[2014/2015]

[Object Oriented Programming Using C#]

Jolyon Joe Yen Ng (1505175)

Table of Contents

[Introduction 2](#_Toc415214154)

[Use Case Model 3](#_Toc415214155)

[*Brainstorm* 3](#_Toc415214156)

[*Use Case Diagram* 4](#_Toc415214157)

[*Scenario Descriptions* 5](#_Toc415214158)

[*Class Identification* 6](#_Toc415214159)

[*CRC Cards* 7](#_Toc415214160)

[*Initial Statechart* 8](#_Toc415214161)

[*Simple Class/ Sequence Diagram* 9](#_Toc415214162)

[Analysis Model 10](#_Toc415214163)

[*Attributes (All Private)* 10](#_Toc415214164)

[*Methods* 11](#_Toc415214165)

[*Analysis Model Sequence Diagram* 12](#_Toc415214166)

[*Analysis Model Class Diagram* 14](#_Toc415214167)

[*Analysis Model Statechart* 15](#_Toc415214168)

[Design Modelling 16](#_Toc415214169)

[*Revisit use-case model* 16](#_Toc415214170)

[*Sequence diagram* 16](#_Toc415214171)

[*Textual description of object to object interaction* 16](#_Toc415214172)

[*Subsystems* 16](#_Toc415214173)

[*Implementation of non-functional requirements* 16](#_Toc415214174)

[*Deployment model* 16](#_Toc415214175)

[*Legacy issues* 16](#_Toc415214176)

[*Reconsider the attributes* 16](#_Toc415214177)

[*Reconsider the associations* 16](#_Toc415214178)

[Summary of how the program will work (Pseudo Algorithm) 18](#_Toc415214179)

[Class Diagram Generated from Code 19](#_Toc415214180)

[Functionality Testing 19](#_Toc415214181)

[Conclusion 22](#_Toc415214182)

[References 23](#_Toc415214183)

# Introduction

This assignment is an object oriented programming task using the C# language. This assignment deals with the well-known ‘travelling salesman problem’ also known as TSP.

The travelling salesman problem is a problem in combinational optimization studies which is important within operations research and theoretical computer science. The concept behind this problem is that given a list of cities and the distances between each cities, find the shortest possible tour around the graph without visiting an individual city more than once. To deal with such a problem certain algorithms will need to be implemented or applied. The two algorithms that will be implemented for this assignment are the greedy algorithm and the ant colony optimization algorithm.

The general theory behind the Greedy algorithm is that given a starting city, the algorithm will determine the distance from the current city to all the different cities. The determined process will illustrate which city is the shortest distance from the current city and move to the next city, this process will loop until the shortest route have been computed, this is when it will move back to the starting point ready for the algorithm to be launched again.

The ant colony optimization algorithm is a lot more complicated because ants make probabilistic decisions based on the amount of locally deposited pheromone they leave behind to move to a different location. Say that an ant is randomly or sequentially allocated on a city, in-order for the ant to move to the next city, a probability of which city to visit will have to be calculated through the formula



Through the previously accumulated pheromone deposits, the next city will be marked as the city with the highest probability. The ant will only move until all the cities have been visited, and the city with the highest pheromone value is marked as the next city.

The ant will move to the next city with the highest amount of pheromone until all the cities have been visited. The pheromone value will then be updates based on the formulas below and the aforementioned steps will then be iterated by the next ant until a fully optimised solution has been generated.

 (2)

 (3)

