Прізвище: Метельський

Ім’я: Всеволод

Група: КНМ-14

Дата прийняття роботи

у системі Git: 24.04.2017

Дисципліна: Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні

Перевірив: Кривий Р.З.

**Звіт до лабораторної роботи № 5**

**«Рішення задачі комівояжера за допомогою генетичного алгоритму»**

**МЕТА РОБОТИ**

Ознайомитися з основними теоретичними відомостями про задачу комівояжера, обрати середовище розробки та мову програмування, реалізувати вирішення задачі комівояжера для [10; 50] міст за допомогою генетичного алгоритму.

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Задача комівояжера (англ. Travelling Salesman Problem)** полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.   
  
Існує маса різновидів узагальненої постановки задачі, зокрема геометрична задача комівояжера (коли матриця відстаней відображає відстані між точками на площині), трикутна задача комівояжера (коли на матриці вартостей виконується нерівність трикутника), симетрична та асиметрична задачі комівояжера.   
  
Прості методи розв'язання задачі комівояжера: повний лексичний перебір, жадібні алгоритми (метод найближчого сусіда), метод включення найближчого міста, метод найдешевшого включення, метод мінімального кістяка дерева. На практиці застосовують різні модифікації ефективніших методів: метод гілок і меж і метод генетичних алгоритмів, а так само алгоритм мурашиної колонії.   
  
Всі ефективні (такі, що скорочують повний перебір) методи розв'язання задачі комівояжера — евристичні. У більшості евристичних методів знаходиться не найефективніший маршрут, а наближений розв'язок. Користуються популярністю так звані any-time алгоритми, тобто алгоритми, що поступово покращують деякий поточний наближений розв'язок.   
  
Задача комівояжера — NP-повна. Часто на ній проводять випробування нових підходів до евристичного скорочення повного перебору.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

За допомогою засобів мови програмування C# розробити програмне забезпечення для розв’язку задачі комівожера.

**Код програми:**

**Form1.cs:**

using System;

using System.Windows.Forms;

namespace TravelingSalesman

{

public partial class Form1 : Form

{

static Random rand = new Random();

public Form1()

{

InitializeComponent();

}

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

{

TourManager.clearCities();

int cityAmount = rand.Next(10, 25);

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

TourManager.addCity(new City());

Population pop = new Population(50, true);

resultsLabel.Text = String.Empty;

resultsLabel.Text = "Initial distance: " + pop.getFittest().getDistance().ToString("F2");

pop = GA.evolvePopulation(pop);

for (int i = 0; i < 100; i++)

{

pop = GA.evolvePopulation(pop);

}

resultsLabel.Text += "\nFinal distance: " + pop.getFittest().getDistance().ToString("F2");

Tour best = pop.getFittest();

resultsChart.Series[0].Points.Clear();

resultsLabel.Text += "\nSolution:";

for (int i = 0; i < best.tourSize(); i++)

{

resultsChart.Series[0].Points.AddXY(best.getCity(i).getX(), best.getCity(i).getY());

resultsChart.Series[0].Points[i].Label = (i+1).ToString();

resultsLabel.Text += "\n"+(i + 1).ToString() + "). " + best.getCity(i).getX() +"; " + best.getCity(i).getY();

}

resultsChart.Series[0].Points.AddXY(best.getCity(0).getX(), best.getCity(0).getY());

}

}

}

**GA.cs:**

using System;

namespace TravelingSalesman

{

public class GA

{

private static double mutationRate = 0.015;

private static int tournamentSize = 5;

private static bool elitism = true;

public static Population evolvePopulation(Population pop)

{

Population newPopulation = new Population(pop.populationSize(), false);

int elitismOffset = 0;

if (elitism)

{

newPopulation.saveTour(0, pop.getFittest());

elitismOffset = 1;

}

for (int i = elitismOffset; i < newPopulation.populationSize(); i++)

{

Tour parent1 = tournamentSelection(pop);

Tour parent2 = tournamentSelection(pop);

Tour child = crossover(parent1, parent2);

newPopulation.saveTour(i, child);

}

for (int i = elitismOffset; i < newPopulation.populationSize(); i++)

{

mutate(newPopulation.getTour(i));

}

return newPopulation;

}

public static Tour crossover(Tour parent1, Tour parent2)

