ELECRONIC，ELECTRICAL AND COMPUTER ENGINEERING



**Object Oriented Programming Using C#**

**Assignment 2014-15**

**Fan Jingqiao**

**ID： 1429155**

Contents

[1. Introduction 3](#_Toc405167035)

[2. Use-Case Model 4](#_Toc405167036)

[2.1 Use-case view 5](#_Toc405167037)

[2.2 Survey description 5](#_Toc405167038)

[2.3 Interaction 5](#_Toc405167039)

[2.4 State-chart 7](#_Toc405167040)

[2.5 Activity diagram 8](#_Toc405167041)

[2.6 CRC Cards 8](#_Toc405167042)

[3. Analysis Model 9](#_Toc405167043)

[3.1 Attributes and methods 9](#_Toc405167044)

[3.2 Sequence Diagram 10](#_Toc405167045)

[3.3 Class Diagram 10](#_Toc405167046)

[3.4 State Chart Diagram 10](#_Toc405167047)

[3.5 Non-fuctional Requirements 10](#_Toc405167048)

[3.6 Packages 11](#_Toc405167049)

[4. Design Model 18](#_Toc405167050)

[4.1 Revisit Use-Case Model 18](#_Toc405167051)

[4.2 Sequence Diagram 18](#_Toc405167052)

[4.3 Textual Description of Object to Object Interaction 18](#_Toc405167053)

[4.4 Subsystems 18](#_Toc405167054)

[4.5 Implementation of Non-functional Requirements 18](#_Toc405167055)

[4.7 Reconsider the attributes 19](#_Toc405167056)

[4.8 Reconsider the Association 19](#_Toc405167057)

[4.9 StateChart 19](#_Toc405167058)

[4.10 Class Design Diagram 19](#_Toc405167059)

[5. Test 21](#_Toc405167060)

[5.1 Initial window 21](#_Toc405167061)

[5.2 Load data file 22](#_Toc405167062)

[5.3 ACO solver 23](#_Toc405167063)

[5.4 GREEDY solver 24](#_Toc405167064)

[6. Conclusion 24](#_Toc405167065)

**1 Introduction**

This program is aimed at solving the issue of ant colony optimization (ACO).The ACO is a kind of bionic algorithm generated from the simulation on the hunting food for ants. In the design, there will be many cities, and the ant should visit to all the cities to get its food, then working out the shortest route between the food and the anthill and output the tour length. The task is how to find out the best route. To deal with it , a concept called pheromone should be introduced. It is released by ants, and the quantity of releasing pheromone depends on the distance between the two cities. The longer the distance is, the less the pheromone will be released. With this mechanism, the following ants can easily choose the shorter route.

To complete the design, I suppose to divide the whole project into three bid parts. The first one is the ants, which is the actor in the design. Seeing from the outside, all the actions were done by ants. The second one is the cities. It looks like the skeleton of the design. They are the targets of the ants. And different locations of the cities may output a different result. The last one is pheromone, which could be used to build a feedback mechanism for the system. The more tests are implemented, the more precise the pheromone will be. It is one of the most important message I the whole design.

Above all is the explanation for the design. To complete the whole design, there are still many specific issues such as how to present the pheromone on a certain route, how do the ant judge the pheromone to choose the route, how to calculate the tour length for an certain ant and so on. Those will be discussed in detailed in the following parts.

**2 Use-Case Model**

This part is to define what the customer wants.

2.1 Use-case view

Figure 1. Use-case diagram

In the diagram, we can easily find out the object of in the design. They are cities, ants, and pheromone. But the picture can only show a simple relationship between each class. The specific interaction of them will de descripted in the following parts.

2.2 Survey description

First the customer should define the cities in the coordinate system. Then a certain ant should start its tour at one city and release the pheromone during the trip. The quantity of pheromone released related to the distance of the two adjacent cities. The other ants should take use of the pheromone left buy ants who had been here earlier to choose its next destination. And it will release the pheromone once again. At last, it is supposed to calculate the whole tour length after searching all the cities.

2.3 Interaction

This part will show how the classes are associated.

Figure 2. Collaboration diagram

In this picture, we can not only see the classes of the design , but can also read the message sent from one object to another. Which can explain how the association was built. First, we can see that the system is the core in this picture, and it is also the thing I am going to design. Every ant is controlled by it, and its tour will be recorded in the system after being to a certain city. What is more, it will update the information of pheromone on each route for ants to choose their next station. What is more, the system still can record the tour length for ants which is not displayed in the picture.

2.4State-chart Figure 3. State diagram

In this diagram, we can easily know how does the program progress. In fact, each visiting a city are all treated as a single state. There are as many states as the number of the cities. After visiting all the cities, the progress will end.

2.5 Activity diagram

Figure 4. Activity diagram

In this diagram I only set three cities to show the principle of choosing cities for ants. First of all, I should judge whether the ant has been to the city. If it has been there, I will make the ant visit another adjacent cities (in the picture, I just use city2 and city3 to present the adjacent cities). If it did not visit to the city before, I should refer to the other actors—pheromone. I should compare the quantity of all the adjacent cities. As introduced above, the longer distance, the less pheromone will be left. So I need to make the ant to choose the city with the most pheromone as its next destination.

2.6 CRC Cards

|  |  |
| --- | --- |
| Cities | |
| Responsiblities | Collaborators |
| The cities react on the program as a target for ants. The distribution of them decides the ants searching route. So it is one of the most important variables in the design. | Ants  Pheromone  System |
| Ants | |
| Responsiblities | Collaborators |
| Ant is the main character in the design. Most of the actions are completed by it, and still most of the messages are sent by it. Ant should visit all the cities, should release the pheromone, and distinguish them. It looks like a practitioner in the design. | Cities  Pheromone  System |
| Pheromone | |
| Responsiblities | Collaborators |
| Pheromone plays the role of the media in the design. It helps the ants to deliver the message. When an ant finished a short trip between two cities, it will leave a message to mark the relative distance. The message was load on pheromone. So it is very important in the progress of visiting all the countries in the shortest route. | Cities  Ants  System |
| System | |
| Responsiblities | Collaborators |
| The system is the core of the design. It will record the pheromone left buy ants and update it, then control the ants to find the nearest adjacent city. What is more, it should mark the city which the ant had been to avoid visiting the same city later. Also, it should calculate the distance the ants had visited, and work out the whole trip length. | Cities  Ants  Pheromone |

Figure 5. CRC Cards

**3 Analysis Model**

Creating this part is an incremental growth from the Use-case Model in which use-case realisations are merged.

3.1 Attributes and methods

In my design, there are four classes. I will introduce them in detail.

(1) Searching cities

Potential attributes: cities, ants, pheromone, system data(mark the city);

Potential methods: check the pheromone, check the system data, compare the probabilities;

(2) Visiting cities

Potential attributes: cities, ants, pheromone, system data(mark the city);

Potential methods: calculate the distance, release the pheromone according to the distance, send the message to the system( mark the city, quantity of pheromone);

(3) Updating the information of the map

Potential attribute: cities, pheromone, divisor of the pheromone, system data;

Potential methods: calculate the total pheromone for a certain city left by all ants, calculate the pheromone between two cities, update the mark for each cities;

(4) Outputting the tour length

Potential attribute: cities, a certain ant, system data

Potential methods: choose a city to start, calculate the distance between two cities, accumulate all the distances;

3.2 Sequence Diagram

There are two sequence diagram. The first one is used to describe the how a ant finish its tour. Enough details are included in it. The other one is used to describe how the system update the map information( mark for cities, quantity of pheromone).

