::crossoverAndMutation(Population
↪  &parentalPopulation, Population &newPopulation) {
    Generator generator;
    int indexRandom1;
    int indexRandom2;
    int newPopulationSize = newPopulation.getSize();
    for (int i = 0; i < (populationSize - newPopulationSize)/2;
    ↪  ++i) {
        indexRandom1 =
        ↪  generator.getNumber(0,parentalPopulation.getSize()-1);
        indexRandom2 =
        ↪  generator.getNumber(0,parentalPopulation.getSize()-1);

        //zabezpieczenie przed wylosowaniem tego samego
        ↪  genotypu
        while (indexRandom1 == indexRandom2)
            indexRandom2 =
            ↪  generator.getNumber(0,parentalPopulation.getSize()-1);

        Tour
        ↪  newTour1(parentalPopulation.getTours()->at(indexRandom1));
        Tour
        ↪  newTour2(parentalPopulation.getTours()->at(indexRandom2));

        //według podanego prawdopodobieństwa, robimy
        ↪  krzyżowanie osobników lub też nie
        if (generator.randomZeroToOne() <= crossRate)
        {
            // z 2 rodziców powstaje 2 dzieci
            crossoverStrategy->doCrossover(newTour1, newTour2);
        }

        ///MUTACJE
        if (generator.randomZeroToOne() <= mutationRate)
            mutationStrategy->doMutation(newTour1);
```

```
            if (generator.randomZeroToOne() <= mutationRate)
                mutationStrategy->doMutation(newTour2);
            newPopulation.addTour(newTour1);
            newPopulation.addTour(newTour2);
        }

        if (newPopulation.getSize() < populationSize)

            ↪  newPopulation.addTour(parentalPopulation.getTours()->at(generator.ge

    }
```

3 Plan eksperymentu

Program został napisany w języku C++ w środowisku CLion i testowany byl w wersji Release.
Grafy wejściowe, na których należało przeprowadzić badania to pliki:

- ftv47.atsp (optymalny znany koszt: 1776)
- ftv170.atsp (optymalny znany koszt: 2755)
- rbg403.atsp (optymalny znany koszt: 2465)

Pomiar czasu został przeprowadzony przy pomocy wykorzystania zegara z biblioteki **<chrono>**: std::chrono::high_resolution_clock, a generowanie liczb, zostało zrealizowane za pomocą generatora liczb mt19937 z biblioteki **<random>**.

Dla każdego testu wykonano 5 uruchomień z ustalonymi wcześniej parametrami startowymi.
Założenia:

- rozmiar turnieju = 0.3 * N, gdzie N to rozmiar populacji

Czas wykonania testu dla plików:

- ftv47.atsp - 120s
- ftv170.atsp - 240s
- rbg403.atsp - 360s

4 Wyniki

4.1 Plik FTV47.atsp

4.1.1 Wpływ wielkości populacji na wyniki

Do badań wybrano 3 wartości populacji:

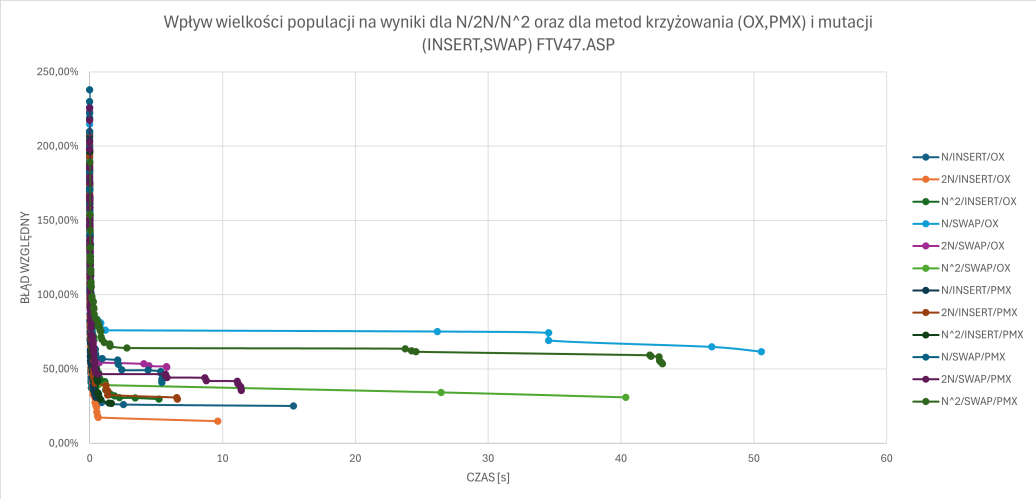
- **N**
- **2N**
- N^2, gdzie **N** oznacz liczbe miast w danym pliku.