# Use Case Model

## Brainstorm

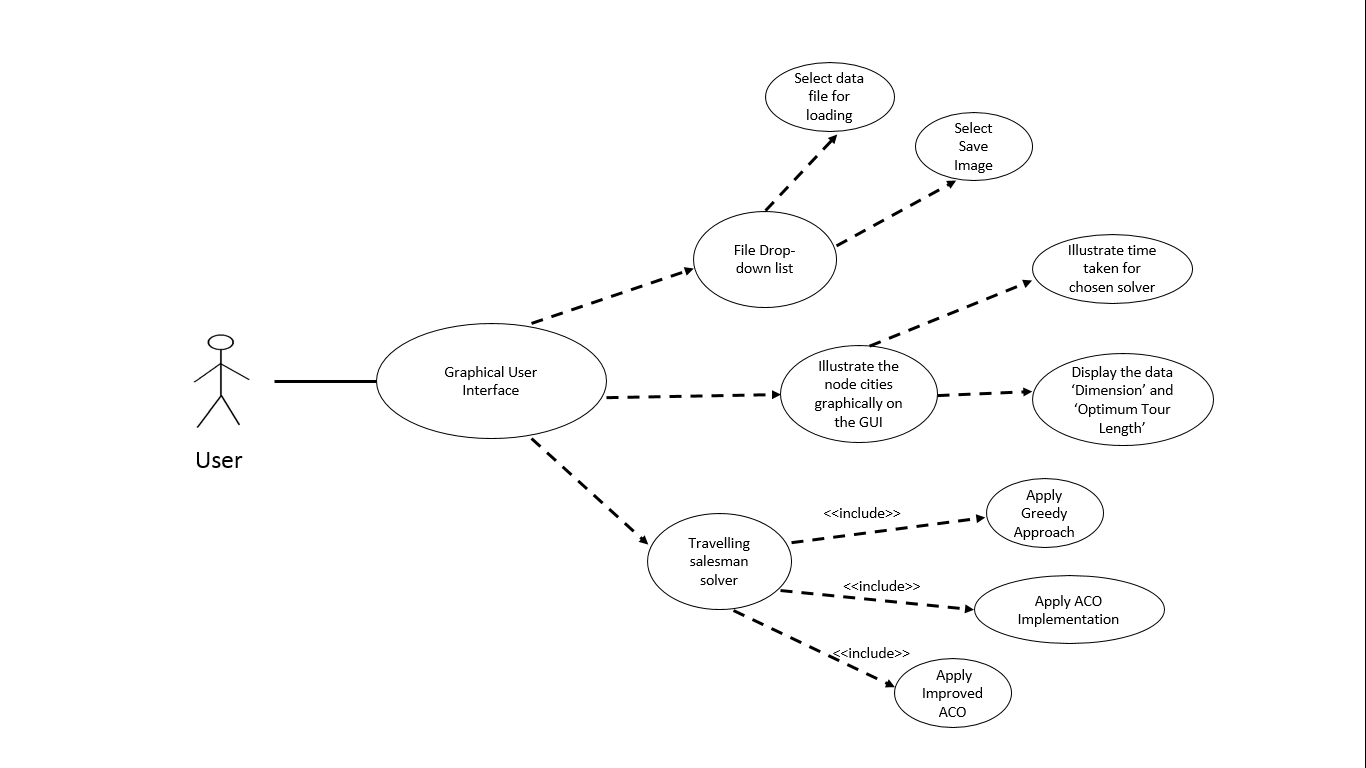
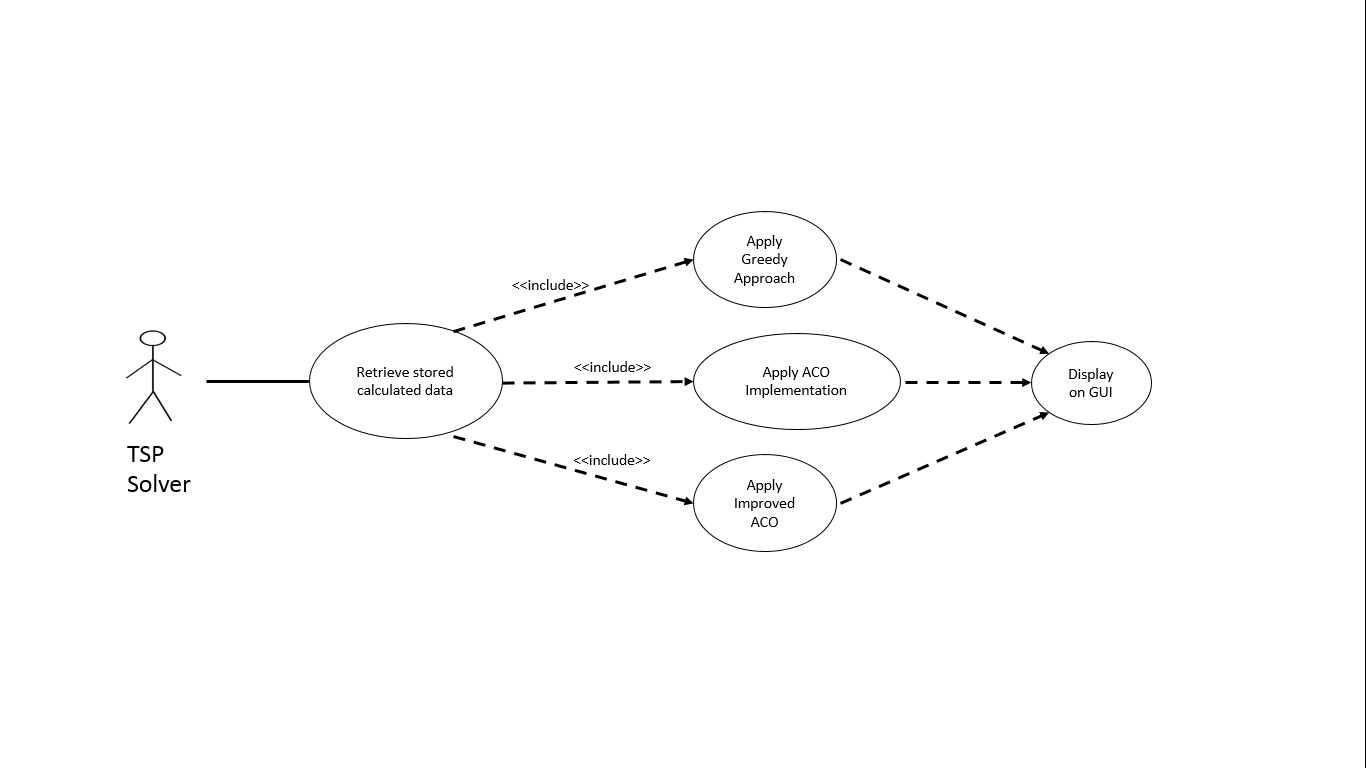
Potential Requirements are illustrated in the table below outlining the general ‘tasks’ needed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Concept** | **Necessity** | **Risk** | **Cost** | **Priority** |
| Develop C# classes to represent the travelling salesman problem involving distance between the cities, coordinates of the cities, a graph that represents the different nodes of the cities, class to generate city results | High | Medium | Medium | High |
| Create and loading data files | High | Low | Low | Medium |
| Select the X and Y coordinates from the data file | High | Low | Low | High |
| Calculate the distance from each city except for the already visited ones (Greedy) | High | Medium | Medium | High |
| Find the minimum distance of a city from the current city (Greedy) | High | Medium | Medium | High |
| Store the result of the data (Greedy) | High | Medium | Medium | High |
| Make a declaration of the number of iterations (t) and number of ants (m) – (ACO) | High | Medium | Low | Medium |
| Calculate the probability of next city to visit for the ant based on the formula from assignment sheet (ACO) | High | Medium | High | High |
| Set the length or arc and its parameters which controls the relative influence of the arc length over the previous accumulated pheromone deposits (ACO) | High | Medium | Medium | Medium |
| Store the calculated data of the next city as the highest possibility (ACO) | High | Medium | Medium | High |
| Move ant to next city and update pheromone for each ant in the city graph based on the formula provided in the assignment (ACO) | High | Medium | High | High |
| Move to next ant and iteration (ACO) | High | Medium | Medium | High |
| Store the data for each city with the maximum amount of pheromone to the results (ACO) | High | Medium | Medium | High |
| Implementation of Graphical User Interface with additional details such as Dimensions, Optimum tour length and time taken. | Medium | Medium | Low | Medium |
| Illustrate the cities as nodes graphically on the GUI | Medium | Medium | Medium | Medium |
| Launch the Greedy or ACO solvers from a ‘drop-down- option’ | Medium | Medium | Low | Medium |

Table 1: Prioritisation of Potential Requirements

Based on Table 1 above, there does not seem to be any great risk associated with the requirements, therefore the design will incorporate all the above requirements.

