

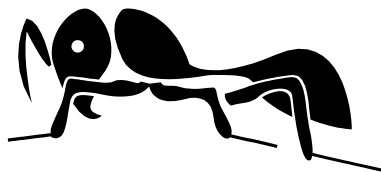
ELECTRONIC, ELECTRICAL AND COMPUTER ENGINEERING



UNIVERSITY OF
BIRMINGHAM

Object Oriented Programming Using C#

Assignment 2014-15



Swarm Intelligence

Student name: Guanshen Yan

Student Number: 1481476

Abstract

The Travelling Salesman Problem (TSP) a challenging discrete optimization problem. Several methods can be used to solve this problem. And Greedy and the ant colony optimisation (ACO) algorithm are two of them. In this project, I need to solve the Travelling Salesman Problem (TSP) via programming the code including two methods: Greedy and the ant colony optimization (ACO) algorithm, and a GUI need to be built to display key information and results.

CONTENTS

1. Introduction.....	1
1.1 Greedy algorithm	1
1.2 The ant colony optimisation algorithm	2
1.2.1 Background	2
1.2.2 Application of ACO algorithm to TSP	3
2. Design.....	5
2.1 Specification.....	5
2.2 Use-case model	5
2.2.1 Use-case diagram.....	5
2.2.2 Scenario Description	6
2.2.3 Class Identification	6
2.2.4 CRC Cards	7
2.2.5 Interaction Diagram	9
2.2.6 State chart Diagram.....	10
2.3 Analysis Model	11
2.3.1 Attributes	11
2.3.2 Methods.....	12
2.3.3 Sequence Diagram.....	13
2.3.4 Class Diagram	14
2.3.5 State Chart Diagram.....	15
2.4 Design Modelling.....	16
2.4.1 Textual Description of Object to Object Interaction	16
2.4.2 Class diagram showing visibility	17
3. Implementation and testing.....	18
3.1 Selecting and opening tsp file.....	18
3.2 Applying algorithm	20
3.2.1 Greedy Algorithm	21
3.2.2 ACO Solver	21
4. Conclusion	22
Reference	23

LIST OF FIGURES

Figure 1A. ants in a pheromone trail between nest and food without obstacle; B. an obstacle in the trail; C. ants find two paths by their own; D. pheromone density helps ants to find the shorter path	2
Figure 2 Process of ACO algorithm	4
Figure 3 Use-Case Diagram	5
Figure 4 Class Diagram (Stereotypes).....	7
Figure 5 Interaction Diagram	9
Figure 6 State chart Diagram	10
Figure 7 Analysis Model Sequence diagram.....	13
Figure 8 Analysis Model Class Diagram.....	14
Figure 9 Analysis Model State chart	15
Figure 10 Analysis Model Class Diagram showing visibility	17
Figure 11 The GUI form.....	18
Figure 12 Clicking 'File' Button	19
Figure 13 The window for choosing file after clicking 'Open File'	19
Figure 14 GUI displays information.....	20
Figure 15 Clicking 'TSP File' Button	20
Figure 16 Greedy Solver.....	21
Figure 17 ACO Solver	22

LIST OF TABLES

Table 1 Class Responsibilities Collaboration Cards	8
Table 2 Attributes of the system	11
Table 3 Method used in system.....	12

1. Introduction

In 1759, Euler had a problem in moving a knight on a chess board. The problem was that the knight must be moved to every position on the board but exactly once on each position. This is the first sample of the travelling salesman problem.[1]

TSP is an idea to find a tour which visits all each city only once and returns to the starting city, and the length of the tour must be minimum.

The symmetric travelling salesman problem can be represented in mathematical way as follows: Assume a graph $G(N, E)$ which is comprising nodes and arcs which connects those all nodes. N and E are two groups of nodes and arcs connecting all nodes of N . d_{ij} is defined to represent the length between city i and j which are included in N . The symmetric travelling salesman problem means that the distance is symmetric: $d_{ij} = d_{ji}$.

Many different methods can be used to solve the travelling salesman problem, but I just used two of them to do this. They are Greedy and the ant colony optimisation (ACO) algorithm.

1.1 Greedy algorithm

Greedy algorithm is a doable method which can be implemented to the travelling salesman problem to get the solution.

Greedy is a simple algorithm to calculate the distance between the starting city and all other cities, and select the nearest unvisited city as the next city until all cities are in the tour.

For example, there are four cities. City 1 is the starting city, and we figure out that the distance between city 1 and city 2 is 10; the distance between city 1 and city 3 is 13; the distance between city 1 and city 4 is 8. So city 4 is the nearest city to city 1, and we choose city 4 as the next city.

Now the unvisited city is city 2 and city 3. After the calculation, we know the distance between city 4 and city 2 is 6; and the distance between city 4 and city 3 is 10. Hence, city 2 is the next city. There is one city unvisited. So city 3 is the next one to city 2, and the distance between city 2

ancity 3 is 12. The tour is a close tour, so it must be back to city 1. As we calculated before, the distance between city 1 and city 3 is 13. In this example, the order of the cities in this tour is city 1, city 4, city 2, city 3, then back to city 1, and the total minimum length is $8 + 6 + 12 + 13 = 39$.

Although the Greedy algorithm can give us the solution to the TSP, the solution is not always best.

1.2 The ant colony optimisation algorithm

As we mentioned before, the Greedy is not always best, though it can give us solution. Hence, I use another algorithm, called the ant colony optimisation (ACO) algorithm.

1.2.1 Background

This algorithm is based on the way real ants using in their hunt for food. There is no direct communication between ants. They communicate with others via pheromone, which is an indirect and efficient way to contact other large number of ants. According to the pheromone, ants will make decision to move through which route. Every ant makes their own decision to go in which direction. But the density of pheromone in the route will change the possibility of ants choosing this route. The stronger the pheromone density, the higher possibility is. In addition, the pheromone density is related to the length of tour. The shorter the tour, the stronger the density is.

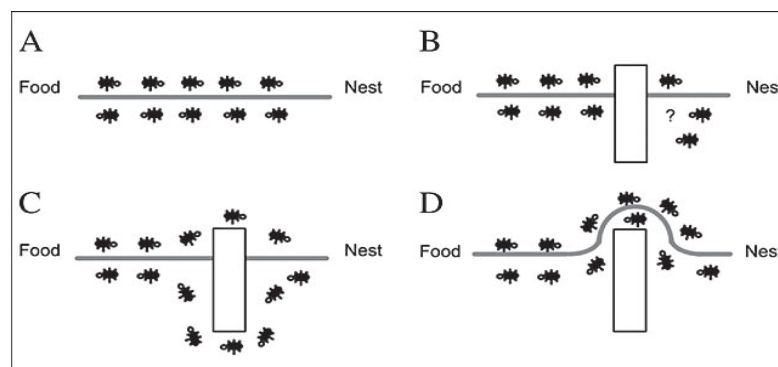


Figure 1A. ants in a pheromone trail between nest and food without obstacle; B. an obstacle in the trail; C. ants find two paths by their own; D. pheromone density helps ants to find the shorter path

[2]

The artificial ants were used to solve discrete optimisation problems are different from the real ants. They can move between discrete adjacent states, and they can remember which state they have been. For that, they can visit every state in their tour once and only once. For artificial ants, they deposit an amount of pheromone, and the pheromone is deposited unless the solution is found. But for real ants, the pheromone is deposited as they go along.

In this algorithm, t_{max} iterations are performed, and m ants are created and they build solutions after they die in each iteration. Then going into next iteration, a new set of ants are created and find solutions again. In this process, the pheromone will be accumulated on each route, and the new sets of ants make decision based on the density of pheromone. There is another feature of pheromone. The pheromone will be evaporated after each iteration.

1.2.2 Application of ACO algorithm to TSP

Assuming that an arc (i, j) connects city i and city j . The pheromone deposited on this arc (i, j) can be defined as $\tau_{ij}(t)$, and t is the index of the current iteration. In each iteration, this quantity of pheromone will be update by all ants when the tour has been completed.

As I said before, the pheromone density on each path will impact the possibility of ants choosing the path. Hence, the possibility can be represented as follow:

$$p_{ij}^k = \frac{[\tau_{ij}(t)][\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)][\eta_{il}]^\beta} \quad j \in N_i^k \quad (1)[3]$$

The meaning of N_i^k is the group of unvisited cities, when the ant k at city i . In this equation, $\eta_{ij} = 1/d_{ij}$ and d_{ij} represents the length between city i and city j . β is the parameter which controls the effect of length of arc over the deposited pheromone. If $\beta = 0$, the algorithm will be stagnation when a route is chosen and amplified. In this situation, all ants will follow this route. When β is greater than 0, the length of arc and the existing pheromone will be combined to calculate the local possibility.

When all ants complete their tour, the pheromone deposited by each ant will be added into local accumulated pheromone.

$$\Delta\tau_{ij}^k(t) = \begin{cases} Q/L^k(t) & \text{if } (i,j) \in T^k(t) \\ 0 & \text{if } (i,j) \notin T^k(t) \end{cases} \quad k = 1 \dots m \quad (2)[3]$$

In this equation, Q is a constant, $T^k(t)$ is the tour which includes the visited cities in sequence, and L^k is the length of the tour.

At the same time, pheromone evaporation is triggered. The quantity of the pheromone will evaporate with a certain rate ρ . Hence, the quantity of pheromone left before next iteration starts can be represented as follow:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{\text{ants } k \text{ which use edge } (i,j)} \frac{Q}{L^k} \quad (3)$$

where m is the number of ants and the value of ρ is between 0 and 1. Normally, when t_{max} , β and ρ are set as 3000, 7 and 0.5 separately, we can get a good result.[lab sheet]

In general, the ACO algorithm can be represented in following figure.

