

ELECTRONIC, ELECTRICAL AND SYSTEM ENGINEERING



UNIVERSITY OF
BIRMINGHAM

Object Oriented Programming Using C#

Assignment 2014-15

Name:

Hanxiao Lin

Student ID Number:

1448679

Abstract Nowadays, plenty of programming languages (C, C++, Java for example) provide more and more choices for human to solve the different problems in our daily life. C# can be considered as the Object Oriented Programming (OOP), which is a kind of programming development method which mainly focus on the objects. As we know, the traditional programming designs are based on the functions, or mathematics, however, in the Object Oriented Programming, each object is able to receive data and send the data to the other objects after analyzing. So, it is obvious that C# allows us to get the solutions in a more convenient way.

This paper is going to design a program by using the Unified Modeling Language (UML) and C# to solve the Traveling Salesman Problem (TSP), which is one of the most typical optimization problems in the world.

In order to find the optimal scenario, the Greedy algorithm and Ant-based optimization (ACO) algorithm will be used in designing. By using the Greedy algorithm and ACO algorithm, the program can provide several different answers to the Traveling Salesman Problem.

Key Words: OOP C# TSP ACO

Table of Content

Abstract.....	I
1. Introduction.....	1
1.1. The Traveling Salesman Problem.....	1
1.2. Greedy Algorithm.....	1
1.3. ACO Algorithm.....	1
2. Use Case View.....	2
2.1 Potential Requirements.....	2
2.2 Scenario Descriptions.....	4
2.3 Class Identification.....	5
2.4. CRC Cards.....	6
2.5. Interaction Diagram.....	8
2.6. Statechart Diagram.....	9
2.7 Class Diagram.....	9
3. Analysis Model.....	11
3.1. Attributes.....	11
3.2. Methods.....	12
3.3. Sequence Diagram.....	13
3.4. Class Diagram.....	14
3.5. Statechart Diagram.....	15
3.6. Non-functional Requirements.....	15
3.7. Packages.....	15

4. Design Modeling.....	16
4.1. Revisit Use-Case Model.....	16
4.2. Sequence Diagram.....	16
4.3. Textual Description of Object to Object Interaction.....	16
4.4. Subsystems.....	17
4.5. Implementation of Non-functional Requirements.....	17
4.6. Legacy Issues.....	17
4.7. Reconsider the Attributes.....	17
4.8. StateChart.....	17
4.9. Key Implementation.....	17
4.10. Class Diagram Showing Visibility.....	18
5. Using C# to Realize the Design.....	19
5.1 Usability of the Graphical User Interface.....	19
5.2 Execute Algorithm.....	21
5.2.1 Greedy Algorithm.....	22
5.2.2 ACO Algorithm.....	22
5.3 Summary.....	23
6. Conclusions.....	24
References.....	25
Appendix.....	26

List of Figures

Figure 2.1. Use-Case Diagram.....	3
Figure 2.3. Class Diagram (Stereotypes).....	6
Figure 2.5. Interaction Diagram: Sequence Diagram.....	8
Figure 2.6. Initial Statechart.....	9
Figure 2.7.1. Class Diagram (Use-Case).....	10
Figure 2.7.2. Class Diagram (Use-Case).....	10
Figure 3.3. Analysis Model Sequence Diagrams.....	13
Figure 3.4. Analysis Model Class Diagram.....	14
Figure 3.5. Analysis Model Statechart.....	15
Figure 4.10. Design Model Class Diagram Showing Visibility.....	18

List of Pictures

Picture 5.1.1. GUI Form.....	19
Picture 5.1.2. Open a TSP File.....	20
Picture 5.1.3. TSP File Loaded.....	20
Picture 5.1.4. 'About' Button Pressed.....	21
Picture 5.2. 'Solvers' Button Pressed.....	21
Picture 5.2.1. Greedy Algorithm.....	22
Picture 5.2.2. ACO Algorithm.....	23

List of Tables

Table 2.1. Potential Requirements.....	2
Table 2.4. Class Responsibility Collaboration Cards.....	7
Table 3.1. Attributes.....	11
Table 3.2. Methods.....	12

1. Introduction

1.1. The Traveling Salesman Problem

The Traveling Salesman Problem is an optimization problem which describing a salesman wants to travel to all the cities in the map and tries to find the shortest path. In order to explain in a more clearly way, we can set of N , E represent cities and the arcs which connect every nodes N . Also we can create cities i and j , so we can use d_{ij} to represent the distance between them ($i, j \in N$). Obviously, if each city must be visited once and once only, we can say that the TSP is looking for the minimal length d_{ij} in a closed tour.

1.2. Greedy Algorithm

Greedy algorithm is an algorithm which only choose the best solution in each step. By doing so, can the algorithm get the optimization solution. In this design, for solving the TSP, Greedy always choose the nearest city as salesman's next destination, finally, salesman can travel all cities. In a word, Greedy algorithm always goes to the city which has the smallest d_{ij} .

1.3. ACO Algorithm

Ant-based optimization (ACO) algorithms give another way to solve the TSP. Such algorithm based on the ants' pheromone as they traveled. It is not hard to image that two ants in city i and both of them need to go to city j . However, there are two ways for them to choose, and they choose the different path. So, in the same time period, ants travel between cities i and j constantly and left same pheromone per step, the shorter path may deposit more amount of pheromone. If we define τ is the current pheromone and it is easy to discover where the τ is highest, the d_{ij} is shortest path.

2. Use Case View

2.1 Potential Requirements

Table 2.1. Potential Requirements.

Concept	Necessity	Risk	Priority
A friendly, convenience interface.	High	Low	High
Showing the cities clearly.	High	Low	Med
Showing the TSP file's information.	Low	Low	Med
Giving the optimization path and its length.	High	Med	High
Different plans should be provided.	Medium	Med	Med
Program can read another TSP file at any time.	Medium	Low	Med
Information of the program can be found.	Low	Low	Med
User can exit as long as they want.	High	Low	Med

According to the Table 2.1 shown that there is no huge risk with all the requirements mentioned above. Therefore, these additional requirements should be included in the design.

The Use-Case diagram is shown in Figure 2.1 on the following page.

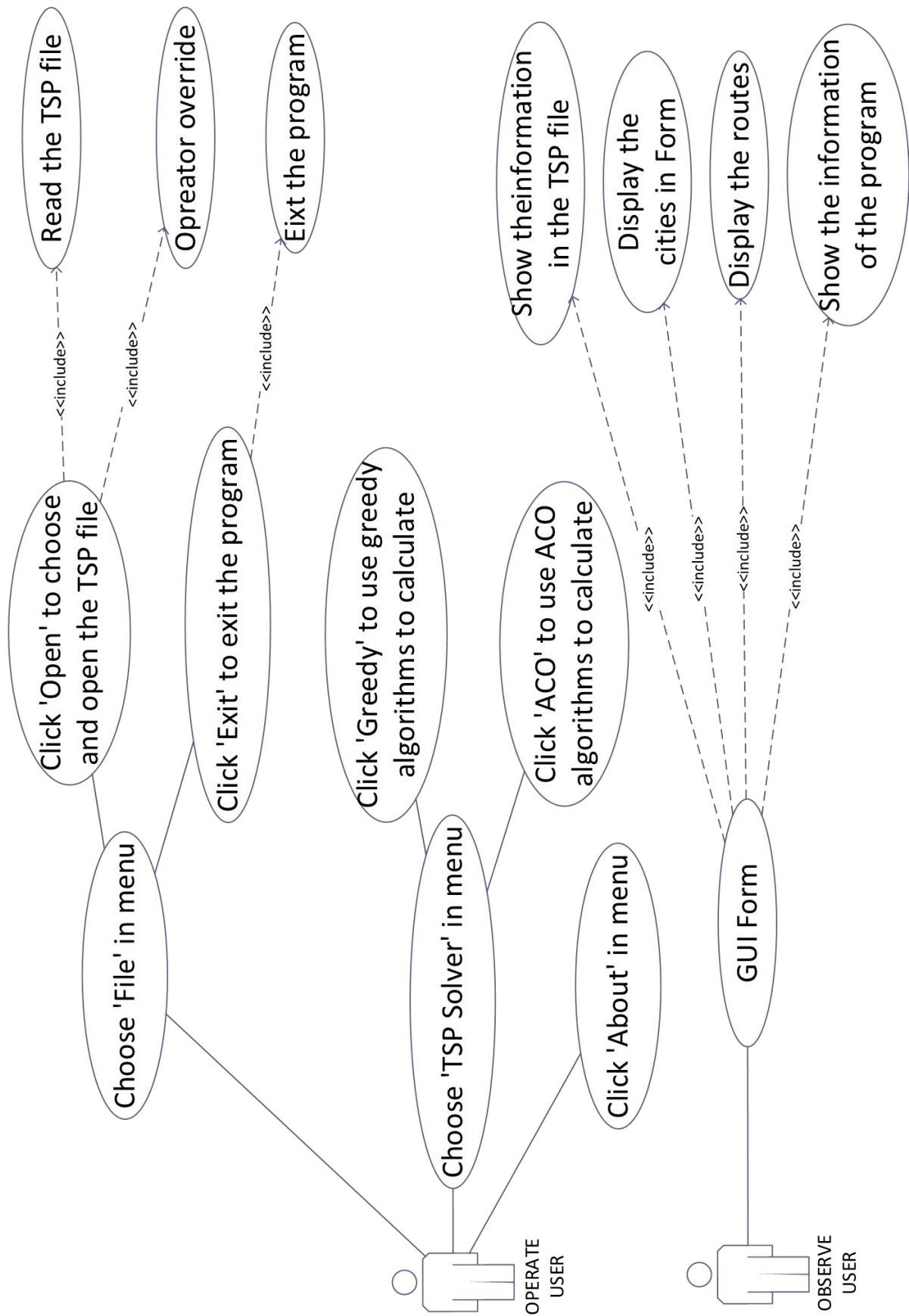


Figure 2.1. Use-Case Diagram.

2.2 Scenario Descriptions

- Operate User(1): The user wants to get the optimization route among the cities, by selecting the 'File'→'Open' in top menu to choose the map in need. User can click the 'About' in top menu to get the information of the program (author or program version) if they want.
- GUI Form: As long as the user choose the TSP file they want to analysis, GUI form shows the cities as points in the display area. The information of TSP file (file's name, cities number, optimization length) should be shown at the mean time. As user click the about button, a message box which contains the program's information should pop in the middle of the screen.
- Map area: The information of TSP file (file's name, cities number, optimization length) should be shown at the mean time which allow the observe user to check.
- Observe User(1): Check the cities, file information shown on the map area, GUI form respectively. As everything is checked, go on to operate the GUI form.
- Operate User(2): As the cities shown in the display area, user can choose the algorithm they want to use (Greedy algorithm or ACO algorithm).
- GUI Form: After the user choose the algorithm, program allow the codes begin to operate.
- Map area: When the calculate finish, the path should give and draw as lines in the display area between the cities points.
- Observe User(2): Check the every detail shows on the map area and GUI form (cities and paths shown on the map, Information shown on the GUI form).
- Operate User(3): User may stop the program at any time they want by clicking the 'File'→'Exit' in top menu.
- GUI Form: Allow the program stop and exit when system receive the exit command.