## Use Case Diagram



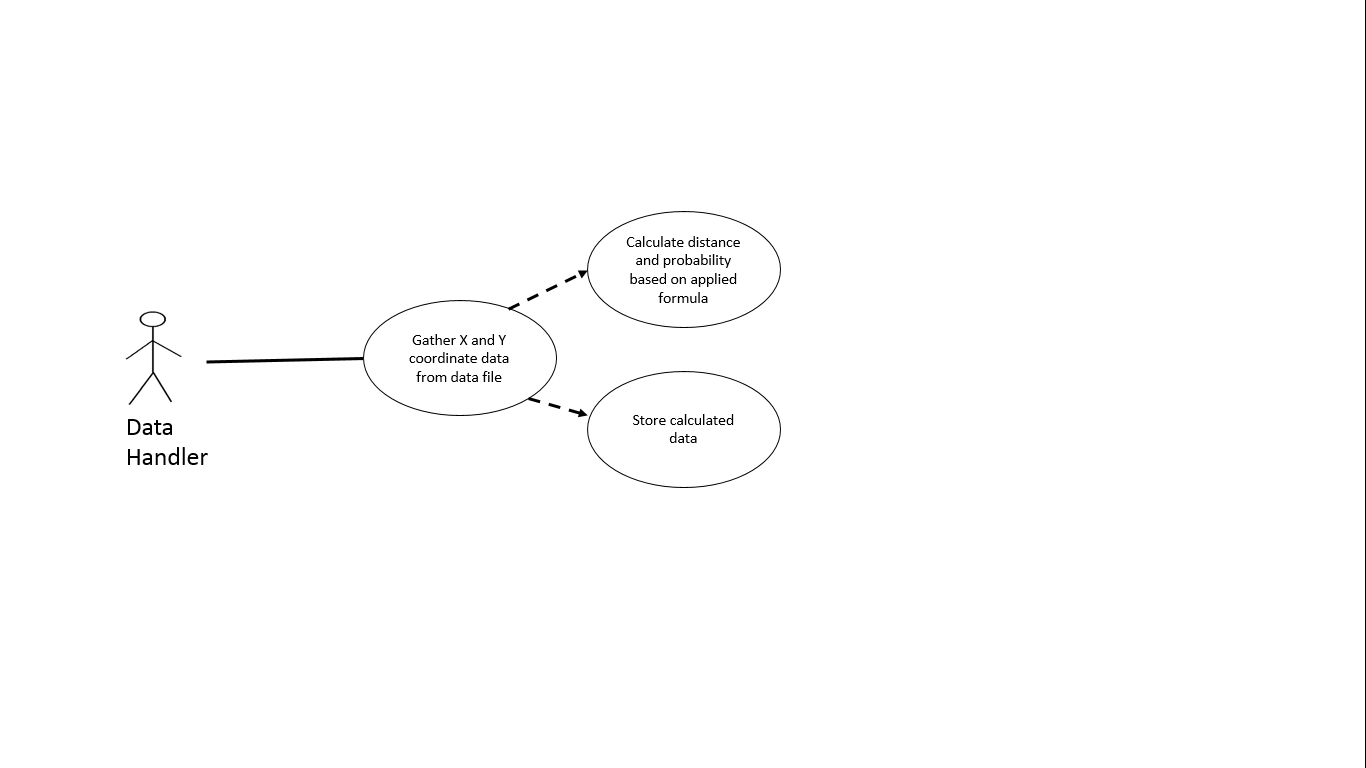
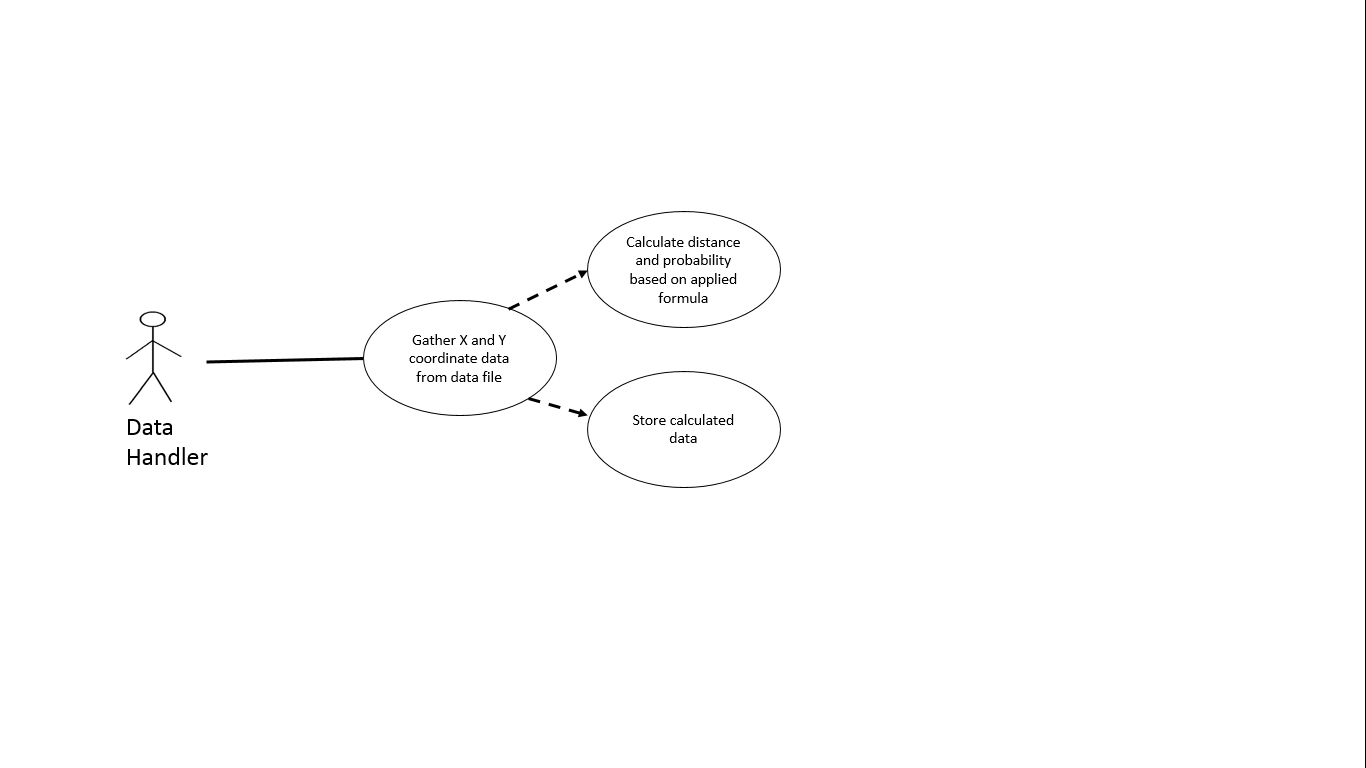


Figure 1 : Use-Case Diagram for User, TSP Solver and Data Handler



## Scenario Descriptions

User :

The user on the graphical user interface, selects a data file from the file drop-down list to upload the coordinates of the cities graphically. The city displayed as nodes will await the selection between the different TSP solvers; Greedy solver, ACO solver or Improved ACO solver. When the user selects the appropriate solution from the TSP solver drop-down list, the GUI produces an optimisation of the graphical representation of the individual city on the city graph and represent the relevant information based on the problem; Dimension, Optimum Tour Length and Time Taken.

When the optimisation have been completed, the user has an option to save the image or exit the graphical user interface through the file drop-down list.

Data Controller :

The data controller gathers the X and Y coordinates from the data file in-order to calculate the distance between each city for the greedy solver and to calculate the probability of the next city to visit for the ant based on the ACO solver. The calculated data will be stored as the city result and ready for displaying results to the GUI through the selection of the different algorithms from the TSP solver list within the GUI (Greedy or ACO).

TSP Solver :

The TSP Solver retrieves the calculated stored data from the data handler and applies the data to the relevant TSP implementation either through the use of Greedy or the ACO. This selection is solely up to the user of the graphical user interface where the result will be displayed on the city graph.

## Class Identification

Nouns

User Scenario :

Data file, drop-down list, upload, coordinates, city, selection, TSP solver, greedy solver, ACO solver, Improved ACO solver, solution, optimisation, problem, exit, dimension, Optimum Tour Length, Time Taken, GUI.

Data Handler Scenario :

Handler, coordinate, distance, city, city result, ant, ACO solver, greedy solver, graph, GUI.

TSP Solver :

TSP Solver, data, handler, greedy solver, ACO solver, user, result, city graph.

