Kamil Kamyszek, 234932 Wrocław, dn. 5 grudnia 2018

WT-7.30 prowadzący: Dr inż. Jarosław Mierzwa

Projektowanie Efektywnych Algorytmów

Algorytm Genetyczny

**1 Wstęp teoretyczny**

* 1. **Definicja algorytmu genetycznego**

Problem definiuje środowisko, w którym istnieje pewna populacja osobników. Każdy z osobników ma przypisany pewien zbiór informacji stanowiących jego genotyp, a będących podstawą do utworzenia fenotypu. Fenotyp to zbiór cech podlegających ocenie funkcji przystosowania modelującej środowisko. Innymi słowy - genotyp opisuje proponowane rozwiązanie problemu, a funkcja przystosowania ocenia, jak dobre jest to rozwiązanie.

Genotyp składa się z chromosomów, gdzie zakodowany jest fenotyp i ewentualnie pewne informacje pomocnicze dla algorytmu genetycznego. Chromosom składa się z genów.

Wspólnymi cechami algorytmów ewolucyjnych, odróżniającymi je od innych, tradycyjnych metod optymalizacji, są:

- stosowanie operatorów genetycznych, które dostosowane są do postaci rozwiązań,

- przetwarzanie populacji rozwiązań, prowadzące do równoległego przeszukiwania przestrzeni rozwiązań z różnych punktów,

- w celu ukierunkowania procesu przeszukiwania wystarczającą informacją jest jakość aktualnych rozwiązań,

- celowe wprowadzenie elementów losowych.

**1.2 Opis działania algorytmu**

Najczęściej działanie algorytmu przebiega następująco:

1. Losowana jest pewna populacja początkowa.

2. Populacja poddawana jest ocenie (selekcja). Najlepiej przystosowane osobniki biorą udział w procesie reprodukcji.

3. Genotypy wybranych osobników poddawane są operatorom ewolucyjnym:

są ze sobą kojarzone poprzez złączanie genotypów rodziców (krzyżowanie),

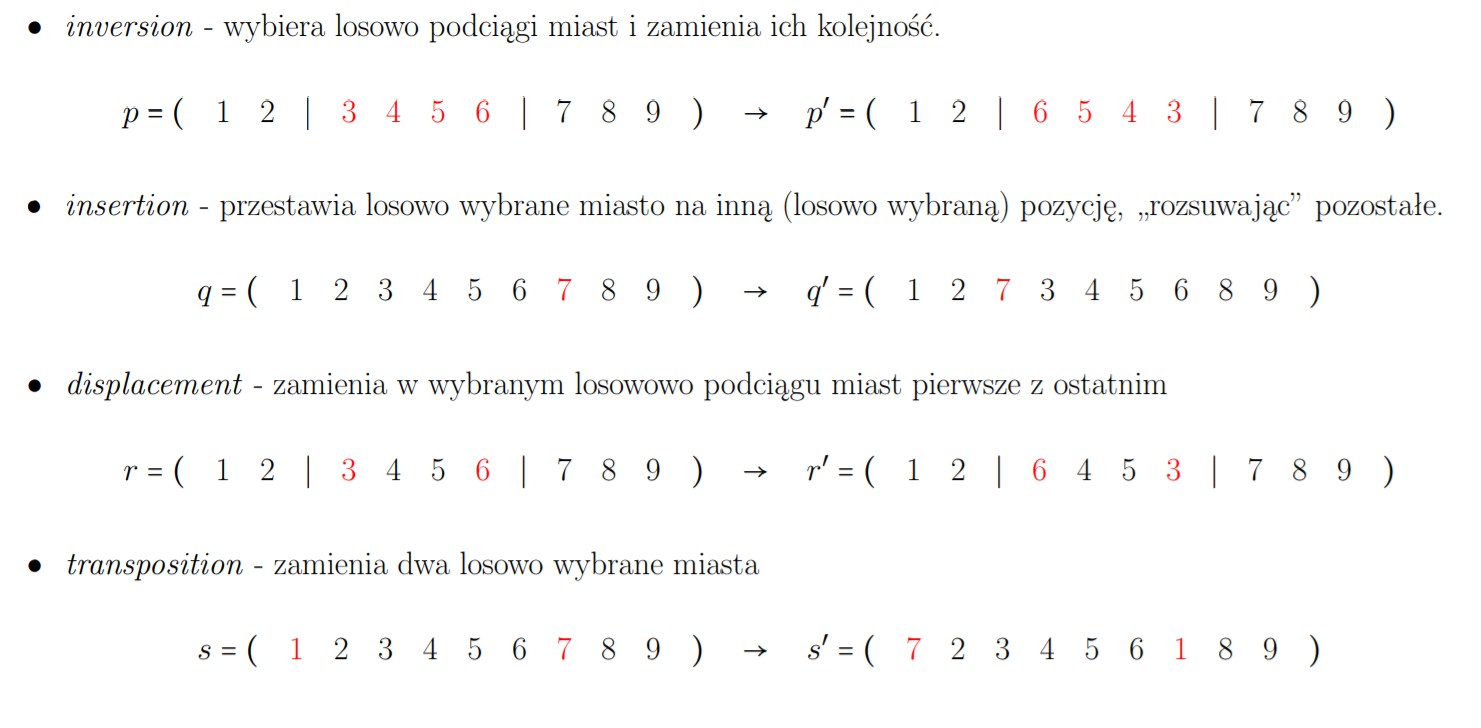
przeprowadzana jest mutacja, czyli wprowadzenie drobnych losowych zmian.