Override: When another TSP file is loaded, system should erase the old cities and stop calculate no matter what is going on. Then, the information and cities in new TSP file should be shown in the certain area.

2.3 Class Identification

Then 'Nouns' mentioned above are: TSP File, Program, Cities, Paths, Greedy algorithm, ACO algorithm, GUI Form, Click Buttons, Map, Cities' Distance, Ants, Pheromone, User. The noun 'User' is external to the software to be design.

As we taught that nouns might have too few attributes or methods associated with them, that's why stereotypical classes are needed.

Stereotypical classes:

Boundary: Click Buttons, GUI Form, Map

Analysis: Cities List, Cities' Distance, Pheromone

Control: Codes

According to the 'Nouns' and the stereotypical classes, it is clear that we can get a minimal set of classes: Codes, GUI Form, Map.

The GUI Form and Map are interface class. GUI form contains the buttons and Map cluster the cities and paths. Codes class is the core of the design, it not only manage the action of Map class, but also contains algorithms which are used for analyzing TSP data. So, the code class can be defined as entity and control class. As shown in Figure2.3.

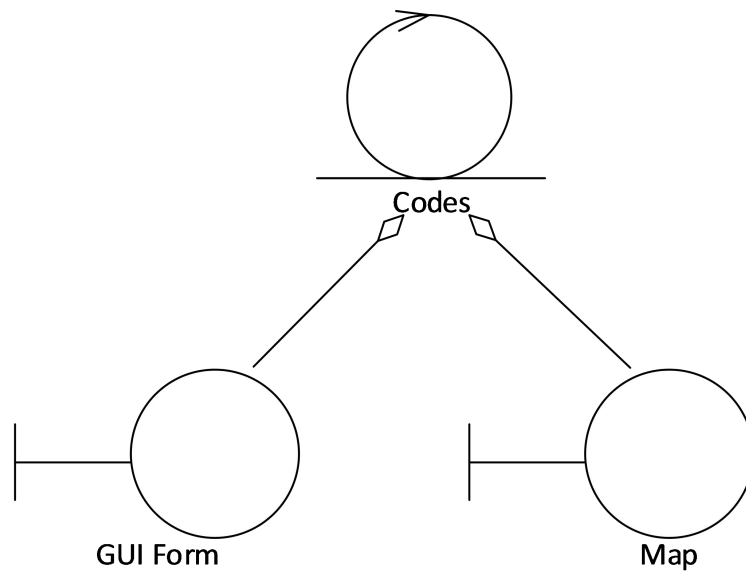


Figure 2.3. Class Diagram (Stereotypes).

2.4. CRC Cards

For identifying attributes, methods and inter-class relationships. CRC (Class Responsibility Collaboration) cards not only show the collaborators of each class, but also describe its responsibilities. That's why CRC cards are quite significant in Analysis stage and should be drafted during this modeling.

Table 2.4. Class Responsibility Collaboration Cards.

Class: GUI Form	
Responsibilities	Collaborators
The GUI Form is responsible for updating the information in TSP file which is loaded. This class not only should include the function choices, but also included algorithms. Exit button and Copyright button should be used too. Solver button operate the program to start using the algorithms the user chosen to calculate the optimal path. GUI Form also allow the user load the TSP file in any cases.	Codes, Map

Class: Map	
Responsibilities	Collaborators
The map class is responsible for showing the points which represent the cities. When the program get the solution, map should draw lines represent the optimal path. By using lines, the points be linked together and when all points be linked, the mapping is finished. As long as there is any TSP file be loaded, map should erases everything and draw points as cities immediately.	Codes, GUI Form

Class: Codes	
Responsibilities	Collaborators
The codes class is responsible for calculate the optimal path between the cities. The codes class messages the map which city is going to visit next. If all cities have been visited, the program stops to wait for other commands. The codes class also responds to the new TSP file. When new file loaded, the algorithms override all operate.	GUI Form, Map

2.5. Interaction Diagram

Collaboration and sequence diagram or called Interaction diagram, describes how instances of use-case interact each other and how classes interact. These diagrams can help us to understand the user requirements in a more clearly way.

The Sequence Diagrams are shown in Figure 2.5 on the following page.

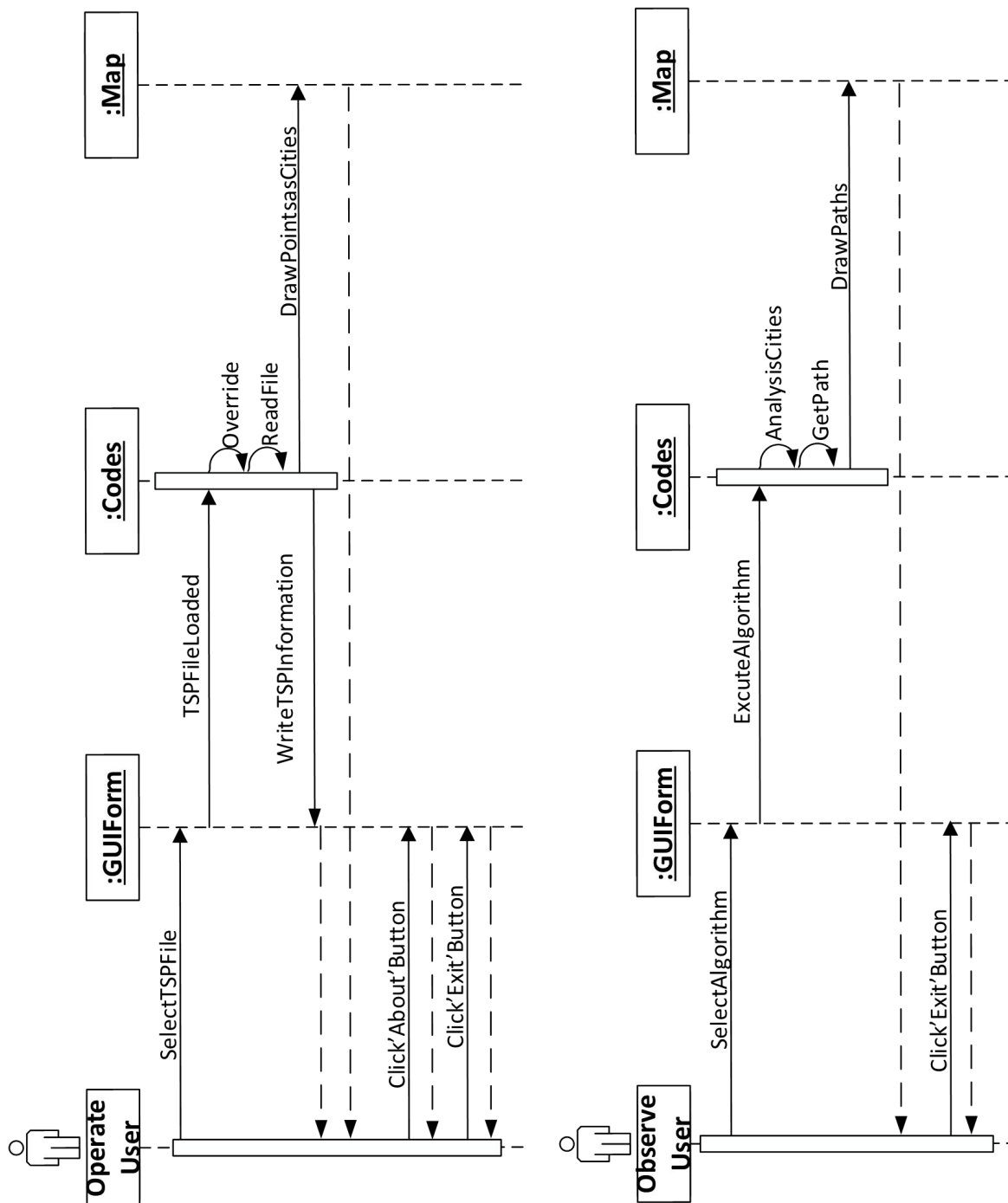


Figure 2.5. Interaction Diagram: Sequence Diagram.

2.6. Statechart Diagram

This diagram describes the transitions between different states. The states are: Idle, Draw Point as Cities, Algorithms, Draw Lines as Paths, Exit response.

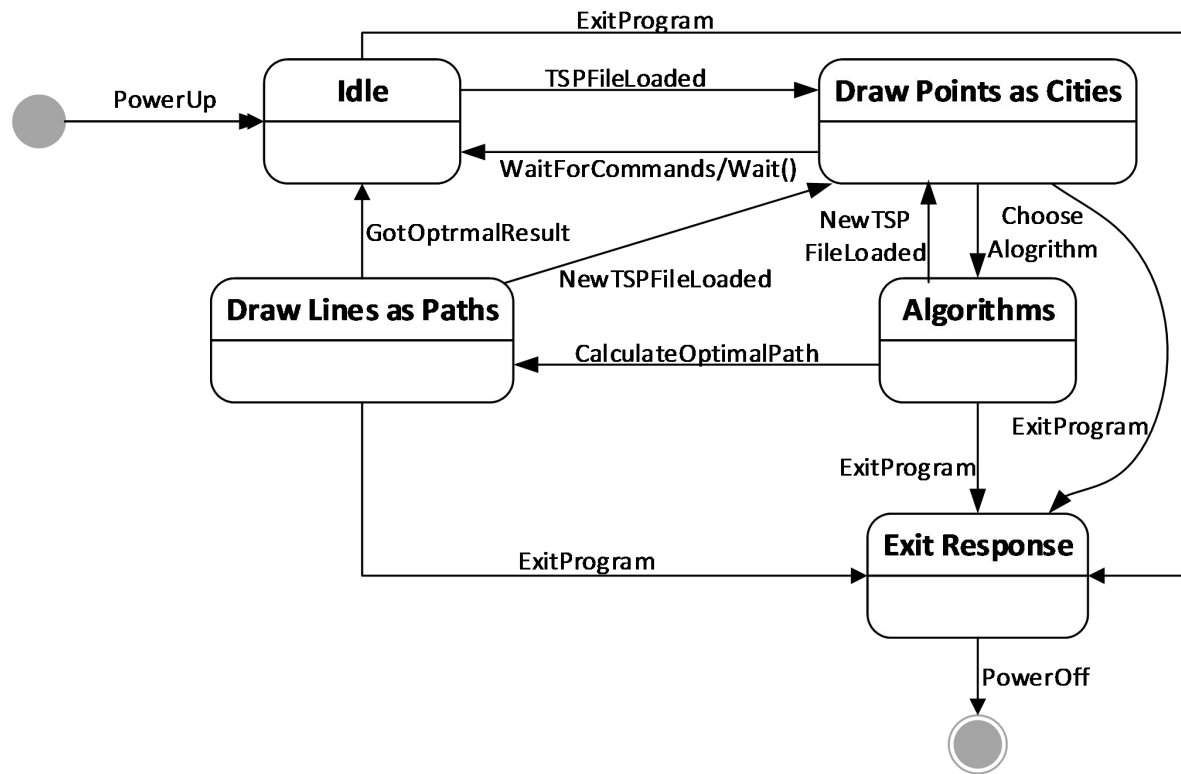


Figure 2.6. Initial Statechart.