Pooling the initial class list:

* Controller
* GUI
* Coordinates
* City
* City result
* City graph
* TSP Solver
* Greedy
* ACO

Stereotypical Classes

Boundary : GUI, City graph

Control : Controller

Entity : Greedy, ACO, City result, coordinates, city, TSP Solver

GUI and City graph can be rolled into the Controller class whereas Greedy, ACO, City result and City will be submerged into the TSP Solver Class, doing so will help maintain the number of classes to a reasonable level and to avoid excessive detail at such an early stage in the design process. Therefore the initial class list will be:

* Controller
* TSP Solver
* Coordinates

## CRC Cards

|  |  |
| --- | --- |
| **Coordinates** | |
| Responsibilities | Collaborators |
| The coordinates class computes the X and Y coordinates of the cities for the TSP Solver to utilise the data during calculations and used by the controller to illustrate the coordinates of the cities to the city graph to be displayed on the GUI | Controller  TSP Solver |

|  |  |
| --- | --- |
| **TSP Solver** | |
| Responsibilities | Collaborators |
| The TSP Solver class contains the mathematically calculated city result data where the distance between the cities are computed and greedy plus ACO algorithm are utilised to solve the TSP problem through the use of the coordinates class by providing the X and Y coordinates for calculation. The TSP Solver will then pass the computed data onto the Controller for it to display the data result to the city graph within the GUI. | Controller  Coordinates |

|  |  |
| --- | --- |
| **Controller** | |
| Responsibilities | Collaborators |
| The responsibility of the Controller class is to display information to the user through a graph. In-order to display the different city onto a graph, the coordinate class is involved as it provides the controller with the relevant X and Y coordinates. This also relates to the TSP Solver as the TSP solver provides algorithmic solutions to the TSP problem which is then passed onto the controller to be displayed graphically and informatively. | Controller  Coordinates |

Table 2: CRC Cards for Coordinates, TSP Solver and Controller Classes

## Initial Statechart

Figure 2 : Initial Statechart:

## Simple Class/ Sequence Diagram

Figure 3 : Class/ Sequence Diagram for Load Data File

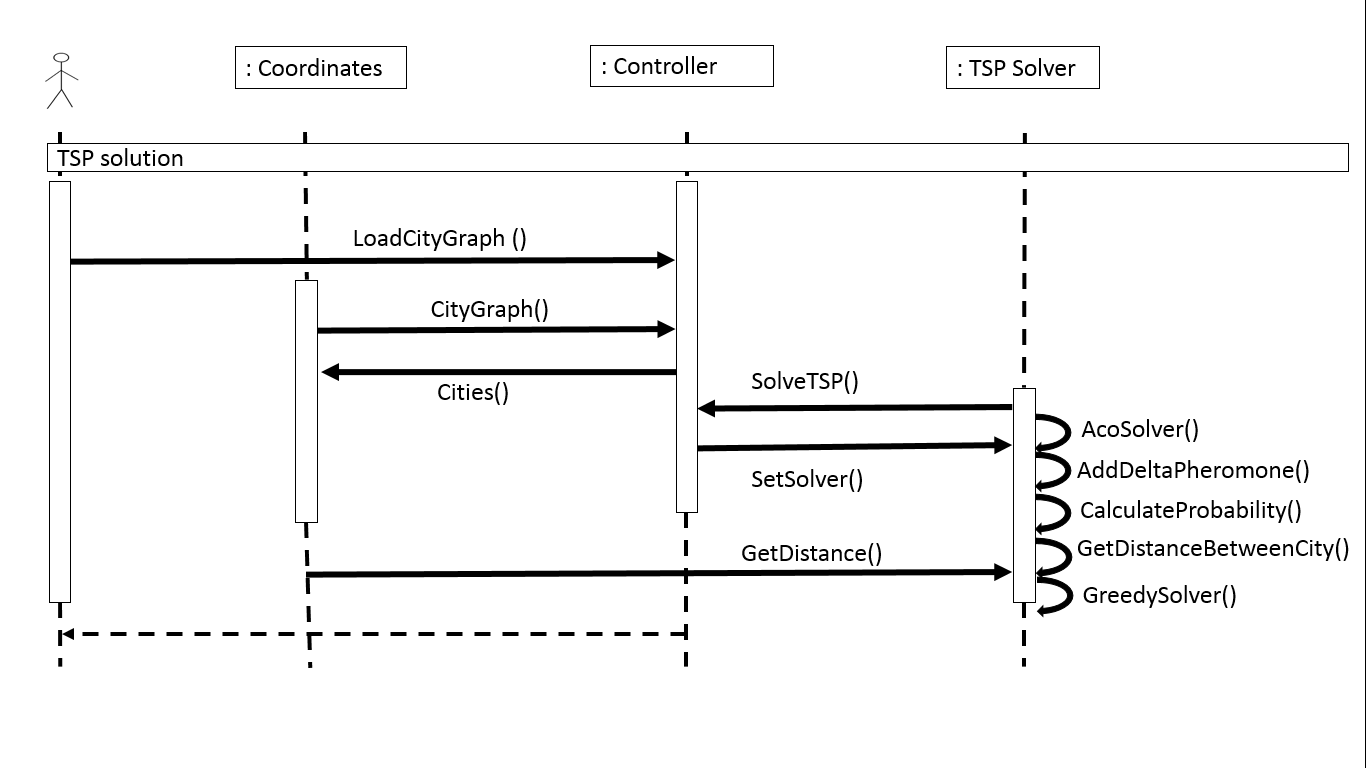


Figure 4: Simple Class/ Sequence Diagram for TSP Solution

# Analysis Model

## Attributes (All Private)

## Methods

|  |  |  |
| --- | --- | --- |
| **Class** | **Method** | **Comment** |
| TSP Solver | GetDistanceBetweenCity() | Used for getting the distance between two cities. |
| SolveTSP() | Method to solve the travelling salesman problem. |
| GreedySolver() | Used to carry out the greedy algorithm by selecting the closest city based on the distance. |
| AcoSolver() | Performs the ACO algorithm accounting for other ‘methods’ within the code CalculateProbability(), AddDeltaPheromone(), GetPheromoneValue(). |
| AddDeltaPheromone() | Used to add a delta value of pheromone on all of the arcs. |
| CalculateProbability() | It calculates the probability of an ant to move from one city to another. |
|  | GetAcoRunResult() | To get the ACO generated results after all iterations of all the ants have been completed |
| Coordinates | Coordintes() | Identifies the X and Y coordinates and saves the coordinates. |
| Controller | CityGraph() | This is used for creating the graph to illustrate the cities. |
| Cities() | Used to list all the cities. |
| GetDistance() | To gather distance between cities. |
|  | LoadCityGraph() | Loads the cities from the file to the GUI. |
|  | SetSolver() | Setting the selected TSP solver (Greedy or ACO) |

