# Sprawozdanie – Projektowanie Efektywnych Algorytmów – Projekt

*Projekt nr 3 – Implementacja i analiza efektywności algorytmu genetycznego (ewolucyjnego) dla problemu komiwojażera*



| | |
|---|---|
| **Autor:** | Adam Czekalski |
| **Nr indeksu:** | 264488 |
| **Termin:** | Piątek, godz. 13:15 – 15:00 |
| **Prowadzący:** | dr inż. Jarosław Mierzwa |
| **Data oddania:** | 26.01.2024 r. |

# 1. Wstęp teoretyczny

## 1.1. Opis ogólny algorytmu

Nazwa i działanie algorytmu genetycznego nawiązuje do zjawiska ewolucji biologicznej. Dana jest populacja o określonej wielkości. Każdy z osobników ma przypisany do siebie chromosom, który składa się z genów, oraz wartość funkcji przystosowania. Osobniki podlegają procesie selekcji (oceny). Następnie wybrane osobniki podlegają procesie reprodukcji – mogą się skrzyżować pod pewnym prawdopodobieństwem a następnie zmutować pod pewnym prawdopodobieństwem. Powstaje SUBpopulacja, która podlega ocenie. Na koniec uruchamiany jest proces sukcesji – najgorzej przystosowane do środowiska wymierają, najsilniejsze natomiast przetrwają. Rodzi się nowa populacja. Proces powtarza się.

Dla problemu komiwojażera wygląda to następująco: na początku zostaje wygenerowana losowa populacja o określonej wielkości. Dla każdego z osobników, chromosom jest reprezentowany przez ciąg wierzchołków (genów) reprezentujących cykl Hamiltona. Natomiast wartość funkcji przystosowania jest reprezentowana przez długość ścieżki.

## 1.2. Opis użytych rozwiązań w algorytmie

- Selekcja

Użyto selekcji rankingowej, polegającej na tym, że populacja sortowana jest od najlepszej (najmniejszej) wartości funkcji przystosowania do najgorszej (największej) wartości funkcji przystosowania. Następnie *n* najlepiej przystosowanych osobników przechodzi do procesu reprodukcji (reszta zostaje usunięta z populacji, elityzm), gdzie:

$$n = 0{,}25 * initialPopulationSize$$

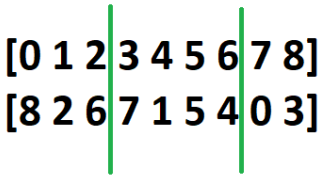
- Krzyżowanie

Losowane są 2 różne losowe osobniki z pozostałej populacji. Następnie krzyżowanie zachodzi pod prawdopodobieństwem $p_k = 0.8$ . Użyto metody OX, która przebiega następująco:

Np. dane są 2 chromosomy (ścieżki): track1={0,1,2,3,4,5,6,7,8}, track2={8,2,6,7,1,5,4,0,3}

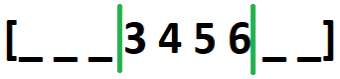
firstCrossoverPoint = 3 (wylosowany z przedziału [0; track1.size())

secondCrossoverPoint = 6 (wylosowany z przedziału [firstCrossoverPoint+1; track1.size())

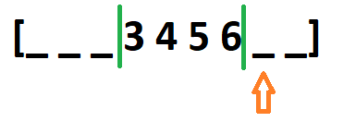


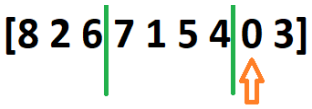

Dziecko wypełniane jest wartościami „-1”.

Na początku do potomka kopiowany jest segment z rodzica nr 1:

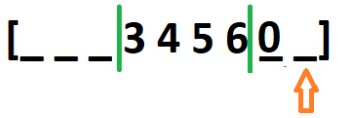



Następnie rozpoczynając z miejsca *secondCrossoverPoint* + *1* w potomku i rodzicu nr 2:

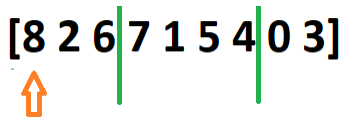



Sprawdzamy czy kolejne miasta z rodzica nr 2 występują w potomku. Robimy to do momentu dojścia do końca ścieżki w rodzicu nr 2. 0 nie występuje w potomku, więc zostaje wstawione; 3 już występuje, więc nie zostanie wstawione:

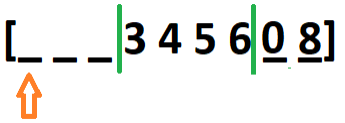



W rodzicu nr 2 doszliśmy już do końca ścieżki, więc wracamy do jej początku:

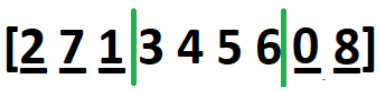



Iterujemy przez nią aż do *secondCrossoverPoint* włącznie.