2.7 Class Diagram

An initial class diagram showing the relationship (composition or aggregation) between classes. Not all of the classes are contain in this diagram and there are not any privacy, visibility or inheritance of methods and attributes described. Shown as Figure 2.7.1 and Figure 2.7.2.

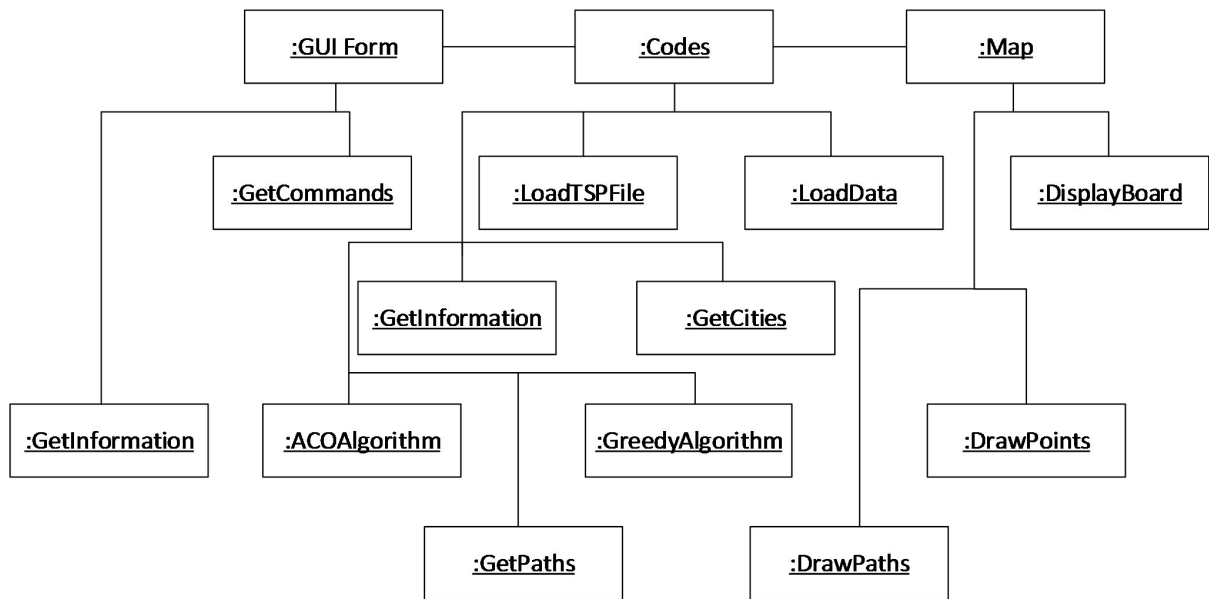


Figure 2.7.1. Class Diagram (Use-Case).

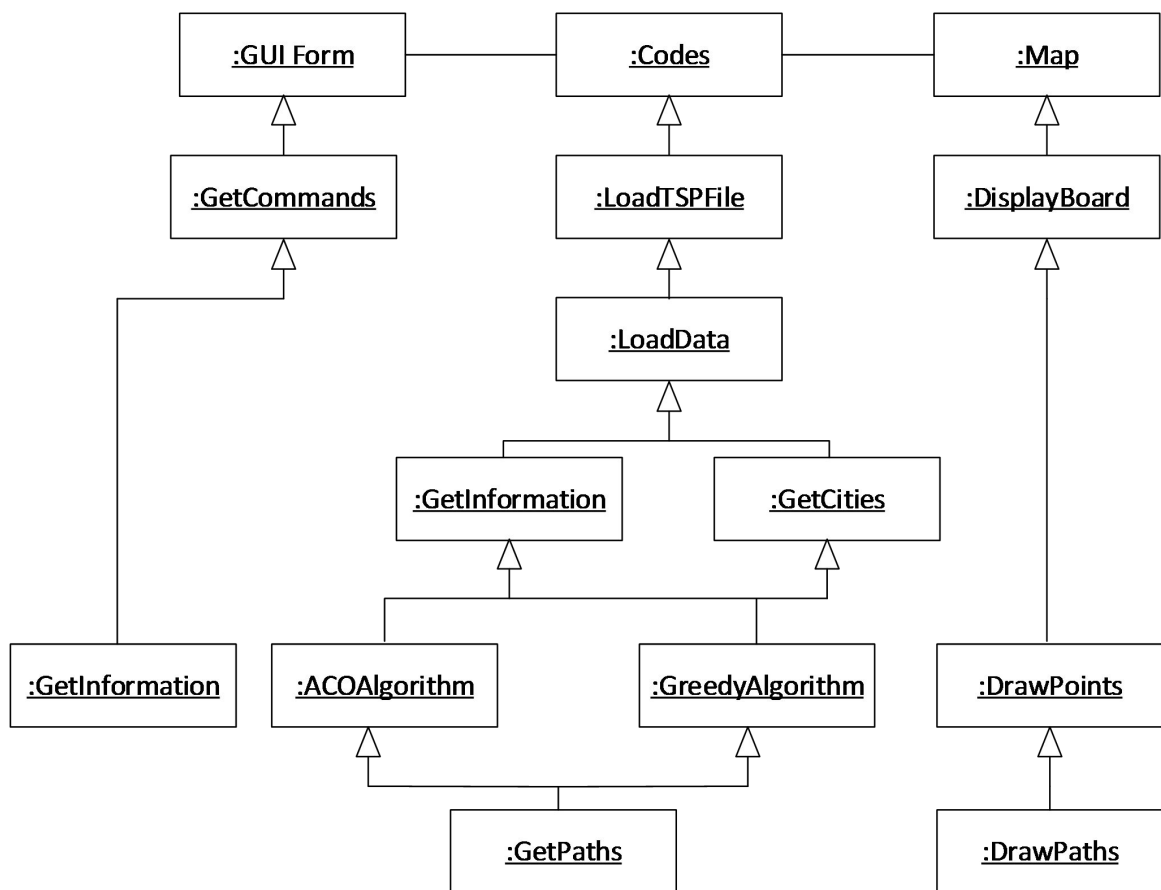


Figure 2.7.2. Class Diagram (Use-Case).

3. Analysis Model

3.1. Attributes

Table 3.1. Attributes.

Class	Attribute	Comment
GUI Form	FileButtonPressed	Boolean
	OpenButtonPressed	Boolean
	DataRead	Status
	SelectTSPFile	Status
	CitiesPosition	Double Array
	ShowInformation	String
	SolverButtonPressed	Boolean
	AlgorithmButtonPressed	ACO/Greedy
	AboutButtonPressed	Boolean
	ExitButtonPressed	Boolean
Map	CitiesPositions	Double Array
	Paths	Double Array
	ClearMap	Boolean
Codes	LoadTSPFile	Status
	CitiesList	Double Array
	SelectDestination	Double Array
	Algorithm	ACO/Greedy
	TotalLength	Integer

3.2. Methods

Table 3.2. Methods.

Class	Methods	Comment
GUI Form	ShowToolStripMenu()	
	ShowOpenFileDialog()	
	OpenTSPFile()	Load the TSP file
	LoadData()	
	SetCities()	
	WriteInformation()	Name, length, number of cities
	ChooseAlgorithm()	
	ShowProgramInformation()	Provide author's information
Map	Eixt()	
	DrawCitiesPoints()	According to TSP file
	DrawPathsLines()	According to algorithm
	EraseMap()	Clean cities and paths
Codes	FileLoaded()	Information read
	ExecuteAlgorithm()	Program calculating
	SelectNextCity()	Choose an un-visited city
	DeleteVisitedCity()	
	GetTotalLength()	
	ExitResponse()	Manage exit response

3.3. Sequence Diagram

In this part, more details are considered and the triggers are added. The diagram also has some changes according to the consideration of attributes and methods. The Diagram shown as Figure 3.3 on the following pages.

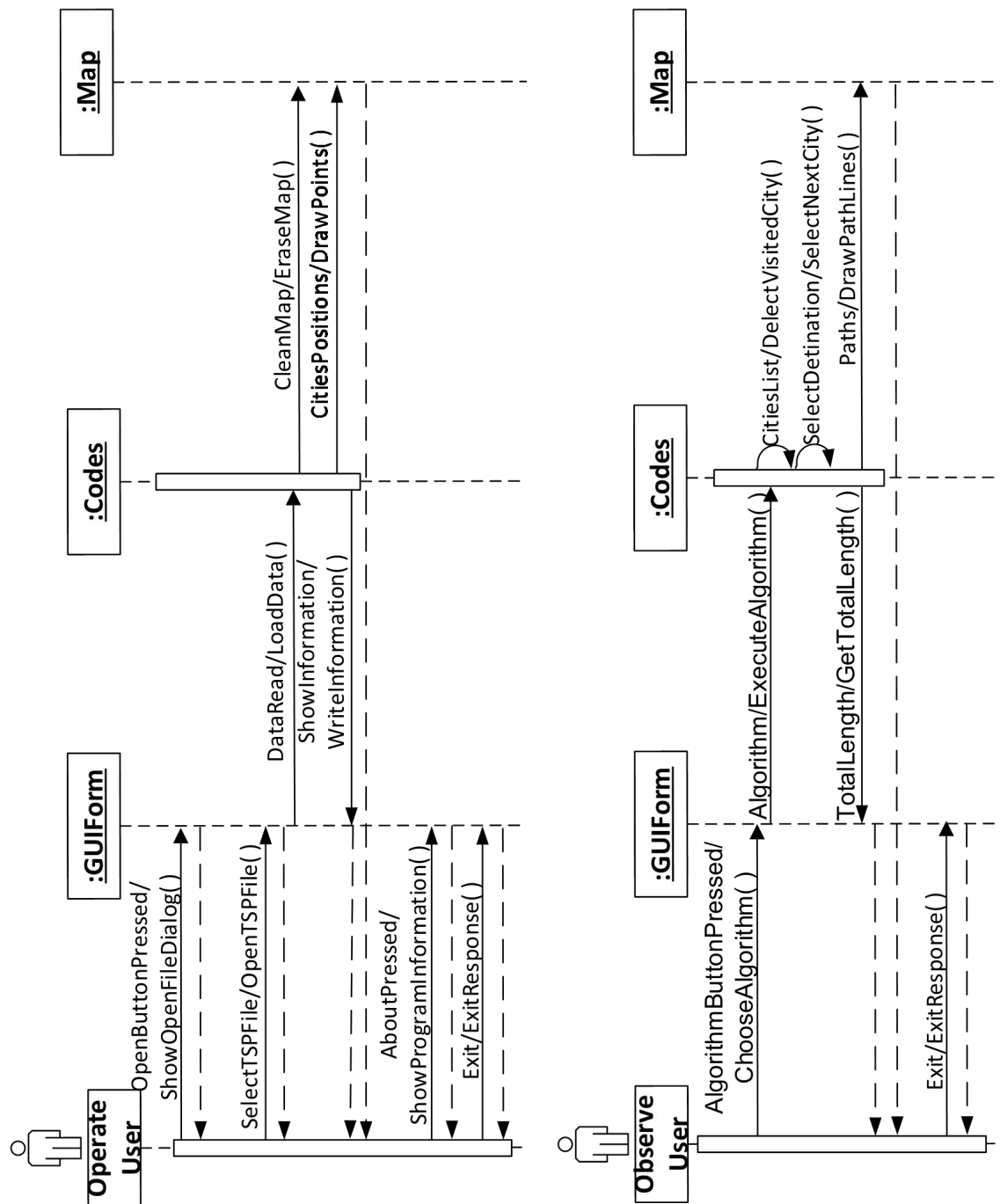


Figure 3.3. Analysis Model Sequence Diagrams.

