# ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

### Кафедра АСУ

### 

Отчёт

о лабораторной работе №3

по курсу: «Решение задачи коммивояжёра с помощью генетических алгоритмов»

# Вариант «2»

Выполнил:

ст. гр. ИСм-22

Мариничев И.И

Проверила:

Васяева Т.А

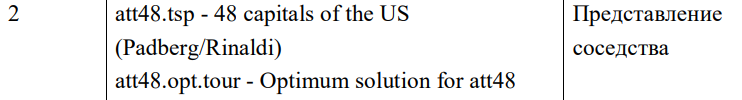
# 

# Донецк 2022

**Ход работы:**

1. Разработать генетический алгоритм для решение задачи коммивояжера. Индивидуальное задание выбирается по таблице 3.1 выбирается файл с данными и решением задачи (оптимальным маршрутом) по ссылке http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/. В файле с \*.tsp координаты городов, а в файле \*.opt.tour – оптимальный маршрут.
2. Выполнить программную реализацию разработанного алгоритма на языке высокого уровня. Рекомендованный язык Python с применением DEAP. Предусмотреть возможность просмотра процесса поиска решения.
3. Представить графически найденное решение. Предусмотреть возможность пошагового просмотра процесса поиска решения.
4. Сравнить найденное решение с представленным в условии задачи оптимальным решением.
5. Исследовать сходимость от параметров ГА.

*Задание:*

**

Хромосомой будет являться вектор с номерами городов в представлении соседства:

В представлении соседства тур является списком из n городов. Город J находится на позиции I только в том случае, если маршрут проходит из города I в город J. Например, вектор (2 4 8 3 9 7 1 5 6) представляет следующий тур: 1-2-4-3-8-5-9-6-7 .

В качестве операции кроссовера был выбран эвристический метод:

Эвристический кроссинговер (heuristic crossover) строит потомков, выбирая случайный город как стартовую точку для маршрута – потомка. Потом он сравнивает два соответствующих ребра от каждого из родителей и выбирает белее короткое. Затем конечный город выбирается как начальный для выбора следующего более короткого ребра из этого города. Если на каком–то шаге получается замкнутый тур, тур продолжается любым случайным городом, который ещё не посещался.

**НЕОБЪОДИМО НАЙТИ ТАКОЙ ВЕКТОР, ТУР ГОРОДОВ КОТОРОГО БУДЕТ ИМЕТЬ НАИМЕНЬШУЮ ДЛИНУ!**

Целевая функция вычисляется следующим образом:

(x,x,x,x,x,x,x) – вектор городов в соседском представлении

X – город с координатами (x;y)

y-y-y-y-y-y-y – расшифрованный тур городов (НАЧИНАЕТСЯ И ЗАКАНЧИВАЕТСЯ 1цей)

Сначала вектор городов преобразуется в тур городов или же их представление по порядку следования:

1. (5,6,4,2,3,7,1)

1-

1. (5,6,4,2,3,7,1)

1-5-

1. (5,6,4,2,3,7,1)

1-5-3-

1. (5,6,4,2,3,7,1)

1-5-3-4-

1. (5,6,4,2,3,7,1)

1-5-3-4-2-

1. (5,6,4,2,3,7,1)

1-5-3-4-2-6-

1. (5,6,4,2,3,7,1)

1-5-3-4-2-6-7

1. (5,6,4,2,3,7,1)

1-5-3-4-2-6-7-1

После того, как маршрут построен, мы высчитываем сумму расстояний между соседними городами, т.е. расстояние между 1 и 5, плюс расстояние между 5 и 3 и т.д.

Формула расчёта расстояния:

R = , где R – расстояние между городами, а «x» и «y» – координаты городов.

**ХРОМОСОМА С НАИМЕНЬШЕЙ СУММОЙ РАССТОЯНИЙ – ЯВЛЯЕТСЯ ОТВЕТОМ!**

**СХЕМА СОЗДАНИЯ ХРОМОСОМ:**

(1,2,3,4,5,6,7) – пулл городов

(x,x,x,x,x,x,x) – вектор городов в соседском представлении

y-y-y-y-y-y-y – расшифрованный тур городов (НАЧИНАЕТСЯ И ЗАКАНЧИВАЕТСЯ 1цей)

1. Я

(1,2,3,4,5,6,7) – пулл городов

(x,x,x,x,x,x,x) – 1-y-y-y-y-y-y

1. й

(1,,3,4,5,6,7) – пулл городов

(2,x,x,x,x,x,x) – 1-2-y-y-y-y-y

1. й

(1,,3,4,,6,7) – пулл городов

(2,5,x,x,x,x,x) – 1-2-5-y-y-y-y

И так далее, пока не будет заполнен вектор.

