Прізвище: Ладченко

Ім’я: Олена

Група: КНС-13

Дата прийняття роботи

у системі Git: ХХ.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 комівояжераполягає 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. 

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

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

**Код програми:**

**Cities.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.IO;

using System.Globalization;

namespace Tsp

{

public class Cities : List<City>

{

public void CalculateCityDistances( int numberOfCloseCities )

{

foreach (City city in this)

{

city.Distances.Clear();

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

{

city.Distances.Add(Math.Sqrt(Math.Pow((double)(city.Location.X - this[i].Location.X), 2D) +

Math.Pow((double)(city.Location.Y - this[i].Location.Y), 2D)));

}

}

foreach (City city in this)

{

city.FindClosestCities(numberOfCloseCities);

}

}

public void OpenCityList(string fileName)

{

DataSet cityDS = new DataSet();

try

{

this.Clear();

cityDS.ReadXml(fileName);

DataRowCollection cities = cityDS.Tables[0].Rows;

foreach (DataRow city in cities)

{

this.Add(new City(Convert.ToInt32(city["X"], CultureInfo.CurrentCulture), Convert.ToInt32(city["Y"], CultureInfo.CurrentCulture)));

}

}

finally

{

cityDS.Dispose();

}

}

}

}

**Population.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

namespace Tsp

{

class Population : List<Tour>

{

private Tour bestTour = null;

public Tour BestTour

{

set

{

bestTour = value;

}

get

{

return bestTour;

}

}

public void CreateRandomPopulation(int populationSize, Cities cityList, Random rand, int chanceToUseCloseCity)

{

int firstCity, lastCity, nextCity;

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

{

Tour tour = new Tour(cityList.Count);

firstCity = rand.Next(cityList.Count);

lastCity = firstCity;

for (int city = 0; city < cityList.Count - 1; city++)

{

do

{

if ((rand.Next(100) < chanceToUseCloseCity) && ( cityList[city].CloseCities.Count > 0 ))

{

nextCity = cityList[city].CloseCities[rand.Next(cityList[city].CloseCities.Count)];

}

else

{

nextCity = rand.Next(cityList.Count);

}

} while ((tour[nextCity].Connection2 != -1) || (nextCity == lastCity));

tour[lastCity].Connection2 = nextCity;

tour[nextCity].Connection1 = lastCity;

lastCity = nextCity;

}

tour[lastCity].Connection2 = firstCity;

tour[firstCity].Connection1 = lastCity;

tour.DetermineFitness(cityList);

Add(tour);

if ((bestTour == null) || (tour.Fitness < bestTour.Fitness))

{

BestTour = tour;

}

}

}

}

}

**Tsp.cs**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using System.Text;

using System.Data;

using System.Drawing;

namespace Tsp

{

class Tsp

{

public delegate void NewBestTourEventHandler(Object sender, TspEventArgs e);

public event NewBestTourEventHandler foundNewBestTour;

Random rand;

Cities cityList;

Population population;

private bool halt = false;

public bool Halt

{

get

{

return halt;

}

set

{

halt = value;

}

}

public Tsp()

{

}

public void Begin(int populationSize, int maxGenerations, int groupSize, int mutation, int seed, int chanceToUseCloseCity, Cities cityList)

{

rand = new Random(seed);

this.cityList = cityList;

population = new Population();

population.CreateRandomPopulation(populationSize, cityList, rand, chanceToUseCloseCity);

displayTour(population.BestTour, 0, false);

bool foundNewBestTour = false;

int generation;

for (generation = 0; generation < maxGenerations; generation++)

{

if (Halt)

{

break;

}

foundNewBestTour = makeChildren(groupSize, mutation);

if (foundNewBestTour)

{

displayTour(population.BestTour, generation, false);

}

}

displayTour(population.BestTour, generation, true);

}

bool makeChildren(int groupSize, int mutation)

{

int[] tourGroup = new int[groupSize];

int tourCount, i, topTour, childPosition, tempTour;

for (tourCount = 0; tourCount < groupSize; tourCount++)

{

tourGroup[tourCount] = rand.Next(population.Count);