3.3 Class Diagram

In this diagram, the interactions between the classes are shown in detail.

3.4 State Chart Diagram

The same with Sequence Diagram, there are two State Chart Diagrams. Also, the concentration of each one is the same with the two Sequence Diagrams.

3.5 Non-fuctional Requirements

There is no Non-fuctional Requirements in the design.

3.6 Packages

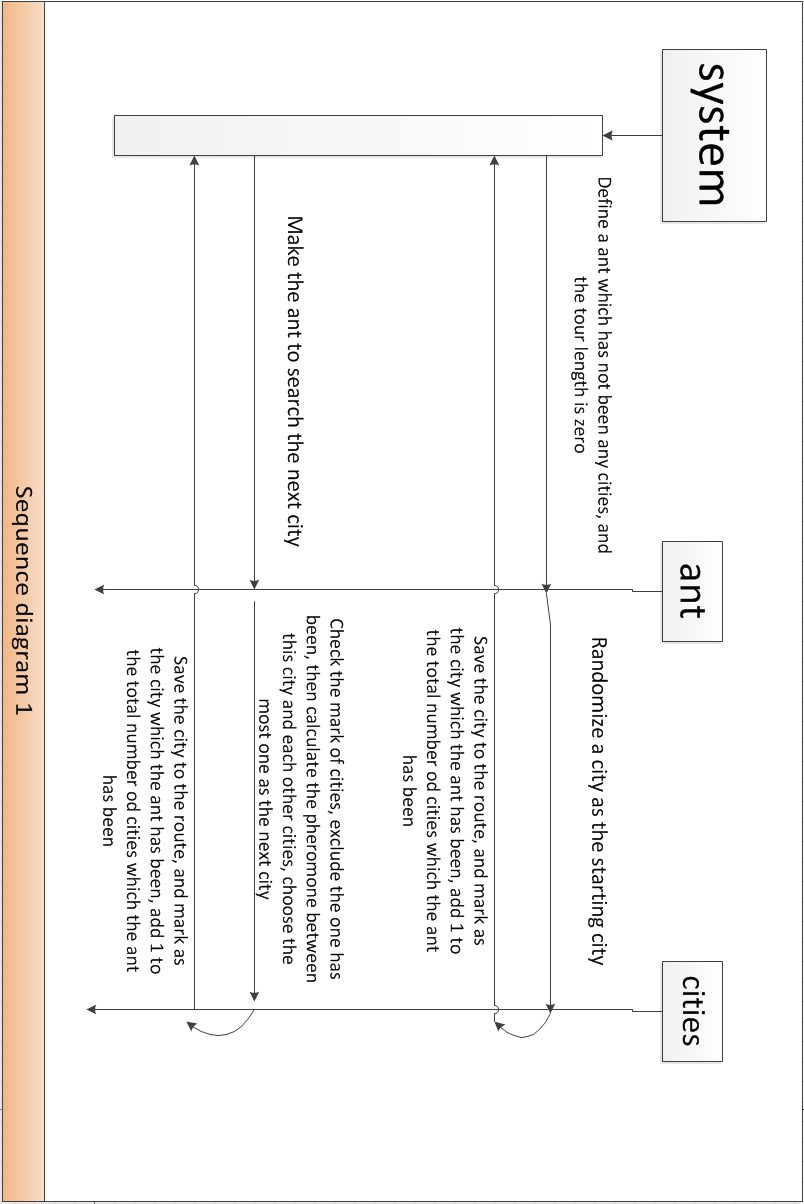
There is no packages in the design.

|  |  |  |
| --- | --- | --- |
| Class | Attributes | Comment |
| Searching cities | Cities | Targets |
| Ants | Objects |
| Pheromone | Messages |
| System data | Controller |
| Visiting cities | Cities | Targets |
| Ants | Objects |
| Pheromone | Messages |
| System data | Messages collector |
| Updating the information of the map | Cities | Loading the messages |
| Pheromone | Media |
| Divisor of the pheromone | Factors |
| System date | Messages collector |
| Outputting the tour length | Cities | Targets |
| A certain ant | Object |
| System data | Calculator |

Figure 6. Attributes table

|  |  |  |
| --- | --- | --- |
| Class | Method | Comment |
| Searching cities | check the pheromone | Compare the pheromone of adjacent cities |
| check the system data | To check whether has been to the city |
| compare the probabilities |  |
| Visiting cities | calculate the distance | Calculation |
| release the pheromone according to the distance | Function |
| send the message to the system( mark the city, quantity of pheromone) |  |
| Updating the information of the map | calculate the total pheromone for a certain city left by all ants | Function |
| calculate the pheromone between two cities | Function |
| update the mark for each cities | Status |
| Outputting the tour length | choose a city to start | Random |
| calculate the distance between two cities | Function |
| accumulate all the distances | Function |