Затем, идёт проверка вектора на зацикливание:

(1,2,3,4,5,6,7) – пулл городов

1. (,2,3,4,5,6,7) – пулл городов

(5,6,4,1,3,7,2) – 1-

1. (,2,3,4,,6,7) – пулл городов

(5,6,4,1,3,7,2) – 1-5-

1. (,2,,4,,6,7) – пулл городов

(5,6,4,1,3,7,2) – 1-5-3-

1. (,2,,,,6,7) – пулл городов

(5,6,4,1,3,7,2) – 1-5-3-4-

1. (,2,,,,6,7) – пулл городов

(5,6,4,1,3,7,2) – 1-5-3-4-1 Происходит зацикливание тура!

1. (,2,,,,6,7) – пулл городов

(5,6,4,1,3,7,2) – Выбираем случайный элемент из пула городов

1. (,2,,,,6,7) – пулл городов

(5,6,4,2,3,7,1) – Меняем местами с текущим

1. (,,,,,6,7) – пулл городов

(5,6,4,2,3,7,1) – 1-5-3-4-2-

1. (,,,,,,7) – пулл городов

(5,6,4,2,3,7,1) – 1-5-3-4-2-6-

1. (,,,,,,)– пулл городов

(5,6,4,2,3,7,1) – 1-5-3-4-2-6-7

**КРОССИНГОВЕР:**

Кроссинговер работает по следующей схеме:

В1 (5,6,4,2,3,7,1) – родитель 1

В2 (2,7,4,6,1,5,3) – родитель 2

П (x,x,x,x,x,x,x) – потомок

Т (1,2,3,4,5,6,7) – список городов в туре

1. Выбираем первый город из родителя 1 и добавляем в потомка:

В1 (5,6,4,2,3,7,1) – родитель 1

В2 (2,7,4,6,1,5,3) – родитель 2

ПРОВЕРЯЕМ, ЕСТЬ ЛИ УЖЕ ТАКОЙ ЭЛЕМЕНТ В ПОТОМКЕ:

П (y,y,y,y,y,y,y)

ВЕКТОР ПОТОМКА ПУСТ, ПО ЭТОМУ ЭЛЕМЕНТ НЕ ПОВТОРЯЕТСЯ.

ДОБАВЛЯЕМ ЕГО В ВЕКТОР ПОТОМКА:

П (5,x,x,x,x,x,x) – потомок

УБИРАЕМ ЕГО ИЗ СПИСКА ГОРОДОВ В ТУРЕ:

Т (1,2,3,4,х,6,7)

1. Смотрим в обоих родителях элемент под этим номером:

№5

В1 (5,6,4,2,3,7,1) – родитель 1

№5

В2 (2,7,4,6,1,5,3) – родитель 2

П (5,x,x,x,x,x,x) – потомок

1. Проверяем наличие этих элементов в векторе потомка:

1 и 3 **∉** П

1. Сравниваем расстояние между координатами города в потомке и городом в каждом родителе поочерёдно по формуле:

R = , где R – расстояние между городами, а «x» и «y» – координаты городов.

И выбираем ближайший город

1. Выбранный город располагаем в массиве под соответствующим номером:

Город №5(3;4) – координаты

Город №3(1;2)

Город №1(5;7)

R1 = = 2,83 город №3 из первого родителя – ближе!

R2 = = 3,6

№5

П (5,x,x,x,3,x,x) – потомок

1. Удаляем город 3 из
2. Заносим добавленный город в список городов тура:

Т (1,2,х,4,х,6,7)

1. Далее выбираем из родителей города, находящиеся в массиве на 3ем месте:

№3

В1 (5,4,6,2,3,7,1) – родитель 1

№3

В2 (2,7,4,6,1,5,3) – родитель 2

И так далее…

ЧАСТНЫЕ СЛУЧАИ:

**Если выбираемый из векторов родителей город уже есть в списке тура и образует замкнутую петлю:**

1. Выбираем города из родителей:

№6

В1 (5,7,6,2,3,4,1) – родитель 1

№6

В2 (2,7,4,6,1,5,3) – родитель 2

П (5,x,4,6,3,x,x) – потомок

Т (1,2,х,х,х,х,7)– список городов в туре

1. Проверяем наличие городов №4 и №5 в потомке:

П (5,x,4,6,3,x,x) – в потомке уже есть эти города

1. Поскольку эти города уже есть в потомке выбираем случайный оставшийся из списка городов в туре и заносим его в потомка:

Т (1,2,х,х,х,х,7)– список городов в туре

№6

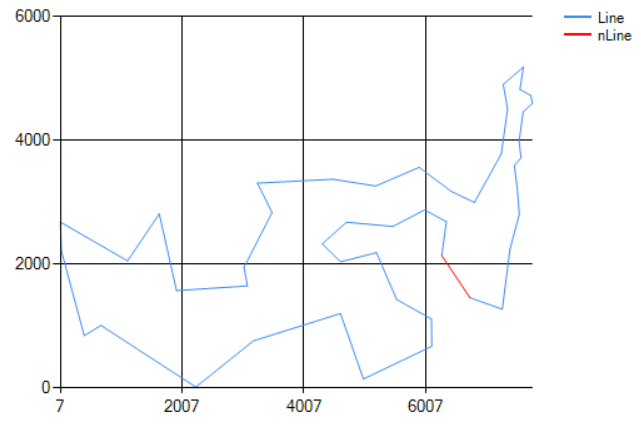
П (5,x,4,6,3,2,x) – потомок

МУТАЦИЯ:

Для мутации хромосома разделяется на 2 половины на случайной позиции и вторая половина заполняется заново оставшимися элементами.