3.4. Class Diagram

Whole class diagram with data types in Figure3.4 on the following pages.

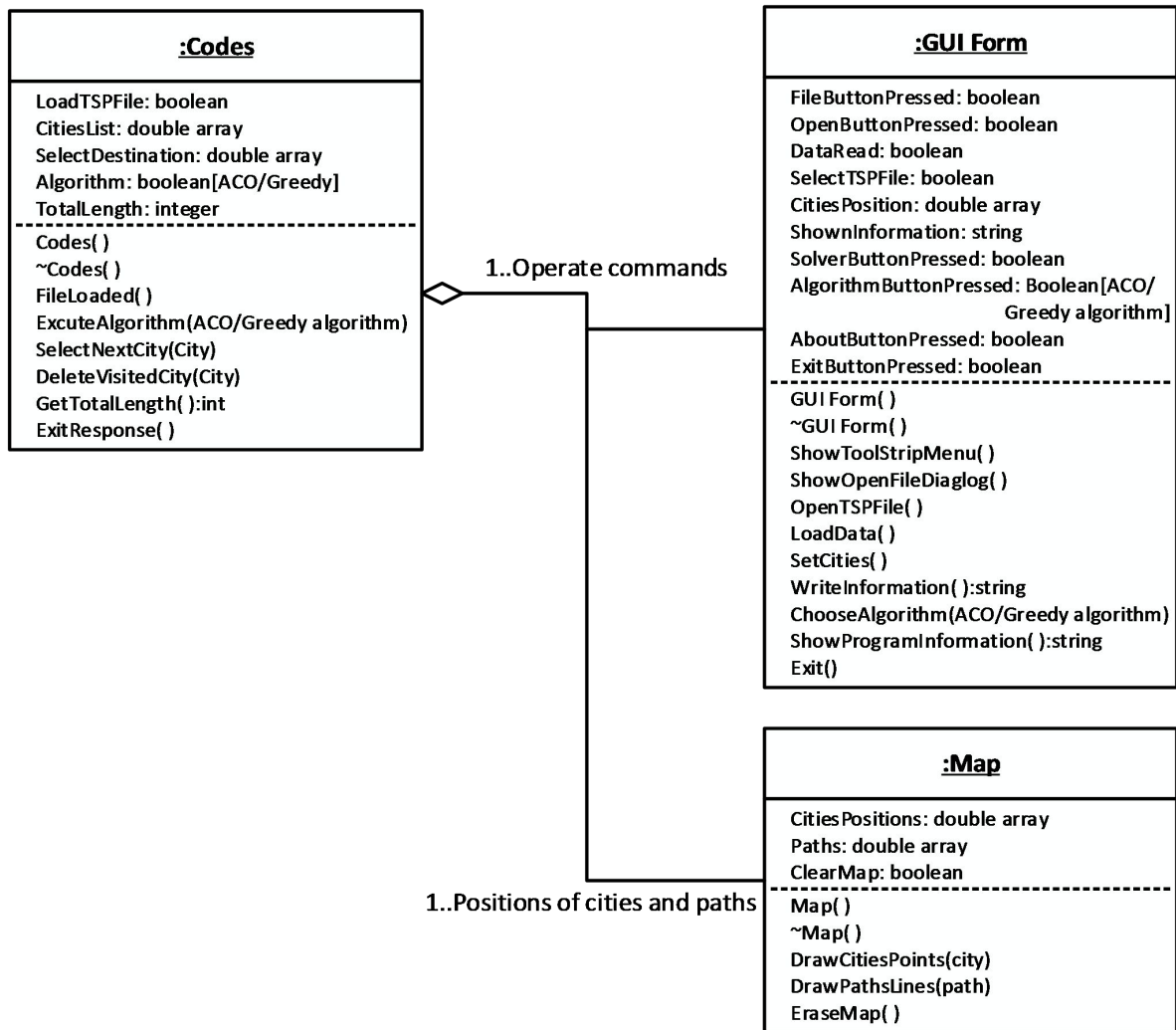


Figure 3.4. Analysis Model Class Diagram.

3.5. Statechart Diagram

The changed statechart in Figure3.5 on the following pages.

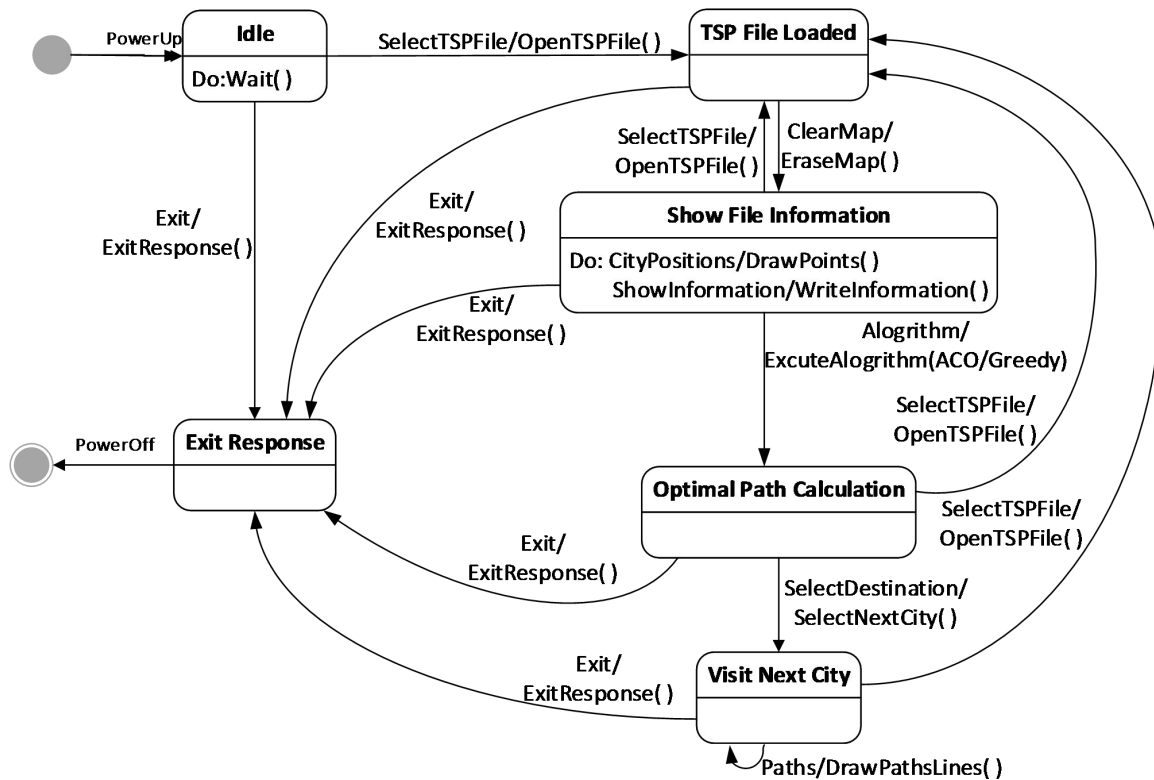


Figure 3.5. Analysis Model Statechart.

3.6. Non-functional Requirements

There is nothing to do when the power fail.

3.7. Packages

Because there are only three classes in this design, so the packages is unnecessary.

4. Design Modeling

4.1. Revisit Use-Case Model

According to the analysis above and consider about the use requirements shown as Use-Case Diagram in Figure 2.1, all the requirements are met.

4.2. Sequence Diagram

No more details needed in this case.

4.3. Textual Description of Object to Object Interaction

Just consider about the user requirements, the system is not so difficult and the diagram showed above also provide a clear description.

The GUI Form class handles interaction with user and make possible for user to control the whole program. User not only can let program load TSP file, but also shows information of file and allow user exit at any time.

The Map class shows the positions and paths for the user which make everything becomes easier to understand.

The Codes class is the core of the design. The class contains the ACO algorithm, Greedy algorithm and other interface code which are the most complex part of the design.

4.4. Subsystems

The design does not need the subsystems due to its property.

4.5. Implementation of Non-functional Requirements

Showing the program's information is the non-functional requirement in this design, and the implementation of it can be found in GUI Form class, though its codes are in the Codes class.

4.6. Legacy Issues

This design is based on C#, which allow other to rewrite the codes. So, others can use this design to solve the similar problem with a little changes in codes.

4.7. Reconsider the Attributes

The attributes are incorporated in the Class Diagram showed above in Figure3.4.

4.8. StateChart

No further revisions required.

4.9. Key Implementation

It is obvious that the key of implementation of this design is the algorithm which located in the codes.

4.10. Class Diagram Showing Visibility

Both Greedy algorithm and ACO algorithm need to delete the data when another TSP file is loaded so additional Method required. The diagram shown in Figure 4.10.