8 nie występuje w potomku, więc je wstawiamy. W potomku doszliśmy do końca ścieżki, więc wracamy na jego początek:




2 także wstawiamy; 6 nie wstawiamy, ponieważ już występuje w ścieżce; 7 wstawiamy, ponieważ nie występuje w ścieżce; 1 także. Finalna ścieżka w dziecku wygląda następująco:




- Mutacje

Zachodzą z prawdopodobieństwem $p_m = 0.01$ .

  - Transposition mutation

    Ta mutacja opiera się na zamianie 2 losowo wybranych miast w ścieżce:

    randomTownIndex = 2

randomTownIndex = 6

[1 2 3 4 5 6 7 8 9]

Wywołujemy metodę swap() na danych miastach. W efekcie:

[1 2 7 4 5 6 3 8 9]

- Inversion mutation

Ta mutacja opiera się na wybraniu 2 losowych punktów w ciągu miast i odwróceniu ich kolejności

randomTownIndex1 = 2

randomTownIndex2 = 6

[1 2 3 4 5 6 7 8 9]

W tym celu odkładamy wszystkie miasta z wylosowanego przedziału na stos. Po czym w tym przedziale podmieniamy wartości biorąc je ze stosu. W efekcie:

[1 2 7 6 5 4 3 8 9]

# 2. Opis programu – opis najważniejszych klas w projekcie

Klasa GeneticAlgorithm:

Pola klasy:

- int stopCriteria – zmienna przechowująca kryterium stopu (czas określony w sekundach)
- int initialPopulationSize – zmienna przechowująca początkową wielkość populacji
- double mutationRate – zmienna przechowująca wartość współczynnika mutacji
- int mutationMethod – zmienna przechowująca wybrany numer metody mutacji (1 – transposition mutation, 2 – inversion mutation)
- double crossOverRate – zmienna przechowująca wartość współczynnika krzyżowania
- int crossOverMethod – zmienna przechowująca wybrany numer metody krzyżowania (1 – OX)
- std::vector<std::pair<int, std::vector<int>>> population – wektor przechowujący aktualną populację. Przechowywana jest w postaci: <wartość funkcji przystosowania, ścieżka>

- std::vector<std::pair<int, std::vector<int>>> children – wektor przechowujący nowo powstającą populację. Przechowywana jest w tej samej postaci co w wektorze aktualnej populacji
- double whenFound – zmienna przechowująca czas w sekundach, kiedy zostało znalezione najlepsze rozwiązanie
- std::vector<int> bestSolution – wektor przechowujący ciąg wierzchołków ścieżki najlepszego znalezionego rozwiązania
- int bestObjectiveFunction – zmienna przechowująca najlepszą znalezioną wartość funkcji przystosowania
- std::list<std::pair<double,int>> save – lista wykorzystywana w trakcie testów – służy do zapisu każdej poprawy ścieżki w postaci (czas znalezienia w sekundach, wartość funkcji przystosowania)
- metoda tworząca losowe rozwiązanie początkowe:

```
std::vector<int> GeneticAlgorithm::generateBegSolutionRandom() {
    std::vector<int> track;
    std::vector<std::pair<bool, int>> visited; //vector to help to decide if town was already visited

    for (int i = 0; i < matrix->nrV; ++i) { //filling the vector with false
        visited.push_back(std::make_pair( x: false,  &: i));
    }

    int x;

    for (int i = 0; i < matrix->nrV; ++i) {
        x = rand() % (matrix->nrV); //get random number

        while (visited[x].first) { //if it exists in track1, generate once again
            x = rand() % (matrix->nrV);
        }

        track.push_back(x);
        visited[x].first = true; //we visit every node once

    }

    return track;
}
```

- metoda wykonująca krzyżowanie typu OX (Order Crossover)

```
std::vector<int> GeneticAlgorithm::OX(std::vector<int> track1, std::vector<int> track2) {
    std::vector<int> child;

    //1. two random cross-breading points [x1,x2]
    firstCrossoverPoint = rand() % (track1.size() - 1);
    secondCrossoverPoint = firstCrossoverPoint + 1 + rand() % (track1.size() - firstCrossoverPoint -
                                                               1);
    //fill the child with -1 value
    child.assign( n: track1.size(),  val: -1);

    //vector with visited nodes
    std::vector<bool> visited;
    visited.assign( n: track1.size(),  x: false);

    //copy towns from random range from parent1 to child
    for (int i = firstCrossoverPoint; i <= secondCrossoverPoint; i++) {
        child[i] = track1[i];
        //it becomes visited
        visited[track1[i]] = true;
    }

    //index in parent2
    int begin;

    //if second crossover point is at the end of the path, we will begin the comparison from index 0 of parent 2
    if(secondCrossoverPoint == track1.size() - 1){
        begin = 0;
    }
    //otherwise, we just take index one after second crossover point
    else{
        begin = secondCrossoverPoint + 1;
    }

    //j <- index in child
    int j = begin;
```

```
    //in range [begin, track1.size()-1]
    for (int i = begin; i < track1.size(); i++) {
        //if it's not visited, we put it in path
        if (!visited[track2[i]]) {
            child[j] = track2[i];
            visited[track2[i]] = true;
            j++;
        }
    }

    //if we reached last index in child, we must change it to 0
    if(j == track1.size()){
        j = 0;
    }

    //in range [0, secondCrossoverPoint]
    for (int i = 0; i <= secondCrossoverPoint; i++) {
        if (!visited[track2[i]]) {
            child[j] = track2[i];
            visited[track2[i]] = true;
            j++;
            //if we reached last index in child, we must change it to 0
            if(j == track1.size()){
                j=0;
            }
        }
    }

    return child;


}
```