Результат выполнения программы:

Известный лучший результат:



(красным указан последний отрезок в цепочке)

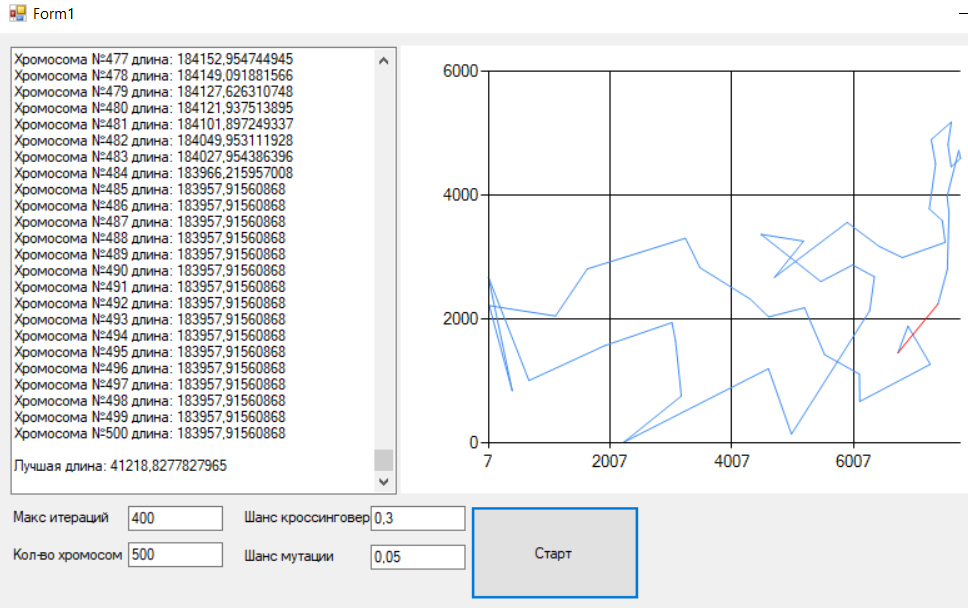
Расстояние = 33523.708

Результат выполнения алгоритма:

Лучшая длина при генерации: 127289,11348937

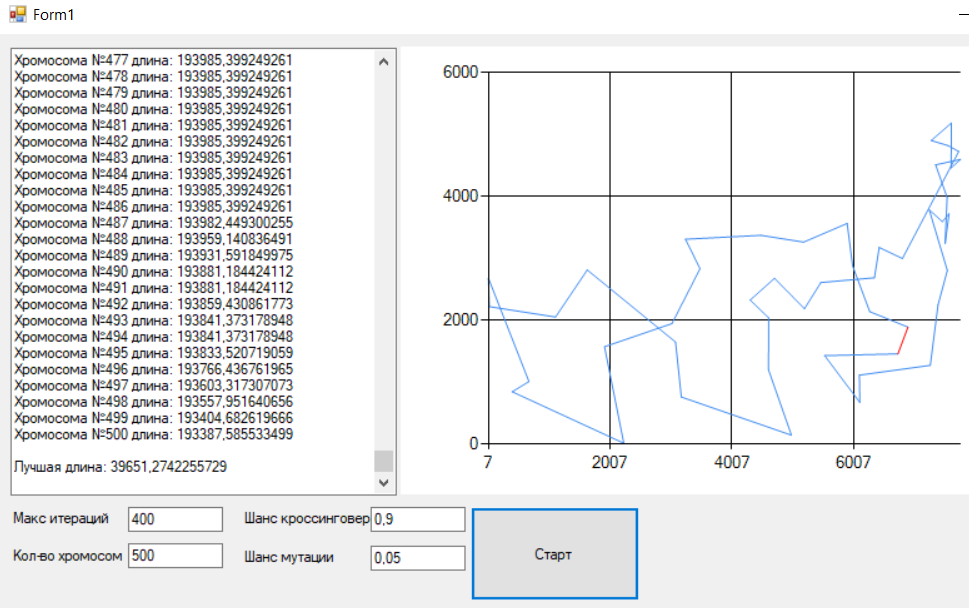
Лучшая длина найденная ГА: 41218,8277827965