4. Rodzi się drugie (kolejne) pokolenie. Aby utrzymać stałą liczbę osobników w populacji te najlepsze (według funkcji oceniającej fenotyp) są powielane, a najsłabsze usuwane. Jeżeli nie znaleziono dostatecznie dobrego rozwiązania, algorytm powraca do kroku drugiego. W przeciwnym wypadku wybieramy najlepszego osobnika z populacji - jego genotyp to uzyskany wynik.

**1.3 Metoda Selekcji**

**Metoda rankingowa** - Obliczamy dla każdego osobnika *funkcję oceny* i ustawiamy je w szeregu najlepszy-najgorszy. Pierwsi na liście dostają prawo do rozmnażania, a reszta usuwana jest z populacji. Wadą metody jest niewrażliwość na różnice pomiędzy kolejnymi osobnikami w kolejce. Może się okazać, że sąsiadujące na liście rozwiązania mają różne wartości funkcji oceny, ale dostają prawie taką samą ilość potomstwa.

**1.4 Rodzaje mutacji (z instrukcji zamieszczonej na stronie prowadzącego)**



W moim programie zostały użyte metody ***insertion***oraz ***transposition.***

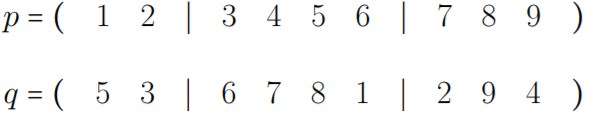
**Insertion** – tworzę zmienną, która ma przypisaną losową wartość z przedziału od **0** do ***ilość miast.*** Następnie uprzednio wylosowane miasto, które chce poddać mutacji umieszczam na wylosowanym przeze mnie miejscu w liście miast i przesuwam je.

**Transposition** – losuję dwa miasta z listy i zamieniam je miejscami.

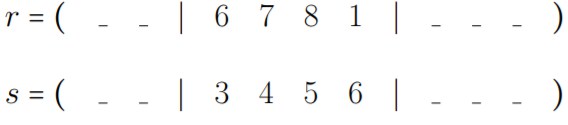
**1.5 Krzyżowanie PMX** **(z instrukcji zamieszczonej na stronie prowadzącego)**

Podczas krzyżowania zastosowałem modyfikację w postaci zamieszczania na wolnych miejscach w liście miast tych miast które nie występują w losowej kolejności, a nie używając tabeli odwzorowań.

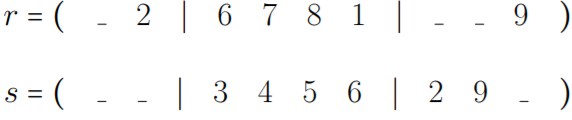
Rodzice:



Dzieci:



Wstawienie miast niepowodujących konfliktów:



Resztę miast uzupełniam na zasadzie stworzenia pomocniczej listy z miastami, które nie występują w dziecku i wklejam je na wolne miejsca list.

1. **Najważniejsze metody**

***Mutacja – umieszczenie losowego miasta z listy na losowym miejscu listy***

public ArrayList<Integer> insertion(Nodes n, int i)

{

int value = n.cityPath.get(i);

Random rand = new Random();

int random = rand.nextInt(n.cityPath.size());

n.cityPath.remove(i);

n.cityPath.add(random,value);

return n.cityPath;

}

***Mutacja – zamiana losowych 2 miast miejscami***

public ArrayList<Integer> transposition (Nodes n, int i ,int j){ //mutation

int tmp = n.cityPath.get(i); //value of i

n.cityPath.set(i,n.cityPath.get(j)); //on i place set j element

n.cityPath.set(j,tmp); //on j place set i element

n.countValue(lf);

return n.cityPath;

}

***SELEKCJA MIAST***

ArrayList<Nodes> populationSelection(int populationSize){ //choosing new population from children population and previous population

ArrayList<Nodes> newPopulation = new ArrayList<>();

ArrayList<Nodes> currentPopulation;

ArrayList<Nodes> currentPopulationChildren;

currentPopulation=(ArrayList<Nodes>)population.clone(); //cloning actual population

currentPopulationChildren =(ArrayList<Nodes>)childrenPopulation.clone(); //cloning children population

for(int i=0; i<populationSize;i++){

if(currentPopulationChildren.isEmpty() && !currentPopulation.isEmpty()){ //if there is no children but the population exist

newPopulation.add(Nodes.createNewInstance(currentPopulation.get(0),lf)); //adding population and removing first element

currentPopulation.remove(0);

continue;

}

if(!currentPopulationChildren.isEmpty() && currentPopulation.isEmpty()){ //if there are childrens and the population is empty

newPopulation.add(Nodes.createNewInstance(currentPopulationChildren.get(0),lf)); //adding children population

currentPopulation.remove(0);

continue;

}

if(currentPopulationChildren.get(0).pathCost<currentPopulation.get(0).pathCost){ //if the children cost is lower than current population cost

newPopulation.add(Nodes.createNewInstance(currentPopulationChildren.get(0),lf));

currentPopulationChildren.remove(0);