Przyjęte współczynniki krzyżowania:

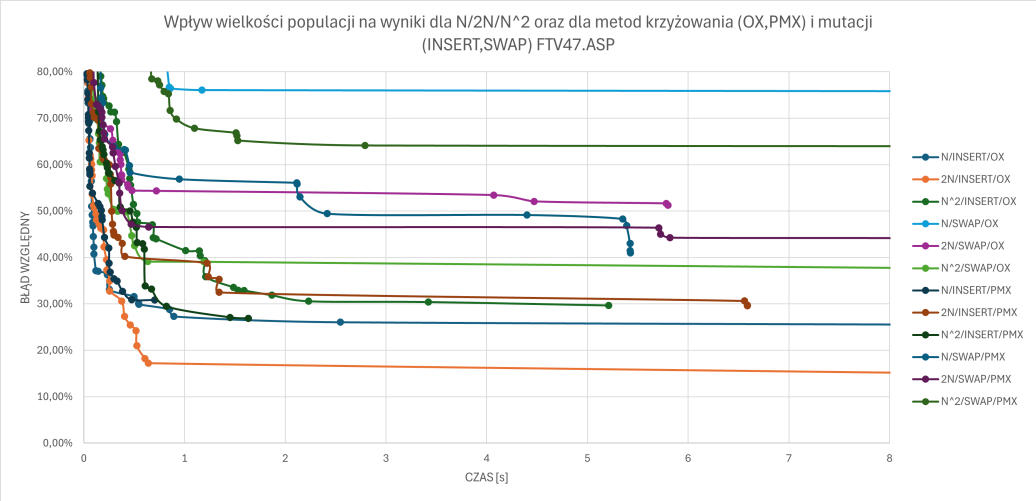
- współczynnik mutacji = **0.01**
- współczynnik krzyżowania = **0.8**

Wykorzystano 2 metody krzyżowania (**PMX**, **OX**) oraz 2 metody mutacji (**INSERT**, **SWAP**).

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Mutacja	Krzyżowanie
1776	2222	25,11%	15,34	47	INSERT	OX
1776	2038	14,75%	9,63	94	INSERT	OX
1776	2303	29,67%	5,21	2209	INSERT	OX
1776	2871	61,66%	50,56	47	SWAP	OX
1776	2686	51,24%	5,80	94	SWAP	OX
1776	2324	30,86%	40,36	2209	SWAP	OX
1776	2324	30,86%	0,70	47	INSERT	PMX
1776	2302	29,62%	6,59	94	INSERT	PMX
1776	2253	26,86%	1,63	2209	INSERT	PMX
1776	2504	40,99%	5,43	47	SWAP	PMX
1776	2407	35,53%	11,41	94	SWAP	PMX
1776	2727	53,55%	43,11	2209	SWAP	PMX



Rysunek 1: 12 krzywych/Wszystkie kominacje 1/2 - ftv47



Rysunek 2: 12 krzywych/Wszystkie kominacje 2/2 - ftv47