{

Tour child = new Tour();

int startPos = new Random().Next(0, 2) \* parent1.tourSize(),

endPos = new Random().Next(0, 2) \* parent1.tourSize();

for (int i = 0; i < child.tourSize(); i++)

{

if (startPos < endPos && i > startPos && i < endPos)

{

child.setCity(i, parent1.getCity(i));

}

else if (startPos > endPos)

{

if (!(i < startPos && i > endPos))

{

child.setCity(i, parent1.getCity(i));

}

}

}

for (int i = 0; i < parent2.tourSize(); i++)

{

if (!child.containsCity(parent2.getCity(i)))

{

for (int ii = 0; ii < child.tourSize(); ii++)

{

if (child.getCity(ii) == null)

{

child.setCity(ii, parent2.getCity(i));

break;

}

}

}

}

return child;

}

private static void mutate(Tour tour)

{

for (int tourPos1 = 0; tourPos1 < tour.tourSize(); tourPos1++)

{

if (new Random().Next(0, 2) < mutationRate)

{

int tourPos2 = new Random().Next(0, tour.tourSize());

City city1 = tour.getCity(tourPos1);

City city2 = tour.getCity(tourPos2);

tour.setCity(tourPos2, city1);

tour.setCity(tourPos1, city2);

}

}

}

private static Tour tournamentSelection(Population pop)

{

Population tournament = new Population(tournamentSize, false);

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

{

int randomId = new Random().Next(0, pop.populationSize());

tournament.saveTour(i, pop.getTour(randomId));

}

Tour fittest = tournament.getFittest();

return fittest;

}

}

}

**Population.cs:**

namespace TravelingSalesman

{

public class Population

{

Tour[] tours;

public Population(int populationSize, bool initialise)

{

tours = new Tour[populationSize];

if (initialise)

{

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

{

Tour newTour = new Tour();

newTour.generateIndividual();

saveTour(i, newTour);

}

}

}

public void saveTour(int index, Tour tour)

{

tours[index] = tour;

}

public Tour getTour(int index)

{

return tours[index];

}

public Tour getFittest()

{

Tour fittest = tours[0];

for (int i = 1; i < populationSize(); i++)

{

if (fittest.getFitness() <= getTour(i).getFitness())

{

fittest = getTour(i);

}

}

return fittest;

}

public int populationSize()

{

return tours.Length;

}

}

}

**Tour.cs:**

using System;

using System.Collections.Generic;

namespace TravelingSalesman

{

public class Tour

{

private List<City> tour = new List<City>();

private double fitness = 0;

private double distance = 0;

public Tour()

{

for (int i = 0; i < TourManager.numberOfCities(); i++)

{

tour.Add(null);

}

}

public Tour(List<City> tour)

{

this.tour = tour;

}

public void generateIndividual()

{

for (int cityIndex = 0; cityIndex < TourManager.numberOfCities(); cityIndex++)

{

setCity(cityIndex, TourManager.getCity(cityIndex));

}

ShuffleCities(tour);

}

List<City> ShuffleCities(List<City> array)

{

Random r = new Random();

for (int i = array.Count; i > 0; i--)

{

City firstCity = array[r.Next(i)],

secondCity = array[r.Next(i)],

temp = firstCity;

firstCity = secondCity;

secondCity = temp;

}

return array;

}

public City getCity(int tourPosition)

{

return tour[tourPosition];

}

public void setCity(int tourPosition, City city)

{

tour[tourPosition] = city;

fitness = 0;

distance = 0;

}

public double getFitness()

{

if (fitness == 0)

{

fitness = 1 / (double)getDistance();

}

return fitness;

}

public double getDistance()

{

if (distance == 0)

{

double tourDistance = 0;

for (int cityIndex = 0; cityIndex < tourSize(); cityIndex++)

{

City fromCity = getCity(cityIndex);

City destinationCity;

if (cityIndex + 1 < tourSize())

{

destinationCity = getCity(cityIndex + 1);

}

else

{

destinationCity = getCity(0);

}

tourDistance += fromCity.distanceTo(destinationCity);

}

distance = tourDistance;

}

return distance;

}

public int tourSize()

{

return tour.Count;

}

public bool containsCity(City city)

{

return tour.Contains(city);