}

else{

newPopulation.add(Nodes.createNewInstance(currentPopulation.get(0),lf));

currentPopulation.remove(0);

}

}

return newPopulation;

}

***Sorting population from best to worst***

void sortingPopulation(ArrayList<Nodes> population){ //sort the population from the best to the worst

for(int i=0;i<population.size();i++){ //counting value of the path for all population

population.get(i).countValue(lf); //counting cityPath cost method

}

Collections.sort(population,Nodes.ValueComparator); //sorting population ascending

Collections.reverse(population); //reverse the population

}

**Crossover PMX**

ArrayList<Nodes> crossParent(ArrayList<Nodes> parentsPair){ //crossingover PMX

int sizeOfTheChildren = parentsPair.get(0).cityPath.size(); //taking number of cities

int crossPointOne=0;

int crossPointTwo=0; //cross point to do the cross

ArrayList<Integer>child\_1 = new ArrayList<>();

ArrayList<Integer>child\_2=new ArrayList<>();

for(int i=0;i<sizeOfTheChildren;i++){ //filing child lists with -1 to fill the whole size

child\_1.add(-1);

child\_2.add(-1);

}

Random rand = new Random();

while(crossPointOne==crossPointTwo){

crossPointOne=rand.nextInt(sizeOfTheChildren); //lottery of cross points in city list(from 0 to city list size)

crossPointTwo=rand.nextInt(sizeOfTheChildren);

}

if(crossPointOne>crossPointTwo){ //if the first point is bigger than first --> swap points

int tmp = crossPointTwo;

crossPointTwo=crossPointOne;

crossPointOne=tmp;

}

for(int i =crossPointOne;i<crossPointTwo;i++){

child\_1.set(i,Nodes.copyCity(parentsPair.get(1).cityPath,i)); //first child gets some mother genes

child\_2.set(i,Nodes.copyCity(parentsPair.get(0).cityPath,i)); //second child gets some father genes

}

for(int i=0;i<sizeOfTheChildren;i++) //filing the genes that are not in collision with the previous one

{

for(int j =crossPointOne;j<crossPointTwo;j++) {

if (!child\_1.contains(parentsPair.get(0).cityPath.get(i))) //if child\_1 doesn't contains element --> add

{

child\_1.set(i,parentsPair.get(0).cityPath.get(i)); //setting element on free place

}

}

for(int j =crossPointOne;j<crossPointTwo;j++) {

if (!child\_2.contains(parentsPair.get(1).cityPath.get(i)))

{

child\_2.set(i,parentsPair.get(1).cityPath.get(i));

}

}

}

ArrayList<Integer> helpList\_1 = new ArrayList<>();

ArrayList<Integer> citiesThatAreNotInList\_1 = new ArrayList<>();

int iterator\_1=0;

for(int h=0; h<sizeOfTheChildren;h++)

{

helpList\_1.add(h); //adding list of cities to list

}

for(int h=0; h<sizeOfTheChildren;h++)

{

if(!child\_1.contains(helpList\_1.get(h))) //if child doesn't have this city-->

{

citiesThatAreNotInList\_1.add(helpList\_1.get(h)); //add to the second help list

}

}

for(int i=0;i<sizeOfTheChildren;i++) //adding rest of the cities

{

if(child\_1.get(i)==-1){

child\_1.set(i,citiesThatAreNotInList\_1.get(iterator\_1));

iterator\_1++;

}

}

ArrayList<Integer> helpList\_2 = new ArrayList<>();

ArrayList<Integer> citiesThatAreNotInList\_2 = new ArrayList<>();

int iterator\_2=0;

for(int h=0; h<sizeOfTheChildren;h++)

{

helpList\_2.add(h);

}

for(int h=0; h<sizeOfTheChildren;h++)

{

if(!child\_2.contains(helpList\_2.get(h)))

{

citiesThatAreNotInList\_2.add(helpList\_2.get(h));

}

}

for(int i=0;i<sizeOfTheChildren;i++)

{

if(child\_2.get(i)==-1){

child\_2.set(i,citiesThatAreNotInList\_2.get(iterator\_2));

iterator\_2++;

}

}

ArrayList<Nodes> newNodeList = new ArrayList<>(); //adding new nodes to the list

Nodes newNode = new Nodes(child\_1,lf);

newNodeList.add(Nodes.createNewInstance(newNode,lf));

newNode = new Nodes(child\_2,lf);

newNodeList.add(Nodes.createNewInstance(newNode,lf));

return newNodeList;