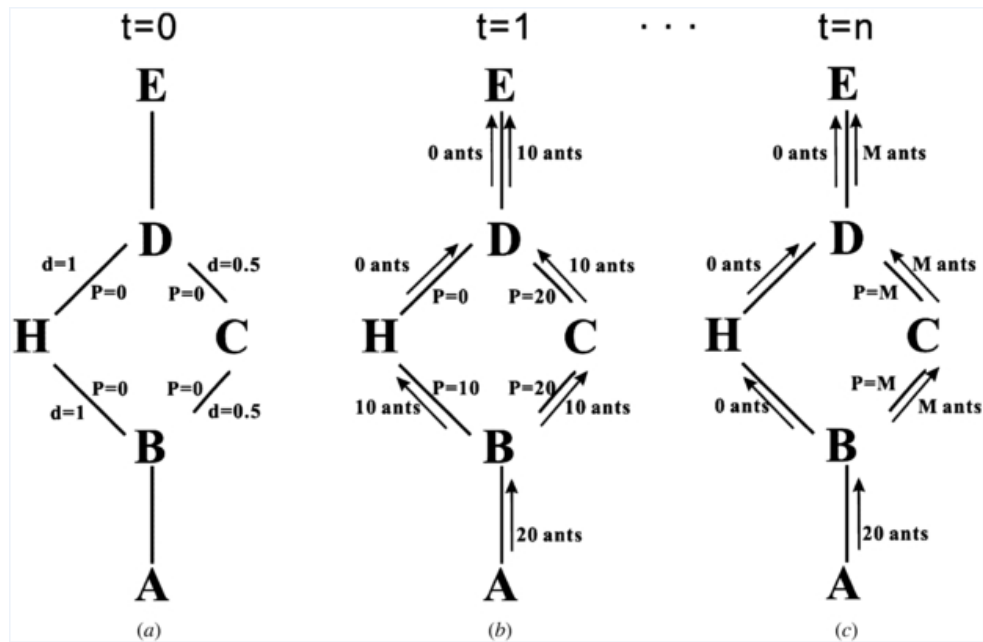


Figure 2 Process of ACO algorithm

[4]

2. Design

I designed the system by using unified modelling language.

2.1 Specification

In this system, Greedy and the ant colony optimisation algorithm are implemented to solve the travelling salesman problem. User can choose tsp file and the solver on interface, and the solver can figure out the shortest route. The interface can display all information which included in tsp file, such as file name, number of cities and optimum route length, and it also displays all results figured out by solvers and maps the route.

2.2 Use-case model

Use-case model is a way to define requirements and added value scenarios of usage, and provide the understanding of the interaction between user and system.

2.2.1 Use-case diagram

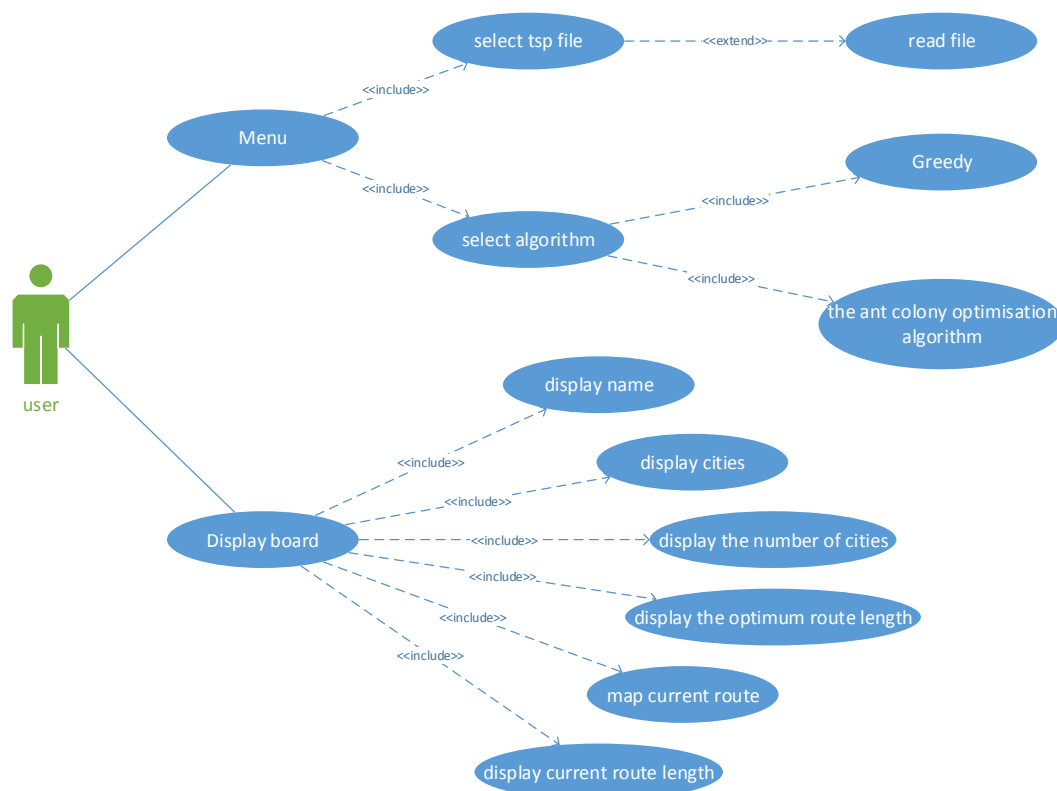


Figure 3 Use-Case Diagram

2.2.2 Scenario Description

User: Before running this system, user need to select a tsp file by clicking file button. And system can read the data in this file. User can make a decision to select which algorithm to solve the travelling salesman problem. Two algorithms can be chosen on interface by clicking solver button, including Greedy and the ant colony optimisation algorithm. Greedy is the way to figure out the solution by calculating the length between each city to find the nearest city. The ACO algorithm is another way to find the best route according to the pheromone on each route. When user selects an algorithm, the system starts running to find the solution to the travelling salesman problem.

Display board: After the process, all information and results will be displayed on display board, including the name of file, cities, number of cities, the optimum route length and the current route length. In additional, the current route will be mapped on display board.

Default Action: If user chooses another tsp file when the system is running, this process will stop immediately and the information of the new file will be displayed on display board.

2.2.3 Class Identification

Nouns:

User, Tsp file, Algorithm, Display board, Button, Code.

Stereotypical classes:

Boundary: Buttons, Display board

Analysis: Algorithm, TSP file

Control: Solver

I can minimise the set of classes as: Menu, Display board, Code.

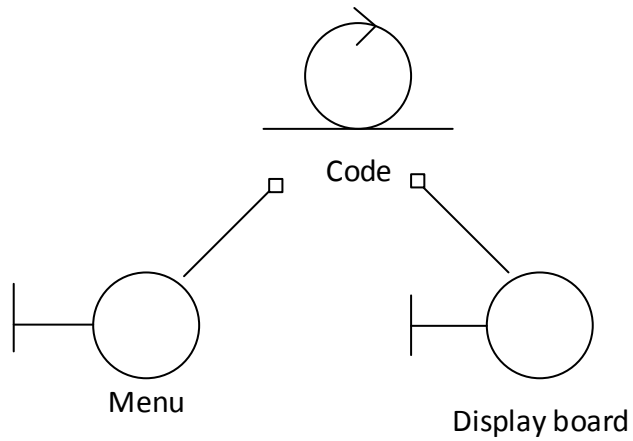


Figure 4 Class Diagram (Stereotypes)

2.2.4 CRC Cards

I describe the responsibilities and collaborators of each class by using CRC Cards.

Class: Menu	
Responsibility	Collaborators
<p>Menu class is responsible for clicking button to choose the tsp file and algorithm, reading the data in the file, and calling the selected algorithm to figure out the solution to travelling salesman problem.</p> <p>When the new tsp file is chosen, the process which is running will be stopped and the information of new tsp file will be displayed on display board.</p>	Code, Display board

Class: Display board	
Responsibility	Collaborators
Display board class is responsible for displaying the information which is included in tsp file after the system reading the file, such as the file name, cities, number of cities, and the optimum route length. The display board also display the results which are figured out by program, such as the current route, the length of current route.	Code, Menu

Class: Solver	
Responsibility	Collaborators
The code class is responsible for loading the data in the tsp file selected by user, using the selected algorithm to figure out the solution to travelling salesman problem, and calling the results which be figured out to display on the display board.	Menu, Display board

Table 1 Class Responsibilities Collaboration Cards

2.2.5 Interaction Diagram

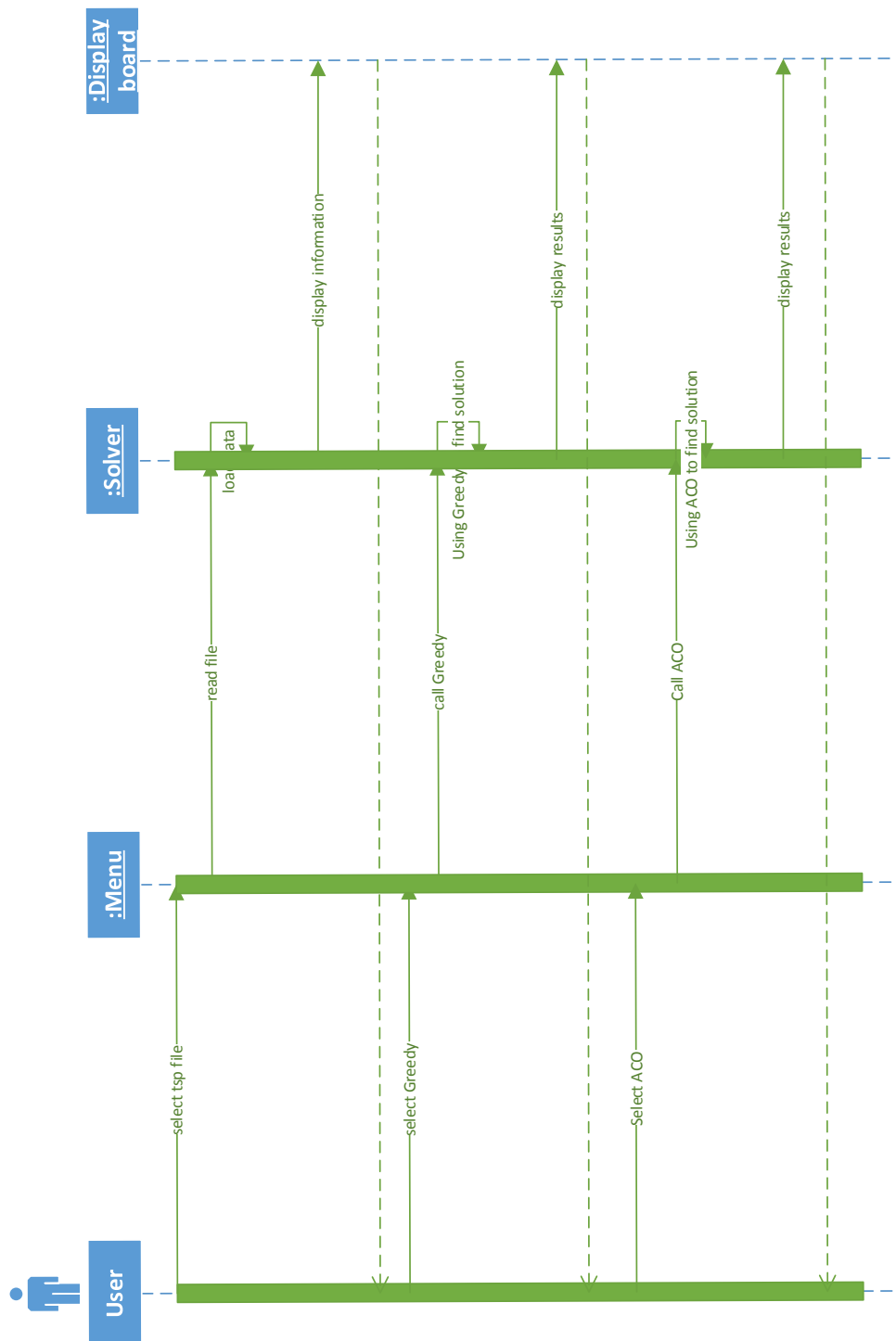


Figure 5 Interaction Diagram

This diagram represents the relation and behaviour between each class and user.