Table 3 : Analysis Model Methods

## Analysis Model Sequence Diagram

Figure 5 : Analysis Model Sequence Diagram for Load Data File

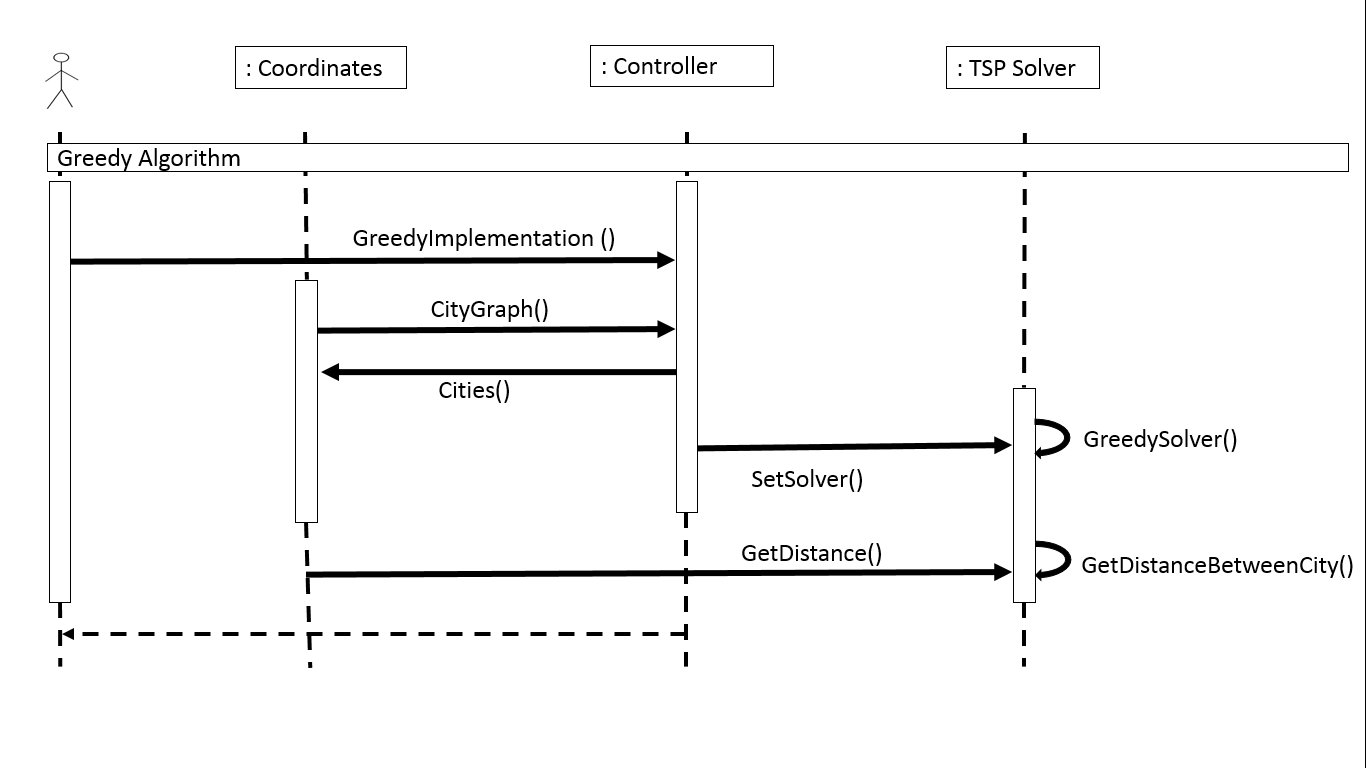


Figure 6 : Analysis Model Sequence Diagram for Greedy Algorithm

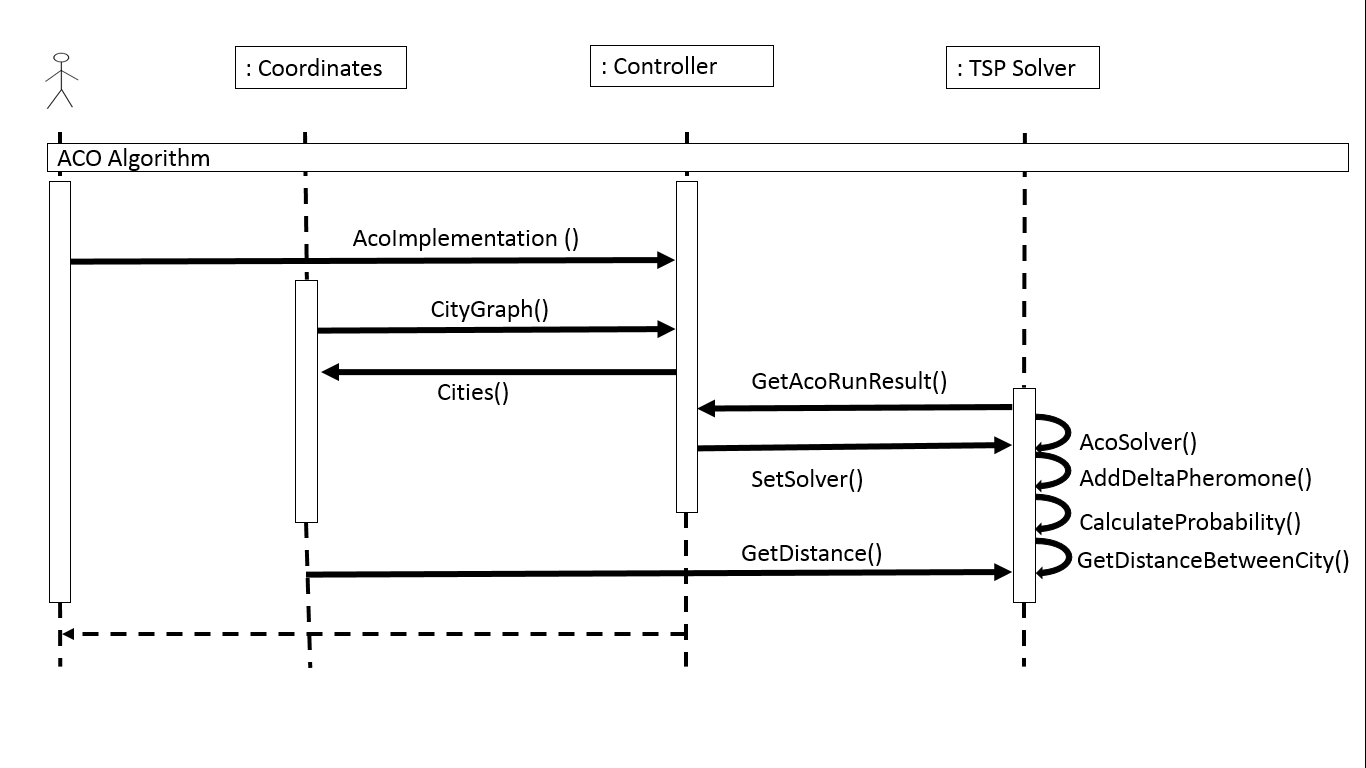


Figure 7 : Analysis Model Sequence Diagram for ACO Algorithm

## Analysis Model Class Diagram

: Controller

Cities()  
GetDistance()  
LoadCityGraph()  
SetSolver()

: TSP Solver

GetDistanceBetweenCity()  
SolveTSP()  
GreedySolver()  
AcoSolver()  
AddDeltaPheromone()  
CalculateProbability()  
GetAcoRunResult()

: Coordinates

Coordintes()