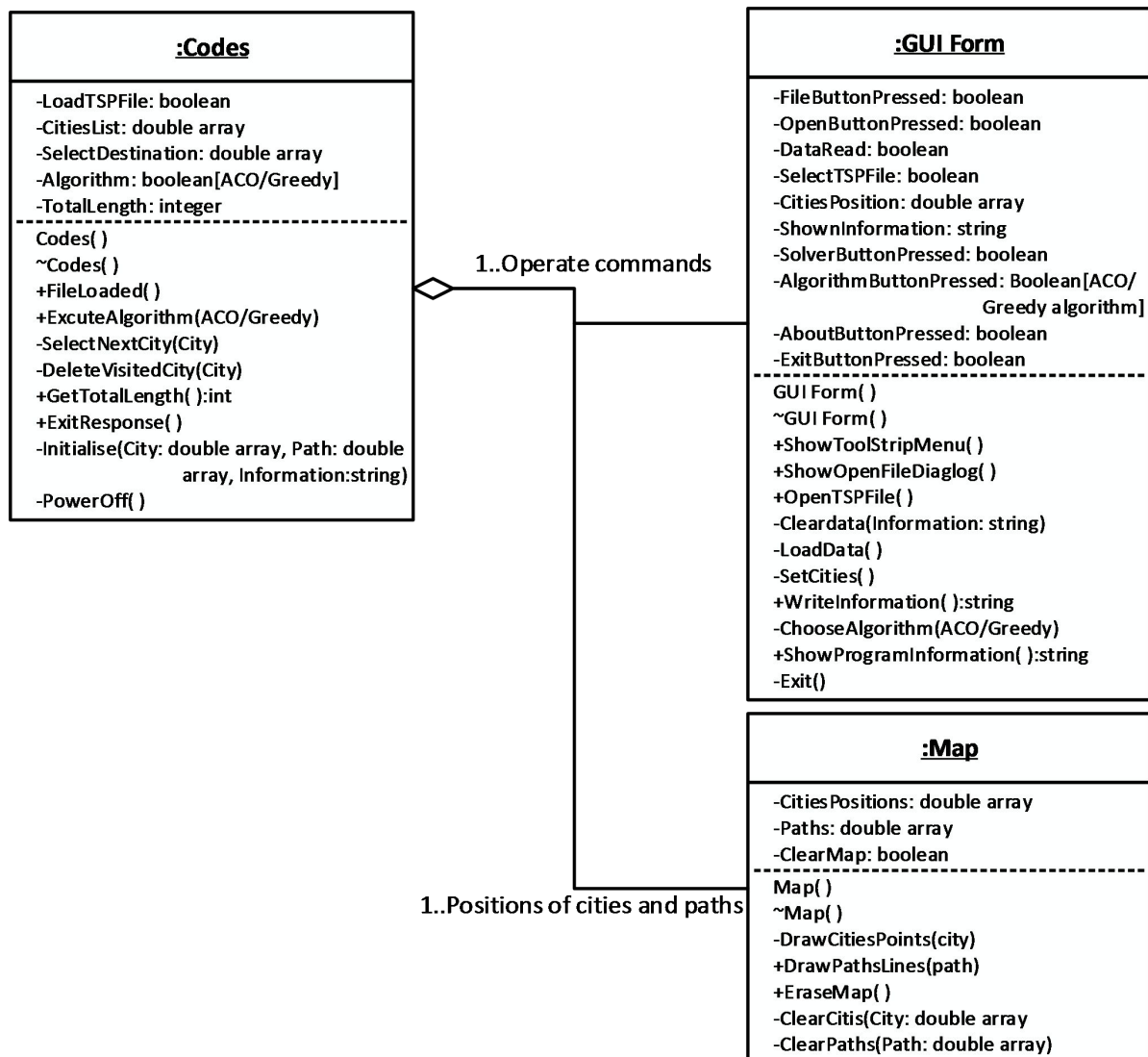
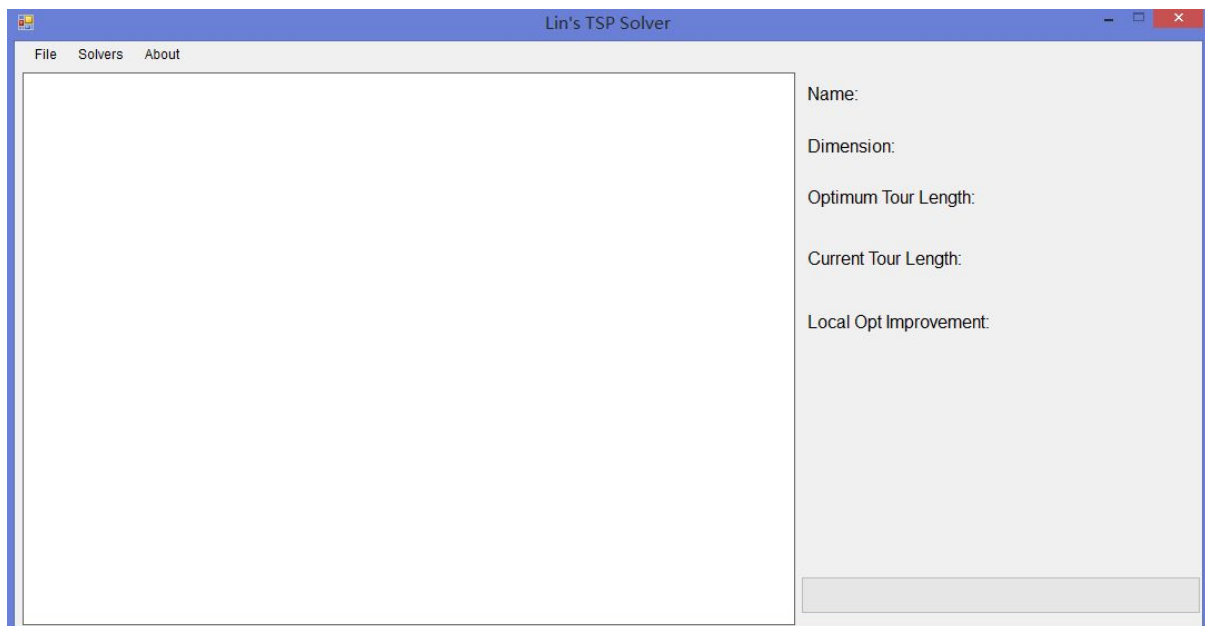


Figure 4.10. Design Model Class Diagram Showing Visibility.

5. Using C# to Realize the Design

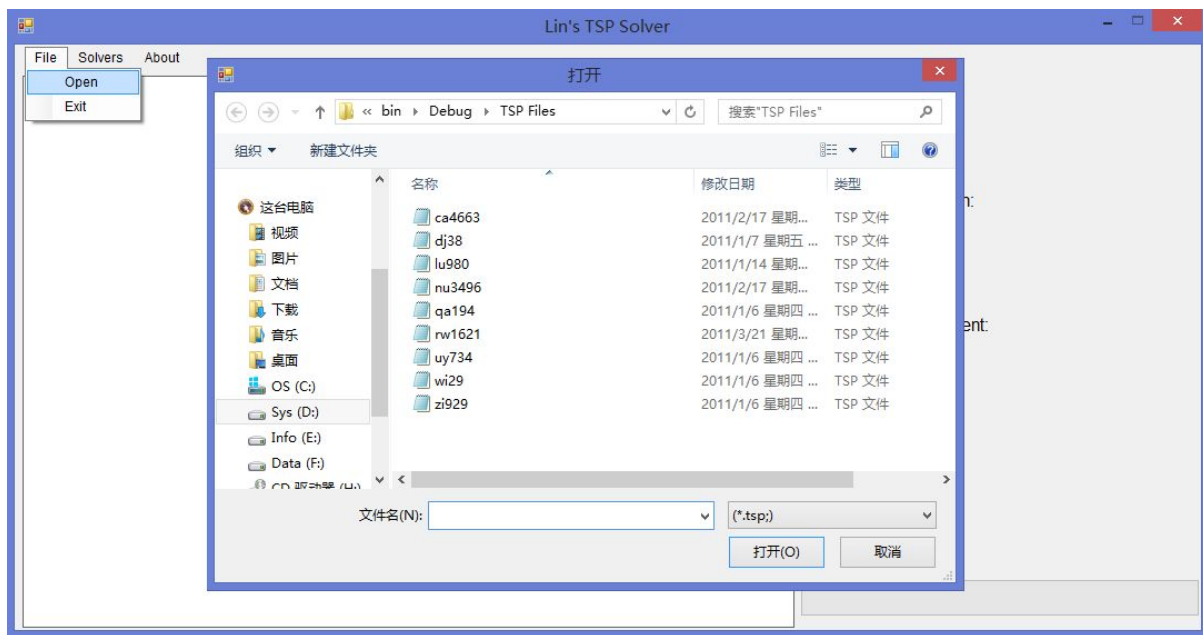
5.1 Usability of the Graphical User Interface

This part will give out the efficiency of the Graphical User Interface (GUI). According to the user requirements and the design shown above, the GUI should consist of friendly interface and commands clearly. As shown in Picture 5.1.1.



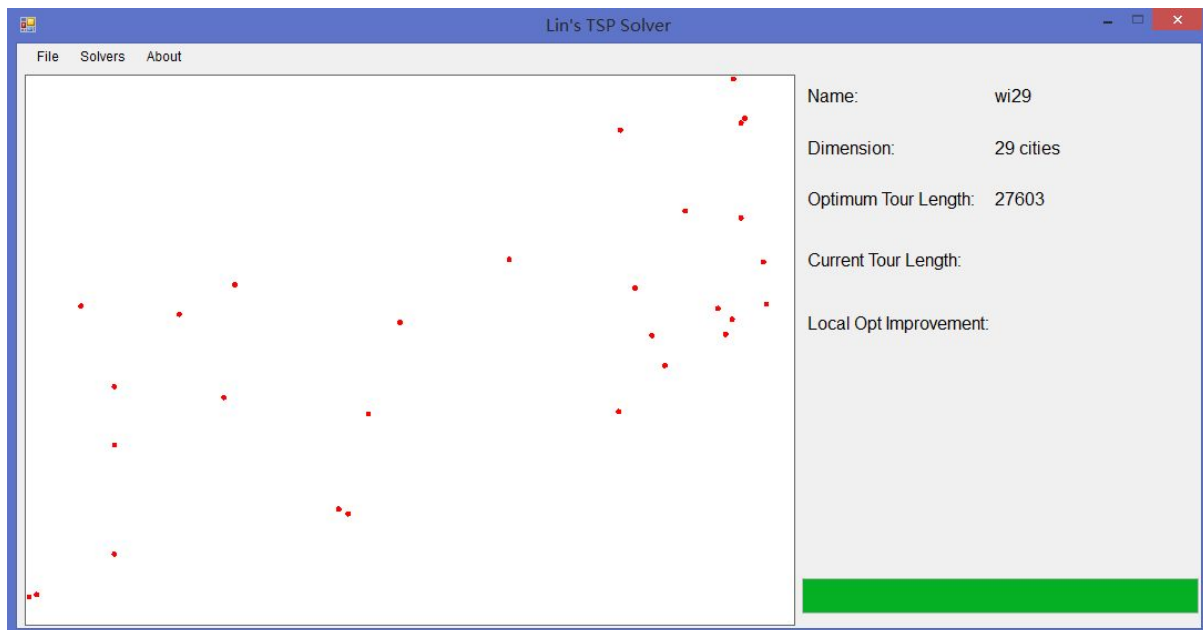
Picture 5.1.1. GUI Form.