Najlepszy wynik uzyskano dla:

- populacji - **2N**
- krzyżowania - **OX**
- mutacji - **INSERT**

4.1.2 Wpływ współczynnika mutacji na wyniki

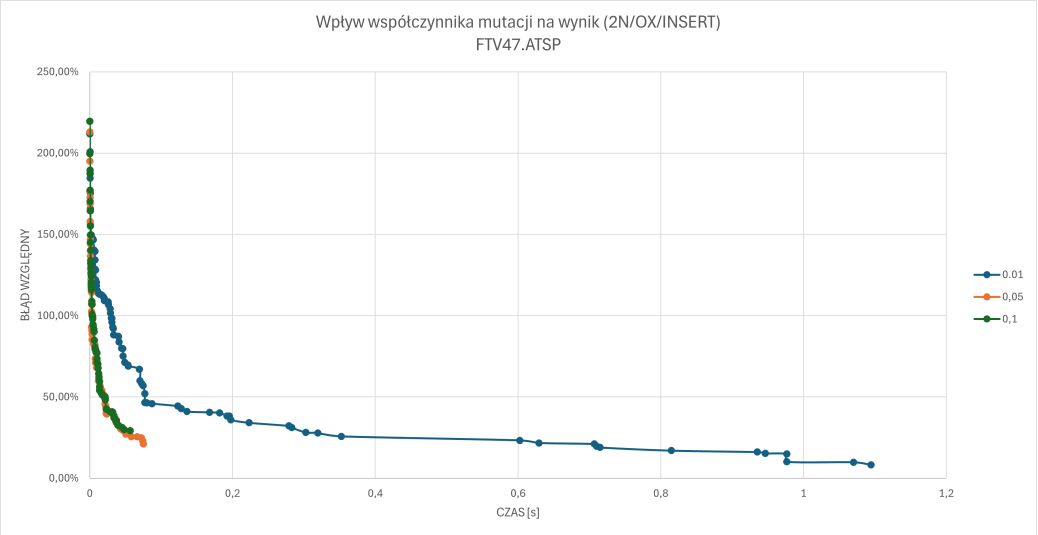
Do badań wybrano 3 wartości współczynnika mutacji:

- **0.01**
- **0.05**
- **0.10**

Pozostałe wartości:

- współczynnik krzyżowania = **0.8**
- wielkość populacji = **2N**
- metoda krzyżowania = **OX**
- metoda mutacji = **INSERT**

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Współczynnik mutacji
1776	1923	8,28%	1,09	94	0.01
1776	2151	21,11%	0,08	94	0.05
1776	2295	29,22%	0,06	94	0.1



Rysunek 3: 3 współczynniki mutacji - ftv47

4.1.3 Wpływ współczynnika krzyżowania na wyniki

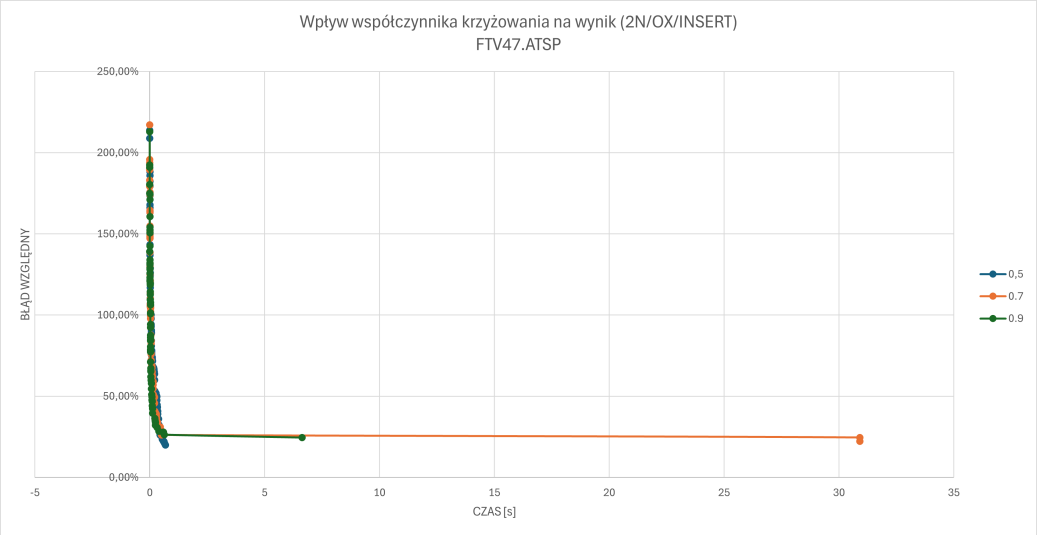
Do badań wybrano 3 wartości współczynnika krzyżowania:

- **0.50**
- **0.70**
- **0.90**

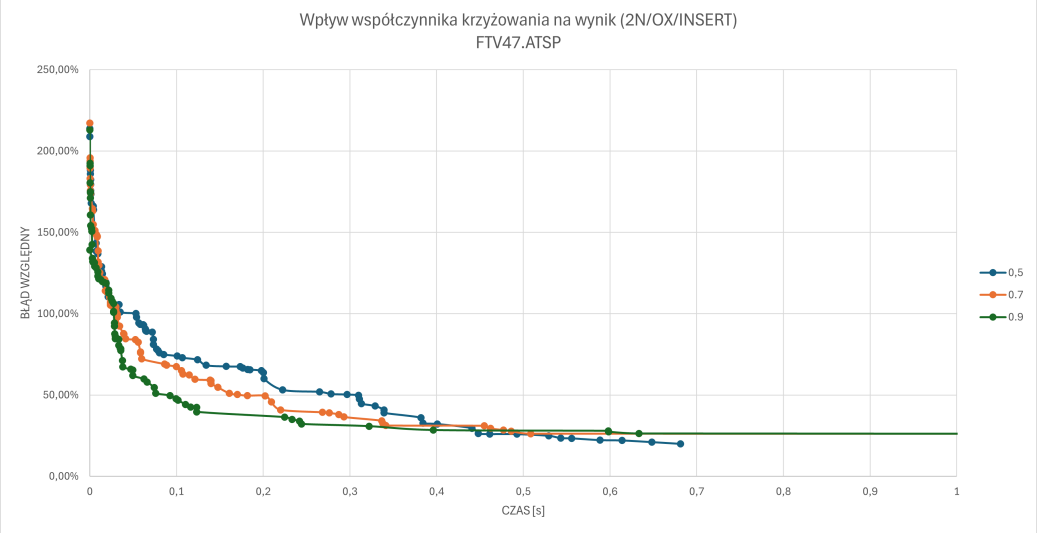