Figure 7. Method table

 Figure 8. Analysis Model sequence diagram1

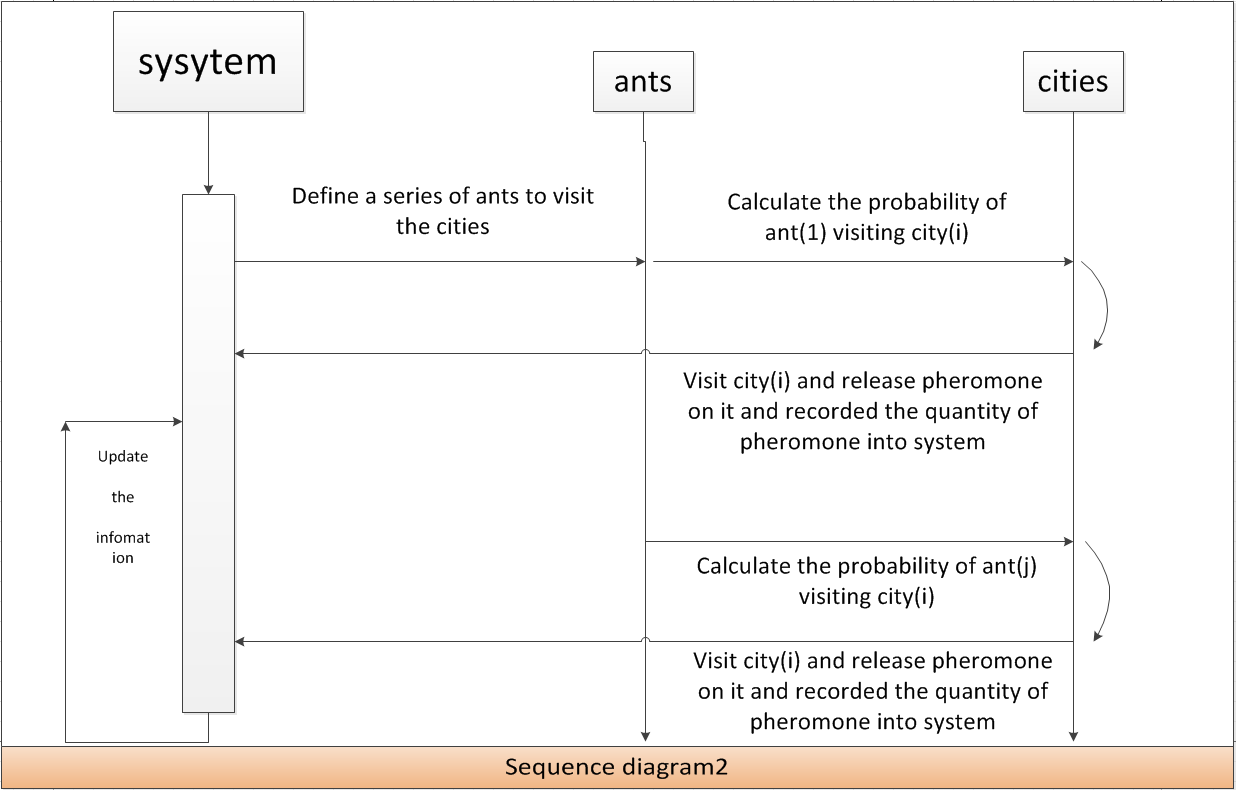
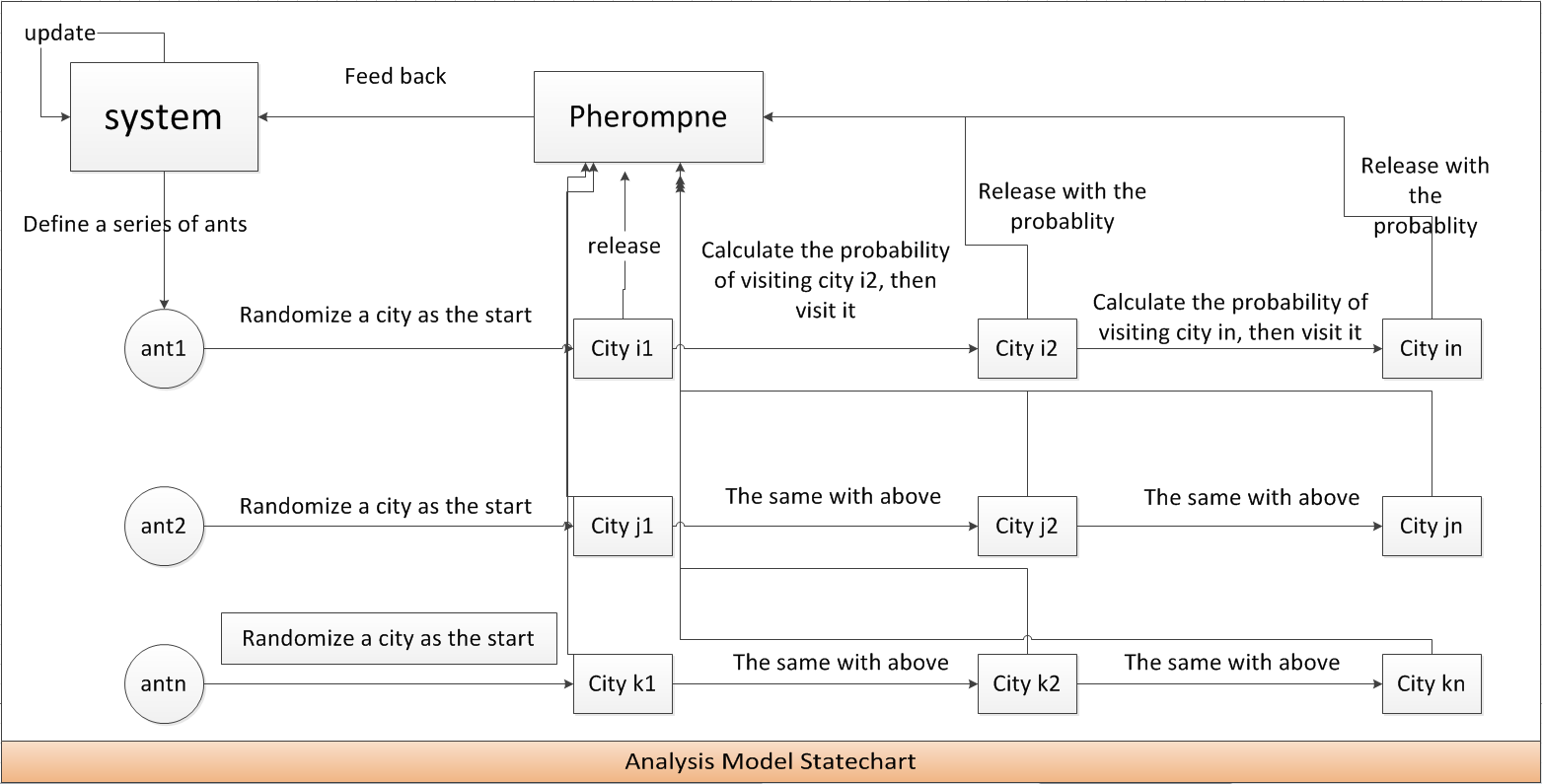
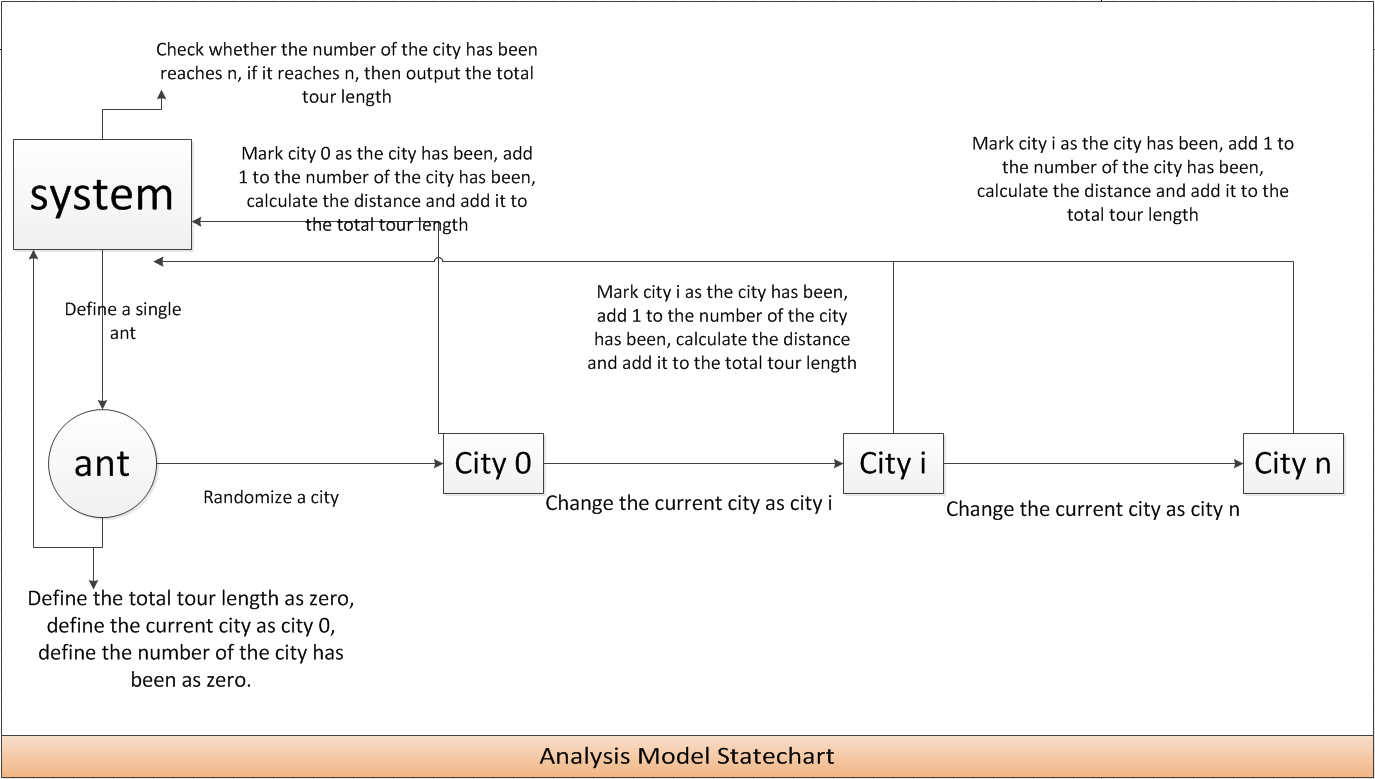


Figure 9. Analysis Model sequence diagram2



Figure 10. Analysis Model Class Diagram

 Figure 11. Analysis Model Statechart1

 Figure 12. Analysis Model Statechart2

Statechart1 describe how the system updates the information of the pheromone. As it shows, the system makes a lots of ants to visit cities and release the pheromone. The system will add all the ants’ pheromone. Then the pheromone will be more correctly to show the distance of the two adjacent cities.Statechar2 shows how to calculate the shortest tour of visiting all the cities. But it omit the progress of choosing city, which has been shown in Statechart1.