}

1. **Wyniki**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ftv47.atsp | | | | Populacja: 10 | | |
| Mutacja: transposition | | | | czas: 1min | | |
| Koszt znaleziony | | Moment znalezienia(s) | | Koszt optymalny | Bład względny(%) | |
| 2021 | | 59 | | 1776 | 13,79504505 | |
| 2001 | | 26 | | 1776 | 12,66891892 | |
| 2245 | | 54 | | 1776 | 26,40765766 | |
| 2021 | | 58 | | 1776 | 13,79504505 | |
| 2245 | | 48 | | 1776 | 26,40765766 | |
| 1888 | | 53 | | 1776 | 6,306306306 | |
| 1852 | | 48 | | 1776 | 4,279279279 | |
| 1986 | | 42 | | 1776 | 11,82432432 | |
| 1995 | | 48 | | 1776 | 12,33108108 | |
| 1898 | | 58 | | 1776 | 6,869369369 | |
| ftv47.atsp | | | Populacja: 100 | | | |
| Mutacja: transposition | | | czas: 1min | | | |
| Koszt znaleziony | Moment znalezienia(s) | | Koszt optymalny | | | Bład względny(%) |
| 1889 | 57 | | 1776 | | | 6,362612613 |
| 1845 | 53 | | 1776 | | | 3,885135135 |
| 1799 | 59 | | 1776 | | | 1,295045045 |
| 2021 | 56 | | 1776 | | | 13,79504505 |
| 2146 | 52 | | 1776 | | | 20,83333333 |
| 1888 | 51 | | 1776 | | | 6,306306306 |
| 1852 | 57 | | 1776 | | | 4,279279279 |
| 1856 | 29 | | 1776 | | | 4,504504505 |
| 1797 | 57 | | 1776 | | | 1,182432432 |
| 1867 | 42 | | 1776 | | | 5,123873874 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ftv47.atsp | | | | | | Populacja: 1000 | | | |
| Mutacja: transposition | | | | | | czas: 1min | | | |
| Koszt znaleziony | | | Moment znalezienia(s) | | | Koszt optymalny | | | Bład względny(%) |
| 1826 | | | 52 | | | 1776 | | | 2,815315315 |
| 1798 | | | 54 | | | 1776 | | | 1,238738739 |
| 1796 | | | 43 | | | 1776 | | | 1,126126126 |
| 1995 | | | 53 | | | 1776 | | | 12,33108108 |
| 2050 | | | 44 | | | 1776 | | | 15,42792793 |
| 1887 | | | 56 | | | 1776 | | | 6,25 |
| 1835 | | | 56 | | | 1776 | | | 3,322072072 |
| 1792 | | | 59 | | | 1776 | | | 0,900900901 |
| 1883 | | | 53 | | | 1776 | | | 6,024774775 |
| 1879 | | | 51 | | | 1776 | | | 5,79954955 |
| ftv47.atsp | | | | | Populacja: 10 | | | | |
| Mutacja: insertion | | | | | czas: 1min | | | | |
| Koszt znaleziony | | Moment znalezienia(s) | | | Koszt optymalny | | Bład względny(%) | | |
| 2524 | | 57 | | | 1776 | | 42,11711712 | | |
| 2322 | | 48 | | | 1776 | | 30,74324324 | | |
| 2299 | | 53 | | | 1776 | | 29,4481982 | | |
| 2355 | | 55 | | | 1776 | | 32,60135135 | | |
| 2188 | | 56 | | | 1776 | | 23,1981982 | | |
| 2223 | | 56 | | | 1776 | | 25,16891892 | | |
| 2118 | | 54 | | | 1776 | | 19,25675676 | | |
| 2304 | | 47 | | | 1776 | | 29,72972973 | | |
| 2139 | | 50 | | | 1776 | | 20,43918919 | | |
| 2382 | | 50 | | | 1776 | | 34,12162162 | | |
| ftv47.atsp | | | | Populacja: 100 | | | | | |
| Mutacja: insertion | | | | czas: 1min | | | | | |
| Koszt znaleziony | Moment znalezienia(s) | | | Koszt optymalny | | | | Bład względny(%) | |
| 2050 | 57 | | | 1776 | | | | 15,42792793 | |
| 2300 | 59 | | | 1776 | | | | 29,5045045 | |
| 2192 | 52 | | | 1776 | | | | 23,42342342 | |
| 2225 | 37 | | | 1776 | | | | 25,28153153 | |
| 1999 | 52 | | | 1776 | | | | 12,55630631 | |
| 2223 | 49 | | | 1776 | | | | 25,16891892 | |
| 1986 | 51 | | | 1776 | | | | 11,82432432 | |
| 1899 | 58 | | | 1776 | | | | 6,925675676 | |
| 2137 | 41 | | | 1776 | | | | 20,32657658 | |
| 2060 | 59 | | | 1776 | | | | 15,99099099 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ftv47.atsp | | Populacja: 1000 | | |
| Mutacja: insertion | | czas: 1min | | |
| Koszt znaleziony | Moment znalezienia(s) | Koszt optymalny | Bład względny(%) | |
| 1889 | 58 | 1776 | | 6,362612613 |
| 1798 | 46 | 1776 | | 1,238738739 |
| 2050 | 51 | 1776 | | 15,42792793 |
| 1888 | 38 | 1776 | | 6,306306306 |
| 1897 | 40 | 1776 | | 6,813063063 |
| 1987 | 45 | 1776 | | 11,88063063 |
| 1799 | 55 | 1776 | | 1,295045045 |
| 1898 | 48 | 1776 | | 6,869369369 |
| 1948 | 46 | 1776 | | 9,684684685 |
| 2030 | 51 | 1776 | | 14,3018018 |