}

}

}

**TourManager.cs:**

using System.Collections.Generic;

using System.Linq;

namespace TravelingSalesman

{

class TourManager

{

private static List<City> destinationCities = new List<City>();

public static void addCity(City city)

{

destinationCities.Add(city);

}

public static City getCity(int index)

{

return destinationCities[index];

}

public static int numberOfCities()

{

return destinationCities.Count();

}

public static void clearCities()

{

destinationCities.Clear();

}

}

}

**City.cs:**

using System;

namespace TravelingSalesman

{

public class City

{

int x, y;

static Random rand = new Random();

public City()

{

x = rand.Next(0, 301);

y = rand.Next(0, 301);

}

public City(int x, int y)

{

this.x = x;

this.y = y;

}

public int getX()

{

return x;

}

public int getY()

{

return y;

}

public double distanceTo(City city)

{

int xDistance = Math.Abs(getX() - city.getX());

int yDistance = Math.Abs(getY() - city.getY());

double distance = Math.Sqrt((xDistance \* xDistance) + (yDistance \* yDistance));

return distance;

}

public override string ToString()

{

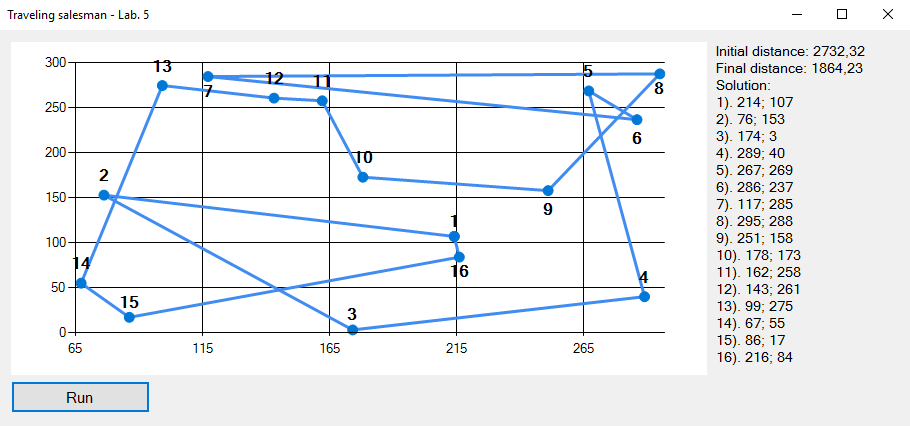
return getX() + ", " + getY();