**4 Design Model**

4.1 Revisit Use-Case Model

The design meets the requirements.

4.2 Sequence Diagram

The sequence Diagram shown above has added the principal qualifiers.

4.3 Textual Description of Object to Object Interaction

The design is not so hard, and enough diagrams have been shown. And I will go to illustrate the classes in the design.

The core of the design is the system. It the center of control, and is also the center of information. First, it will manage the updating of map information, which is an important factor for ants choosing cities. Then, it will control the ant to read the message and choose its route.

The ants play the role of visitor in the design. Its duty is to visit the city with the controlling of system and release pheromone.

The cities are the destinations for ants. And it will store the pheromone for system.

4.4 Subsystems

There is no subsystem in the design for the design is not so complex.

4.5 Implementation of Non-functional Requirements

There is no Non-functional Requirements in the design.

4.6 Deployment Model

The TSP issue is implemented on a single processor.

4.7 Reconsider the attributes

4.8 Reconsider the Association

All the details have been shown in the sequence diagram above.

4.9 StateChart

There is no need for further details.

4.10 Class Design Diagram

Figure 13.Design Model Class Diagram

**5 Test**

In this part, I will show the pictures of running the program, and attach some explanation.

5.1 Initial window

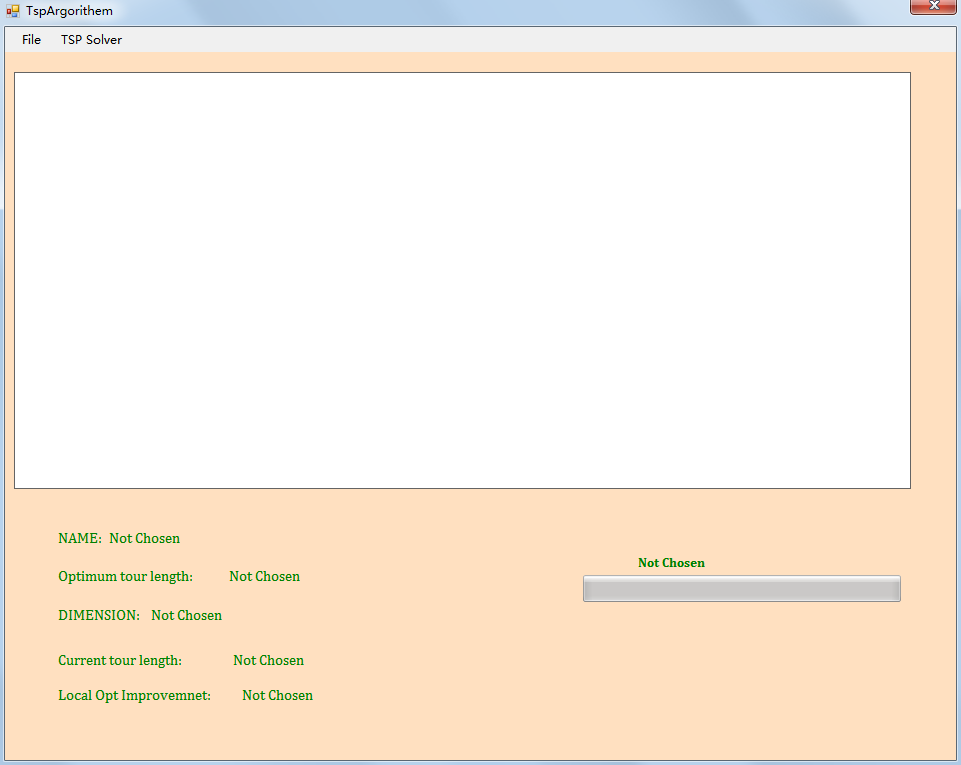


Figure 14.Initial window

This is the interface of the program. When open it first time, there is no data in it. So the map is empty, the name is shown as not chosen.

5.2 Load data file

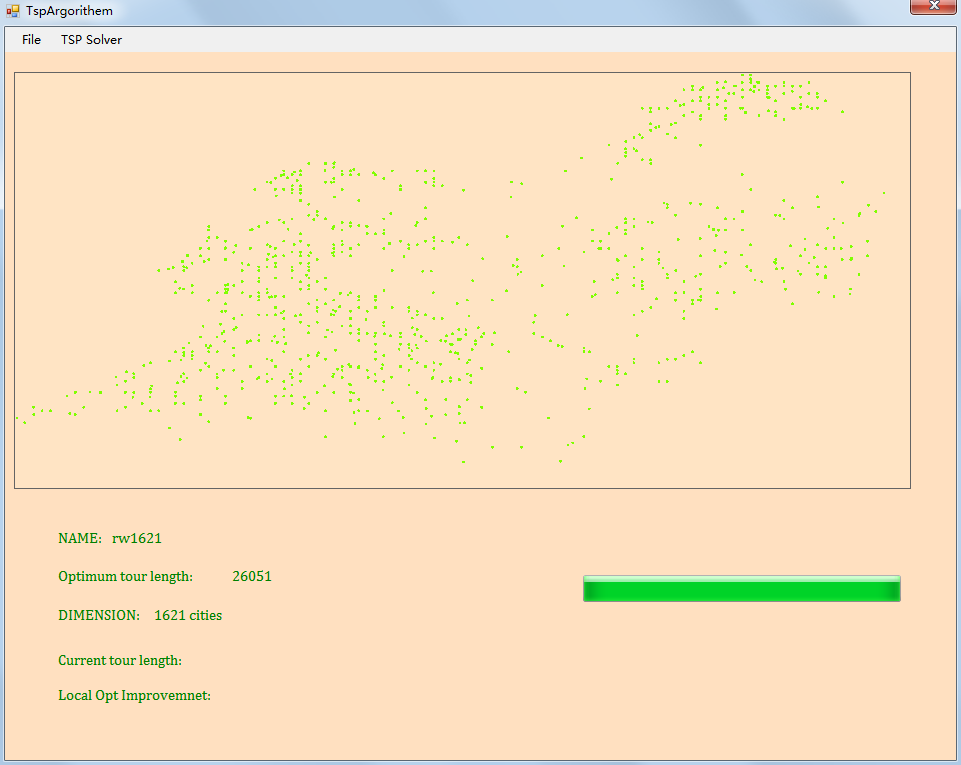


Figure 15.Load data file

After loading the city file, the cities will located in the map with its coordinate, the name will show the file’s name. The Dimension will show the number of the cities. The progress bar is full, which indicates that the loading has been finished.