- metoda wykonująca mutację typu transposition:

```
void GeneticAlgorithm::transpositionMutation(std::vector<int> &track) {

    //mutation
    randomTownIndex1 = rand() % (track.size()); //1. random town
    randomTownIndex2 = rand() % (track.size()); //2. random town

    while (randomTownIndex1 == randomTownIndex2) { //it has to be 2 different towns
        randomTownIndex2 = rand() % (track.size()); //2. random town
    }

    std::swap( &track[randomTownIndex1], &track[randomTownIndex2]); //swap

}
```

- metoda wykonująca mutację typu inversion:

```cpp
void GeneticAlgorithm::inversionMutation(std::vector<int> &track) {

    //mutation
    randomTownIndex1 = rand() % (track.size() - 1); //we have to have at least 2 elements to invert
    randomTownIndex2 = randomTownIndex1 + 1 + rand() % (track.size() - randomTownIndex1 -
                                                        1); //2. random town from range (randomTownIndex1; track.size())

                                                        //put numbers from random range on stack
    for (int j = randomTownIndex1; j <= randomTownIndex2; j++) {
        substring.emplace( & track[j]);
    }

    //get numbers from stack in the same range
    for (int j = randomTownIndex1; j <= randomTownIndex2; j++) {
        track[j] = substring.top();
        substring.pop();
    }

}
```

Klasa Timer:

Czas mierzony jest w mikrosekundach za pomocą klasy Timer, za pomocą biblioteki <windows.h>. Po wywołaniu metody startTimer() zostaje odczytana zostaje liczba „tiknięć” od ostatniego restartu systemu. Następnie po wywołaniu metody stopTimer(), znowu odczytana zostaje liczba „tiknięć” a następnie zostaje ona odjęta od wartości odczytanej z wywołaniem metody startTimer() i podzielona przez wartość frequency = 10MHz.

```cpp
#include "Timer.h"

long long int Timer::readQPC(){
    LARGE_INTEGER count;
    QueryPerformanceCounter( lpPerformanceCount: &count); //read number of ticks since the last restart of the system
    return((long long int)count.QuadPart); //64b integer
}

void Timer::startTimer(){
    QueryPerformanceFrequency( lpFrequency: (LARGE_INTEGER*)&frequency); //10MHz
    start = readQPC(); //number of ticks since the last restart of the system
}

long Timer::stopTimer(){
    elapsed = readQPC() - start; //read ticks once again and subtract from read ticks at the start

    long time = (1000000.0 * elapsed) / frequency;
    return time;
}

void Timer::printTime(){
    std::cout << "Time [s] = " << std::fixed << std::setprecision( n: 6) << (float)elapsed / frequency << std::endl;
    std::cout << "Time [ms] = " << std::fixed << std::setprecision( n: 4) << (1000.0 * elapsed) / frequency << std::endl;
    std::cout << "Time [us] = " << std::fixed << std::setprecision( n: 1) << (1000000.0 * elapsed) / frequency << std::endl;
    std::cout << "Time [ns] = " << std::fixed << std::setprecision( n: 0) << (1000000000.0 * elapsed) / frequency << std::endl << std::endl;
}

Timer::Timer() {

}
```

Klasa FileWriter:

Po każdym teście zapisuje każdą poprawę ścieżki i czas znalezienia do pliku w formacie: „nazwaPlikuZGrafem_rozmiarPoczątkowejPopulacji_metodaMutacji_nrTestu.txt”, np. „ftv170.atsp_2500_1_1.txt”. Zapisuje także najlepszą znalezioną ścieżkę z danego testu do pliku w formacie: „nazwaPlikuZGrafem_rozmiarPoczątkowejPopulacji_metodaMutacji_nrTestu_path.txt”, np. „ftv170.atsp_2500_1_1_path.txt.

```
void FileWriter::write(GeneticAlgorithm* geneticAlgorithm, int numberOfTest){

    //saving every improvement of path and when it was improved to file in format: "ftv170.atsp_2500_1_1.txt" (filename_populationSize_mutationMethod_nrOfTest.txt)
    std::string name = filename + "_" + std::to_string( val initialPopulationSize) + "_" +
            std::to_string( val mutationMethod) + "_" + std::to_string( val numberOfTest) + ".txt";
    std::ofstream file( s name);

    //<time, path> in every line
    for(auto entry : pair<double,int> : geneticAlgorithm->save){
        file << entry.first << " " << entry.second << std::endl;
    }

    file.close();

    //if that's the best found route in test, we save that info
    if(geneticAlgorithm->bestObjectiveFunction < bestObjectiveFunction){
        bestObjectiveFunction = geneticAlgorithm->bestObjectiveFunction;
        filenameOfBestSolution = name;
    }

    //saving path in format: "ftv170.atsp_2500_1_1_path.txt"
    name = filename + "_" + std::to_string( val initialPopulationSize) + "_" + std::to_string( val mutationMethod) + "_" +
            std::to_string( val numberOfTest) + "_path" + ".txt";
    std::ofstream file1( s name);

    //size of graph
    file1 << geneticAlgorithm->bestSolution.size() << std::endl;

    //path
    for(int town : geneticAlgorithm->bestSolution){
        file1 << town << " ";
    }

    file1.close();

    //if it's last test, we save where the best solution has been found
    if(numberOfTest == 9){
        name = filename + "_" + std::to_string( val initialPopulationSize) + "_" + std::to_string( val mutationMethod) + "_best" + ".txt";
        std::ofstream file2( s name);
        file2<<filenameOfBestSolution;
        file2.close();
    }
```

# 3. Dane w postaci tabel i wykresów

Wykonano testy dla następujących rozmiarów populacji: . Dla każdego rozmiaru populacji, pliku z grafem oraz sposobu mutacji uruchomiono 5 testów. Zawsze przyjmowano współczynnik mutacji $p_m = 0.01$ oraz współczynnik krzyżowania $p_k = 0.8$ .

## 3.1. Graf ftv47.atsp

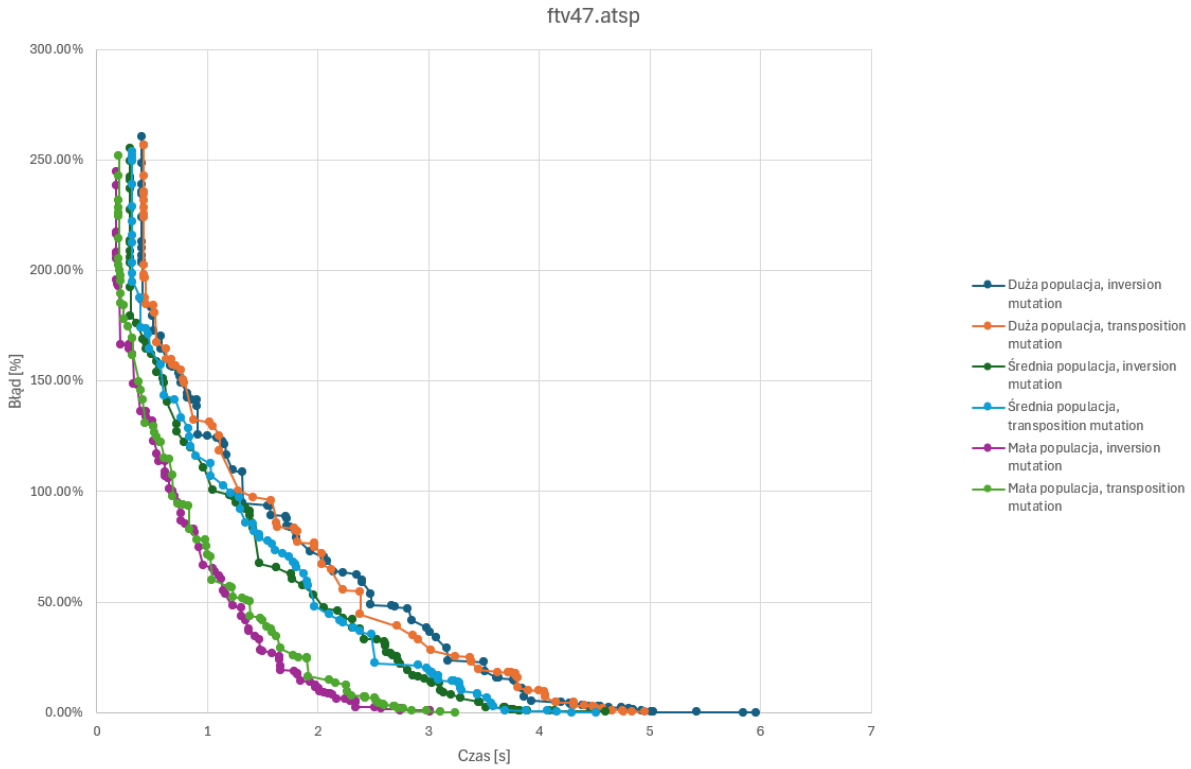
Jako kryterium stopu przyjęto 2 minuty.