Podsumowując wyniki dla 47 miast można dojść do wniosku, iż obydwie metody mutacji dają zbliżone wyniki. Bez względu na wielkość populacji czas znalezienia najlepszego wyniku oscyluje w granicach 35-59 sekund. Wielkość populacji ma wpływ na dokładność wyniku. Im większa populacja tym wynik jest lepszy. Najlepszy znaleziony wynik to 1799 co daje błąd względny na poziomie 1,2%.

|  |  |  |  |
| --- | --- | --- | --- |
| ftv170.atsp | | Populacja: 10 | |
| Mutacja: transposition | | czas: 1min | |
| Koszt znaleziony | Moment znalezienia(s) | Koszt optymalny | Bład względny(%) |
| 3000 | 57 | 2755 | 8,89292196 |
| 3012 | 48 | 2755 | 9,328493648 |
| 2992 | 53 | 2755 | 8,602540835 |
| 2985 | 55 | 2755 | 8,34845735 |
| 3015 | 56 | 2755 | 9,43738657 |
| 3215 | 56 | 2755 | 16,6969147 |
| 3333 | 54 | 2755 | 20,9800363 |
| 3546 | 47 | 2755 | 28,71143376 |
| 3597 | 50 | 2755 | 30,56261343 |
| 2999 | 50 | 2755 | 8,856624319 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ftv170.atsp | | | | Populacja: 100 | | |
| Mutacja: transposition | | | | czas: 1min | | |
| Koszt znaleziony | Moment znalezienia(s) | | | Koszt optymalny | | Bład względny(%) |
| 2899 | 48 | | | 2755 | | 5,226860254 |
| 3001 | 48 | | | 2755 | | 8,929219601 |
| 2989 | 42 | | | 2755 | | 8,493647913 |
| 2987 | 54 | | | 2755 | | 8,421052632 |
| 2968 | 58 | | | 2755 | | 7,731397459 |
| 2846 | 59 | | | 2755 | | 3,303085299 |
| 2895 | 35 | | | 2755 | | 5,081669691 |
| 2986 | 49 | | | 2755 | | 8,384754991 |
| 3002 | 59 | | | 2755 | | 8,965517241 |
| 2989 | 46 | | | 2755 | | 8,493647913 |
| ftv170.atsp | | | | Populacja: 1000 | | |
| Mutacja: transposition | | | | czas: 1min | | |
| Koszt znaleziony | | | Moment znalezienia(s) | Koszt optymalny | | Bład względny(%) |
| 2799 | | | 43 | 2755 | | 1,597096189 |
| 2789 | | | 59 | 2755 | | 1,234119782 |
| 2856 | | | 48 | 2755 | | 3,666061706 |
| 2854 | | | 56 | 2755 | | 3,593466425 |
| 2836 | | | 55 | 2755 | | 2,940108893 |
| 2899 | | | 48 | 2755 | | 5,226860254 |
| 2789 | | | 43 | 2755 | | 1,234119782 |
| 2888 | | | 57 | 2755 | | 4,827586207 |
| 2999 | | | 51 | 2755 | | 8,856624319 |
| 2874 | | | 49 | 2755 | | 4,319419238 |
| ftv170.atsp | | | | | Populacja: 10 | |
| Mutacja:insert | | | | | czas: 1min | |
| Koszt znaleziony | | Moment znalezienia(s) | | | Koszt optymalny | Bład względny(%) |
| 3222 | | 48 | | | 2755 | 16,95099819 |
| 3248 | | 54 | | | 2755 | 17,89473684 |
| 2989 | | 47 | | | 2755 | 8,493647913 |
| 3521 | | 45 | | | 2755 | 27,80399274 |
| 3944 | | 59 | | | 2755 | 43,15789474 |
| 3032 | | 53 | | | 2755 | 10,05444646 |
| 3512 | | 47 | | | 2755 | 27,47731397 |
| 3356 | | 49 | | | 2755 | 21,81488203 |
| 3498 | | 56 | | | 2755 | 26,96914701 |
| 3000 | | 51 | | | 2755 | 8,89292196 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ftv170.atsp | | | Populacja: 100 | | |
| Mutacja:insert | | | czas: 1min | | |
| Koszt znaleziony | Moment znalezienia(s) | | Koszt optymalny | Bład względny(%) | |
| 2999 | 49 | | 2755 | 8,856624319 | |
| 2984 | 56 | | 2755 | 8,31215971 | |
| 3023 | 54 | | 2755 | 9,727767695 | |
| 3000 | 51 | | 2755 | 8,89292196 | |
| 3221 | 47 | | 2755 | 16,91470054 | |
| 2899 | 53 | | 2755 | 5,226860254 | |
| 2998 | 45 | | 2755 | 8,820326679 | |
| 2864 | 49 | | 2755 | 3,956442831 | |
| 2799 | 39 | | 2755 | 1,597096189 | |
| 3002 | 46 | | 2755 | 8,965517241 | |
| ftv170.atsp | | | Populacja: 1000 | | |
| Mutacja:insert | | | czas: 1min | | |
| Koszt znaleziony | | Moment znalezienia(s) | Koszt optymalny | | Bład względny(%) |
| 2888 | | 49 | 2755 | | 4,827586207 |
| 2874 | | 46 | 2755 | | 4,319419238 |
| 2799 | | 43 | 2755 | | 1,597096189 |
| 2768 | | 54 | 2755 | | 0,471869328 |
| 2998 | | 39 | 2755 | | 8,820326679 |
| 2777 | | 38 | 2755 | | 0,798548094 |
| 2899 | | 45 | 2755 | | 5,226860254 |
| 2798 | | 58 | 2755 | | 1,560798548 |
| 2768 | | 57 | 2755 | | 0,471869328 |
| 2847 | | 46 | 2755 | | 3,33938294 |