2.2.6 State chart Diagram

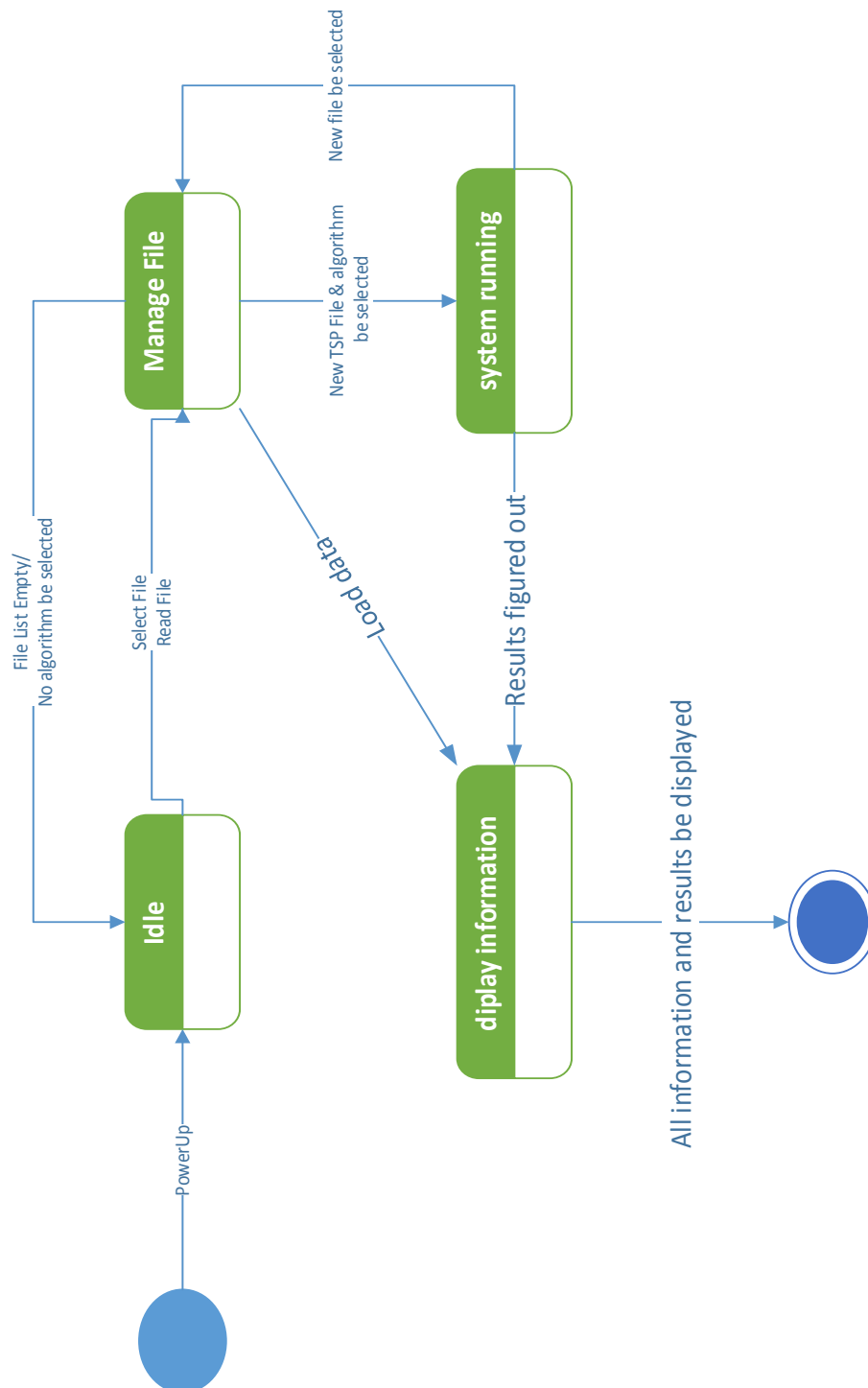


Figure 6 State chart Diagram

The state changing of the system can be represented as the figure above.

2.3 Analysis Model

At this stage, I need to analysis this system from the perspective of the designer. If we consider about the system form the internal view, it is more helpful for us to design the system. It can tell us how the function to be realised.

2.3.1 Attributes

The attributes of each class are listed in the following table:

Class	Attribute	Comment
Menu	ButtonPress	Boolean
	Dataload	Boolean
	Algorithmchoose	Boolean
Display board	City	Status
	CityName	String
	CityNumber	Integer
	CurrentRoute	Status
	OptimumRouteLength	Integer
	CurrentRouteLength	Integer
Solver	TSPFile	Integer
	Algorithm	Greedy/ACO

Table 2 Attributes of the system

2.3.2 Methods

Methods are used in each class are listed in the following table:

Class	Method	Comment
Menu	PressFileButton()	
	PressAlgorithmButton()	
	ReadFile()	
	CallAlgorithm()	
Display board	CallFileName()	
	CallCityCoordinate()	
	CallCityNumber()	
	CallOptimumRouteLength()	
	CallCurrentRoute()	
	CallCurrentRouteLength()	
	Display()	Display information and results
	Map()	Map cities and route
Solver	RunAlgorithm()	Run the algorithm which is selected by user in the menu
	ReturnResult()	Return results

Table 3 Method used in system

2.3.3 Sequence Diagram

The sequence diagram includes triggers, parameters and some detail needed.