}

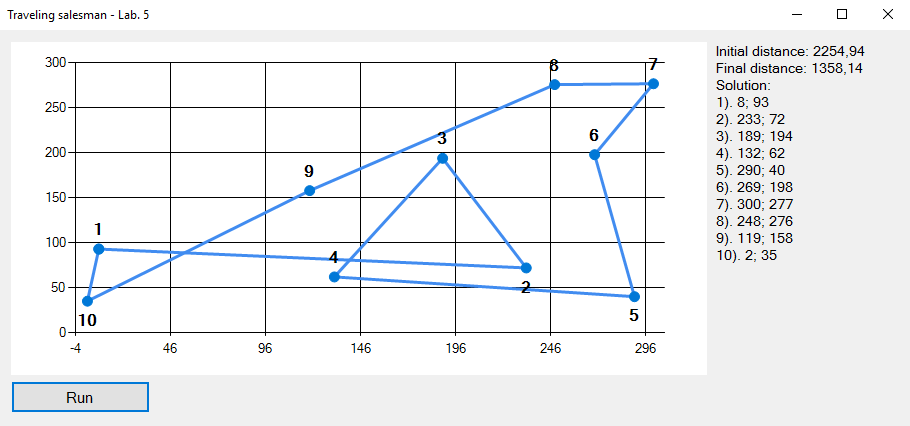
}

}

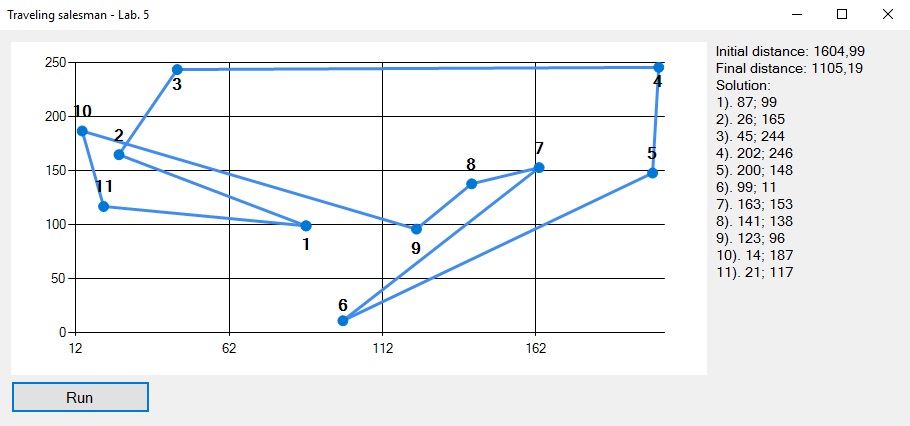
**Отримані результати:**



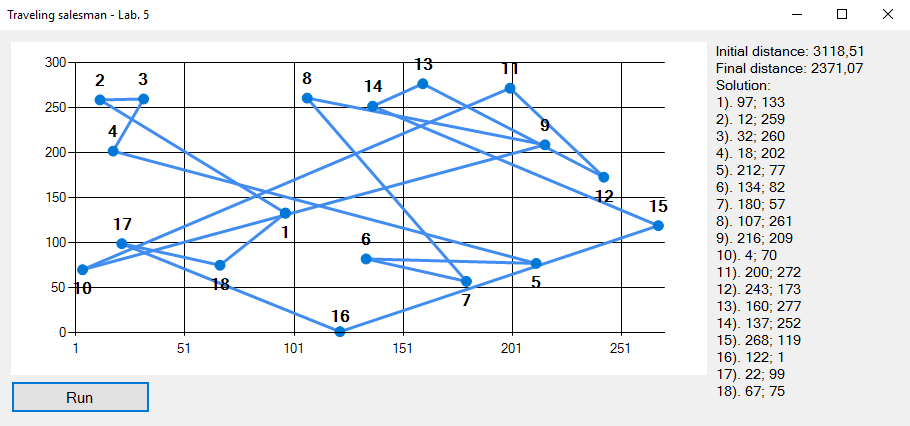
**Рис 1.** Розв`язок задачі при першому наборі випадково згенерованих міст.



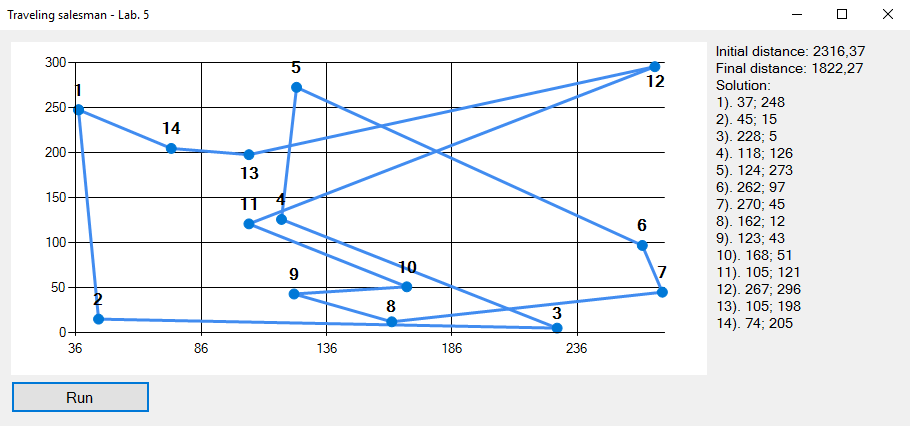
**Рис 2.** Розв`язок задачі при другому наборі випадково згенерованих міст.



**Рис 3.** Розв`язок задачі при третьому наборі випадково згенерованих міст.



**Рис 4.** Розв`язок задачі при четвертому наборі випадково згенерованих міст.



**Рис 5.** Розв`язок задачі при п`ятому наборі випадково згенерованих міст.

**Висновок:** виконуючи дану лабораторну роботу я ознайомився із теоретичними відомостями, а також, використовуючи засоби мови програмування C# та середовища розробки Visual Studio 2015, розробив програмне забезпечення для розв’язку задачі комівояжера за допомогою генетичного алгоритму.