Pozostałe wartości:

- współczynnik mutacji = **0.01**
- wielkość populacji = **2N**
- metoda krzyżowania = **OX**
- metoda mutacji = **INSERT**

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Współczynnik krzyżowania
1776	2130	19,93%	0,68	94	0.5
1776	2171	22,24%	30,90	94	0.7
1776	2211	24,49%	6,63	94	0.9



Rysunek 4: 3 współczynniki krzyżowania 1/2 - ftv47



Rysunek 5: 3 współczynniki krzyżowania 2/2 - ftv47

4.2 Plik FTV170.atsp

4.2.1 Wpływ wielkości populacji na wyniki

Do badań wybrano 3 wartości populacji:

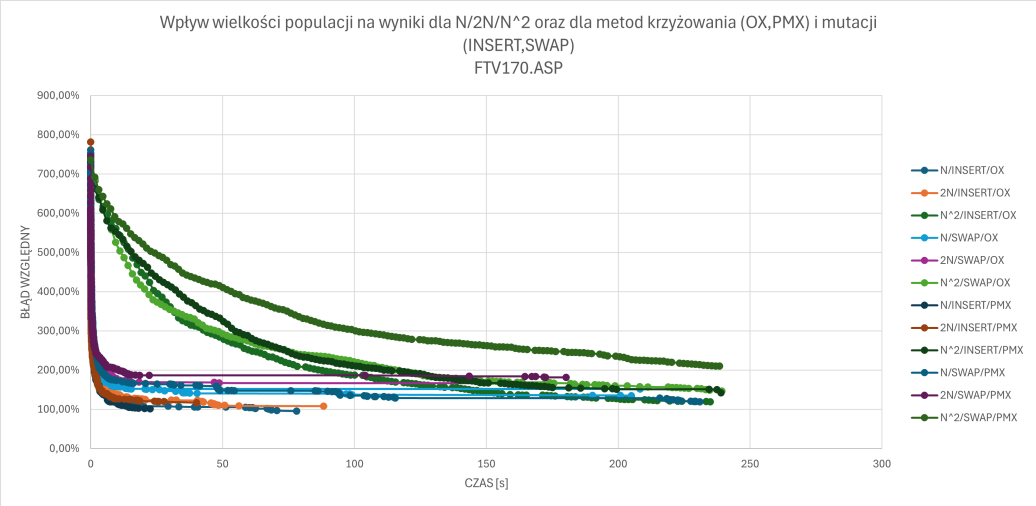
- **N**
- **2N**
- N^2, gdzie **N** oznacz liczbe miast w danym pliku.

Przyjęte współczynniki krzyżowania:

- współczynnik mutacji = **0.01**
- współczynnik krzyżowania = **0.8**

Wykorzystano 2 metody krzyżowania (**PMX**, **OX**) oraz 2 metody mutacji (**INSERT**, **SWAP**).