5.3 ACO solver

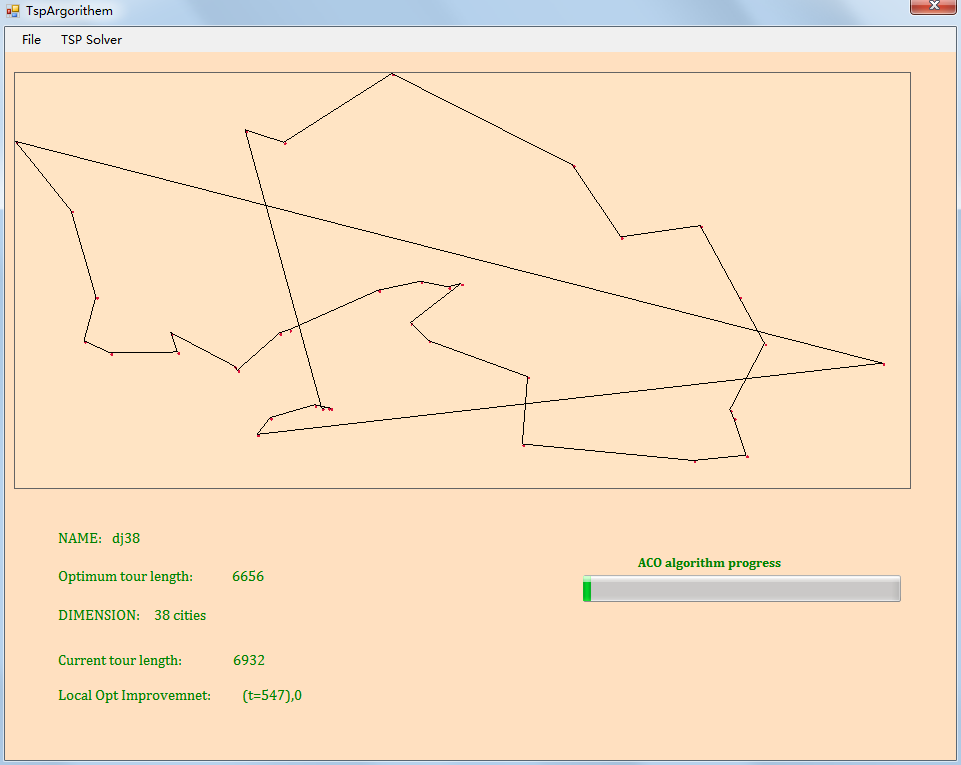


Figure 16.ACO progressing

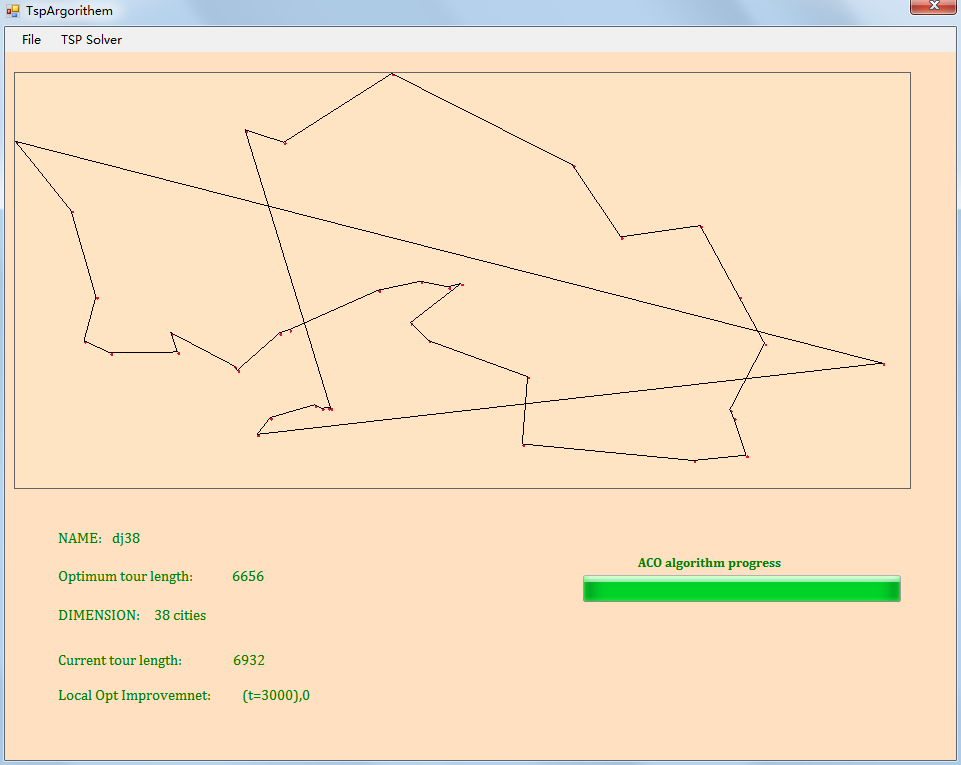


Figure 17.ACO finished

5.4 GREEDY solver

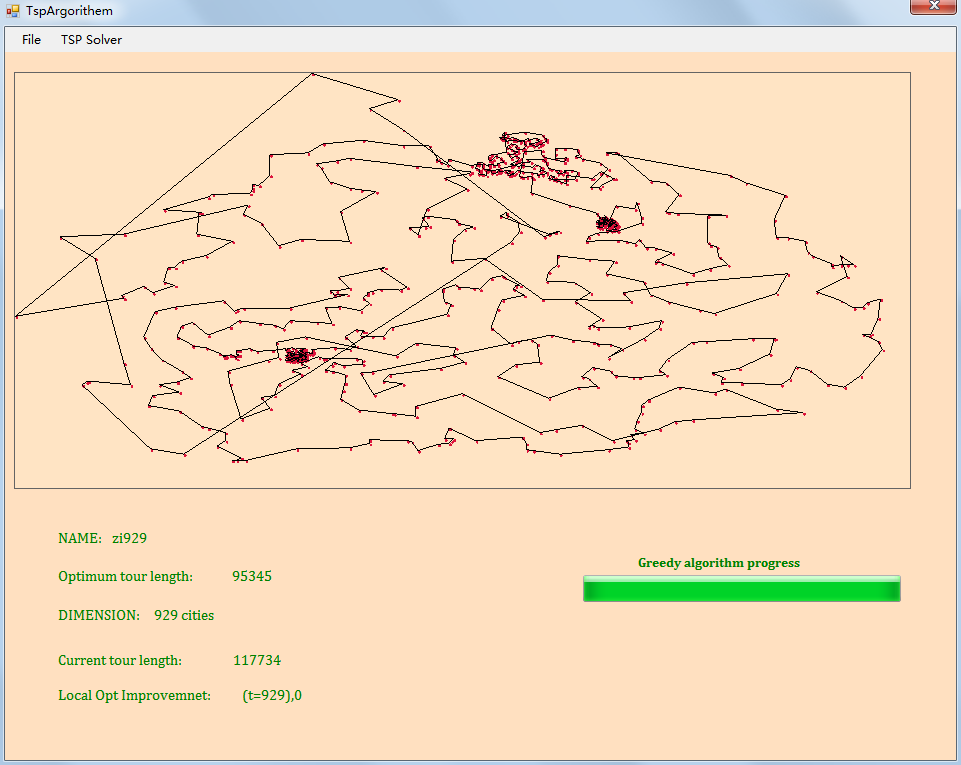


Figure 18.GREEDY solver finished

**6 Conclusion**

Though the design is a bit difficult foe, I still try my best to finish it. After the design, I gain a lot. The biggest improvement for me is to learn to make use of the “use case”, “analysis model” and “design model” to design a program. At the beginning, I can not imagine what the program would be like. While following the steps of use case”, “analysis model” and “design model”, the program appears clearly gradually. The way to design is very useful. And I will keep it for the designs in the future.