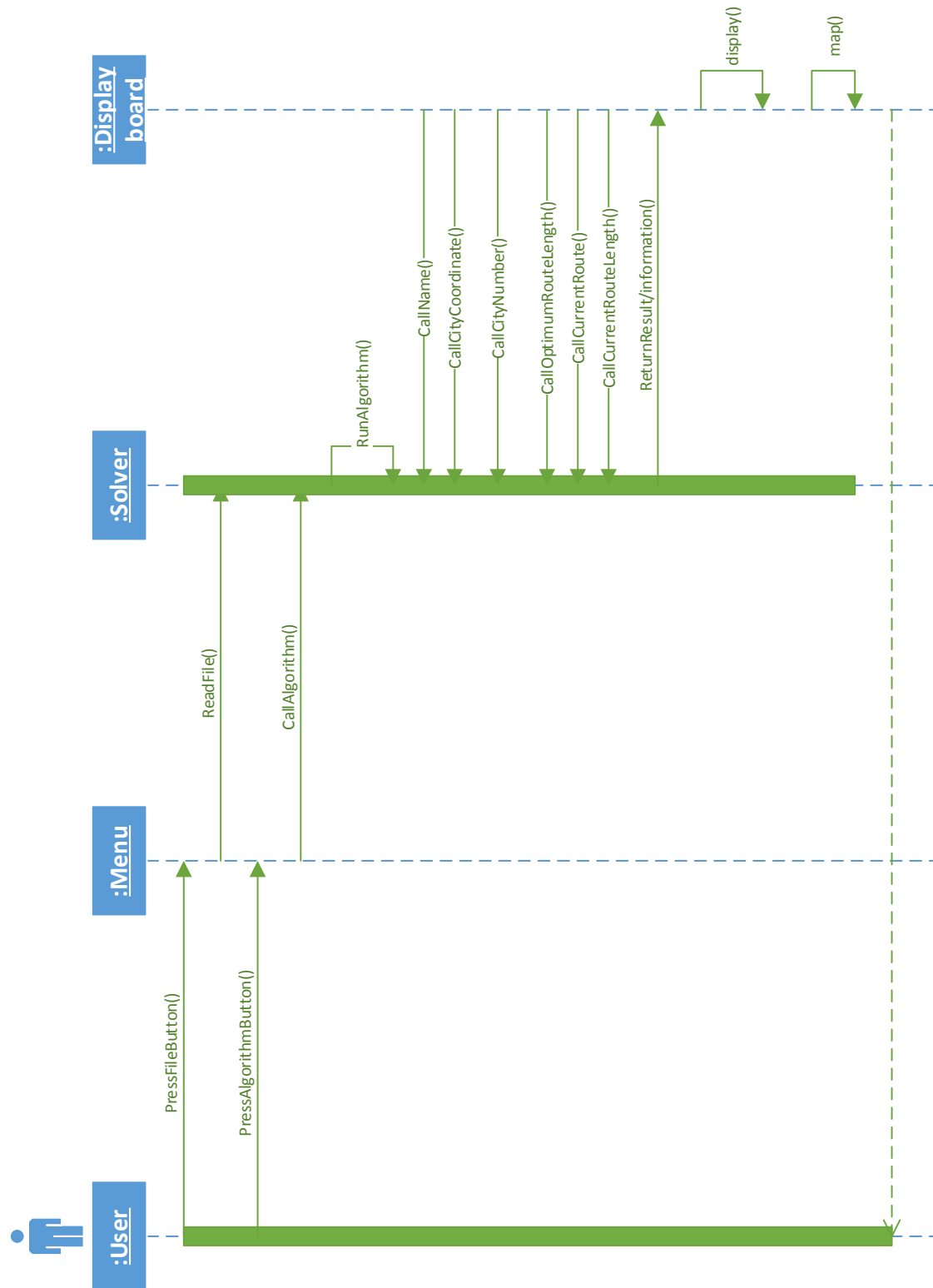


Figure 7 Analysis Model Sequence diagram

2.3.4 Class Diagram

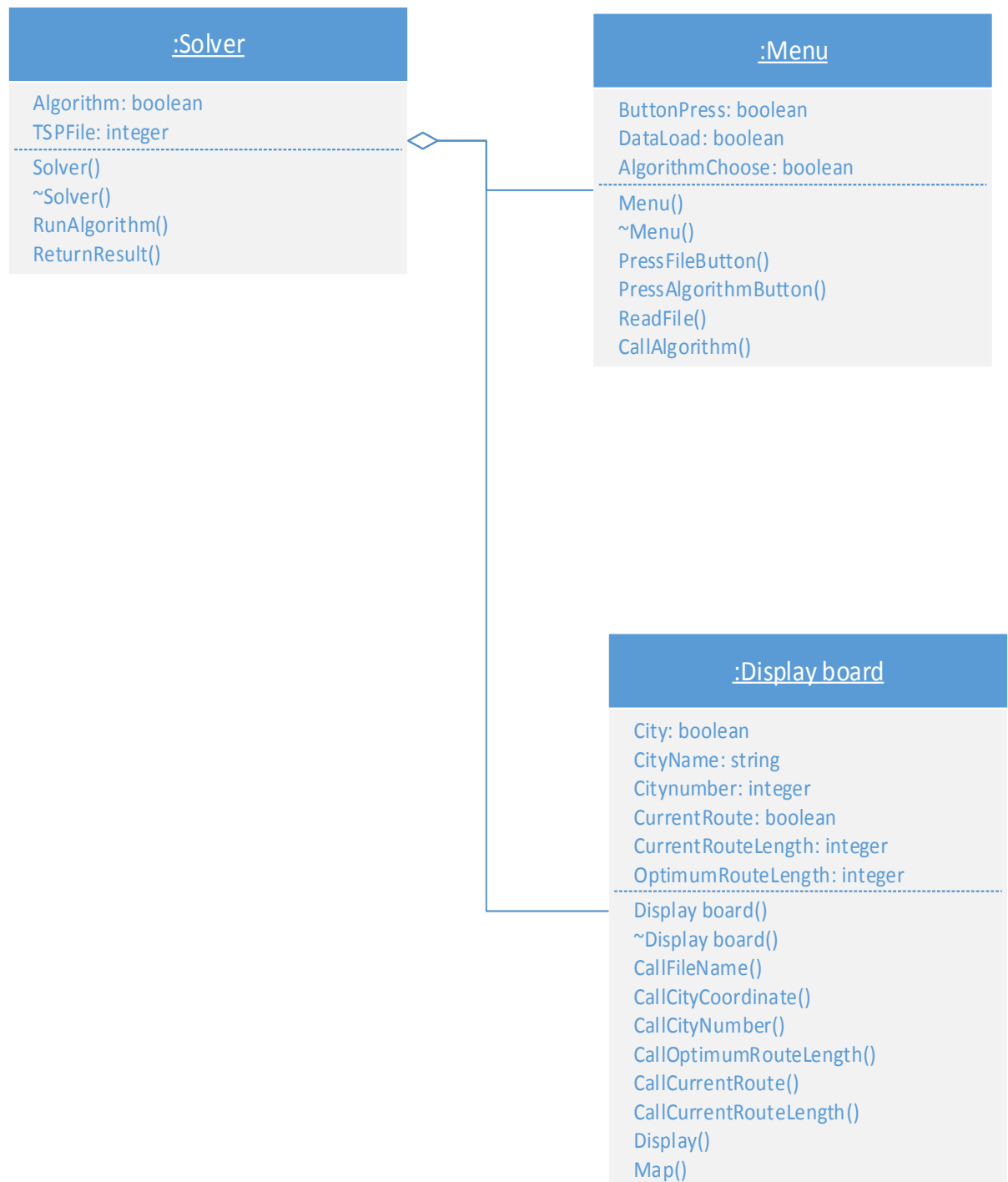


Figure 8 Analysis Model Class Diagram

2.3.5 State Chart Diagram

The state chart in analysis model represents the state changing and the behaviour in each state.

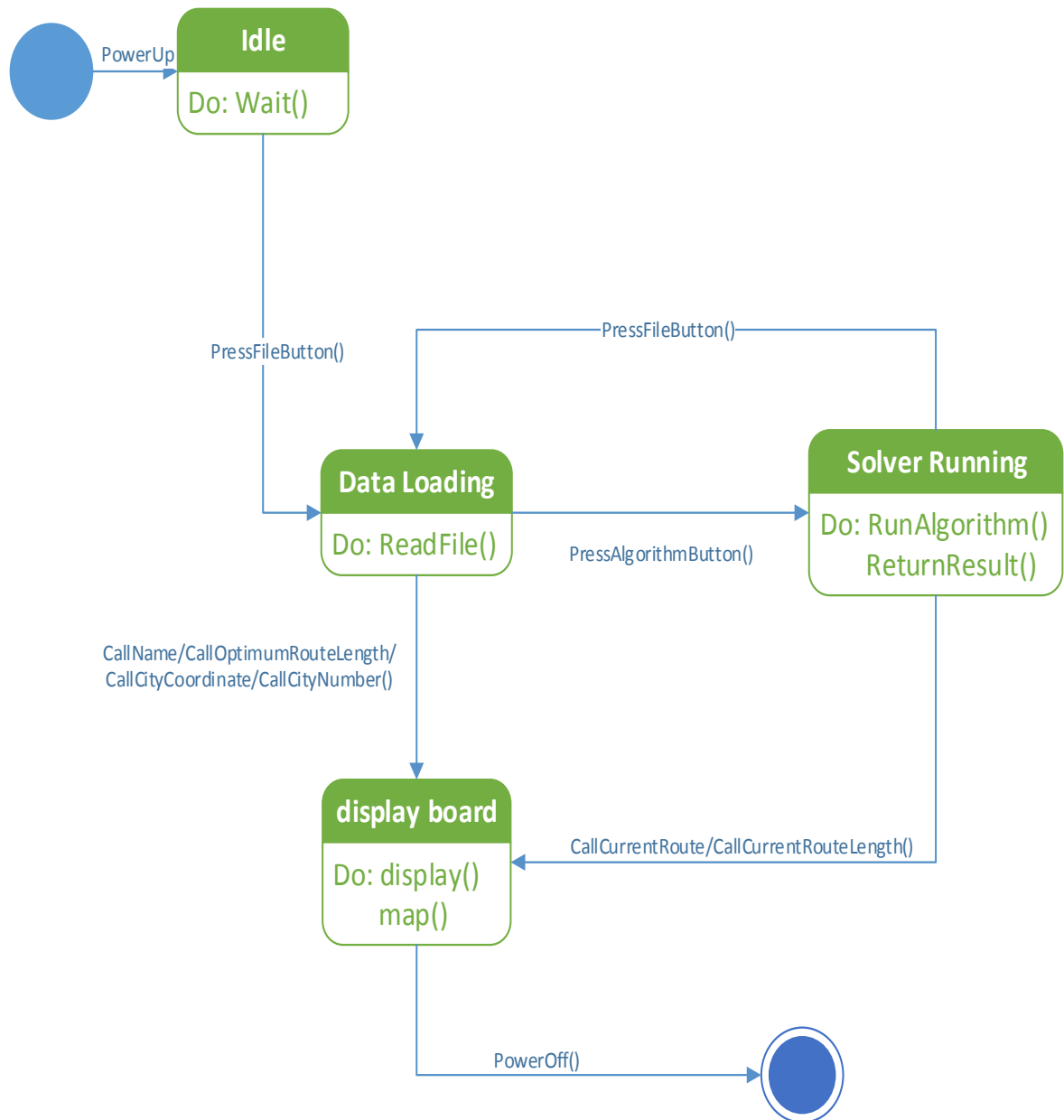


Figure 9 Analysis Model State chart

2.4 Design Modelling

After analysing, I need to draw a physical and implementation blueprint. In this stage, the design must be in logic and in detail.

For the Use-Case, all requirements are met by the design. And the design has added the principal qualifiers already as well. There are no subsystem required in a simple design. However, some non-functional requirements need to be implemented with a method in the solver class to read the coordinate of cities and the commands from Menu. The solver I built is implemented on a single processor in development model. All attributes are incorporated in the class diagram. In additional, according to the analysis, the revisions of the state chart are not required.

2.4.1 Textual Description of Object to Object Interaction

The Menu class acts as an interface between user and system and send information of tsp file to display board. User can select file and algorithm in this class, and solver can get the command from menu to read file and choose algorithm.

The solver class uses the algorithm selected by users to get the optimum route form the tsp file, and return the result to display board.

The display board class is an interface to call information of the file and the results from solver and show them to user.