| Wielkość instancji | Rodzaj mutacji | Nr pomiaru | Czas znalezienia | Wynik |
|---|---|---|---|---|
| 50000 | transposition | 1 | 2.52127 | 1782 |
| 50000 | transposition | 2 | 2.52127 | 1784 |
| 50000 | transposition | 3 | 3.07182 | 1862 |

| | | | | |
|---|---|---|---|---|
| 50000 | transposition | 4 | 3.24279 | 1777 |
| 50000 | transposition | 5 | 2.67337 | 1875 |
| 50000 | inversion | 1 | 3.01038 | 1782 |
| 50000 | inversion | 2 | 2.98912 | 1784 |
| 50000 | inversion | 3 | 2.7581 | 1789 |
| 50000 | inversion | 4 | 3.00696 | 1796 |
| 50000 | inversion | 5 | 4.51584 | 1777 |
| 75000 | transposition | 1 | 4.04603 | 1782 |
| 75000 | transposition | 2 | 4.71015 | 1789 |
| 75000 | transposition | 3 | 4.9678 | 1790 |
| 75000 | transposition | 4 | 6.49792 | 1842 |
| 75000 | transposition | 5 | 6.02432 | 1784 |
| 75000 | inversion | 1 | 4.59643 | 1784 |
| 75000 | inversion | 2 | 4.84858 | 1791 |
| 75000 | inversion | 3 | 4.64307 | 1855 |
| 75000 | inversion | 4 | 5.41941 | 1790 |
| 75000 | inversion | 5 | 4.48546 | 1789 |
| 100000 | transposition | 1 | 4.95766 | 1782 |
| 100000 | transposition | 2 | 5.57118 | 1826 |
| 100000 | transposition | 3 | 5.24899 | 1816 |
| 100000 | transposition | 4 | 8.48607 | 1803 |
| 100000 | transposition | 5 | 5.60842 | 1799 |
| 100000 | inversion | 1 | 5.36096 | 1789 |
| 100000 | inversion | 2 | 5.29906 | 1800 |
| 100000 | inversion | 3 | 4.99819 | 1789 |
| 100000 | inversion | 4 | 5.21359 | 1789 |
| 100000 | inversion | 5 | 5.95924 | 1776 |

Najlepsze wyniki:

| Wielkość instancji | Sposób krzyżowania | Sposób mutacji | Wartość funkcji przystosowania | Błąd |
|---|---|---|---|---|
| 50000 | OX | transposition | 1777 | 0.06% |
| 75000 | OX | transposition | 1777 | 0.06% |
| 100000 | OX | transposition | 1782 | 0.34% |
| 50000 | OX | inversion | 1782 | 0.34% |
| 75000 | OX | inversion | 1784 | 0.45% |
| 100000 | OX | inversion | 1776 | 0.00% |