Znane optimum	**Znalezione rozwiązanie**	**Błąd względny**	**Czas znalezienia rozwiązania [s]**	**Liczba populacji**	**Mutacja**	**Krzyżowanie**
2755	5391	95,68%	78,08	170	INSERT	OX
2755	5739	108,31%	88,33	340	INSERT	OX
2755	6034	119,02%	234,90	28900	INSERT	OX
2755	6459	134,45%	204,95	170	SWAP	OX
2755	7344	166,57%	163,71	340	SWAP	OX
2755	6788	146,39%	239,26	28900	SWAP	OX
2755	5545	101,27%	22,54	170	INSERT	PMX
2755	5985	117,24%	40,08	340	INSERT	PMX
2755	6665	141,92%	238,95	28900	INSERT	PMX
2755	6041	119,27%	230,89	170	SWAP	PMX
2755	7751	181,34%	180,26	340	SWAP	PMX
2755	8543	210,09%	238,46	28900	SWAP	PMX



Rysunek 6: 12 krzywych/Wszystkie kominacje - ftv170

Najlepszy wynik uzyskano dla:

- populacji - **N**
- krzyżowania - **OX**
- mutacji - **INSERT**

4.2.2 Wpływ współczynnika mutacji na wyniki

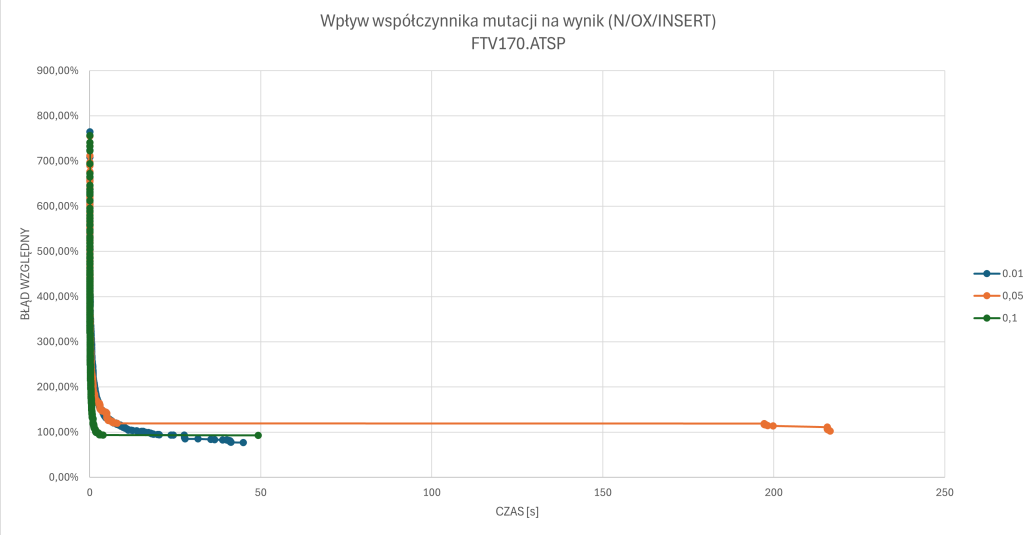
Do badań wybrano 3 wartości współczynnika mutacji:

- **0.01**
- **0.05**
- **0.10**

Pozostałe wartości:

- współczynnik krzyżowania = **0.8**
- wielkość populacji = **N**
- metoda krzyżowania = **OX**
- metoda mutacji = **INSERT**

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Współczynnik mutacji	Mutacja	Krzyżowanie	Współczynnik krzyżowania
2755	4876	76,99%	44,92	170	0.01	INSERT	OX	0.8