Figure 8: Analysis Model Class Diagram

## Analysis Model Statechart

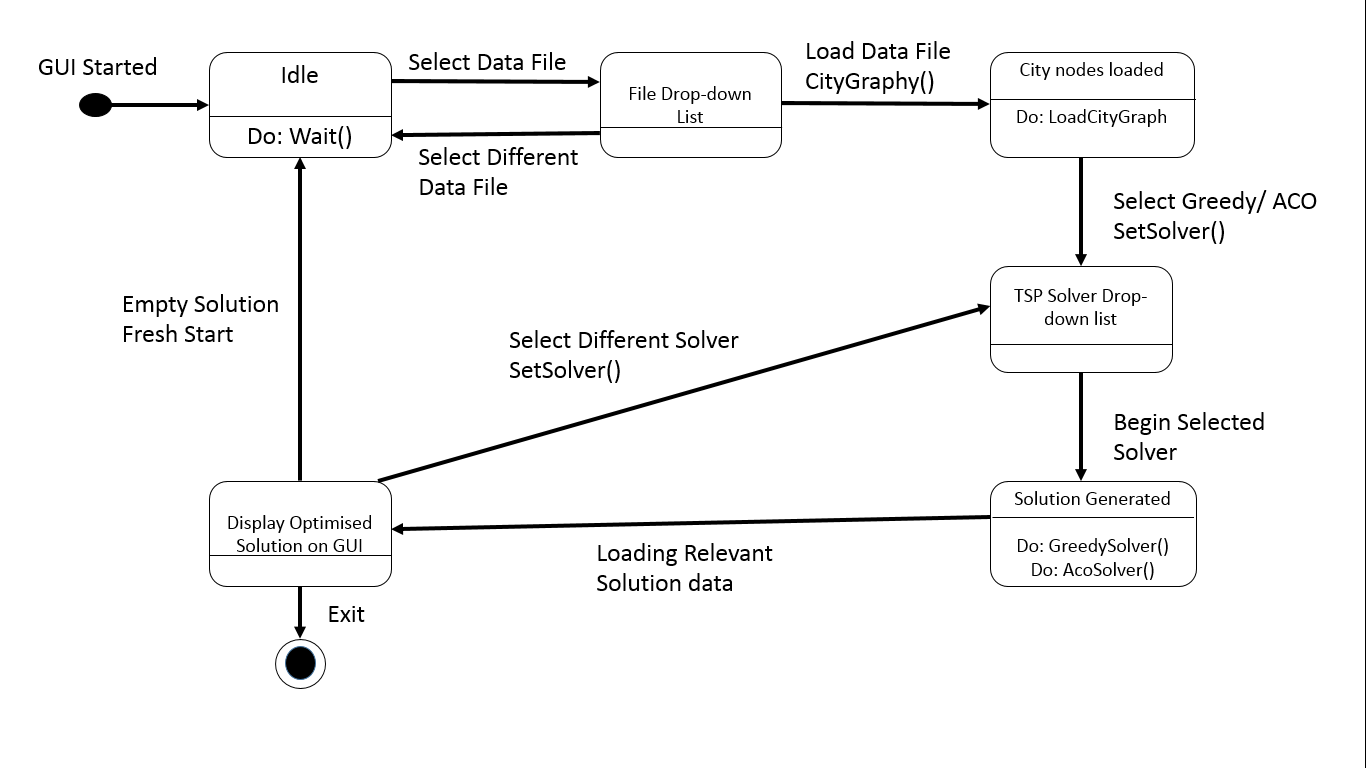


Figure 9 : Analysis Model Statechart

# Design Modelling

## Revisit use-case model

The design meets the minimum requirements necessary for implementation.

## Sequence diagram

Qualifiers have already been applied to the sequence diagram.

## Textual description of object to object interaction

The Controller class handles the display of information to the user through a graph. In-order to display the different city onto a graph, the coordinate class is involved as it provides the controller with the relevant X and Y coordinates. This also relates to the TSP Solver as the TSP solver provides algorithmic solutions to the TSP problem which is then passed onto the controller to be displayed graphically and informatively.

The coordinates class computes the X and Y coordinates of the cities for the TSP Solver to utilise the data during calculations and used by the controller to illustrate the coordinates of the cities to the city graph to be displayed on the GUI.

The TSP Solver class contains the mathematically calculated city result data where the distance between the cities are computed and greedy plus ACO algorithm are utilised to solve the TSP problem through the use of the coordinates class by providing the X and Y coordinates for calculation. The TSP Solver will then pass the computed data onto the Controller for it to display the data result to the city graph within the GUI.

## Subsystems

No subsystems are needed for this design.

## Implementation of non-functional requirements

The improved ACO versions will be implemented once the Greedy and normal ACO algorithms are working accordingly, this will be done by altering the normal ACO algorithm.

## Deployment model

The Controller class will be implemented on a single processor as data will be altered after every iteration especially within the ACO algorithm.

## Legacy issues

There will be no legacy issues as the program will be developed on Visual Studio 2013 considering the development software is constantly updating and the coding will be writing by myself.

## Reconsider the attributes

Incorporated within the Class Diagram.

## Reconsider the associations

Relationship between the classes remain the same for now, however as a thought for consideration, a few classes may well need to be added to incorporate a better functionality of the code as the design so far may well consist of too little classes, this will therefore cause future associations between classes to be different.

Statechart

No further revisions required.

# Summary of how the program will work (Pseudo Algorithm)

Pseudo Algorithm-

Greedy Algorithm:

1. Iterate over each city
   1. Find distance from each city except already visited one
   2. Find minimum distance from a city to the current city
   3. Add the distance to the result
   4. Mark this city as visited.

ACO Algorithm:

1. Declare the number of iteration (t) and the number of ants (m)
2. For each iteration
   1. For each ant
      1. Allocate ant on any city randomly or sequentially
      2. Calculate the probability of next city to visit for this ant according to following formula



Where the length of the arc and its parameter which control the relative influence of the arc length over the previously accumulated pheromone deposits

* + 1. save next city as the highest probability city
    2. move ant to the next city until all the cities have been visited
  1. update pheromone value (global variable) for each arc in the city graph as per formula

 (2)

 (3)

* 1. move to the next ant

1. move to the next iteration
2. for each city
   1. find arc with maximum pheromone from global variable
   2. add this arc to result.