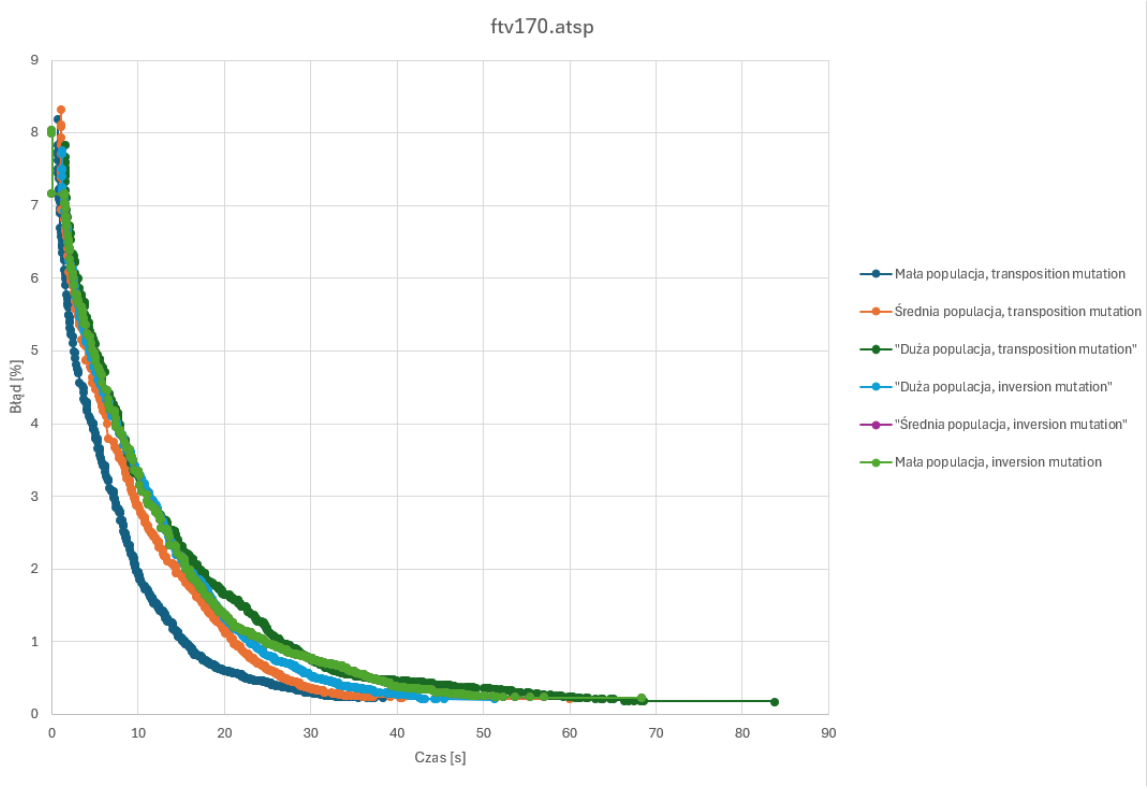

## 3.2. Graf ftv170.atsp

Jako kryterium stopu przyjęto 4 minuty.

| Wielkość instancji | Rodzaj mutacji | Nr pomiaru | Czas znalezienia | Wynik |
|---|---|---|---|---|
| 50000 | transposition | 1 | 47.8481 | 3795 |
| 50000 | transposition | 2 | 38.5218 | 3584 |
| 50000 | transposition | 3 | 52.4212 | 3327 |
| 50000 | transposition | 4 | 38.9035 | 3438 |
| 50000 | transposition | 5 | 26.9782 | 3354 |
| 50000 | inversion | 1 | 22.4215 | 3769 |
| 50000 | inversion | 2 | 41.2585 | 3847 |
| 50000 | inversion | 3 | 35.2858 | 3408 |
| 50000 | inversion | 4 | 26.6795 | 4043 |
| 50000 | inversion | 5 | 36.1508 | 3634 |

| | | | | |
|---|---|---|---|---|
| 75000 | transposition | 1 | 48.2823 | 3684 |
| 75000 | transposition | 2 | 59.9892 | 3678 |
| 75000 | transposition | 3 | 60.0426 | 3335 |
| 75000 | transposition | 4 | 55.7297 | 3650 |
| 75000 | transposition | 5 | 82.2517 | 3540 |
| 75000 | inversion | 1 | 68.2729 | 3610 |
| 75000 | inversion | 2 | 68.3282 | 3388 |
| 75000 | inversion | 3 | 46.2025 | 3617 |
| 75000 | inversion | 4 | 56.5003 | 3497 |
| 75000 | inversion | 5 | 51.0411 | 3489 |
| 100000 | transposition | 1 | 62.0472 | 3669 |
| 100000 | transposition | 2 | 113.805 | 3656 |
| 100000 | transposition | 3 | 71.8887 | 3646 |
| 100000 | transposition | 4 | 107.71 | 3873 |
| 100000 | transposition | 5 | 83.7614 | 3248 |
| 100000 | inversion | 1 | 51.2049 | 3316 |
| 100000 | inversion | 2 | 68.6252 | 3660 |
| 100000 | inversion | 3 | 54.8819 | 3665 |
| 100000 | inversion | 4 | 63.204 | 3414 |
| 100000 | inversion | 5 | 102.11 | 3473 |

| Wielkość instancji | Sposób krzyżowania | Sposób mutacji | Wartość funkcji przystosowania | Błąd |
|---|---|---|---|---|
| 50000 | OX | transposition | 3355 | 21.78% |
| 75000 | OX | transposition | 3335 | 21.05% |
| 100000 | OX | transposition | 3248 | 17.89% |
| 50000 | OX | inversion | 3408 | 23.70% |
| 75000 | OX | inversion | 3388 | 22.98% |
| 100000 | OX | inversion | 3316 | 20.36% |