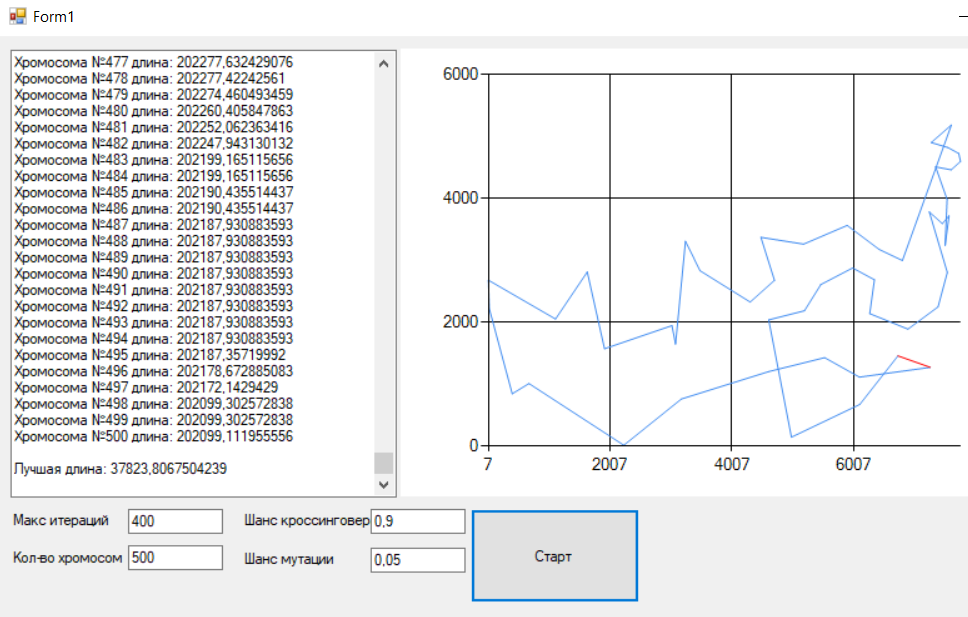
Вывод: ГА - работает

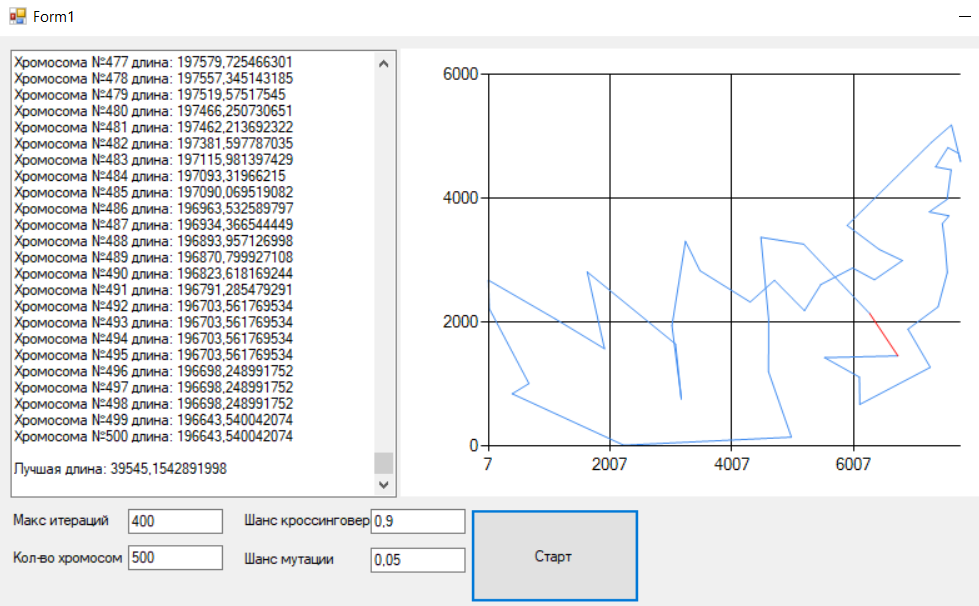


Проверим результат работы ГА при увеличении шанса кроссинговера

При 500 хромосомах и 400 циклах:

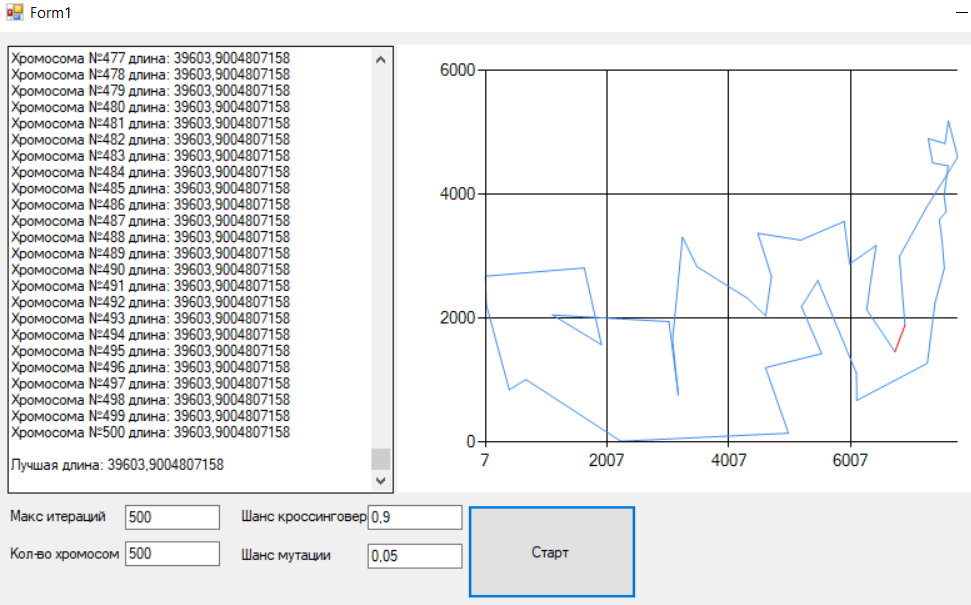






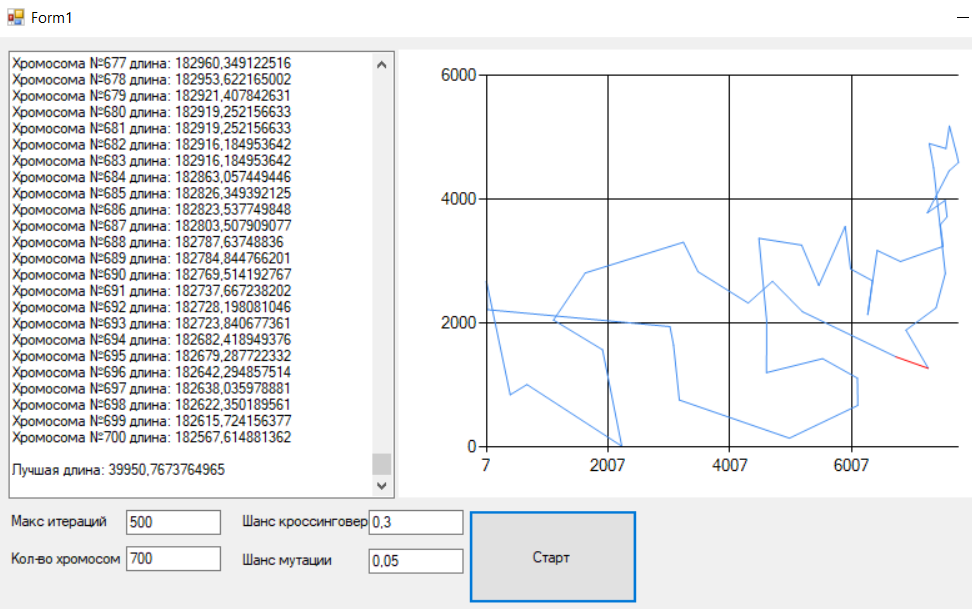
Вывод: увеличение шанса кроссинговера немного повысила качество вычислений.

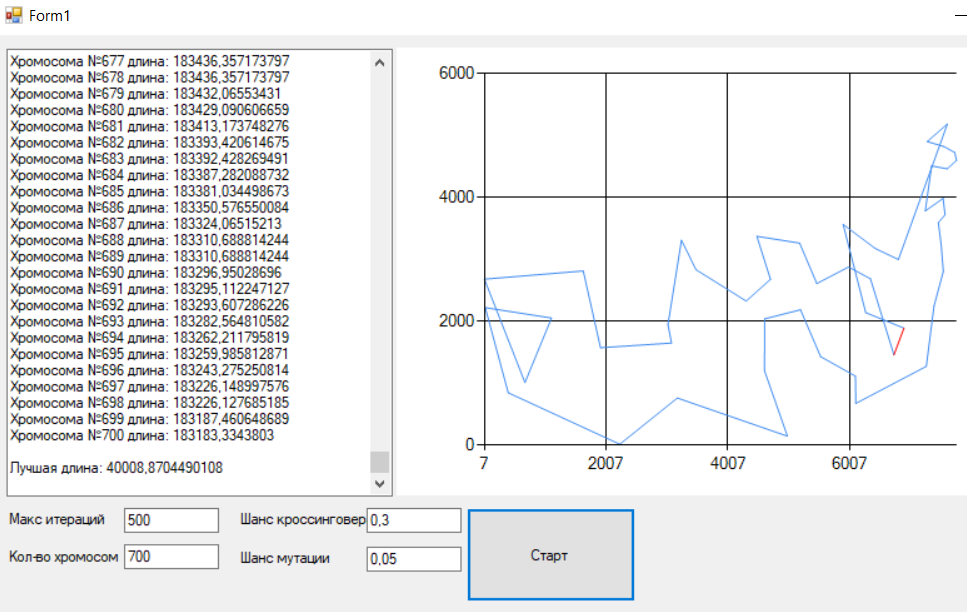
Попробуем увеличить количество итераций:

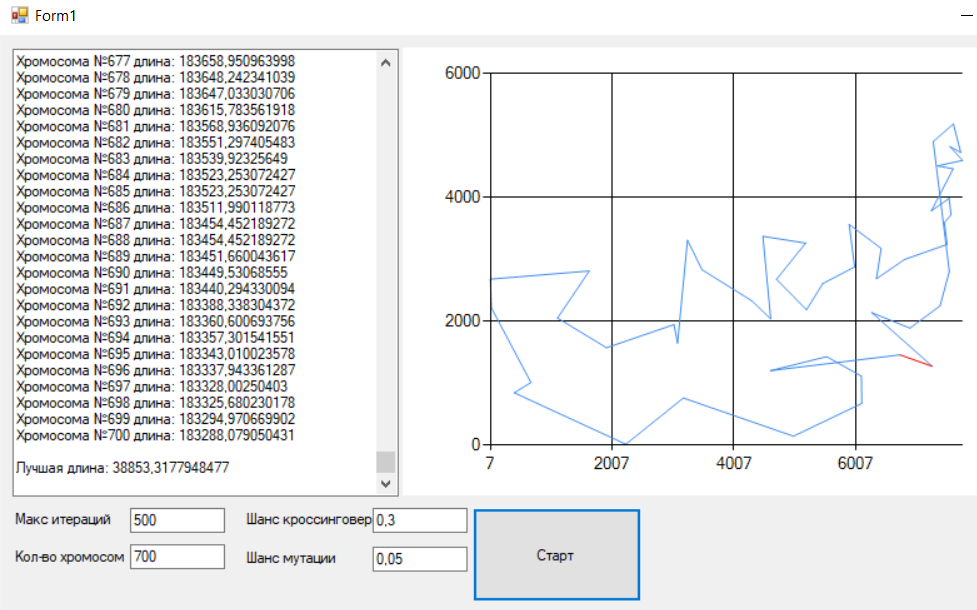


Уже на 500й итерации все хромосомы сошлись.

Попробуем увеличить количество хромосом, при шансе кроссинговера 0,3 и 500 итерациях:







Вывод: увеличение количества хромосом незначительно улучшает качество работы ГА.

ОБЩИЙ ВЫВОД: увеличение сила хромосом и шанса кроссинговера улучшают работу ГА.

Код программы:

Классы алгоритма:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Лаба\_4\_ИИ

{

class City

{

public int number;

public int x;

public int y;

public City()

{

x = 0;

y = 0;

number = 0;

}

public City(int xx, int yy, int numb)

{

x = xx;

y = yy;

number = numb;

}

}

class Chromosome

{

public int[] mass;

public int[] sortedmass;

public double length;

public Chromosome(int col, int[] mass2, int[] smass)

{

mass = new int[col];

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

{

mass[i] = mass2[i];

}

sortedmass = new int[col];

smass.CopyTo(sortedmass, 0);

int a = 0;

}

public Chromosome()

{

mass = new int[48];

}

}

class GeneticAlgorithm

{

public City[] cities;

public List<Chromosome> chromosomes;

public StringBuilder textAll = new StringBuilder();

private Random rand = new Random();

public GeneticAlgorithm()

{

}

public void init(int colc, int colh, int[,] mass2)

{

chromosomes = new List<Chromosome>();

List<int> list = new List<int>();

cities = new City[colc];

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

{

cities[i] = new City();

cities[i].number = mass2[i, 0];

cities[i].x = mass2[i, 1];

cities[i].y = mass2[i, 2];

list.Add(mass2[i, 0]);

}

chromosomes = new List<Chromosome>();

int[] cline = new int[colc];

int tmp, tmp2;

List<int> tmplist = new List<int>();

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

{

tmplist.AddRange(list);

for (int j = 0; j < colc; j++)

{

l1: tmp = tmplist[rand.Next(0, colc - j)];

if ((tmp == j + 1) && (tmplist.Count > 1)) goto l1;

cline[j] = tmp;

tmplist.Remove(cline[j]);

}

chromosomes.Add(road(list, cline));

chromosomes[i].length = count\_length(chromosomes[i]);

textAll.Append(" Хромосома №" + (i + 1) + " длина: " + chromosomes[i].length + Environment.NewLine);

}

textAll.Append(" Лучшая длина: " + chromosomes.Min(c => c.length) + Environment.NewLine);

}

public Chromosome road(List<int> list, int[] cline)

{

int[] massprov = new int[cities.Length];

List<int> tmplist = new List<int>();

int tmp, tmp2;

massprov[0] = 1;

tmplist.AddRange(list);

tmplist.Remove(1);

for (int j = 0; j < cities.Length - 1; j++)

{

tmp = cline[massprov[j] - 1];

if (massprov.Contains(tmp))