Podsumowując obliczenia dla 170 miast można dojść do wniosku, iż czas znalezienia najlepszej drogi oscyluje między 45-59 sekund. Rodzaj mutacji nie dał odmiennych wyników i bez względu na jej wybór dają podobne rezultaty. Największy wpływ na wynik algorytmu na wybór populacji. Im większa tym wynik dokładniejszy co można zobaczyć, po wielkości błędu względnego. Najlepszy wynik jaki udało się znaleźć to 2768 co daje błąd względny 0,47% czyli bardzo blisko wyniku optymalnego.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rgb403.atsp | | | | | Populacja: 10 | | | | |
| Mutacja:transposition | | | | | czas: 1min | | | | |
| Koszt znaleziony | | Moment znalezienia(s) | | | Koszt optymalny | | Bład względny(%) | | |
| 6740 | | 60 | | | 2465 | | 173,4279919 | | |
| 6969 | | 60 | | | 2465 | | 182,7180527 | | |
| 6851 | | 60 | | | 2465 | | 177,9310345 | | |
| 6909 | | 59 | | | 2465 | | 180,2839757 | | |
| 6579 | | 39 | | | 2465 | | 166,8965517 | | |
| 6351 | | 56 | | | 2465 | | 157,6470588 | | |
| 6910 | | 54 | | | 2465 | | 180,3245436 | | |
| 6784 | | 58 | | | 2465 | | 175,2129817 | | |
| 6819 | | 56 | | | 2465 | | 176,63286 | | |
| 6435 | | 59 | | | 2465 | | 161,0547667 | | |
| rgb403.atsp | | | | Populacja: 100 | | | | | |
| Mutacja:transposition | | | | czas: 1min | | | | | |
| Koszt znaleziony | Moment znalezienia(s) | | | Koszt optymalny | | | | Bład względny(%) | |
| 4869 | 60 | | | 2465 | | | | 97,52535497 | |
| 4857 | 56 | | | 2465 | | | | 97,03853955 | |
| 4358 | 45 | | | 2465 | | | | 76,79513185 | |
| 5867 | 43 | | | 2465 | | | | 138,0121704 | |
| 3254 | 56 | | | 2465 | | | | 32,00811359 | |
| 3222 | 53 | | | 2465 | | | | 30,70993915 | |
| 3045 | 57 | | | 2465 | | | | 23,52941176 | |
| 4435 | 58 | | | 2465 | | | | 79,9188641 | |
| 3586 | 35 | | | 2465 | | | | 45,47667343 | |
| 3956 | 59 | | | 2465 | | | | 60,48681542 | |
| rgb403.atsp | | | | | | Populacja: 1000 | | | |
| Mutacja:transposition | | | | | | czas: 1min | | | |
| Koszt znaleziony | | | Moment znalezienia(s) | | | Koszt optymalny | | | Bład względny(%) |
| 2999 | | | 43 | | | 2465 | | | 21,663286 |
| 3221 | | | 53 | | | 2465 | | | 30,6693712 |
| 3896 | | | 42 | | | 2465 | | | 58,05273834 |
| 2658 | | | 46 | | | 2465 | | | 7,829614604 |
| 2768 | | | 56 | | | 2465 | | | 12,29208925 |
| 2956 | | | 53 | | | 2465 | | | 19,9188641 |
| 2867 | | | 45 | | | 2465 | | | 16,30831643 |
| 2786 | | | 58 | | | 2465 | | | 13,02231237 |
| 2999 | | | 45 | | | 2465 | | | 21,663286 |
| 3054 | | | 39 | | | 2465 | | | 23,89452333 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rgb403.atsp | | | | | Populacja: 10 | | | | |
| Mutacja:insertion | | | | | czas: 1min | | | | |
| Koszt znaleziony | | Moment znalezienia(s) | | | Koszt optymalny | | Bład względny(%) | | |
| 6759 | | 57 | | | 2465 | | 174,198783 | | |
| 5968 | | 60 | | | 2465 | | 142,1095335 | | |
| 5468 | | 54 | | | 2465 | | 121,8255578 | | |
| 5387 | | 45 | | | 2465 | | 118,5395538 | | |
| 5777 | | 45 | | | 2465 | | 134,3610548 | | |
| 5458 | | 53 | | | 2465 | | 121,4198783 | | |
| 5689 | | 48 | | | 2465 | | 130,7910751 | | |
| 5687 | | 53 | | | 2465 | | 130,7099391 | | |
| 5783 | | 47 | | | 2465 | | 134,6044625 | | |
| 4558 | | 46 | | | 2465 | | 84,90872211 | | |
| rgb403.atsp | | | | Populacja: 100 | | | | | |
| Mutacja:insertion | | | | czas: 1min | | | | | |
| Koszt znaleziony | Moment znalezienia(s) | | | Koszt optymalny | | | | Bład względny(%) | |
| 4356 | 45 | | | 2465 | | | | 76,71399594 | |
| 4586 | 60 | | | 2465 | | | | 86,04462475 | |
| 4773 | 56 | | | 2465 | | | | 93,63083164 | |
| 4583 | 59 | | | 2465 | | | | 85,92292089 | |
| 3576 | 39 | | | 2465 | | | | 45,07099391 | |
| 4537 | 48 | | | 2465 | | | | 84,05679513 | |
| 3576 | 55 | | | 2465 | | | | 45,07099391 | |
| 3567 | 58 | | | 2465 | | | | 44,70588235 | |
| 3941 | 44 | | | 2465 | | | | 59,87829615 | |
| 4586 | 39 | | | 2465 | | | | 86,04462475 | |
| rgb403.atsp | | | | | | Populacja: 1000 | | | |
| Mutacja:insertion | | | | | | czas: 1min | | | |
| Koszt znaleziony | | | Moment znalezienia(s) | | | Koszt optymalny | | | Bład względny(%) |
| 2786 | | | 45 | | | 2465 | | | 13,02231237 |
| 2867 | | | 60 | | | 2465 | | | 16,30831643 |
| 2968 | | | 54 | | | 2465 | | | 20,40567951 |
| 2786 | | | 58 | | | 2465 | | | 13,02231237 |
| 2864 | | | 59 | | | 2465 | | | 16,18661258 |
| 3023 | | | 52 | | | 2465 | | | 22,63691684 |
| 2969 | | | 45 | | | 2465 | | | 20,44624746 |
| 3567 | | | 56 | | | 2465 | | | 44,70588235 |
| 3456 | | | 48 | | | 2465 | | | 40,20283976 |
| 3564 | | | 56 | | | 2465 | | | 44,5841785 |