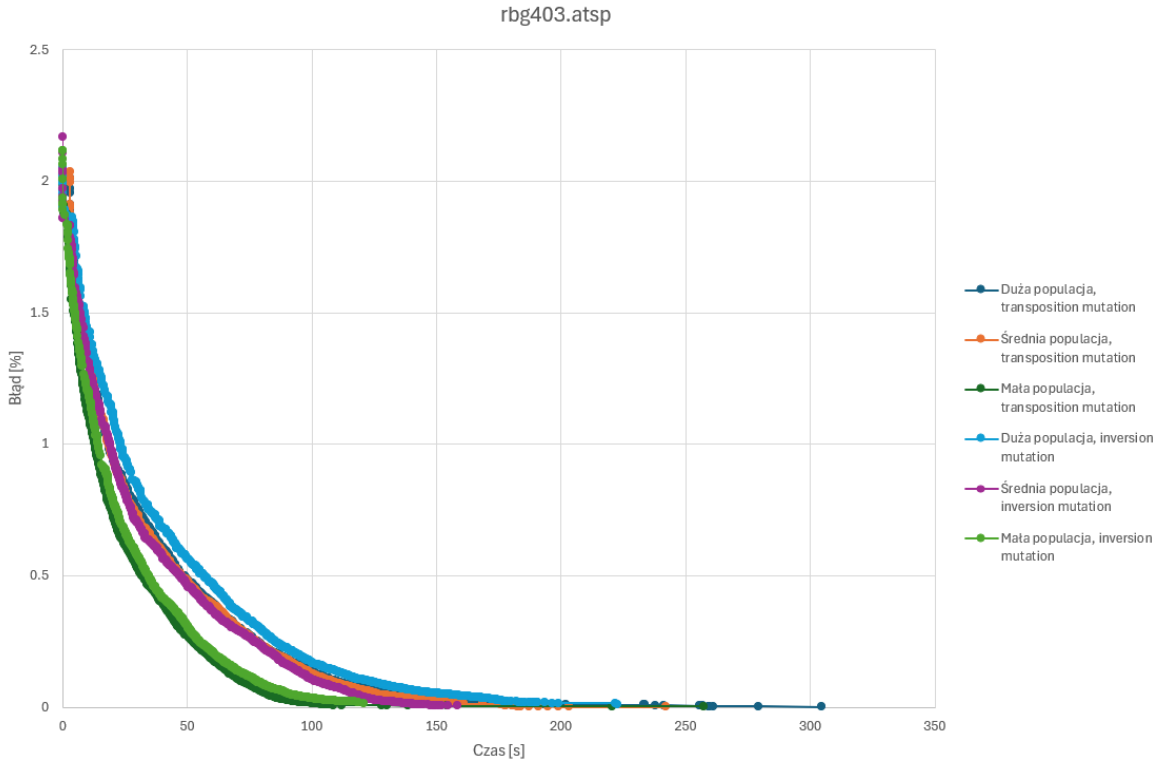

## 3.3. Graf rbg403.atsp

Jako kryterium stopu przyjęto 6 minut.

| Wielkość instancji | Rodzaj mutacji | Nr pomiaru | Czas znalezienia | Wynik |
|---|---|---|---|---|
| 50000 | transposition | 1 | 282.55 | 2470 |
| 50000 | transposition | 2 | 295.362 | 2501 |
| 50000 | transposition | 3 | 298.075 | 2467 |
| 50000 | transposition | 4 | 192.065 | 2485 |
| 50000 | transposition | 5 | 257.264 | 2474 |
| 50000 | inversion | 1 | 106.496 | 2527 |
| 50000 | inversion | 2 | 120.74 | 2504 |
| 50000 | inversion | 3 | 107.41 | 2514 |
| 50000 | inversion | 4 | 132.684 | 2505 |
| 50000 | inversion | 5 | 107.717 | 2517 |
| 75000 | transposition | 1 | 241.926 | 2466 |