2.4.2 Class diagram showing visibility

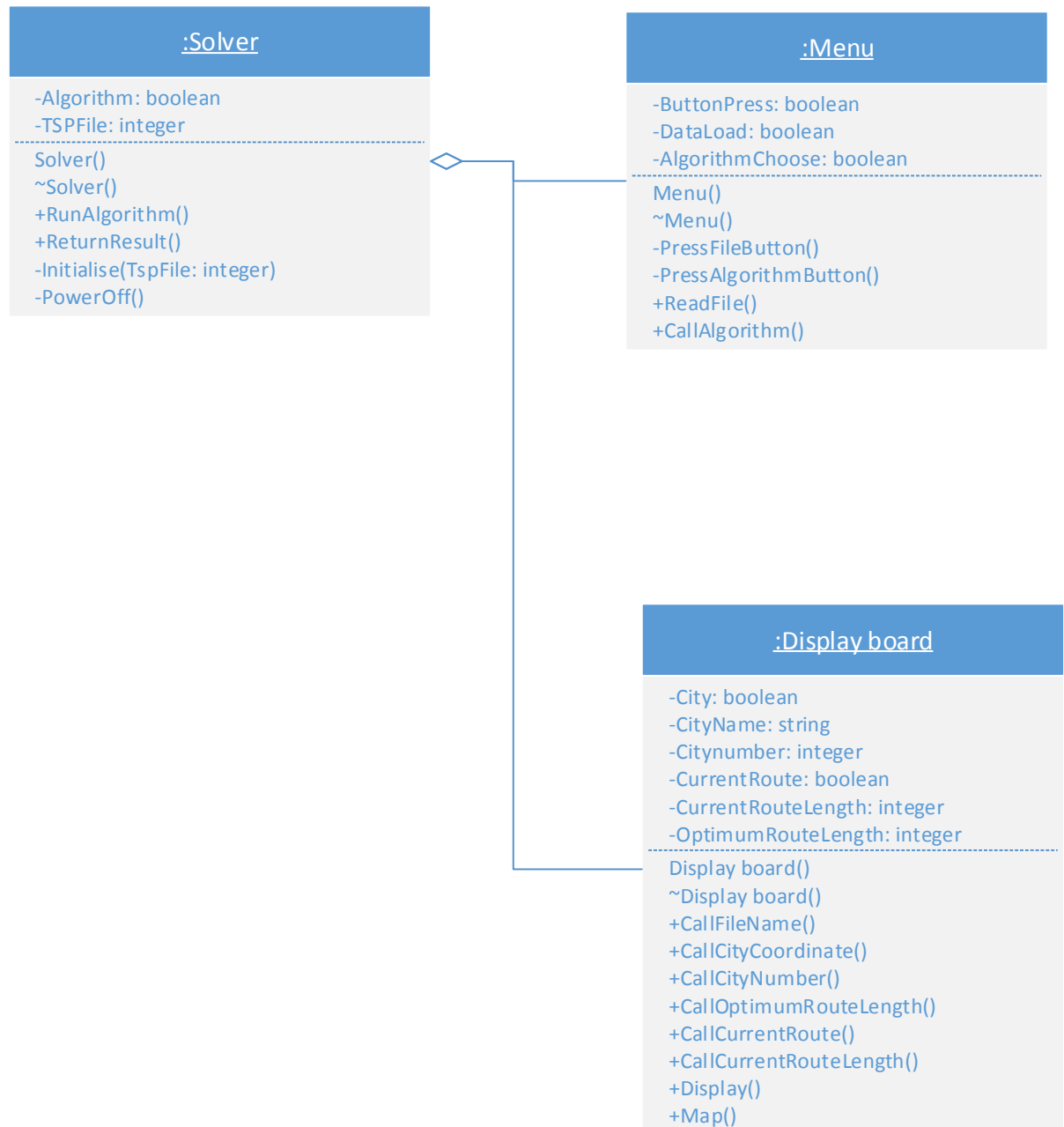


Figure 10 Analysis Model Class Diagram showing visibility

3. Implementation and testing

I built a code to realise the system designed before, and used the tsp files to test my code if it works. The GUI is also need to be built for user to control the system.

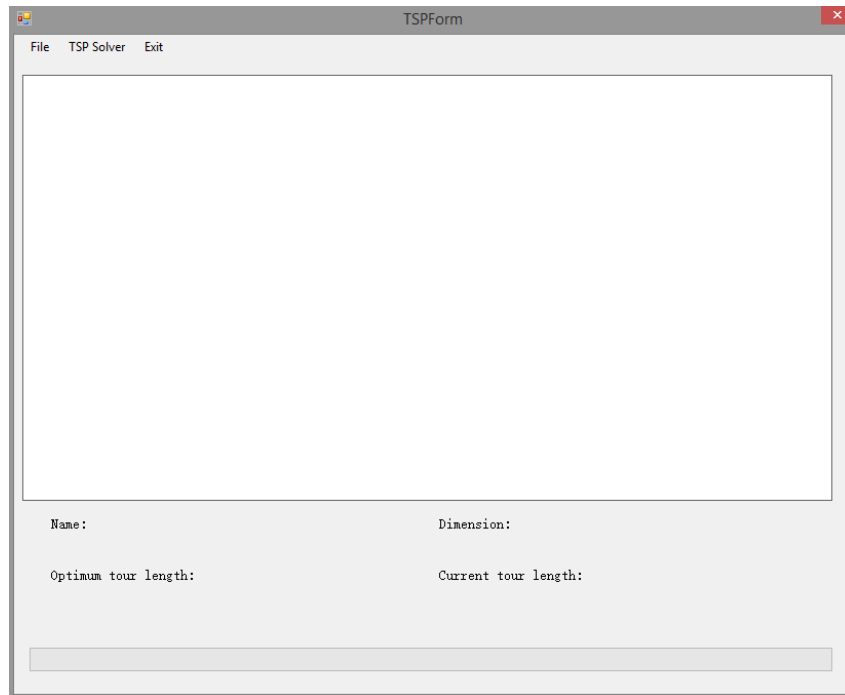


Figure 11 The GUI form

3.1 Selecting and opening tsp file

First thing needs to be done after opening the application is choosing a file which the algorithm can be applied in.

The file can be selected by clicking the 'File' button. After this step, a window appears. User can choose file from this window.

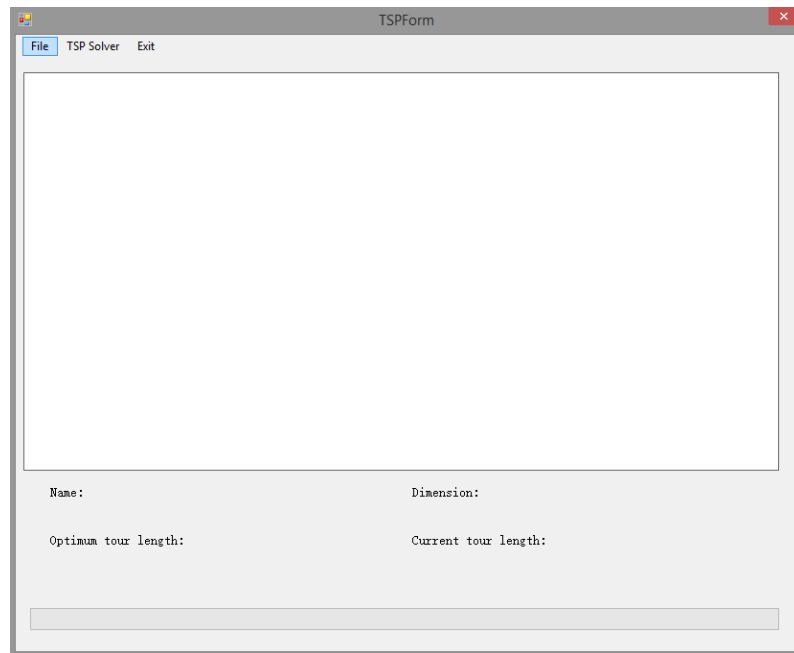


Figure 12 Clicking 'File' Button

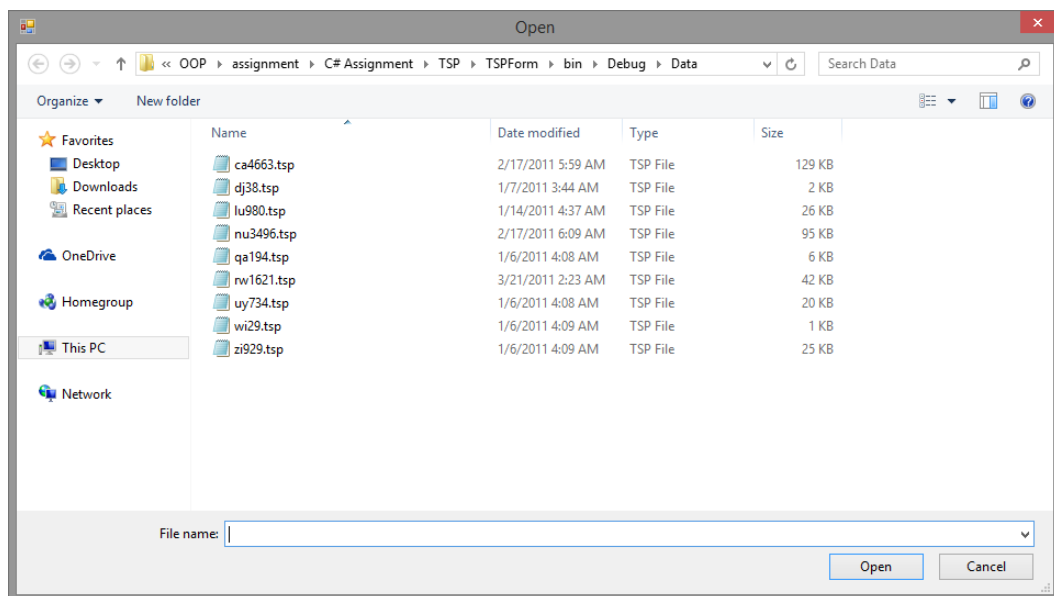


Figure 13 The window for choosing file after clicking 'Open File'

The file can be opened by double clicking a tsp file, and the cities can be plotted in the picture box. In addition, the GUI can display the information included in the file. For example, I choose 'wi29.tsp'.

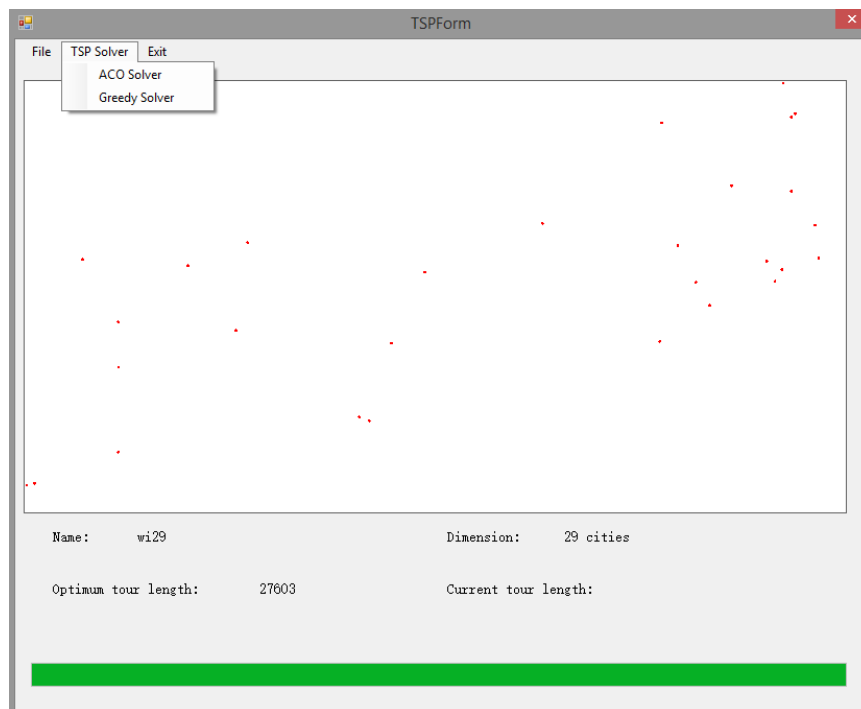


Figure 14 GUI displays information

3.2 Applying algorithm

After choosing the file, the algorithm needs to be chosen to find the solution to the tsp by clicking 'TSP Solver' button.