It is easy to find that there are three commands can be selected at menu bar. 'File', 'Solvers' and 'About' respectively. By clicking 'File' button, can we find 'Open' and 'Exit' buttons, as clicking 'Open', an open file dialog pop in the middle of the screen which allow user to choose the TSP file to be loaded. 'Exit' allow user to exit the program. And the priority of them is the highest which means whether system is calculating or drawing map, it must load another TSP file or exit the program as the commands from the user immediately. As shown in Picture 5.1.2.



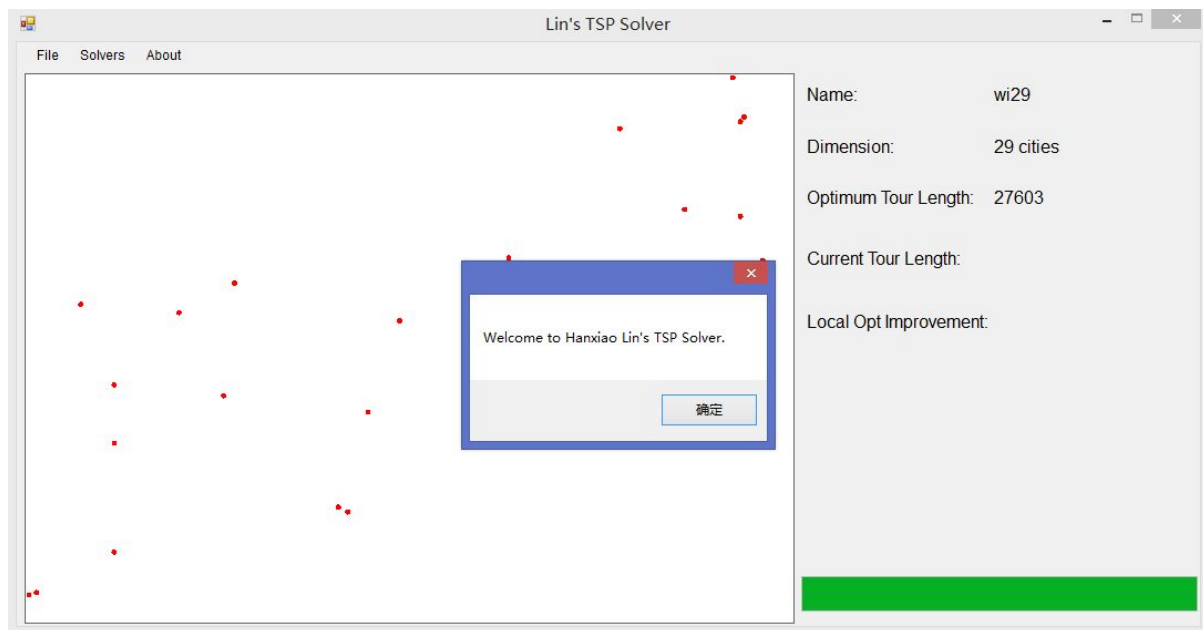
Picture 5.1.2. Open a TSP File.

After a TSP file loaded, the cities and information should be shown in display area as shown in Picture 5.1.3 (In this case, we use 'wi29.tsp' for testing).



Picture 5.1.3. TSP File Loaded.

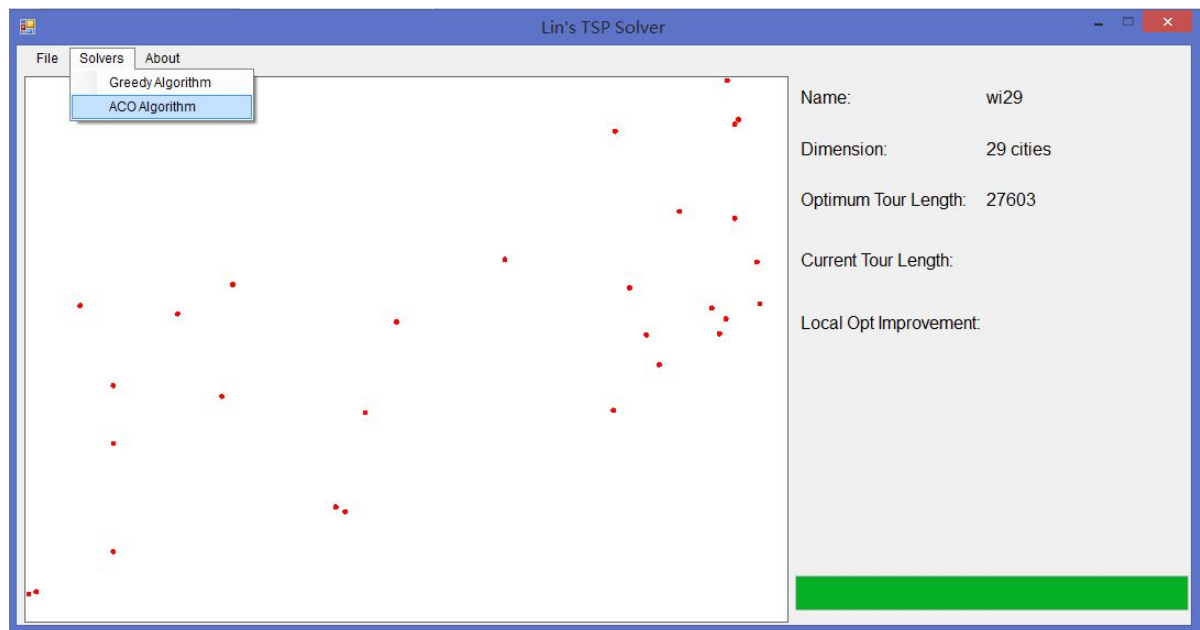
Button 'About' on the menu bar can operate at any time to show the information of the program by popping a message box in the screen as shown in Picture 5.1.4.



Picture 5.1.4. 'About' Button Pressed.

5.2 Execute Algorithm

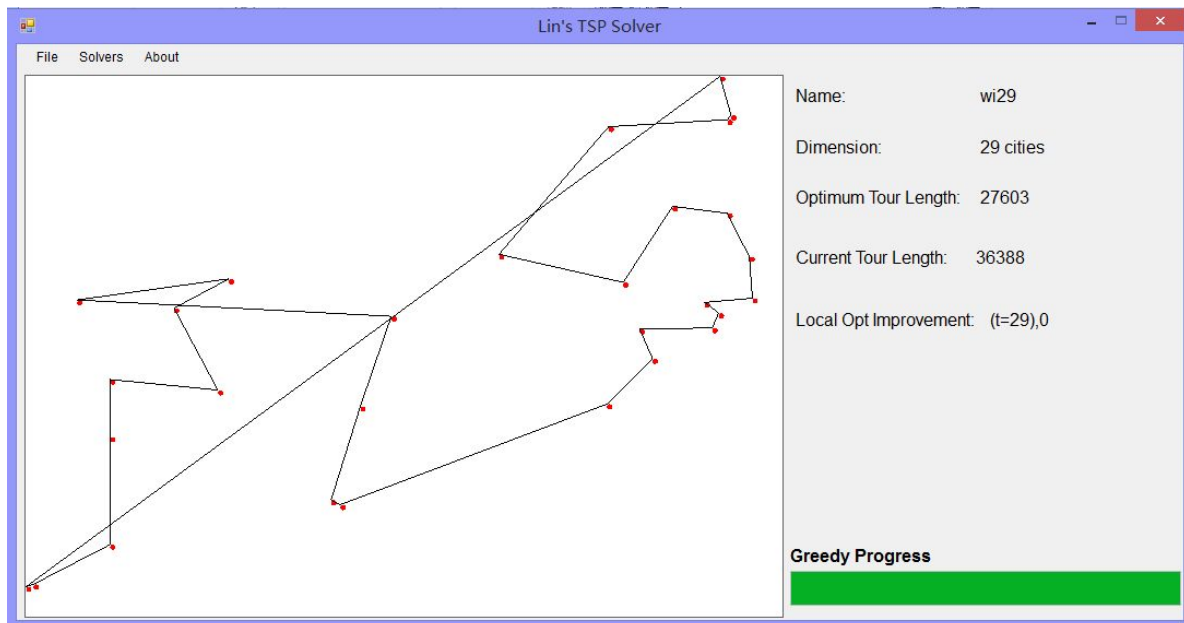
The program provides two algorithms which allow the user to compare the different paths and length between algorithms. User can choose whether Greedy algorithm or ACO algorithm by clicking the 'Solvers' in the menu bar. As shown in Picture 5.2.



Picture 5.2. 'Solvers' Button Pressed.

5.2.1 Greedy Algorithm

Greedy algorithm begin to execute as the user click the 'Greedy Algorithm' in the 'Solvers' in menu. The optimal paths display in the map area, as shown below.



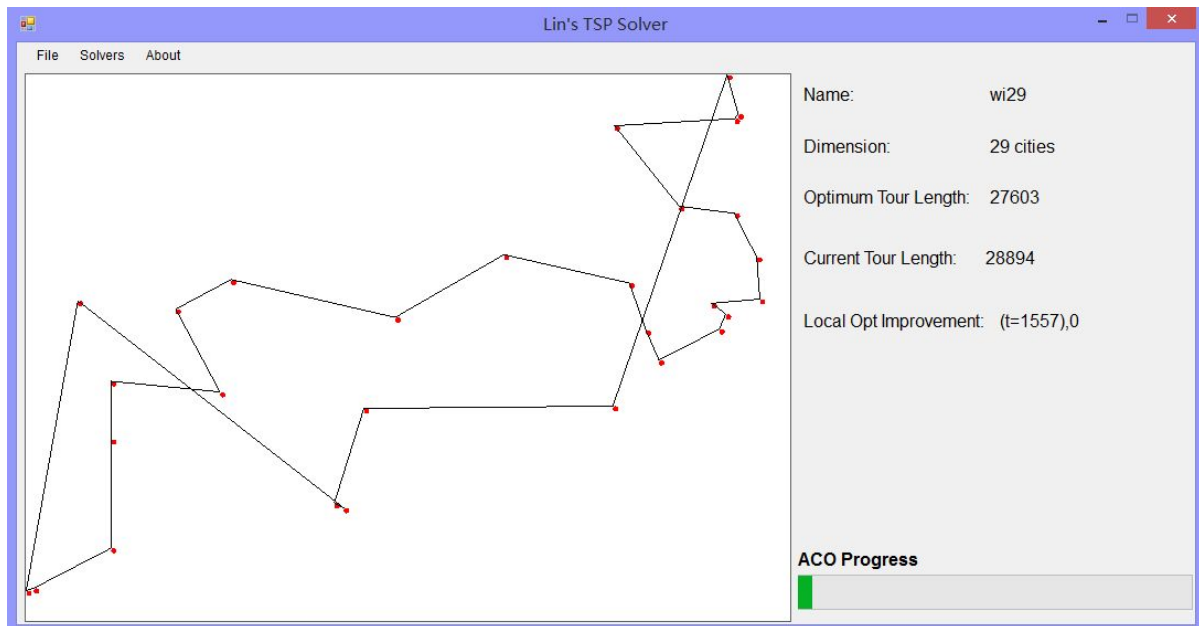
Picture 5.2.1. Greedy Algorithm.