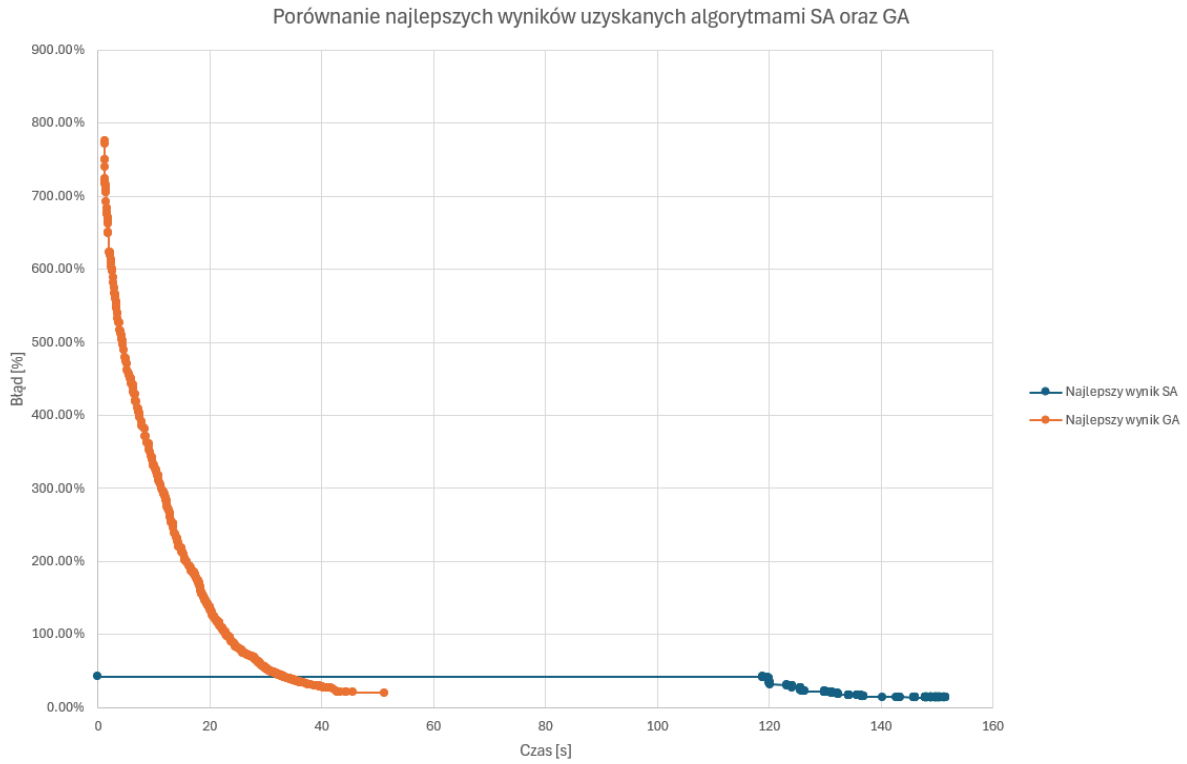
| | | | | |
|---|---|---|---|---|
| 75000 | transposition | 2 | 341.419 | 2474 |
| 75000 | transposition | 3 | 307.801 | 2485 |
| 75000 | transposition | 4 | 359.767 | 2493 |
| 75000 | transposition | 5 | 246.158 | 2478 |
| 75000 | inversion | 1 | 171.977 | 2485 |
| 75000 | inversion | 2 | 173.121 | 2515 |
| 75000 | inversion | 3 | 158.57 | 2478 |
| 75000 | inversion | 4 | 196.159 | 2483 |
| 75000 | inversion | 5 | 165.555 | 2512 |
| 100000 | transposition | 1 | 304.539 | 2469 |
| 100000 | transposition | 2 | 299.236 | 2481 |
| 100000 | transposition | 3 | 295.866 | 2479 |
| 100000 | transposition | 4 | 321.828 | 2470 |
| 100000 | transposition | 5 | 269.971 | 2488 |
| 100000 | inversion | 1 | 211.063 | 2533 |
| 100000 | inversion | 2 | 170.099 | 2504 |
| 100000 | inversion | 3 | 222.633 | 2492 |
| 100000 | inversion | 4 | 215.382 | 2494 |
| 100000 | inversion | 5 | 191.908 | 2502 |

| Wielkość instancji | Sposób krzyżowania | Sposób mutacji | Wartość funkcji przystosowania | Błąd |
|---|---|---|---|---|
| 50000 | OX | transposition | 2474 | 0.37% |
| 75000 | OX | transposition | 2466 | 0.04% |
| 100000 | OX | transposition | 2469 | 0.16% |
| 50000 | OX | inversion | 2504 | 1.58% |
| 75000 | OX | inversion | 2478 | 0.53% |
| 100000 | OX | inversion | 2492 | 1.10% |



## 3.4. Porównanie algorytmu genetycznego z symulowanym wyżarzaniem

| Długość ścieżki GA | Błąd | Czas | Długość ścieżki SA | Błąd | Czas |
|---|---|---|---|---|---|
| 3316 | 20.36% | 51.2049 | 3141 | 14.01% | 151.563 |




# 4. Wnioski

Rozwiązanie problemu komiwojażera przez algorytm symulowanego wyżarzania okazał się bardziej wydajny niż przez algorytm genetyczny. Może to wynikać z faktu że zaimplementowany algorytm genetyczny tworzy na początku losową populację, a w symulowanym wyżarzaniu początkowe rozwiązanie generowane jest zachłannie (przeszukiwaniem lokalnym typu greedy). Algorytm genetyczny mimo startu z błędu prawie 800%, potrafi szybko polepszać rozwiązanie przy odpowiedniej wielkości populacji.

# 5. Literatura


- https://www.youtube.com/watch?v=Pg4HP6Ayijs&pp=ygUdbWFjaWVqIGtvbW9zacWEc2tpIGdlbmV0eWN6bmU%3D <- wykład teoretyczny
- https://www.aragorn.wi.pb.edu.pl/~wkwedlo/EA5.pdf <- działanie metody krzyżowania OX oraz mutacji transposition i inversion

- https://sound.eti.pg.gda.pl/student/isd/isd03-algorytmy_genetyczne.pdf <- wykład teoretyczny; selekcja rankingowa, elityzm; sukcesja
- http://www.alife.pl/gp/p/AGelem.html <- współczynniki krzyżowania i mutacji
- https://pl.wikipedia.org/wiki/Algorytm_genetyczny <- teoria