{

l2: int randomm = rand.Next(0, cities.Length - j - 1);

tmp = tmplist[randomm];

if (massprov.Contains(tmp))

{

goto l2;

}

else

{

tmp2 = cline[massprov[j] - 1];

cline[massprov[j] - 1] = tmp;

cline[Array.IndexOf(cline, tmp)] = tmp2;

}

}

massprov[j + 1] = tmp;

tmplist.Remove(tmp);

}

return new Chromosome(cline.Length, cline, massprov);

}

public List<Chromosome> algorithm(int colsteps, double crosschans, double mutchans)

{

List<Chromosome> newgen = new List<Chromosome>();

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

{

for (int j = 0; j < chromosomes.Count; j++)

{

if (rand.NextDouble() <= crosschans)

{

int[] abc = crossingover(j, rand.Next(0, chromosomes.Count - 1));

newgen.Add(road(chromosomes[0].mass.ToList(), abc));

newgen.Last().length = count\_length(newgen.Last());

;

}

if (rand.NextDouble() <= mutchans)

{

mutation(chromosomes[j]);

}

}

chromosomes.AddRange(newgen);

Chromosome temp;

for (int k = 0; k < chromosomes.Count; k++)

{

for (int l = i + 1; l < chromosomes.Count; l++)

{

if (chromosomes[k].length > chromosomes[l].length)

{

temp = chromosomes[k];

chromosomes[k] = chromosomes[l];

chromosomes[l] = temp;

}

}

}

for (int k = chromosomes.Count - newgen.Count; k < chromosomes.Count;)

{

chromosomes.RemoveAt(k);

}

newgen.Clear();

}

return chromosomes;

}

public int[] crossingover(int num1, int num2)

{

Chromosome parent1 = chromosomes[num1];

Chromosome parent2 = chromosomes[num2];

int[] masschild1 = new int[parent1.mass.Length];

int[] masschild2 = new int[parent2.mass.Length];

List<int> tmpcities1 = new List<int>();

tmpcities1.AddRange(parent1.mass.ToList());

List<int> tmpcities2 = new List<int>();

tmpcities2.AddRange(parent2.mass.ToList());

double tmpchild1;

double tmpchild2;

for (int i = 0; i < parent1.mass.Length; i++)

{

masschild1[i] = -1;

}

masschild1[0] = parent1.mass[0];

int num = 0;

tmpcities1.Remove(masschild1[0]);

tmpcities1.Remove(1);

while (tmpcities1.Count > 0)

{

if (parent1.mass[masschild1[num] - 1] == 1 || parent2.mass[masschild1[num] - 1] == 1)

{

int abc;

crosserror: abc = tmpcities1[rand.Next(0, tmpcities1.Count)];

if (abc == masschild1[num]) goto crosserror;

if (masschild1[abc - 1] != -1) goto crosserror;

masschild1[masschild1[num] - 1] = abc;

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

else if (!masschild1.Contains(parent1.mass[masschild1[num] - 1]) && !masschild1.Contains(parent2.mass[masschild1[num] - 1]))

{

tmpchild1 = Math.Sqrt(Math.Pow((cities[masschild1[num] - 1].x - cities[parent1.mass[masschild1[num] - 1] - 1].x), 2)

+ Math.Pow((cities[masschild1[num] - 1].y - cities[parent1.mass[masschild1[num] - 1] - 1].y), 2));

tmpchild2 = Math.Sqrt(Math.Pow((cities[masschild1[num] - 1].x - cities[parent2.mass[masschild1[num] - 1] - 1].x), 2)

+ Math.Pow((cities[masschild1[num] - 1].y - cities[parent2.mass[masschild1[num] - 1] - 1].y), 2));

if (tmpchild1 <= tmpchild2)

{

masschild1[masschild1[num] - 1] = parent1.mass[masschild1[num] - 1];

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

if (tmpchild1 > tmpchild2)

{

masschild1[masschild1[num] - 1] = parent2.mass[masschild1[num] - 1];

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

}

else

{

if (!masschild1.Contains(parent1.mass[masschild1[num] - 1]))

{

masschild1[masschild1[num] - 1] = parent1.mass[masschild1[num] - 1];

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

else

{

if (!masschild1.Contains(parent2.mass[masschild1[num] - 1]))

{

masschild1[masschild1[num] - 1] = parent2.mass[masschild1[num] - 1];

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

else

{

if (tmpcities1.Count == 1)

{

masschild1[masschild1[num] - 1] = tmpcities1[0];

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

else

{

int abc;

crosserror: abc = tmpcities1[rand.Next(0, tmpcities1.Count)];

if (abc == masschild1[num]) goto crosserror;

if (masschild1[abc - 1] != -1) goto crosserror;

masschild1[masschild1[num] - 1] = abc;

tmpcities1.Remove(masschild1[masschild1[num] - 1]);

num = masschild1[num] - 1;