According to the figure and operation, it is not hard to discover that using Greedy algorithm allow user to get the result in a very short time, however, the paths are much farther than the optimum tour length. The one of the main reason of the time-saving is that such algorithm just needs to compare the distances between cities and always pick up the shortest one, and the distances between cities are loaded and calculated when the TSP file is read.

5.2.2 ACO Algorithm

According to the report introduction, author get the basic parameters which are ' $t_{\max}=3000$ ', ' $\beta \approx 7$ ' and ' $p \approx 0.5$ '. In order to find the shortest path and ensure the map would not flash, author use ' $t_{\max}=3000$ ', ' $\beta=6.5$ ' and ' $p \approx 0.45$ ' which may lead to better results.

Ant-based optimization (ACO) algorithm begin to execute when user click the 'ACO Algorithm'. The optimal paths display in the map area, as shown in Picture 5.2.2.



Picture 5.2.2. ACO Algorithm.

From figures and operation, we can find that using ACO algorithm allow user to get a better result than Greedy algorithm. The optimum length calculated in this part is not so far from the optimum length which is given by the TSP file. However, one of the most obvious disadvantage is ACO algorithm takes much time compared with the Greedy algorithm. Also, ACO cannot guarantee the same result at any time and paths may change though long time past.

5.3 Summary

Comparing with the two algorithms it is easy to discover that Greedy algorithm can provide a steady result though is not the best choice. ACO algorithm has better accuracy than Greedy but takes much more time especially when the number of cities is large. In summary, ACO algorithm is more reliable and might be the better choice for the users or salesmen.

Above pictures are the screen shots by testing through 'wi29.tsp', other TSP file testing screen shots are shown in appendix. And code can be found in program files which are attached with this report.

6. Conclusions

The first part of the report outlines the idea and analyses requirements. This report uses UML to design the program for meeting demands. Such part designed in detail and all steps complete fully.

The second part of report uses the C# to realize the design. In this part, GUI form and Greedy algorithm have finished completely, but ACO algorithm still needs more promotion. One or more versions of ACO algorithm with different parameters are expected.

The design teach us how to analysis user's requirements and consider all of them in detail. By designing the system, not only can we learn how to use design tools (UML & C#), but also known how to design in a more professional (as an engineer) approach. However, the name used in UML design are not corresponding with the name used in C# completely. This mainly due to lack of experience and may lead to some problems which should be avoided. Also, some details, opening another TSP file (or using another algorithm) when an algorithm is processing for example, program may get stuck which leads to inconvenience for the users and should be avoided.

In conclusion, the design meets the requirements mainly, the usability of the GUI is absolutely fine. Greedy algorithm also can works perfectly. However, the efficiency of ACO algorithm needs improve. So, the program is able to achieve the specification and has full functionality though some details should fixed and improved.

References

Deitel, H. M. and Deitel, P.J. (2009) **Visual C# 2008. How to Program.** 3rded. Upper Saddle River, N.J. : Pearson/Prentice Hall

Song Z. (2009) **TSP Research based on Ant Colony Algorithm.** Master dissertation, JiangXi University of Science and Technology.

Spann, M. (2014) **Object Oriented Programming Using C# Assignment 2014-15.** University of Birmingham.

Spann, M. (2014) **Object Oriented Programming Using Course Slides 2014-15.** University of Birmingham.

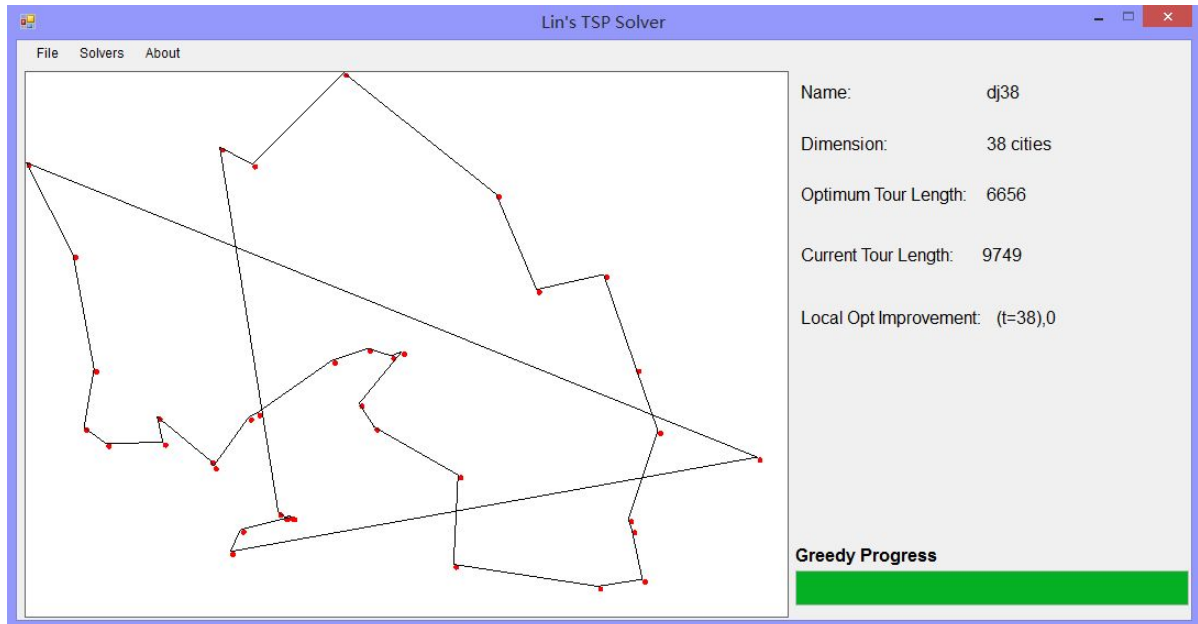
Pycock, D. (2014) **Lift Control System.** University of Birmingham.

Pycock, D. (2014) **Object-Oriented Software Design UML: Unified Modelling Language.** University of Birmingham.

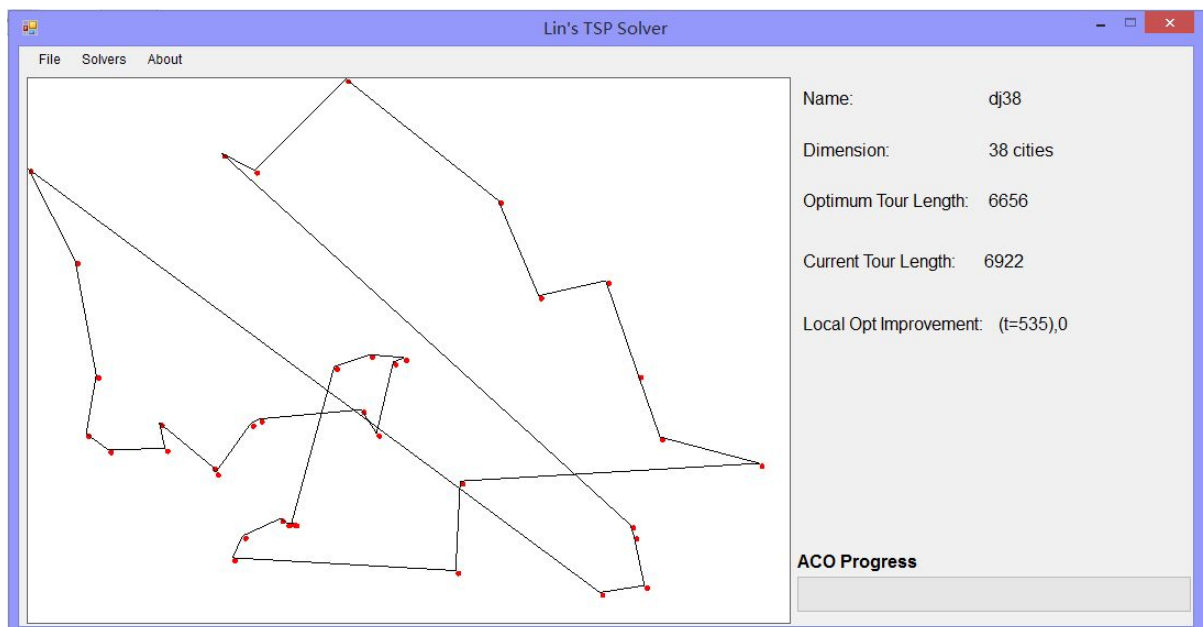
Appendix

Testing Pictures

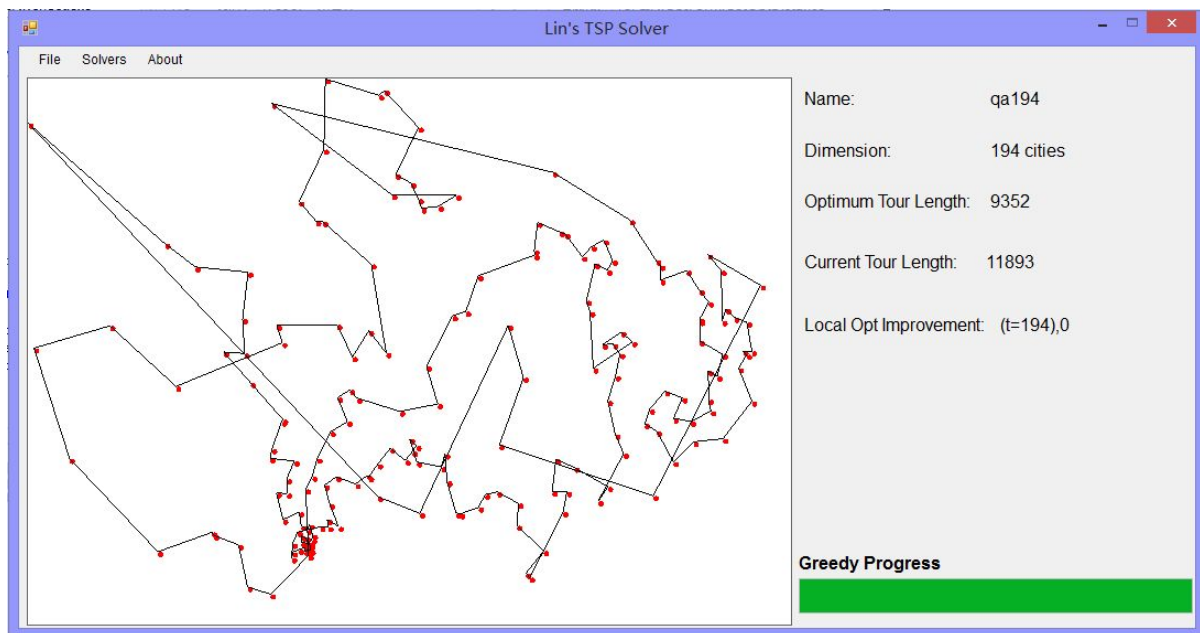
Testing File: dj38; Testing Algorithm: Greedy