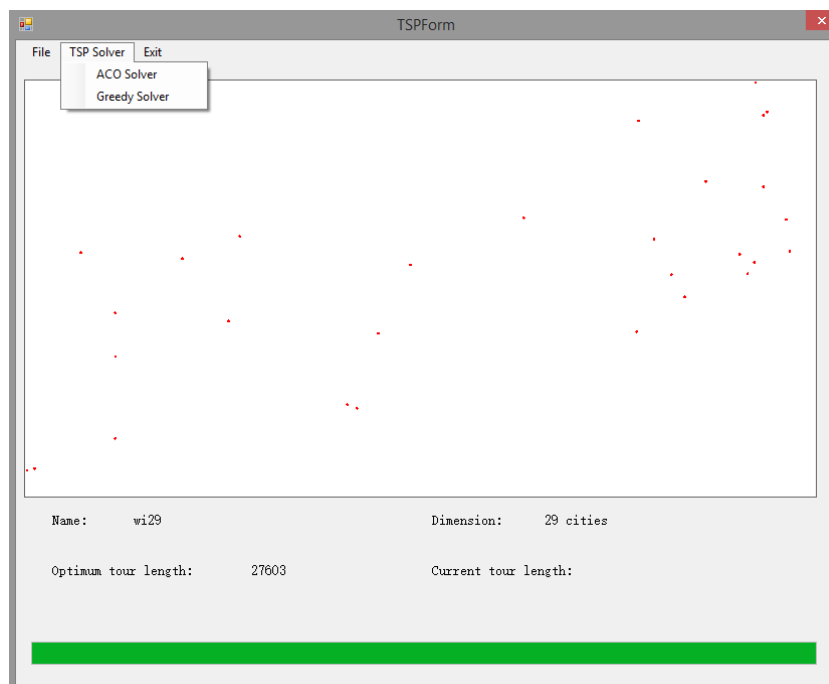


Figure 15 Clicking 'TSP File' Button

3.2.1 Greedy Algorithm

Firstly, I test the Greedy Algorithm. This algorithm is easier than ACO algorithm.

Click 'Greedy Solver' to start the testing.

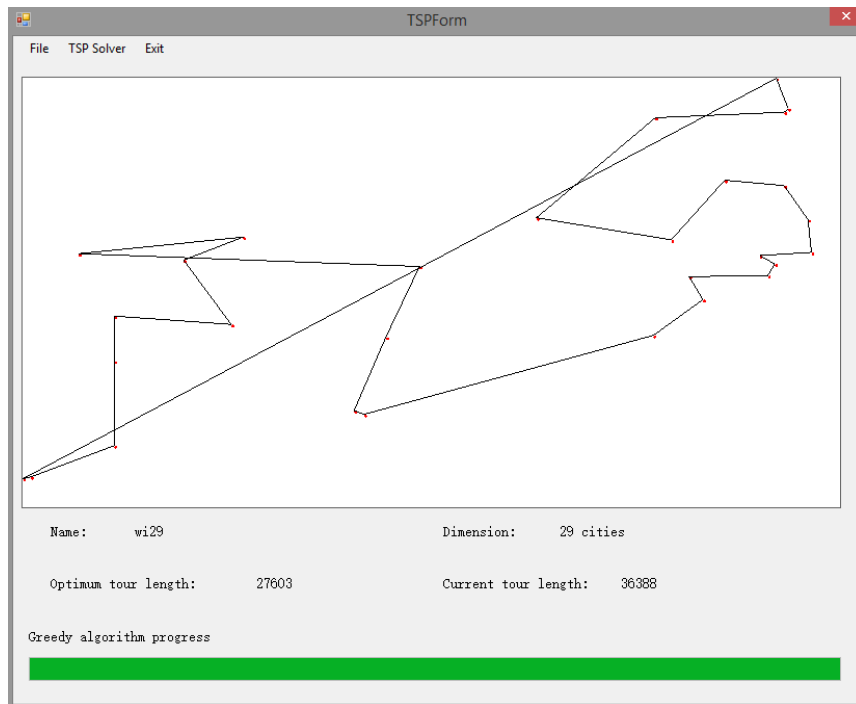


Figure 16 Greedy Solver

Now, I can see the solution figured out by Greedy. The best route length calculated by Greedy is 36388, and the route display in the picture box is the best route figured out by Greedy. However, the optimum route length for this tsp file is 27603. There is still difference from optimum route.

3.2.2 ACO Solver

The other algorithm is the ant colony optimisation algorithm. Clicking 'ACO Solver' can start the ACO algorithm to solve the tsp.

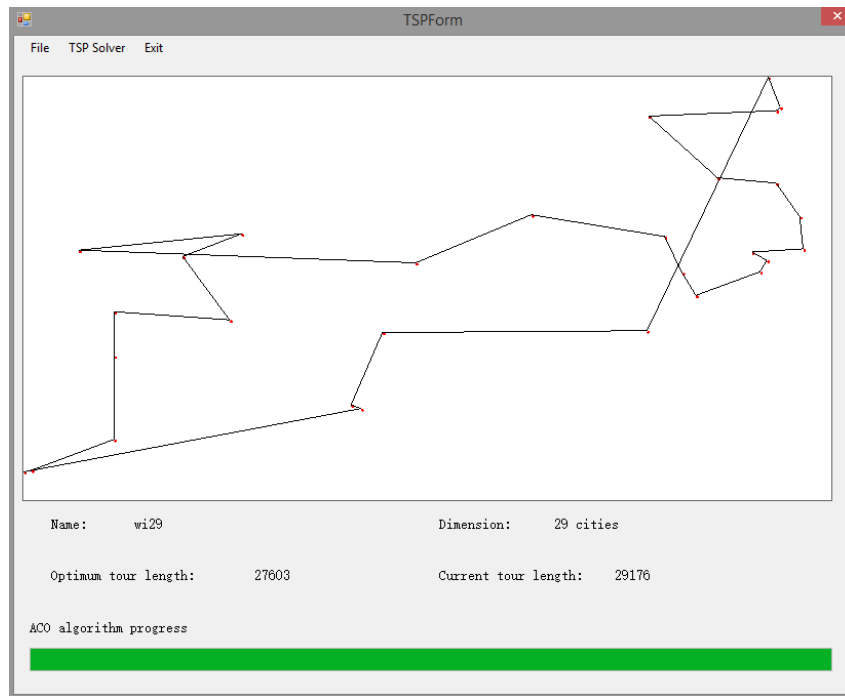


Figure 17 ACO Solver

The route displayed in this figure is the best route figured out by ACO. The length of this route is 29176. Although it is still not an optimum solution, it is better than Greedy. Compared with Greedy, this algorithm takes longer time,

4. Conclusion

In this project, a system is built for solve the travelling salesman problem. User can control this system via the interface, including choosing tsp file, selecting algorithm, exiting the system, and monitoring the information of the file and result figured out by algorithm by mapping and displaying the route and data. These functions are met in this program.

There are only two algorithms I considered, Greedy and basic ACO algorithm. Both of these two algorithms can give us solutions to travelling salesman problem, but they do not always give us the optimum solution. Comparing these two algorithms, ACO algorithm gives the better solution, but Greedy takes shorter time.

Through this project, I learnt how to use the visual studio to build this system using C#. And I also know that how two algorithms work to figure out the solution.

Reference

- [1] A. Benjamin, "Genetic Algorithms and the Traveling Salesman Problem by Kylie Bryant Genetic Algorithms and the Traveling Salesman Problem by Kylie Bryant," no. December, 2000.
- [2] M. Perretto and H. S. Lopes, "Reconstruction of phylogenetic trees using the ant colony optimization paradigm." 2005.
- [3] Dr. M. Spann, "Object Oriented Programming Using C# Assignment 2014-15 Swarm Intelligence." .
- [4] J. Zhao and S. Z. Sun, "Automatic fault extraction using a modified ant-colony algorithm," p. 2, 2013.

Appendix (Code)

```

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 FormTSP : Form
    {
        private int City;
        private double[] xMinMax;
        private double[] yMinMax;
        private double[,] xy;
        private int[] m;           // the index of the good path
        private double[,] d;       //distance between each city
        private int[] CityList;
        private double opt1 = double.PositiveInfinity;
        private double bopt1 = double.PositiveInfinity;
        int[,] VisitedCity; //store the city which has been visited by ant
        int[,] CitiesTabu;
        int[,] CityNoTabu;
        int n;                //ant number
        int TMax = 3000;      //the maximum iteration number
        double[,] Pheromone;  //the amount of pheromone
        double[,] eta;        // reciprocal of the length of arc(i,j)
        double[,] delta;
        double[] dk;          //the shortest length of the tour which finished by ant k
        double alpha = 2.0;
        double beta = 7.0;    //the relative influence of the arc length over the
//previously accumulated pheromone deposit
        double r = 0.5;       //evoraption rate
        double PheromoneMin = 1e-5; // the range of the density of pheromone
        double PheromoneMax = 30;
        public FormTSP()
        {
            InitializeComponent();
        }

        ////////////////////////////////////////////Greedy//////////////////////////////////////////
        ////////////////////////////////////////////

        private void Greedy()
        {
            int[] travelledcity = new int[City]; // vector of visited city
            double q = double.PositiveInfinity; //the shortest distance between
//each city
            for (int i = 0; i < City; i++)
            {
                m[i] = 0;
                CityList[i] = 0;
                travelledcity[i] = -1;
            }
        }
    }
}