}

}

}

}

}

if (masschild1.Contains(-1))

{

masschild1[ Array.IndexOf(masschild1, -1)] = 1;

}

return masschild1;

chromosomes[num1] = road(parent1.mass.ToList(), masschild1);

}

public void mutation(Chromosome chromosome)

{

}

public List<Chromosome> tour(List<Chromosome> n, int diff)

{

List<Chromosome> newgen = new List<Chromosome>();

int tmpn, bestn = 0;

double tmpl, bestl = n[0].length;

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

{

bestn = rand.Next(0, diff - 1); ;

bestl = n[bestn].length;

for (int j = 0; j < 10; j++)

{

tmpn = rand.Next(0, diff - 1);

if (n[tmpn].length < bestl)

{

bestn = tmpn;

bestl = n[tmpn].length;

}

}

newgen.Add(n[bestn]);

}

return newgen;

}

public double count\_length(Chromosome chr)

{

double sum = 0;

for (int i = 0; i < chr.mass.Length - 1; i++)

{

sum += Math.Sqrt(Math.Pow((cities[chr.sortedmass[i + 1] - 1].x - cities[chr.sortedmass[i] - 1].x), 2)

+ Math.Pow((cities[chr.sortedmass[i + 1] - 1].y - cities[chr.sortedmass[i] - 1].y), 2));

}

sum += Math.Sqrt(Math.Pow((cities[chr.sortedmass[chr.mass.Length - 1] - 1].x - cities[chr.sortedmass[0] - 1].x), 2)

+ Math.Pow((cities[chr.sortedmass[chr.mass.Length - 1] - 1].y - cities[chr.sortedmass[0] - 1].y), 2));

return sum;

}

}

}

Главная форма:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace Лаба\_4\_ИИ

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm();

private void button1\_Click(object sender, EventArgs e)

{

int kolChromosome = Convert.ToInt32(textBox3.Text);

int kolIteration = Convert.ToInt32(textBox2.Text);

double VerMyta = Convert.ToDouble(textBox6.Text);

double VerCrosengove = Convert.ToDouble(textBox5.Text);

int[,] mass = new int[48, 3];

var infile = new StreamReader("1.txt");

int i = 0;

while (!infile.EndOfStream)

{

var line = infile.ReadLine();

var ints = line.Split(' ');

for (int j = 0; j < 3; j++)

{

mass[i, j] = Convert.ToInt32(ints[j]);

}

i++;

}

GeneticAlgorithm algorithm = new GeneticAlgorithm();

algorithm.init(48, kolChromosome, mass);

List<Chromosome> chromosomes = algorithm.algorithm(kolIteration, VerCrosengove, VerMyta);

Chromosome c = chromosomes.Find(chr => chr.length == chromosomes.Min(Chromosome => Chromosome.length));

double best = chromosomes.Min(Chromosome => Chromosome.length);

textBox1.Clear();

textBox1.AppendText(algorithm.textAll + Environment.NewLine);

textBox1.AppendText(Environment.NewLine + "Финальный геном:" + Environment.NewLine);

for (int j = 0; j < chromosomes.Count; j++)

{

textBox1.AppendText("Хромосома №" + (j + 1) + " длина: " + chromosomes[j].length + Environment.NewLine);

}

chart1.Series.Clear();

chart1.Series.Add("Line");

for (int j = 0; j < algorithm.cities.Length - 1; j++)

{

chart1.Series["Line"].Points.Add(new DataPoint(algorithm.cities[c.sortedmass[j] - 1].x, algorithm.cities[c.sortedmass[j] - 1].y));

chart1.Series["Line"].ChartType = SeriesChartType.Line;

}

chart1.Series.Add("nLine");

chart1.Series["nLine"].Points.Add(new DataPoint(algorithm.cities[c.sortedmass[0] - 1].x, algorithm.cities[c.sortedmass[0] - 1].y));

chart1.Series["nLine"].Points.Add(new DataPoint(algorithm.cities[c.sortedmass.Last() - 1].x, algorithm.cities[c.sortedmass.Last() - 1].y));

chart1.Series["nLine"].Color = Color.Red;

chart1.Series["nLine"].ChartType = SeriesChartType.Line;

textBox1.AppendText(Environment.NewLine + "Лучшая длина: " + best.ToString() + Environment.NewLine);

}

private void button2\_Click(object sender, EventArgs e)

{

}

public static double getChromosomeValue(string s, double minInterval, double maxInterval)

{

int sum = 0;

double c, znach = 0;

for (int i = 0; i < s.Count(); i++)

{

c = char.GetNumericValue(s[i]);

znach = Math.Pow(2, s.Count() - i - 1);

sum += (int)(c \* znach);

}

return minInterval + sum \* ((maxInterval - minInterval) / (Math.Pow(2, s.Count()) - 1));

}

private void chart1\_Click(object sender, EventArgs e)

{

}

}

}