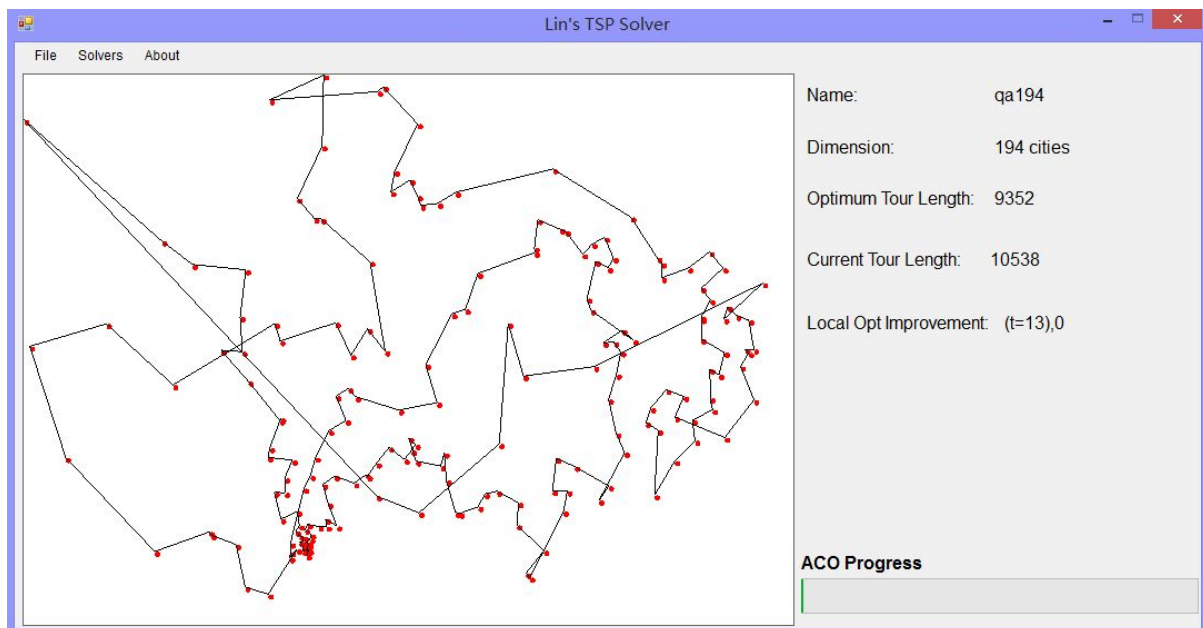
Testing File: dj38; Testing Algorithm: ACO Algorithm



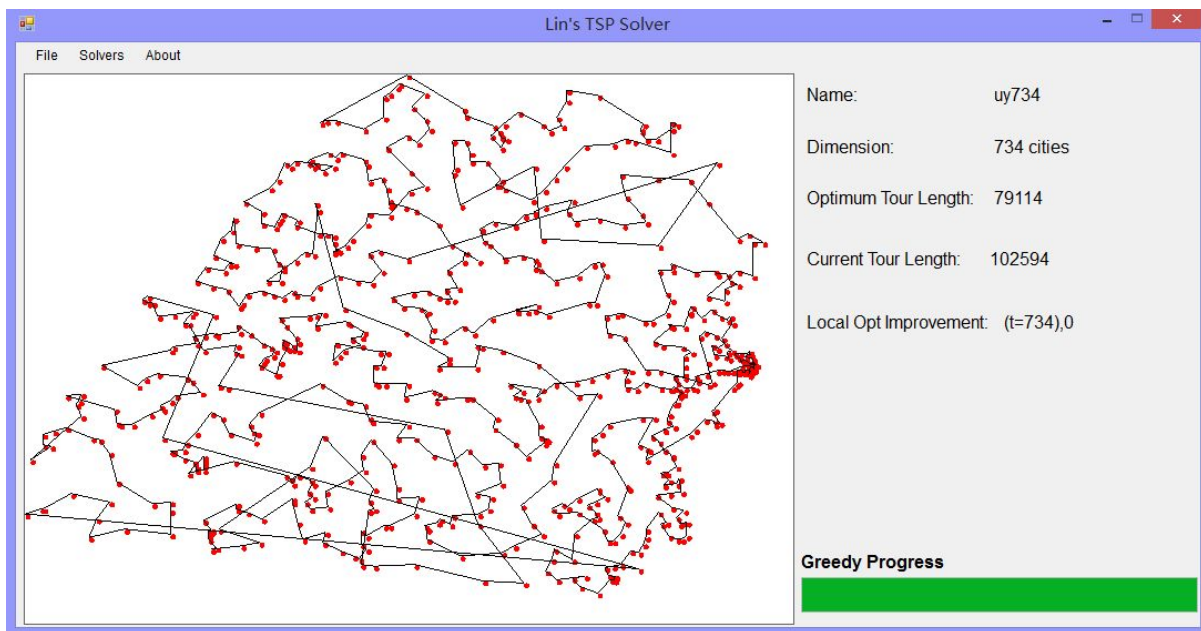
Testing File: qa194; Testing Algorithm: Greedy Algorithm



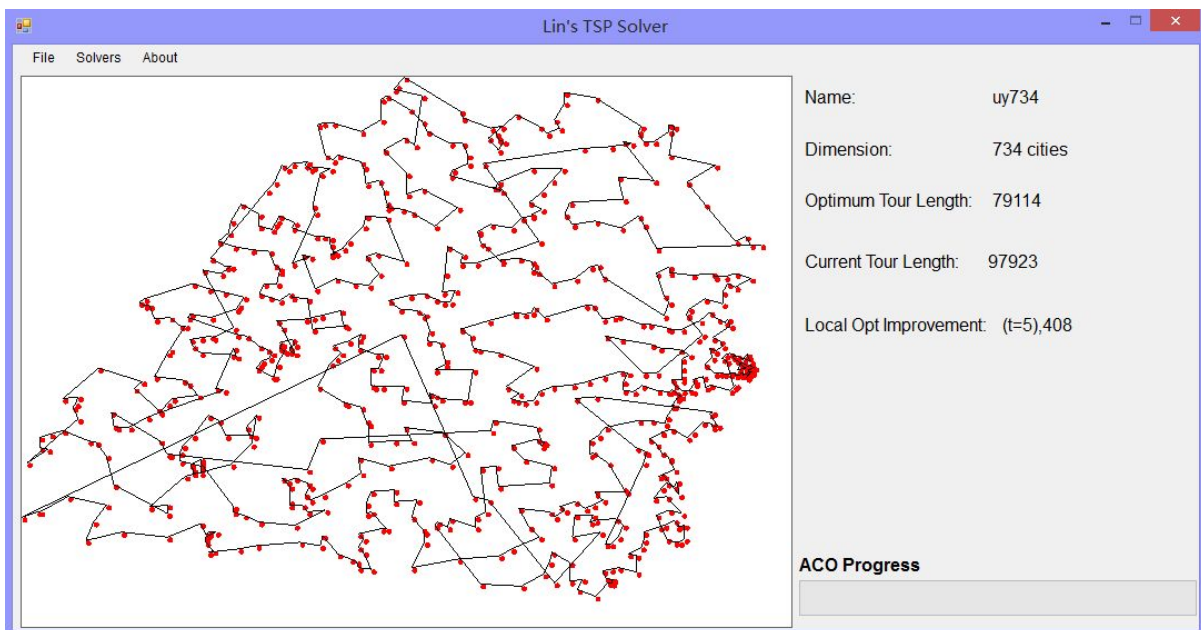
Testing File: qa194; Testing Algorithm: ACO Algorithm



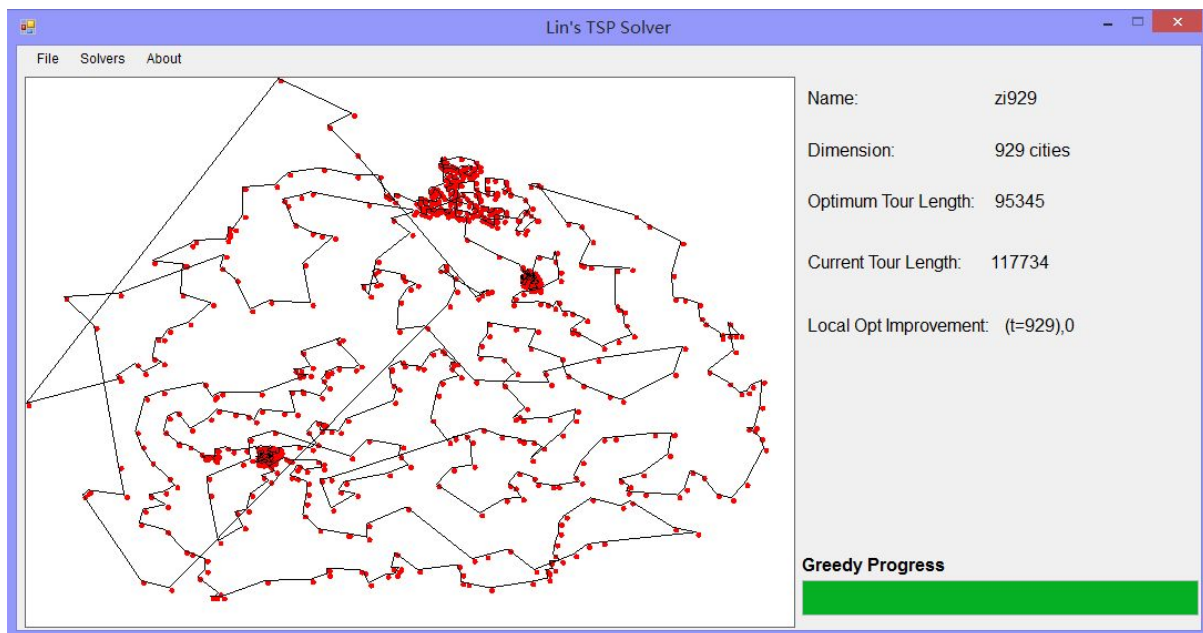
Testing File: uy734; Testing Algorithm: Greedy Algorithm



Testing File: uy734; Testing Algorithm: ACO Algorithm



Testing File: zi929; Testing Algorithm: Greedy Algorithm



Testing File: zi929; Testing Algorithm: ACO Algorithm