2755	5583	102,65%	216,47	170	0.05	INSERT	OX	0.8
2755	5319	93,07%	49,28	170	0.1	INSERT	OX	0.8



Rysunek 7: 3 współczynniki mutacji - ftv170

4.2.3 Wpływ współczynnika krzyżowania na wyniki

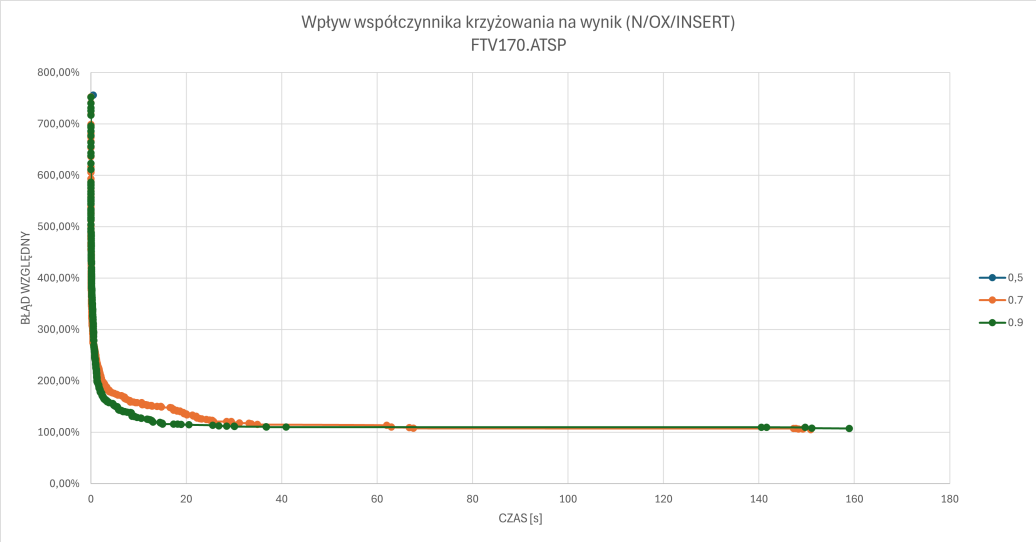
Do badań wybrano 3 wartości współczynnika krzyżowania:

- **0.50**
- **0.70**
- **0.90**

Pozostałe wartości:

- współczynnik mutacji = **0.01**
- wielkość populacji = **N**
- metoda krzyżowania = **OX**
- metoda mutacji = **INSERT**

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Współczynnik mutacji	Mutacja	Krzyżowanie	Współczynnik krzyżowania
2755	5091	84,79%	39,86	170	0.01	INSERT	OX	0.5
2755	5660	105,44%	150,87	170	0.01	INSERT	OX	0.7
2755	5717	107,51%	158,93	170	0.01	INSERT	OX	0.9



Rysunek 8: 3 współczynniki krzyżowania - ftv170

4.2.4 Porównanie najlepszego wyniku z metodą Tabu Search oraz Simulated Annealing

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Algorytm
2755	4876	76,99%	44,92	Genetic algorithm
2755	3680	33,58%	0,001	Simulated Annealing
2755	3350	21,59%	156,948	Tabu Search

4.3 Plik RBG403.atsp

4.3.1 Wpływ wielkości populacji na wyniki

Do badań wybrano 3 wartości populacji:

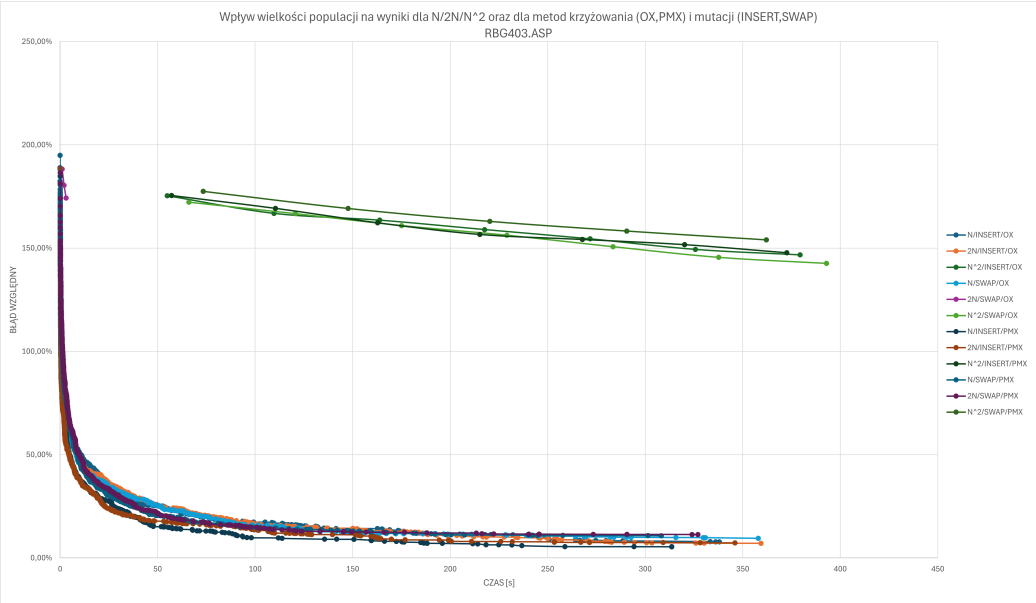
- **N**
- **2N**
- N^2, gdzie **N** oznacz liczbe miast w danym pliku.