Podsumowując wyniki dla 403 miast można dojść do wniosku, że i tym razem wybór metody mutacji nie robi zbyt dużej różnicy w wynikach. Czas oscyluje w granicach 45-60 sekund. Głównym czynnikiem wpływającym na wynik jest liczba populacji. Przy populacjach 10 i 100 wyniki znacząco odbiegają od najkrótszej drogi. Dopiero przy 1000 populacji wynik zbliża się do rozwiązania optymalnego, jednak nadal błąd oscyluje w granicach 10-35%.

|  |  |  |
| --- | --- | --- |
| Porównanie do Tabu Search | | |
| Ilość miast | Najlepszy wynik TS | Najlepszy wynik AG |
| 47 | 1797 | 1792 |
| 170 | 4365 | 2768 |
| 403 | 2524 | 2956 |

Porównując oba algorytmy pominąłem rodzaje sąsiedztwa i mutacje i wybrałem wyniki najlepsze dla danej ilości miast. Porównując wyniki ogółem mogę stwierdzić, iż w tabu search wyniki są mniej chaotyczne niż w algorytmie genetycznym. Najlepszy wynik ogółem udało się znaleźć używając AG, jednakże wyniki tak zbliżone do optymalnego pojawiają się rzadko w przeciwieństwie do tabu search, w którym to wyniki mocno przybliżone (przynajmniej dla malej liczby miast) występują dużo częściej.