**Appendix**

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;

using System.Windows.Forms;

namespace TSPForm

{

public partial class TspArgorithem : Form

{

private int Cities;

private double[] xMinMax;

private double[] yMinMax;

private double[,] xy;

private double a;

private static double b;

private int[] goodPathIndex;

private double[,] CityDistance;

private int[] CityTabu;

private double shortesPathLenth = double.PositiveInfinity;

private double BeforeShortPathLegth = double.PositiveInfinity;

int[,] CityAntHaveTrave;

double[,] PheromoneTau;

double[,] etaCij;

double[,] deltaTau;

int[,] CitiesTabu;

int[,] CityNoTabu;

double[] mAntShortestDistance;

int AntCount;

int NcMax = 3000;

double Q = 100;

double alpha = 2.0;

double beta = 7.0;

double rho = 0.5;

int iHaveNoChange = 50;

int Nc = 0; //The current iteration number

double PhenomenonZero = 1e-5;

double PhenomenonRangeMax =30;

public TspArgorithem()

{

InitializeComponent();

}

private void TspArgorithem\_Load(object sender, EventArgs e)//Window initialization

{

labelCTLength.Text = "Not Chosen";

labelDimension.Text = "Not Chosen";

labelName.Text = "Not Chosen";

labelOPTLength.Text = "Not Chosen";

labelImprovement.Text = "Not Chosen";

labelProgress.Text = "Not Chosen";

}

private void readDataToolStripMenuItem\_Click(object sender, EventArgs e)//choose read data menu

{

labelCTLength.Text = "";

labelDimension.Text = "";

labelName.Text = "";

labelOPTLength.Text = "";

labelImprovement.Text = "";

labelProgress.Text = "";//clear up the display of label

OpenFileDialog myFileDialog = new OpenFileDialog();//menu for choosing data files

myFileDialog.InitialDirectory = Application.StartupPath + "\\Data";//The default path

xMinMax = new double[2] { double.MaxValue, 0 };

yMinMax = new double[2] { double.MaxValue, 0 };

if (myFileDialog.ShowDialog() == DialogResult.OK)//load data file

{

string strPath = myFileDialog.FileName;

StreamReader streamReader = new StreamReader(strPath);

string stringData;

string[] stringArray;

string dataFlag = "no";

int i = 0;

while ((stringData = streamReader.ReadLine()) != "EOF")

{

if (stringData.Contains("NAME"))

{

stringArray = stringData.Split(':');

labelName.Text = stringArray[1];

}

if (stringData.Contains("Optimum"))

{

int iLenthStartIndex;

iLenthStartIndex = stringData.IndexOf("is") + 2;

labelOPTLength.Text = stringData.Substring(iLenthStartIndex);

}

if (stringData.Contains("DIMENSION"))

{

stringArray = stringData.Split(':');

labelDimension.Text = stringArray[1] + " cities";

Cities = int.Parse(stringArray[1].Trim());

xy = new double[Cities, 2];

goodPathIndex = new int[Cities];

CityDistance = new double[Cities, Cities];

CityTabu = new int[Cities];

setBarValue(0, Cities);

}

if (dataFlag == "yes")

{

stringArray = stringData.Split(' ');

xy[i, 0] = double.Parse(stringArray[1]);

xy[i, 1] = double.Parse(stringArray[2]);

if (xMinMax[0] > xy[i, 0])

{

xMinMax[0] = xy[i, 0];

}

if (xMinMax[1] < xy[i, 0])

{

xMinMax[1] = xy[i, 0];

}

if (yMinMax[0] > xy[i, 1])

{

yMinMax[0] = xy[i, 1];

}

if (yMinMax[1] < xy[i, 1])

{

yMinMax[1] = xy[i, 1];

}

i++;

UpadateProgressBar(i);

}

if (stringData.Contains("NODE\_COORD\_SECTION"))

{

dataFlag = "yes";

}

}

streamReader.Close();

DrawCityPoint();

CalculateDistance();

}

}

private void CalculateDistance()//calculate the distance

{

setBarValue(0, Cities);

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

{

for (int k = 0; k < Cities; k++)

{

CityDistance[j, k] = Math.Sqrt((xy[j, 0] - xy[k, 0]) \* (xy[j, 0] - xy[k, 0]) + (xy[j, 1] - xy[k, 1]) \* (xy[j, 1] - xy[k, 1]));

}

UpadateProgressBar(j + 1);

}

}

public void UpadateProgressBar(int iBarValue)//update the progress bar

{

if (progressBarTSP.InvokeRequired)

{

Action<int> actionUpdateProgressBar = delegate(int iTemp)

{

progressBarTSP.Value = iTemp;

};

progressBarTSP.Invoke(actionUpdateProgressBar, iBarValue > progressBarTSP.Maximum ? progressBarTSP.Maximum : iBarValue);

}

else

{

progressBarTSP.Value = iBarValue;

}

}

public void setBarValue(int minValue, int maxValue)//set the progress bar

{

if (progressBarTSP.InvokeRequired)

{

Action<int, int> actionSetBarValue = delegate(int minTemp, int maxTemp)

{

progressBarTSP.Minimum = minTemp;

progressBarTSP.Maximum = maxTemp;

};

progressBarTSP.Invoke(actionSetBarValue, minValue, maxValue);

}

else

{

progressBarTSP.Minimum = minValue;

progressBarTSP.Maximum = maxValue;

}

}

private void DrawCityPoint()//cities located in the map

{

setBarValue(0, Cities);

if (pictureBoxCity.InvokeRequired)

{

Action actionDrawCityPoint = delegate()

{

double fPicboxWidth = pictureBoxCity.Width;

double fPicboxheight = pictureBoxCity.Height;

double PicAxisX, PicAxisY;

PicAxisX = (double)(fPicboxWidth - 30) / (xMinMax[1] - xMinMax[0]);

PicAxisY = (double)(fPicboxheight - 30) / (yMinMax[1] - yMinMax[0]);

SolidBrush RedBrush = new SolidBrush(Color.Chartreuse);

pictureBoxCity.Image = null;

Image imageTSP = new Bitmap(pictureBoxCity.Width, pictureBoxCity.Height);

Graphics GraphicsTSP = Graphics.FromImage(imageTSP);

GraphicsTSP.Clear(Color.Bisque);

double cityPointX;

double cityPointY;

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

{

cityPointX = (xy[j, 0] - xMinMax[0]) \* PicAxisX;

cityPointY = (xy[j, 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityPointX, (float)cityPointY, 3, 3));

UpadateProgressBar(j + 1);

}

pictureBoxCity.Image = imageTSP;

};

pictureBoxCity.Invoke(actionDrawCityPoint);

}

else

{

double fPicboxWidth = pictureBoxCity.Width;

double fPicboxheight = pictureBoxCity.Height;

double PicAxisX, PicAxisY;

PicAxisX = (double)(fPicboxWidth - 30) / (xMinMax[1] - xMinMax[0]);

PicAxisY = (double)(fPicboxheight - 30) / (yMinMax[1] - yMinMax[0]);

SolidBrush RedBrush = new SolidBrush(Color.Chartreuse);

pictureBoxCity.Image = null;

Image imageTSP = new Bitmap(pictureBoxCity.Width, pictureBoxCity.Height);

Graphics GraphicsTSP = Graphics.FromImage(imageTSP);

GraphicsTSP.Clear(Color.Bisque);

double cityPointX;

double cityPointY;