```

```

    }
    travelledcity[0] = 0;
    CityList[travelledcity[0]] = 1;
    int minIndex = travelledcity[0];
    Value(1, City - 1);
    for (int i = 1; i < City; i++)
    {
        q = double.PositiveInfinity;
        for (int j = 0; j < City; j++)
        {
            if (d[travelledcity[i - 1], j] < q && CityList[j] == 0)
            {
                minIndex = j;
                q = d[travelledcity[i - 1], j];
            }
        }
        ProgressBar(i);
        CityList[minIndex] = 1;
        travelledcity[i] = minIndex;
        displayinfo(i, q);
    }
    opt1 = 0;
    for (int k = 0; k < City - 1; k++)
    {
        opt1 += d[travelledcity[k], travelledcity[k + 1]];
    }
    opt1 += d[travelledcity[City - 1], travelledcity[0]];
    displayinfo(City, opt1);
    m = travelledcity;

    if (pictureBox1.InvokeRequired)
    {
        Action actionDraw = delegate()
        {
            double W = pictureBox1.Width;
            double H = pictureBox1.Height;
            double PicX, PicY;
            PicX = (double)(W - 30) / (xMinMax[1] - xMinMax[0]);
            PicY = (double)(H - 30) / (yMinMax[1] - yMinMax[0]);
            SolidBrush RedBrush = new SolidBrush(Color.Red);
            pictureBox1.Image = null;
            Image imageTSP = new Bitmap(pictureBox1.Width,
pictureBox1.Height);
            Graphics GraphicsTSP = Graphics.FromImage(imageTSP);
            GraphicsTSP.Clear(Color.White);
            double cityX;
            double cityY;
            Value(0, City - 1);
            Font myFont1 = new Font("Hacttenschweiler", 7);
            for (int j = 0; j < City; j++)
            {
                cityX = (xy[j, 0] - xMinMax[0]) * PicX;
                cityY = (xy[j, 1] - yMinMax[0]) * PicY;
                GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityX,
(float)cityY, 3, 3));
                ProgressBar(j);
            }

            double X1, Y1, X2, Y2;
            Pen penLine = new Pen(Color.Black, 1);
            Value(1, City - 1);
            for (int i = 1; i < City; i++)

```

```

        {
            X1 = (xy[m[i - 1], 0] - xMinMax[0]) * PicX;
            Y1 = (xy[m[i - 1], 1] - yMinMax[0]) * PicY;
            X2 = (xy[m[i], 0] - xMinMax[0]) * PicX;
            Y2 = (xy[m[i], 1] - yMinMax[0]) * PicY;
            GraphicsTSP.DrawLine(penLine, new PointF((float)X1,
(float)Y1), new PointF((float)X2, (float)Y2));
            ProgressBar(i);
        }

        X1 = (xy[m[0], 0] - xMinMax[0]) * PicX;
        Y1 = (xy[m[0], 1] - yMinMax[0]) * PicY;
        X2 = (xy[m[City - 1], 0] - xMinMax[0]) * PicX;
        Y2 = (xy[m[City - 1], 1] - yMinMax[0]) * PicY;
        GraphicsTSP.DrawLine(penLine, new PointF((float)X1, (float)Y1),
new PointF((float)X2, (float)Y2));
        pictureBox1.Image = imageTSP;
    };
    pictureBox1.Invoke(actionDraw);
}
else
{
    double W = pictureBox1.Width;
    double H = pictureBox1.Height;
    double PicX, PicY;
    PicX = (double)(W - 30) / (xMinMax[1] - xMinMax[0]);
    PicY = (double)(H - 30) / (yMinMax[1] - yMinMax[0]);
    SolidBrush RedBrush = new SolidBrush(Color.Red);
    pictureBox1.Image = null;
    Image imageTSP = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    Graphics GraphicsTSP = Graphics.FromImage(imageTSP);
    GraphicsTSP.Clear(Color.White);
    double cityX;
    double cityY;
    Value(0, City - 1);
    for (int j = 0; j < City; j++)
    {
        cityX = (xy[j, 0] - xMinMax[0]) * PicX;
        cityY = (xy[j, 1] - yMinMax[0]) * PicY;
        GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityX,
(float)cityY, 3, 3));
        ProgressBar(j);
    }

    double X1, Y1, X2, Y2;
    Pen penLine = new Pen(Color.Black, 1);
    Value(1, City - 1);
    for (int i = 1; i < City; i++)
    {
        X1 = (xy[m[i - 1], 0] - xMinMax[0]) * PicX;
        Y1 = (xy[m[i - 1], 1] - yMinMax[0]) * PicY;
        X2 = (xy[m[i], 0] - xMinMax[0]) * PicX;
        Y2 = (xy[m[i], 1] - yMinMax[0]) * PicY;
        GraphicsTSP.DrawLine(penLine, new PointF((float)X1, (float)Y1),
new PointF((float)X2, (float)Y2));
        ProgressBar(i);
    }

    X1 = (xy[m[0], 0] - xMinMax[0]) * PicX;
    Y1 = (xy[m[0], 1] - yMinMax[0]) * PicY;
    X2 = (xy[m[City - 1], 0] - xMinMax[0]) * PicX;
    Y2 = (xy[m[City - 1], 1] - yMinMax[0]) * PicY;

```

```

        GraphicsTSP.DrawLine(penLine, new PointF((float)X1, (float)Y1), new
PointF((float)X2, (float)Y2));
        pictureBox1.Image = imageTSP;
    }
}

```

```

//////////////////////////////////////ACO//////////////////////////////////////
//////////////////////////////////////

```

```

private void ACOAlgorithm()
{
    n = City;
    Pheromone = new double[City, City];
    eta = new double[City, City];
    delta = new double[City, City];
    CitiesTabu = new int[n, City];
    CityNoTabu = new int[n, City];
    VisitedCity = new int[n, City];
    dk = new double[n];
    for (int i = 0; i < City; i++)
    {
        for (int j = 0; j < City; j++)
        {
            Pheromone[i, j] = 1.0;
            if (i != j)
            {
                eta[i, j] = 1.0 / d[i, j];
            }
            delta[i, j] = 0.0;
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < City; j++)
        {
            VisitedCity[i, j] = -1;
            CitiesTabu[i, j] = 0;
        }
        VisitedCity[i, 0] = 0;
        CitiesTabu[i, 0] = 1;
    }

    for (int ic = 1; ic <= TMax; ic++)
    {
        double Pij = 0;
        double PijSum = 0;
        double randomPij = 0;
        Random randData = new Random((int)DateTime.Now.Ticks & 0x0000FFFF);
        for (int Tc = 1; Tc < City; Tc++)
        {
            Value(0, n);
            for (int iAnt = 0; iAnt < n; iAnt++)
            {
                Pij = 0.0;
                PijSum = 0.0;
                randomPij = randData.Next(3000) / 3000.0;

                for (int iCity = 0; iCity < City; iCity++)
                {
                    if (CitiesTabu[iAnt, iCity] == 0)

```

```

        {
            try
            {
                PijSum += Math.Pow(Pheromone[VisitedCity[iAnt, Tc
- 1], iCity], alpha) * Math.Pow(eta[VisitedCity[iAnt, Tc - 1], iCity], beta);
            }
            catch (System.Exception ex)
            {
                continue;
            }
        }
    }
    for (int iCity = 0; iCity < City; iCity++)
    {
        if (CitiesTabu[iAnt, iCity] == 0)
        {
            try
            {
                Pij += Math.Pow(Pheromone[VisitedCity[iAnt, Tc -
1], iCity], alpha) * Math.Pow(eta[VisitedCity[iAnt, Tc - 1], iCity], beta) / PijSum;
                if (Pij > randomPij)
                {
                    CitiesTabu[iAnt, iCity] = 1;
                    VisitedCity[iAnt, Tc] = iCity;
                    break;
                }
            }
            catch (System.Exception ex)
            {
                continue;
            }
        }
    }
    ProgressBar(iAnt + 1);
}

int mk = 0;
bopt1 = opt1;
Value(0, n);
for (int k = 0; k < n; k++)
{
    dk[k] = cd(k);
    if (dk[k] < opt1)
    {
        opt1 = dk[k];
        mk = k;
    }
    ProgressBar(k + 1);
}
Value(0, City);
for (int i = 0; i < City; i++)
{
    m[i] = VisitedCity[mk, i];
    ProgressBar(i + 1);
}

Value(0, n);
for (int k = 0; k < n; k++)
{

```

```

        try
        {
            for (int i = 0; i < City - 1; i++)
            {
                delta[VisitedCity[k, i], VisitedCity[k, i + 1]] += 100 /
dk[k];

            }
            delta[VisitedCity[k, City - 1], VisitedCity[k, 0]] += 100 /
dk[k];

        }
        catch (System.Exception ex)
        {
            continue;
        }
        ProgressBar(k + 1);
    }

    Value(0, City);
    for (int i = 0; i < City; i++)
    {
        for (int j = 0; j < City; j++)
        {
            Pheromone[i, j] = (1 - r) * Pheromone[i, j] + delta[i, j];
            if (Pheromone[i, j] < PheromoneMin)
                Pheromone[i, j] = PheromoneMin;
            if (Pheromone[i, j] > PheromoneMax)
                Pheromone[i, j] = PheromoneMax;
        }
        ProgressBar(i + 1);
    }
    Value(0, n);
    for (int k = 0; k < n; k++)
    {
        for (int c = 0; c < City; c++)
        {
            CitiesTabu[k, c] = 0;
            if (c > 0)
            {
                VisitedCity[k, c] = -1;
            }
        }
        CitiesTabu[k, VisitedCity[k, 0]] = 1;
        ProgressBar(k + 1);
    }
    displayinfo(ic, opt1);

    if (pictureBox1.InvokeRequired)
    {
        Action actionDraw = delegate()
        {
            double W = pictureBox1.Width;
            double H = pictureBox1.Height;
            double PicX, PicY;
            PicX = (double)(W - 30) / (xMinMax[1] - xMinMax[0]);
            PicY = (double)(H - 30) / (yMinMax[1] - yMinMax[0]);
            SolidBrush RedBrush = new SolidBrush(Color.Red);
            pictureBox1.Image = null;
            Image imageTSP = new Bitmap(pictureBox1.Width,
pictureBox1.Height);
            Graphics GraphicsTSP = Graphics.FromImage(imageTSP);
            GraphicsTSP.Clear(Color.White);