}

for (tourCount = 0; tourCount < groupSize - 1; tourCount++)

{

topTour = tourCount;

for (i = topTour + 1; i < groupSize; i++)

{

if (population[tourGroup[i]].Fitness < population[tourGroup[topTour]].Fitness)

{

topTour = i;

}

}

if (topTour != tourCount)

{

tempTour = tourGroup[tourCount];

tourGroup[tourCount] = tourGroup[topTour];

tourGroup[topTour] = tempTour;

}

}

bool foundNewBestTour = false;

childPosition = tourGroup[groupSize - 1];

population[childPosition] = Tour.Crossover(population[tourGroup[0]], population[tourGroup[1]], cityList, rand);

if (rand.Next(100) < mutation)

{

population[childPosition].Mutate(rand);

}

population[childPosition].DetermineFitness(cityList);

if (population[childPosition].Fitness < population.BestTour.Fitness)

{

population.BestTour = population[childPosition];

foundNewBestTour = true;

}

childPosition = tourGroup[groupSize - 2];

population[childPosition] = Tour.Crossover(population[tourGroup[1]], population[tourGroup[0]], cityList, rand);

if (rand.Next(100) < mutation)

{

population[childPosition].Mutate(rand);

}

population[childPosition].DetermineFitness(cityList);

if (population[childPosition].Fitness < population.BestTour.Fitness)

{

population.BestTour = population[childPosition];

foundNewBestTour = true;

}

return foundNewBestTour;

}

void displayTour(Tour bestTour, int generationNumber, bool complete)

{

if (foundNewBestTour != null)

{

this.foundNewBestTour(this, new TspEventArgs(cityList, bestTour, generationNumber, complete));