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

{

cityPointX = (xy[j, 0] - xMinMax[0]) \* PicAxisX;

cityPointY = (xy[j, 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityPointX, (float)cityPointY, 3, 3));

UpadateProgressBar(j + 1);

}

pictureBoxCity.Image = imageTSP;

}

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)//choose exit

{

Application.Exit();

}

private void greedySolverToolStripMenuItem\_Click(object sender, EventArgs e)// choose greedy Solver

{

labelProgress.Text = "Greedy algorithm progress";

shortesPathLenth = double.PositiveInfinity;

BeforeShortPathLegth = double.PositiveInfinity;

Thread threadGreedy = new Thread(new ThreadStart(GreedyAlgorithm));

threadGreedy.IsBackground = true;

threadGreedy.Start();

}

private void GreedyAlgorithm()//greedy solver

{

int[] GreedyhaveTrave = new int[Cities];

double cityShortestDistance = double.PositiveInfinity;

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

{

goodPathIndex[i] = 0;

CityTabu[i] = 0;

GreedyhaveTrave[i] = -1;

}

GreedyhaveTrave[0] = 0;

CityTabu[GreedyhaveTrave[0]] = 1;

int minCityIndex = GreedyhaveTrave[0];

setBarValue(1, Cities - 1);

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

{

cityShortestDistance = double.PositiveInfinity;

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

{

if (CityDistance[GreedyhaveTrave[i - 1], j] < cityShortestDistance && CityTabu[j] == 0)

{

minCityIndex = j;

cityShortestDistance = CityDistance[GreedyhaveTrave[i - 1], j];

}

}

UpadateProgressBar(i);

CityTabu[minCityIndex] = 1;

GreedyhaveTrave[i] = minCityIndex;

WriteProgressInfo(i, cityShortestDistance);

}

shortesPathLenth = 0;

for (int k = 0; k < Cities - 1; k++)

{

shortesPathLenth += CityDistance[GreedyhaveTrave[k], GreedyhaveTrave[k + 1]];

}

shortesPathLenth += CityDistance[GreedyhaveTrave[Cities - 1], GreedyhaveTrave[0]];

WriteProgressInfo(Cities, shortesPathLenth);

goodPathIndex = GreedyhaveTrave;

DrawShortestPath();

}

public void WriteProgressInfo(int iterationCount, double minimumLength)//display the result

{

double fImprovement = 0;

if (labelImprovement.InvokeRequired)

{

Action<int, double> actionIteration = delegate(int iterationCountTemp, double minimumLengthTemp)

{

fImprovement = BeforeShortPathLegth == double.PositiveInfinity ? 0 : BeforeShortPathLegth - minimumLengthTemp;

labelImprovement.Text = "(t=" + iterationCountTemp.ToString() + ")," + fImprovement.ToString("F0");

};

labelImprovement.Invoke(actionIteration, iterationCount, minimumLength);

}

else

{

fImprovement = BeforeShortPathLegth == double.PositiveInfinity ? 0 : BeforeShortPathLegth - minimumLength;

labelImprovement.Text = "(t=" + iterationCount.ToString() + ")," + fImprovement.ToString("F0");

}

if (labelCTLength.InvokeRequired)

{

Action<double> actionLength = delegate(double shortLengh)

{

labelCTLength.Text = shortLengh == 0 ? "" : shortLengh.ToString("F0");

};

labelCTLength.Invoke(actionLength, minimumLength);

}

else

{

labelCTLength.Text = minimumLength == 0 ? "" : minimumLength.ToString("F0");

}

}

public void DrawShortestPath()//draw the shortes length

{

if (pictureBoxCity.InvokeRequired)

{

Action actionDraw = delegate()

{

double fPicboxWidth = pictureBoxCity.Width;

double fPicboxheight = pictureBoxCity.Height;

double PicAxisX, PicAxisY;

PicAxisX = (double)(fPicboxWidth - 30) / (xMinMax[1] - xMinMax[0]);

PicAxisY = (double)(fPicboxheight - 30) / (yMinMax[1] - yMinMax[0]);

SolidBrush RedBrush = new SolidBrush(Color.Crimson);

pictureBoxCity.Image = null;

Image imageTSP = new Bitmap(pictureBoxCity.Width, pictureBoxCity.Height);

Graphics GraphicsTSP = Graphics.FromImage(imageTSP);

GraphicsTSP.Clear(Color.Bisque);

double cityPointX;

double cityPointY;

setBarValue(0, Cities - 1);

Font myFont1 = new Font("Hacttenschweiler", 7);

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

{

cityPointX = (xy[j, 0] - xMinMax[0]) \* PicAxisX;

cityPointY = (xy[j, 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityPointX, (float)cityPointY, 3, 3));

UpadateProgressBar(j);

}

double FirstPointX, FirstPointY, SecondPointX, SecondPointY;

Pen penLine = new Pen(Color.Black, 1);

setBarValue(1, Cities - 1);

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

{

FirstPointX = (xy[goodPathIndex[i - 1], 0] - xMinMax[0]) \* PicAxisX;

FirstPointY = (xy[goodPathIndex[i - 1], 1] - yMinMax[0]) \* PicAxisY;

SecondPointX = (xy[goodPathIndex[i], 0] - xMinMax[0]) \* PicAxisX;

SecondPointY = (xy[goodPathIndex[i], 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.DrawLine(penLine, new PointF((float)FirstPointX, (float)FirstPointY), new PointF((float)SecondPointX, (float)SecondPointY));

UpadateProgressBar(i);

}

FirstPointX = (xy[goodPathIndex[0], 0] - xMinMax[0]) \* PicAxisX;

FirstPointY = (xy[goodPathIndex[0], 1] - yMinMax[0]) \* PicAxisY;

SecondPointX = (xy[goodPathIndex[Cities - 1], 0] - xMinMax[0]) \* PicAxisX;

SecondPointY = (xy[goodPathIndex[Cities - 1], 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.DrawLine(penLine, new PointF((float)FirstPointX, (float)FirstPointY), new PointF((float)SecondPointX, (float)SecondPointY));

pictureBoxCity.Image = imageTSP;

};

pictureBoxCity.Invoke(actionDraw);

}

else

{

double fPicboxWidth = pictureBoxCity.Width;

double fPicboxheight = pictureBoxCity.Height;

double PicAxisX, PicAxisY;

PicAxisX = (double)(fPicboxWidth - 30) / (xMinMax[1] - xMinMax[0]);

PicAxisY = (double)(fPicboxheight - 30) / (yMinMax[1] - yMinMax[0]);

SolidBrush RedBrush = new SolidBrush(Color.Red);

pictureBoxCity.Image = null;

Image imageTSP = new Bitmap(pictureBoxCity.Width, pictureBoxCity.Height);

Graphics GraphicsTSP = Graphics.FromImage(imageTSP);

GraphicsTSP.Clear(Color.White);

double cityPointX;

double cityPointY;

setBarValue(0, Cities - 1);

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

{

cityPointX = (xy[j, 0] - xMinMax[0]) \* PicAxisX;

cityPointY = (xy[j, 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityPointX, (float)cityPointY, 3, 3));

UpadateProgressBar(j);

}

double FirstPointX, FirstPointY, SecondPointX, SecondPointY;

Pen penLine = new Pen(Color.Black, 1);

setBarValue(1, Cities - 1);

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

{

FirstPointX = (xy[goodPathIndex[i - 1], 0] - xMinMax[0]) \* PicAxisX;