# Class Diagram Generated from Code

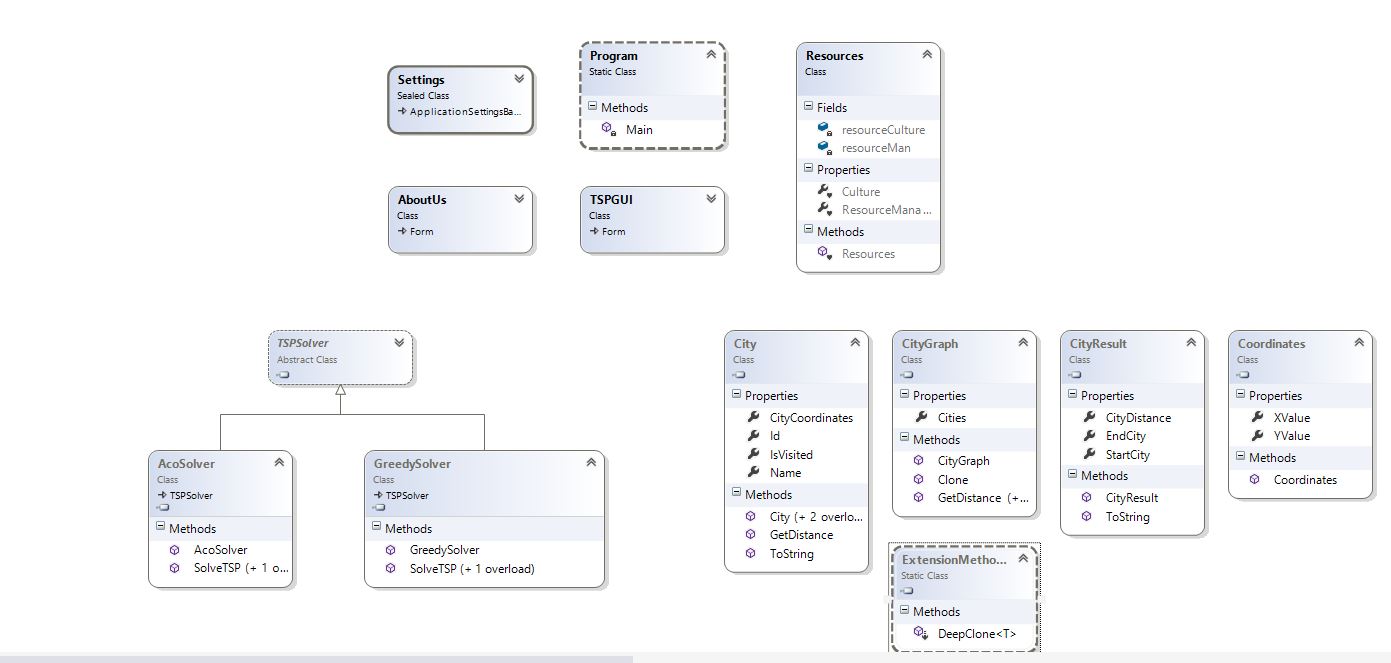


Figure 10 : Class Diagram Generated from Code

# Functionality Testing

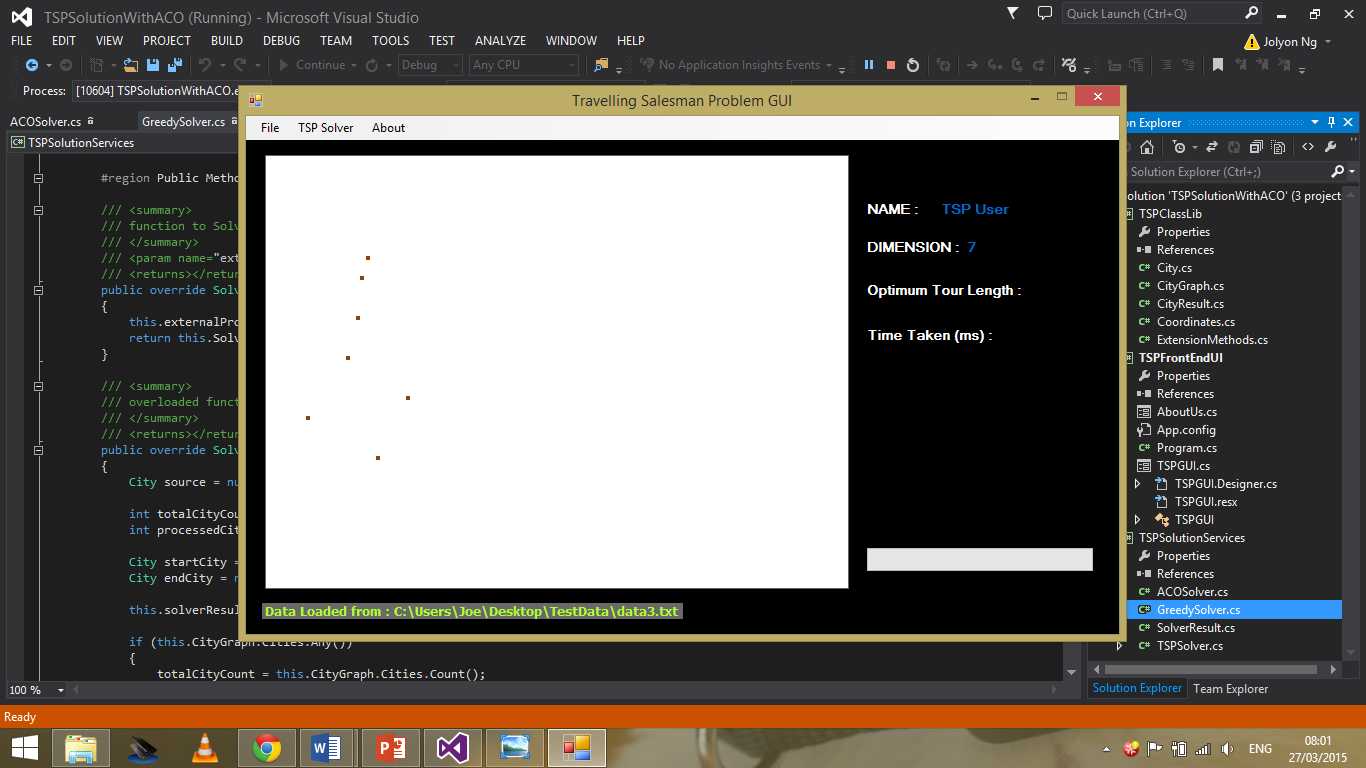


Figure 11 : Testing loading of data with minimal cities

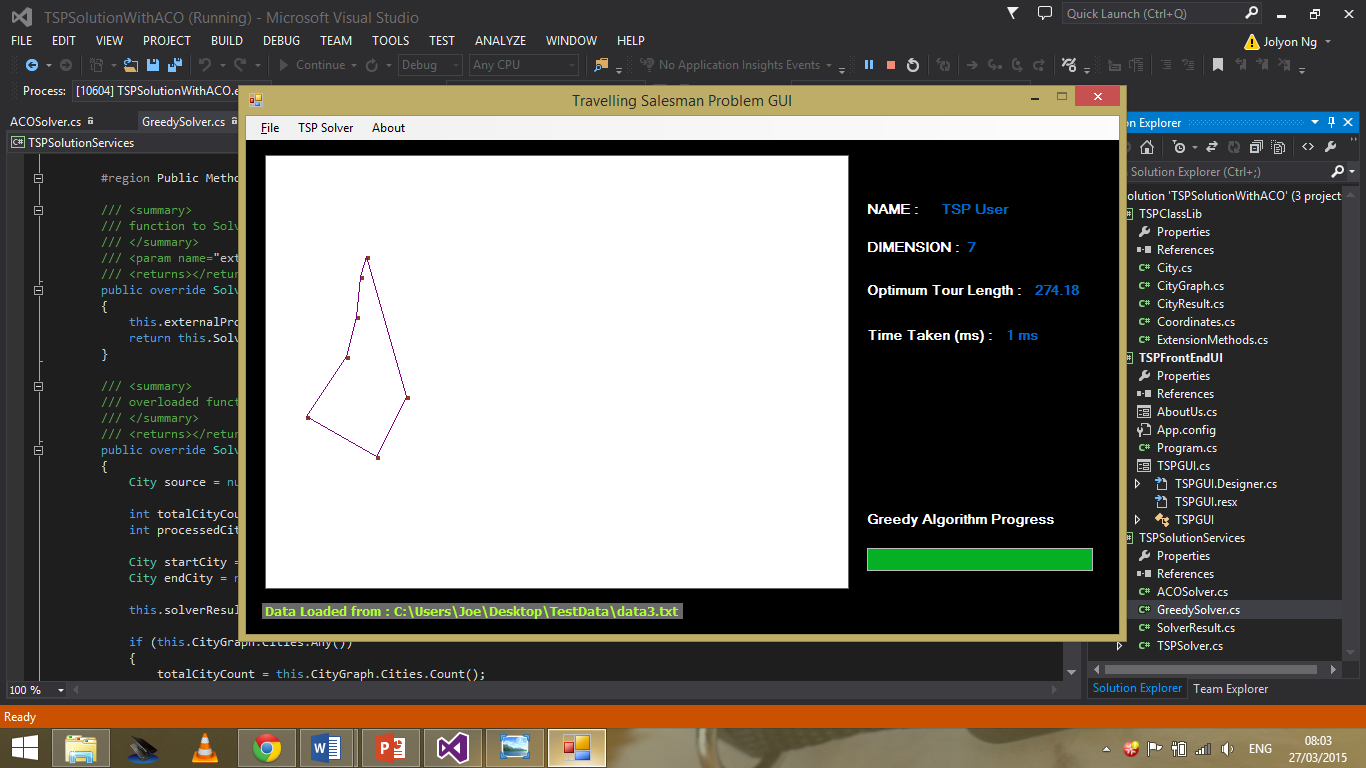