```

```

        double cityX;
        double cityY;
        Value(0, City - 1);
        Font myFont1 = new Font("Hacttenschweiler", 7);
        for (int j = 0; j < City; j++)
        {
            cityX = (xy[j, 0] - xMinMax[0]) * PicX;
            cityY = (xy[j, 1] - yMinMax[0]) * PicY;
            GraphicsTSP.FillEllipse(RedBrush, new
RectangleF((float)cityX, (float)cityY, 3, 3));
            ProgressBar(j);
        }

        double X1, Y1, X2, Y2;
        Pen penLine = new Pen(Color.Black, 1);
        Value(1, City - 1);
        for (int i = 1; i < City; i++)
        {
            X1 = (xy[m[i - 1], 0] - xMinMax[0]) * PicX;
            Y1 = (xy[m[i - 1], 1] - yMinMax[0]) * PicY;
            X2 = (xy[m[i], 0] - xMinMax[0]) * PicX;
            Y2 = (xy[m[i], 1] - yMinMax[0]) * PicY;
            GraphicsTSP.DrawLine(penLine, new PointF((float)X1,
(float)Y1), new PointF((float)X2, (float)Y2));
            ProgressBar(i);
        }

        X1 = (xy[m[0], 0] - xMinMax[0]) * PicX;
        Y1 = (xy[m[0], 1] - yMinMax[0]) * PicY;
        X2 = (xy[m[City - 1], 0] - xMinMax[0]) * PicX;
        Y2 = (xy[m[City - 1], 1] - yMinMax[0]) * PicY;
        GraphicsTSP.DrawLine(penLine, new PointF((float)X1,
(float)Y1), new PointF((float)X2, (float)Y2));
        pictureBox1.Image = imageTSP;
    };
    pictureBox1.Invoke(actionDraw);
}
else
{
    double W = pictureBox1.Width;
    double H = pictureBox1.Height;
    double PicX, PicY;
    PicX = (double)(W - 30) / (xMinMax[1] - xMinMax[0]);
    PicY = (double)(H - 30) / (yMinMax[1] - yMinMax[0]);
    SolidBrush RedBrush = new SolidBrush(Color.Red);
    pictureBox1.Image = null;
    Image imageTSP = new Bitmap(pictureBox1.Width,
pictureBox1.Height);
    Graphics GraphicsTSP = Graphics.FromImage(imageTSP);
    GraphicsTSP.Clear(Color.White);
    double cityX;
    double cityY;
    Value(0, City - 1);
    for (int j = 0; j < City; j++)
    {
        cityX = (xy[j, 0] - xMinMax[0]) * PicX;
        cityY = (xy[j, 1] - yMinMax[0]) * PicY;
        GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityX,
(float)cityY, 3, 3));
        ProgressBar(j);
    }
}

```

```

        double X1, Y1, X2, Y2;
        Pen penLine = new Pen(Color.Black, 1);
        Value(1, City - 1);
        for (int i = 1; i < City; i++)
        {
            X1 = (xy[m[i - 1], 0] - xMinMax[0]) * PicX;
            Y1 = (xy[m[i - 1], 1] - yMinMax[0]) * PicY;
            X2 = (xy[m[i], 0] - xMinMax[0]) * PicX;
            Y2 = (xy[m[i], 1] - yMinMax[0]) * PicY;
            GraphicsTSP.DrawLine(penLine, new PointF((float)X1,
(float)Y1), new PointF((float)X2, (float)Y2));
            ProgressBar(i);
        }

        X1 = (xy[m[0], 0] - xMinMax[0]) * PicX;
        Y1 = (xy[m[0], 1] - yMinMax[0]) * PicY;
        X2 = (xy[m[City - 1], 0] - xMinMax[0]) * PicX;
        Y2 = (xy[m[City - 1], 1] - yMinMax[0]) * PicY;
        GraphicsTSP.DrawLine(penLine, new PointF((float)X1, (float)Y1),
new PointF((float)X2, (float)Y2));
        pictureBox1.Image = imageTSP;
    }
}

private double cd(int k)    //function for calculating the current minimum
distance
{
    double currentd = 0;
    Value(0, City - 1);
    for (int i = 0; i < City - 1; i++)
    {
        ProgressBar(i + 1);

        currentd += d[VisitedCity[k, i], VisitedCity[k, i + 1]];
    }
    currentd += d[VisitedCity[k, City - 1], VisitedCity[k, 0]];
    return currentd;
}

////////////////////////////////////////form////////////////////////////////////////
////////////////////////////////////////

//function of the 'File' Button
private void readDataStripMenuItem_Click(object sender, EventArgs e)
{
    label6.Text = "";           // current route length
    label8.Text = "";           //dimension (number of city)
    label9.Text = "";           //name
    label7.Text = "";           //optimum route length
    label10.Text = "";          //status of solver
    OpenFileDialog myFileDialog = new OpenFileDialog();
    myFileDialog.InitialDirectory = Application.StartupPath + "\\Data";
    xMinMax = new double[2] { double.MaxValue, 0 };
    yMinMax = new double[2] { double.MaxValue, 0 };
    if (myFileDialog.ShowDialog() == DialogResult.OK)
    {
        string strPath = myFileDialog.FileName;
        StreamReader streamReader = new StreamReader(strPath);
        string Data;
        string[] Array;
    }
}

```

```

string dataFlag = "no";
int i = 0;

while ((Data = streamReader.ReadLine()) != "EOF")
{
    if (Data.Contains("NAME"))
    {
        Array = Data.Split(':');
        label19.Text = Array[1];
    }
    if (Data.Contains("Optimum"))
    {
        int si; //starting index
        si = Data.IndexOf("is") + 2;
        label17.Text = Data.Substring(si);
    }
    if (Data.Contains("DIMENSION"))
    {
        Array = Data.Split(':');
        label8.Text = Array[1] + " cities";
        City = int.Parse(Array[1].Trim());
        xy = new double[City, 2];
        m = new int[City];
        d = new double[City, City];
        CityList = new int[City];
        Value(0, City);
    }

    if (dataFlag == "yes")
    {
        Array = Data.Split(' ');
        xy[i, 0] = double.Parse(Array[1]);
        xy[i, 1] = double.Parse(Array[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++;
        ProgressBar(i);
    }
    if (Data.Contains("NODE_COORD_SECTION"))
    {
        dataFlag = "yes";
    }
}
streamReader.Close();
//Draw city points
Value(0, City);
if (pictureBox1.InvokeRequired)
{

```

```

        Action actionDrawCityPoint = delegate()
        {
            double W = pictureBox1.Width;
            double H = pictureBox1.Height;
            double PicX, PicY;
            PicX = (double)(W - 30) / (xMinMax[1] - xMinMax[0]);
            PicY = (double)(H - 30) / (yMinMax[1] - yMinMax[0]);
            SolidBrush RedBrush = new SolidBrush(Color.Red);
            pictureBox1.Image = null;
            Image imageTSP = new Bitmap(pictureBox1.Width,
pictureBox1.Height);
            Graphics GraphicsTSP = Graphics.FromImage(imageTSP);
            GraphicsTSP.Clear(Color.White);
            double cityX;
            double cityY;
            for (int j = 0; j < City; j++)
            {
                cityX = (xy[j, 0] - xMinMax[0]) * PicX;
                cityY = (xy[j, 1] - yMinMax[0]) * PicY;
                GraphicsTSP.FillEllipse(RedBrush, new
RectangleF((float)cityX, (float)cityY, 3, 3));
                ProgressBar(j + 1);
            }
            pictureBox1.Image = imageTSP;
        };
        pictureBox1.Invoke(actionDrawCityPoint);
    }
    else
    {
        double W = pictureBox1.Width;
        double H = pictureBox1.Height;
        double PicX, PicY;
        PicX = (double)(W - 30) / (xMinMax[1] - xMinMax[0]);
        PicY = (double)(H - 30) / (yMinMax[1] - yMinMax[0]);
        SolidBrush RedBrush = new SolidBrush(Color.Red);
        pictureBox1.Image = null;
        Image imageTSP = new Bitmap(pictureBox1.Width,
pictureBox1.Height);
        Graphics GraphicsTSP = Graphics.FromImage(imageTSP);
        GraphicsTSP.Clear(Color.White);
        double cityX;
        double cityY;
        for (int j = 0; j < City; j++)
        {
            cityX = (xy[j, 0] - xMinMax[0]) * PicX;
            cityY = (xy[j, 1] - yMinMax[0]) * PicY;
            GraphicsTSP.FillEllipse(RedBrush, new RectangleF((float)cityX,
(float)cityY, 3, 3));
            ProgressBar(j + 1);
        }
        pictureBox1.Image = imageTSP;
    }

    Value(0, City);
    for (int j = 0; j < City; j++)
    {
        for (int k = 0; k < City; k++)
        {
            d[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]));
        }
        ProgressBar(j + 1);
    }
}

```

```

    }
}

//define the progress bar
public void ProgressBar(int BarValue)
{
    if (progressBarTSP.InvokeRequired)
    {
        Action<int> actionUpdateProgressBar = delegate(int Temp)
        {
            progressBarTSP.Value = Temp;
        };
        progressBarTSP.Invoke(actionUpdateProgressBar, BarValue >
progressBarTSP.Maximum ? progressBarTSP.Maximum : BarValue);
    }
    else
    {
        progressBarTSP.Value = BarValue;
    }
}

//set the value of the progress bar
public void Value(int minValue, int maxValue)
{
    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 FormTSP_Load(object sender, EventArgs e)
{
    label16.Text = "";
    label18.Text = "";
    label19.Text = "";
    label17.Text = "";
    label10.Text = "";
}

//function of the 'Greedy Solver' Button
private void greedySolverToolStripMenuItem_Click(object sender, EventArgs e)
{
    label10.Text = "Greedy algorithm progress";
    opt1 = double.PositiveInfinity;
    bopt1 = double.PositiveInfinity;
    Thread threadGreedy = new Thread(new ThreadStart(Greedy));
    threadGreedy.IsBackground = true;
    threadGreedy.Start();
}

```

```

//function of the 'ACO Solver' Button
private void ACOSolverToolStripMenuItem_Click(object sender, EventArgs e)
{
    label10.Text = "ACO algorithm progress";
    opt1 = double.PositiveInfinity;
    bopt1 = double.PositiveInfinity;
    int T = 0;
    Thread threadACO = new Thread(new ThreadStart(ACOAlgorithm));
    threadACO.IsBackground = true;
    threadACO.Start();
}

public void displayinfo(int p, double m1)
{
    double fImprovement = 0;
    if (label6.InvokeRequired)
    {
        Action<double> l = delegate(double s1)
        {
            label6.Text = s1 == 0 ? "" : s1.ToString("F0");
        };
        label6.Invoke(l, m1);
    }
    else
    {
        label6.Text = m1 == 0 ? "" : m1.ToString("F0");
    }
}

////function of the 'Exit' Button
private void exitToolStripMenuItem_Click_1(object sender, EventArgs e)
{
    Application.Exit();
}

}
}

```