FirstPointY = (xy[goodPathIndex[i - 1], 1] - yMinMax[0]) \* PicAxisY;

SecondPointX = (xy[goodPathIndex[i], 0] - xMinMax[0]) \* PicAxisX;

SecondPointY = (xy[goodPathIndex[i], 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.DrawLine(penLine, new PointF((float)FirstPointX, (float)FirstPointY), new PointF((float)SecondPointX, (float)SecondPointY));

UpadateProgressBar(i);

}

FirstPointX = (xy[goodPathIndex[0], 0] - xMinMax[0]) \* PicAxisX;

FirstPointY = (xy[goodPathIndex[0], 1] - yMinMax[0]) \* PicAxisY;

SecondPointX = (xy[goodPathIndex[Cities - 1], 0] - xMinMax[0]) \* PicAxisX;

SecondPointY = (xy[goodPathIndex[Cities - 1], 1] - yMinMax[0]) \* PicAxisY;

GraphicsTSP.DrawLine(penLine, new PointF((float)FirstPointX, (float)FirstPointY), new PointF((float)SecondPointX, (float)SecondPointY));

pictureBoxCity.Image = imageTSP;

}

}

private void aCOSolverToolStripMenuItem\_Click(object sender, EventArgs e)//choose ACO Solver

{

labelProgress.Text = "ACO algorithm progress";

shortesPathLenth = double.PositiveInfinity;

BeforeShortPathLegth = double.PositiveInfinity;

Nc = 0;

Thread threadACO = new Thread(new ThreadStart(ACOAlgrorithm));

threadACO.IsBackground = true;

threadACO.Start();

}

private void ACOAlgrorithm()//Aco solver

{

AntCount = Cities;

PheromoneTau = new double[Cities, Cities];

etaCij = new double[Cities, Cities];

deltaTau = new double[Cities, Cities];

CitiesTabu = new int[AntCount, Cities];

CityNoTabu = new int[AntCount, Cities];

CityAntHaveTrave = new int[AntCount, Cities];

mAntShortestDistance = new double[AntCount];

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

{

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

{

PheromoneTau[i, j] = 1.0;

if (i != j)

{

etaCij[i, j] = 1.0 / CityDistance[i, j];

}

deltaTau[i, j] = 0.0;

}

}

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

{

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

{

CityAntHaveTrave[i, j] = -1;

CitiesTabu[i, j] = 0;

}

CityAntHaveTrave[i, 0] = 0;

CitiesTabu[i, 0] = 1;

}

for (int iterationCount = 1; iterationCount <= NcMax; iterationCount++)

{

startAntMove();

int minLenthAntNo = 0;

BeforeShortPathLegth = shortesPathLenth;

setBarValue(0, AntCount);

for (int antNo = 0; antNo < AntCount; antNo++)

{

mAntShortestDistance[antNo] = calculateMinDistance(antNo);

if (mAntShortestDistance[antNo] < shortesPathLenth)

{

shortesPathLenth = mAntShortestDistance[antNo];

minLenthAntNo = antNo;

}

UpadateProgressBar(antNo + 1);

}

setBarValue(0, Cities);

for (int cityNO = 0; cityNO < Cities; cityNO++)

{

goodPathIndex[cityNO] = CityAntHaveTrave[minLenthAntNo, cityNO];

UpadateProgressBar(cityNO + 1);

}

UpdatePathIncrementPheromone();

setBarValue(0, Cities);

for (int icityNo = 0; icityNo < Cities; icityNo++)

{

for (int jcityNo = 0; jcityNo < Cities; jcityNo++)

{

PheromoneTau[icityNo, jcityNo] = (1 - rho) \* PheromoneTau[icityNo, jcityNo] + deltaTau[icityNo, jcityNo];

if (PheromoneTau[icityNo, jcityNo] < PhenomenonZero)

PheromoneTau[icityNo, jcityNo] = PhenomenonZero;

if (PheromoneTau[icityNo, jcityNo] > PhenomenonRangeMax)

PheromoneTau[icityNo, jcityNo] = PhenomenonRangeMax;

}

UpadateProgressBar(icityNo + 1);

}

setBarValue(0, AntCount);

for (int antNo = 0; antNo < AntCount; antNo++)

{

for (int cityNo = 0; cityNo < Cities; cityNo++)

{

CitiesTabu[antNo, cityNo] = 0;

if (cityNo > 0)

{

CityAntHaveTrave[antNo, cityNo] = -1;

}

}

CitiesTabu[antNo, CityAntHaveTrave[antNo, 0]] = 1;

UpadateProgressBar(antNo + 1);

}

WriteProgressInfo(iterationCount, shortesPathLenth);

DrawShortestPath();

}

}

private void startAntMove()//ants go to move

{

double antPij = 0;

double PijSum = 0;

double randomPij = 0;

Random randData = new Random((int)DateTime.Now.Ticks & 0x0000FFFF);

for (int haveTraveCity = 1; haveTraveCity < Cities; haveTraveCity++)

{

setBarValue(0, AntCount);

for (int iAnt = 0; iAnt < AntCount; iAnt++)

{

antPij = 0.0;

PijSum = 0.0;

randomPij = randData.Next(3000) / 3000.0;

for (int iCity = 0; iCity < Cities; iCity++)

{

if (CitiesTabu[iAnt, iCity] == 0)

{

try

{

PijSum += Math.Pow(PheromoneTau[CityAntHaveTrave[iAnt, haveTraveCity - 1], iCity], alpha) \* Math.Pow(etaCij[CityAntHaveTrave[iAnt, haveTraveCity - 1], iCity], beta);

}

catch (System.Exception ex)

{

continue;

}

}

}

for (int iCity = 0; iCity < Cities; iCity++)

{

if (CitiesTabu[iAnt, iCity] == 0)

{

try

{

antPij += Math.Pow(PheromoneTau[CityAntHaveTrave[iAnt, haveTraveCity - 1], iCity], alpha) \* Math.Pow(etaCij[CityAntHaveTrave[iAnt, haveTraveCity - 1], iCity], beta) / PijSum;

if (antPij > randomPij)

{

CitiesTabu[iAnt, iCity] = 1;

CityAntHaveTrave[iAnt, haveTraveCity] = iCity;

break;

}

}

catch (System.Exception ex)

{

continue;

}

}

}

UpadateProgressBar(iAnt + 1);

}

}

}

private void UpdatePathIncrementPheromone()//update the information

{

setBarValue(0, AntCount);

for (int antNo = 0; antNo < AntCount; antNo++)

{

try

{

for (int cityNo = 0; cityNo < Cities - 1; cityNo++)

{

deltaTau[CityAntHaveTrave[antNo, cityNo], CityAntHaveTrave[antNo, cityNo + 1]] += 100 / mAntShortestDistance[antNo];

}

deltaTau[CityAntHaveTrave[antNo, Cities - 1], CityAntHaveTrave[antNo, 0]] += 100 / mAntShortestDistance[antNo];

}

catch (System.Exception ex)

{

continue;

}

UpadateProgressBar(antNo + 1);

}

}

private double calculateMinDistance(int antNo)

{

double currentMinDis = 0;

setBarValue(0, Cities - 1);

for (int cityNO = 0; cityNO < Cities - 1; cityNO++)

{

UpadateProgressBar(cityNO + 1);

currentMinDis += CityDistance[CityAntHaveTrave[antNo, cityNO], CityAntHaveTrave[antNo, cityNO + 1]];

}

currentMinDis += CityDistance[CityAntHaveTrave[antNo, Cities - 1], CityAntHaveTrave[antNo, 0]];

return currentMinDis;

}

}

}