}

}

}

}

**Отримані результати:**

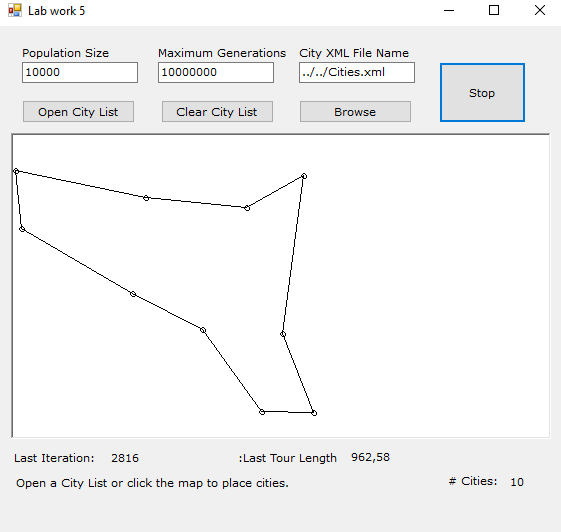


Рис 1. Розв`язок задачі при наявності десяти міст

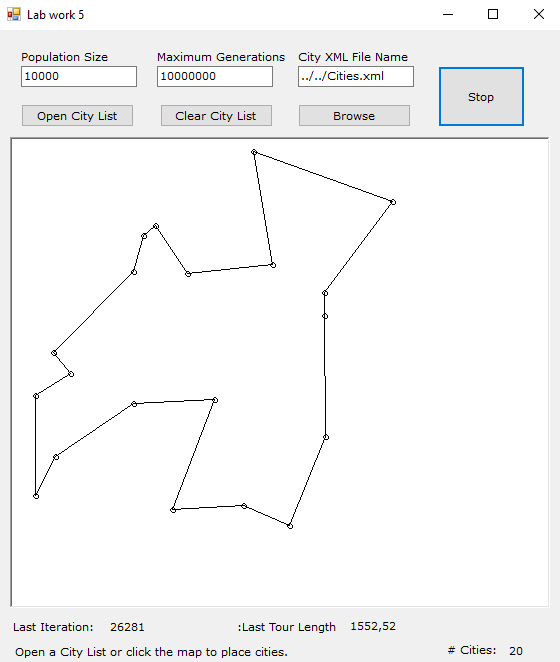


Рис 2. Розв`язок задачі при наявності двадцяти міст

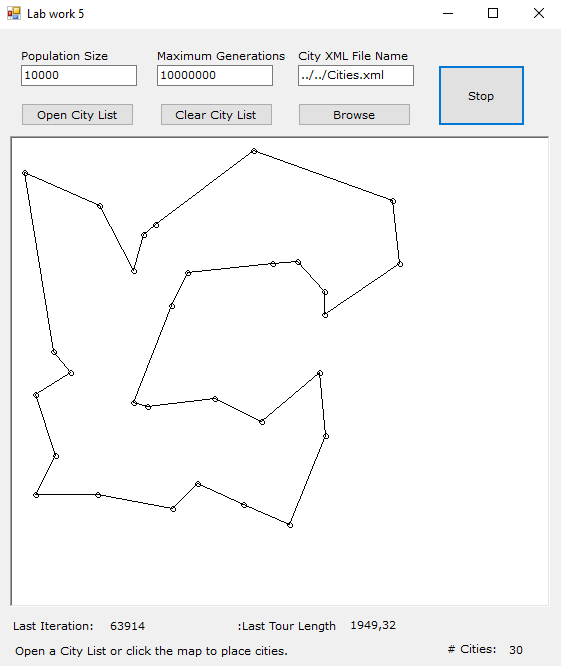


Рис 3. Розв`язок задачі при наявності тридцяти міст.

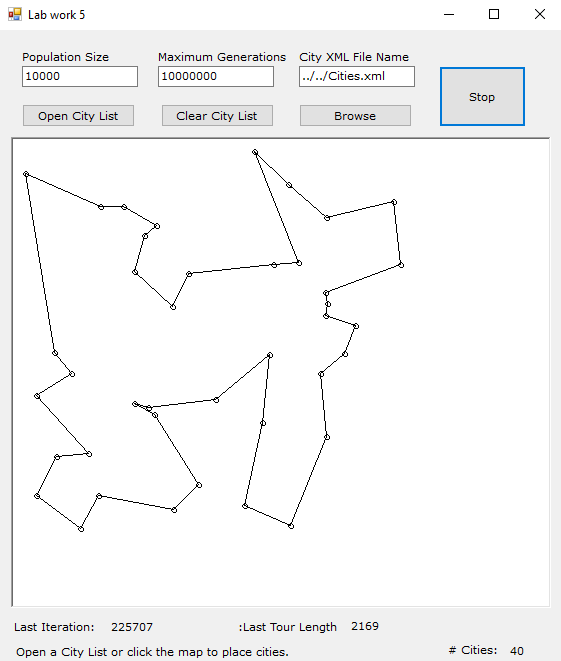


Рис 4. Розв`язок задачі при наявності сорока міст.

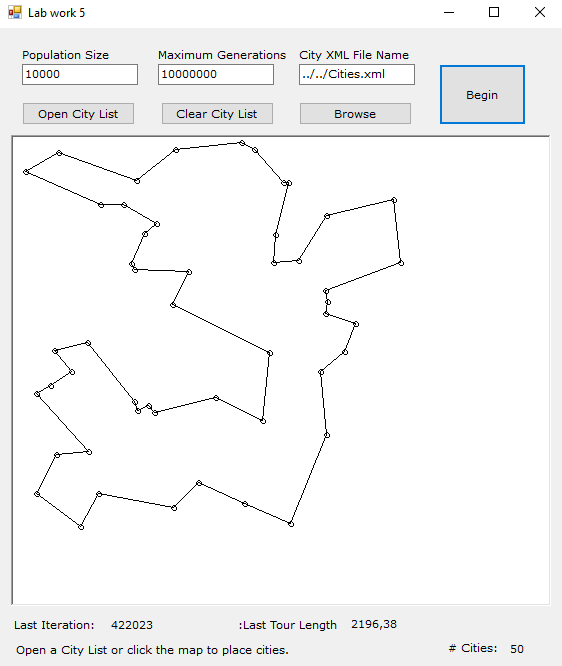


Рис 5. Розв`язок задачі при наявності п’ятдесяти міст.

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