Przyjęte współczynniki krzyżowania:

- współczynnik mutacji = **0.01**
- współczynnik krzyżowania = **0.8**

Wykorzystano 2 metody krzyżowania (**PMX**, **OX**) oraz 2 metody mutacji (**INSERT**, **SWAP**).

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Mutacja	Krzyżowanie
2465	2657	7,79%	337,99	403	INSERT	OX
2465	2639	7,06%	359,35	806	INSERT	OX
2465	6082	146,73%	379,45	162409	INSERT	OX
2465	2698	9,45%	357,91	403	SWAP	OX
2465	2737	11,03%	352,18	806	SWAP	OX
2465	5980	142,60%	392,95	162409	SWAP	OX
2465	2598	5,40%	313,58	403	INSERT	PMX
2465	2645	7,30%	345,97	806	INSERT	PMX
2465	6106	147,71%	372,63	162409	INSERT	PMX
2465	2730	10,75%	308,19	403	SWAP	PMX
2465	2744	11,32%	327,06	806	SWAP	PMX
2465	6261	154,00%	362,10	162409	SWAP	PMX



Rysunek 9: 12 krzywych/Wszystkie kominacje - rbg403

Najlepszy wynik uzyskano dla:

- populacji - **N**
- krzyżowania - **PMX**
- mutacji - **INSERT**

4.3.2 Wpływ współczynnika mutacji na wyniki

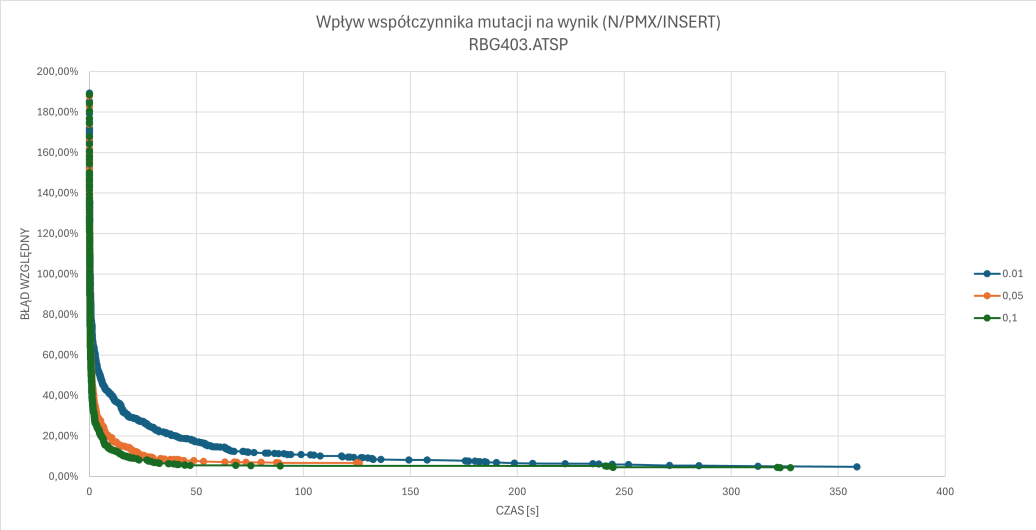
Do badań wybrano 3 wartości współczynnika mutacji:

- **0.01**
- **0.05**
- **0.10**

Pozostałe wartości:

- współczynnik krzyżowania = **0.8**
- wielkość populacji = **N**
- metoda krzyżowania = **PMX**
- metoda mutacji = **INSERT**

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Współczynnik mutacji	Mutacja	Krzyżowanie	Współczynnik krzyżowania
2465	2583	4,79%	358,68	403	0.01	INSERT	PMX	0.8
2465	2629	6,65%	126,11	403	0.05	INSERT	PMX	0.8
2465	2571	4,30%	327,68	403	0.1	INSERT	PMX	0.8



Rysunek 10: 3 współczynniki mutacji - rbg403

4.3.3 Wpływ współczynnika krzyżowania na wyniki

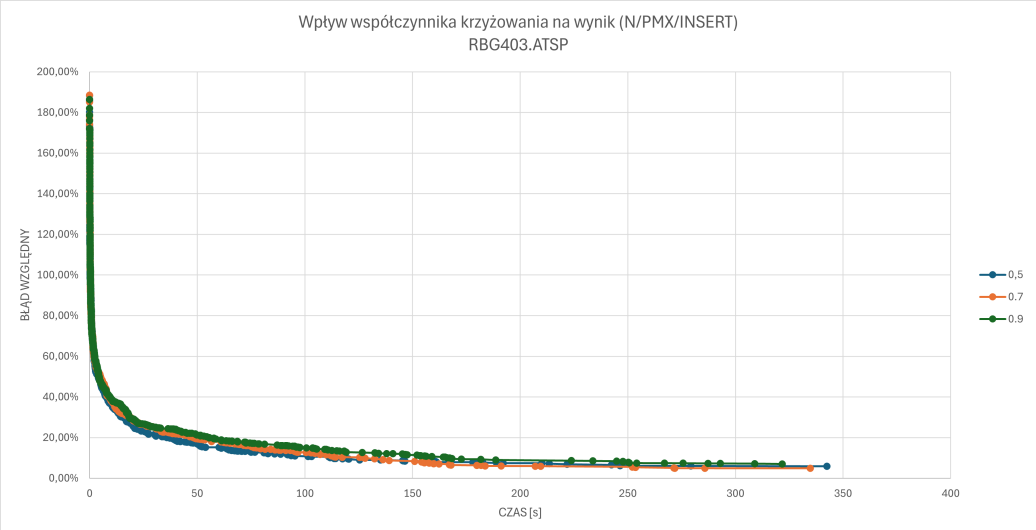
Do badań wybrano 3 wartości współczynnika krzyżowania:

- **0.50**
- **0.70**
- **0.90**

Pozostałe wartości:

- współczynnik mutacji = **0.01**
- wielkość populacji = **N**
- metoda krzyżowania = **PMX**
- metoda mutacji = **INSERT**

Znane optimum	Znalezione rozwiązanie	Błąd względny	Czas znalezienia rozwiązania [s]	Liczba populacji	Współczynnik mutacji	Mutacja	Krzyżowanie	Współczynnik krzyżowania
2465	2610	5,88%	342,47	403	0.01	INSERT	PMX	0.5
2465	2584	4,83%	334,77	403	0.01	INSERT	PMX	0.7
2465	2638	7,02%	321,67	403	0.01	INSERT	PMX	0.9



Rysunek 11: 3 współczynniki krzyżowania - rbg403

5 Wnioski

Analizując wszystkie wykresy oraz wyniki, dochodzę do wniosków:

- Algorytm genetyczny wykazuje dość dobre wynik, jak na to że jego początkowa populacja jest całkowicie losowa, bez żadnych ścieżek zachłannych.
- Wypadł gorzej od poprzednich algorytmów, lecz prawodpobodobnie jest to spowodowane następującymi rzeczami:
  - zbyt duży rozmiar turniejów - powoduje to że słabsze osobniki (o niskiej wartości funkcji przystosowania), mają małe szanse na wejście do puli rodzicielskiej, czego skutkiem jest mała eksploracja przestrzeni rozwiązań, oraz mała możliwość wyjścia z minimum lokalnego.

    Lepszym rozwiązaniem w przypadku tych plików, byłby wybór rozmiaru turnieju w okolicach 5-20 osobników.
  - sukcesja z reprodukcją częściową - za każdym razem przechodzi 10% najlepszych osobników ze starej populacji. Powoduje to elitaryzm i jeszcze większą dominację nad słabszymi osobnikami, przez co algorytm mógł nie wychodzić z minimów lokalnych.
- Biorąc pod uwagę popełnione błędy i wyciągnięte wnioski, dochodzę do wniosku, że z lekkimi poprawkami algorytm mógłby pokonać poprzednie algorytmy.