**Wykresy funkcji od czasu**

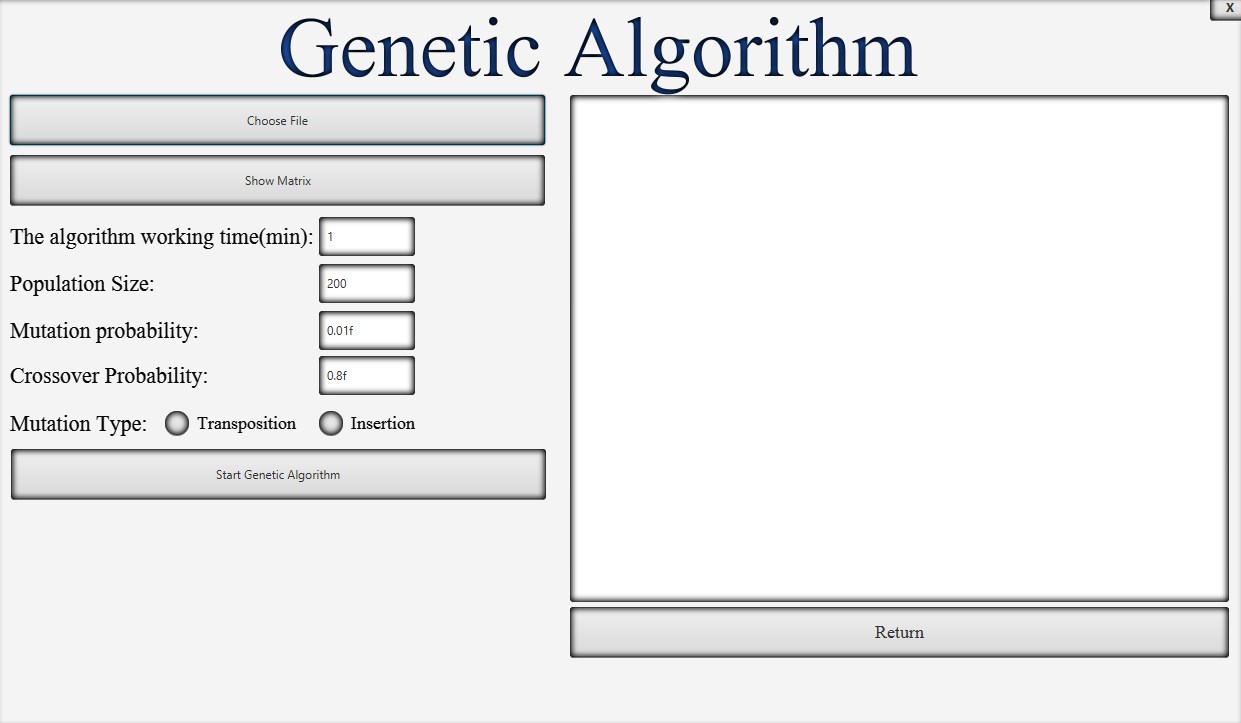
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ftv47.atsp | | | | Mutacja:transposition | Populacja:1000 | |
| Koszt znaleziony | | Koszt optymalny | | Błąd względny (%) | Czas (min) | |
| 2888 | | 1776 | | 62,61261261 | 1 | |
| 2728 | | 1776 | | 53,6036036 | 2 | |
| 2643 | | 1776 | | 48,81756757 | 4 | |
| 2619 | | 1776 | | 47,46621622 | 6 | |
| ftv170.atsp | | | | Mutacja:transposition | | Populacja:1000 |
| Koszt znaleziony | Koszt optymalny | | | Błąd względny (%) | | Czas (min) |
| 4532 | 2755 | | | 64,50090744 | | 1 |
| 3568 | 2755 | | | 29,50998185 | | 2 |
| 2987 | 2755 | | | 8,421052632 | | 4 |
| 3054 | 2755 | | | 10,85299456 | | 6 |
| 403rgb.atsp | | | | Mutacja:transposition | | Populacja:1000 |
| Koszt znaleziony | | | Koszt optymalny | Błąd względny (%) | | Czas (min) |
| 7058 | | | 2465 | 186,3286004 | | 1 |
| 6587 | | | 2465 | 167,2210953 | | 2 |
| 3045 | | | 2465 | 23,52941176 | | 4 |
| 4685 | | | 2465 | 90,06085193 | | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ftv47.atsp | | | | Mutacja:insertion | Populacja:1000 |
| Koszt znaleziony | | Koszt optymalny | | Błąd względny (%) | Czas (min) |
| 2463 | | 1776 | | 38,68243243 | 1 |
| 3054 | | 1776 | | 71,95945946 | 2 |
| 1989 | | 1776 | | 11,99324324 | 4 |
| 1886 | | 1776 | | 6,193693694 | 6 |
| ftv170.atsp | | | | Mutacja:insertion | Populacja:1000 |
| Koszt znaleziony | Koszt optymalny | | | Błąd względny (%) | Czas (min) |
| 5896 | 2755 | | | 114,0108893 | 1 |
| 4658 | 2755 | | | 69,07441016 | 2 |
| 3958 | 2755 | | | 43,66606171 | 4 |
| 2998 | 2755 | | | 8,820326679 | 6 |
| 403rgb.atsp | | | | Mutacja:insertion | Populacja:1000 |
| Koszt znaleziony | | | Koszt optymalny | Błąd względny (%) | Czas (min) |
| 7435 | | | 2465 | 201,6227181 | 1 |
| 5369 | | | 2465 | 117,8093306 | 2 |
| 4526 | | | 2465 | 83,61054767 | 4 |
| 3054 | | | 2465 | 23,89452333 | 6 |

Analizując wykresy błędu względnego od czasu można zauważyć, że algorytm im dłużej działa tym lepszy wynik daje na koniec. Jest to obarczone pewną losowością, dlatego błąd względny nie maleje zawsze, gdy czas rośnie, jednak czas przeszukiwania ma duży wpływ na końcowy wynik działania algorytmu.

1. **Implementacja Graficzna**

W celu łatwiejszego dokonywania pomiarów stworzony został przyjazny użytkownikowi interfejs graficzny wykonany w technologii JavaFX.



**5 Uwagi końcowe**

Ze względu na to, iż projekt był pisany w języku Java możliwe są błędy w wynikach spowodowane działaniem maszyny wirtualnej (zawyżone/zaniżone wyniki). Komentarze w kodzie są wykonane w języku angielskim, gdyż są to komentarze, które pisałem podczas wykonywania projektu.

**6 Źródła**

<http://algorytmy.ency.pl/tutorial/problem_komiwojazera_algorytm_genetyczny?fbclid=IwAR3GypfmBZZYP1jO64Xjytavi-3P3QjXcR8FPcMyWjqRZz7R8j3mhPOITJw>

<http://home.agh.edu.pl/~vlsi/AI/gen_t/?fbclid=IwAR2YDfa_8mDHkcfjOt0G3eCBwg9qbxVi8f2ecuN1kIk3isXynpn8voeb9gA>

<https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35?fbclid=IwAR0F7-B57LOWN5AjdGPOnbiafDtrTRglunPEfKzSDUgbAxGwUBVEnVRMm2I>

<https://pl.wikipedia.org/wiki/Algorytm_genetyczny>