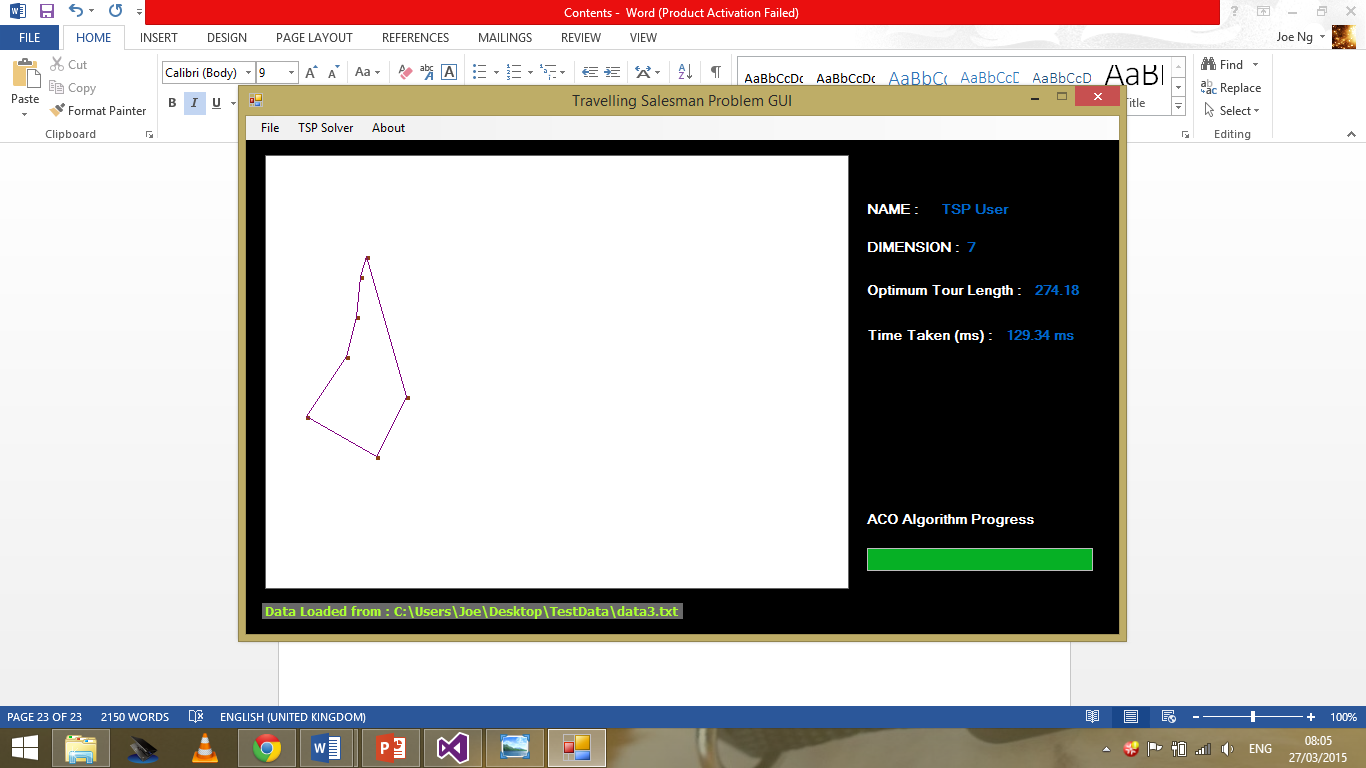


Figure 12 : Testing data with minial cities with greedy approach

**

*Figure 13: Testing data with minial cities with ACO Implementation*

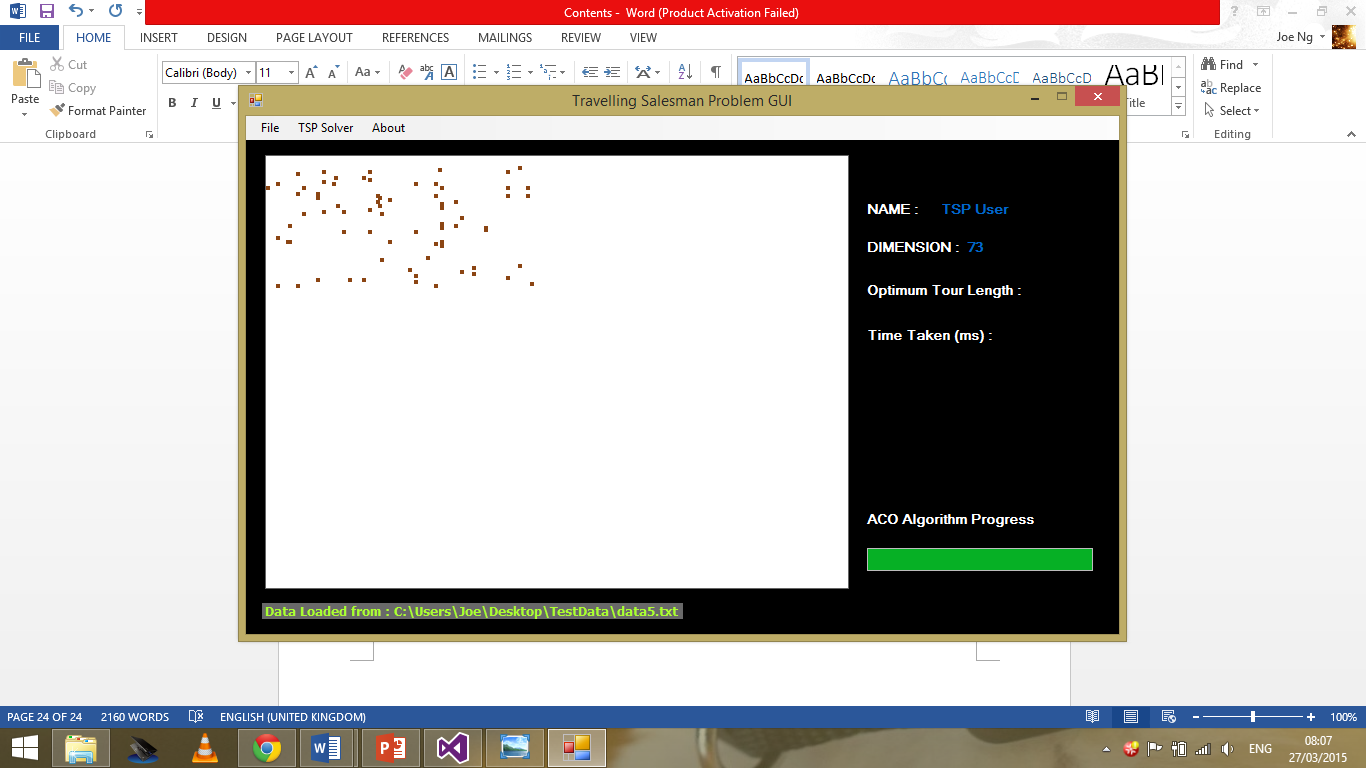


Figure 14 : Testing data with more cities

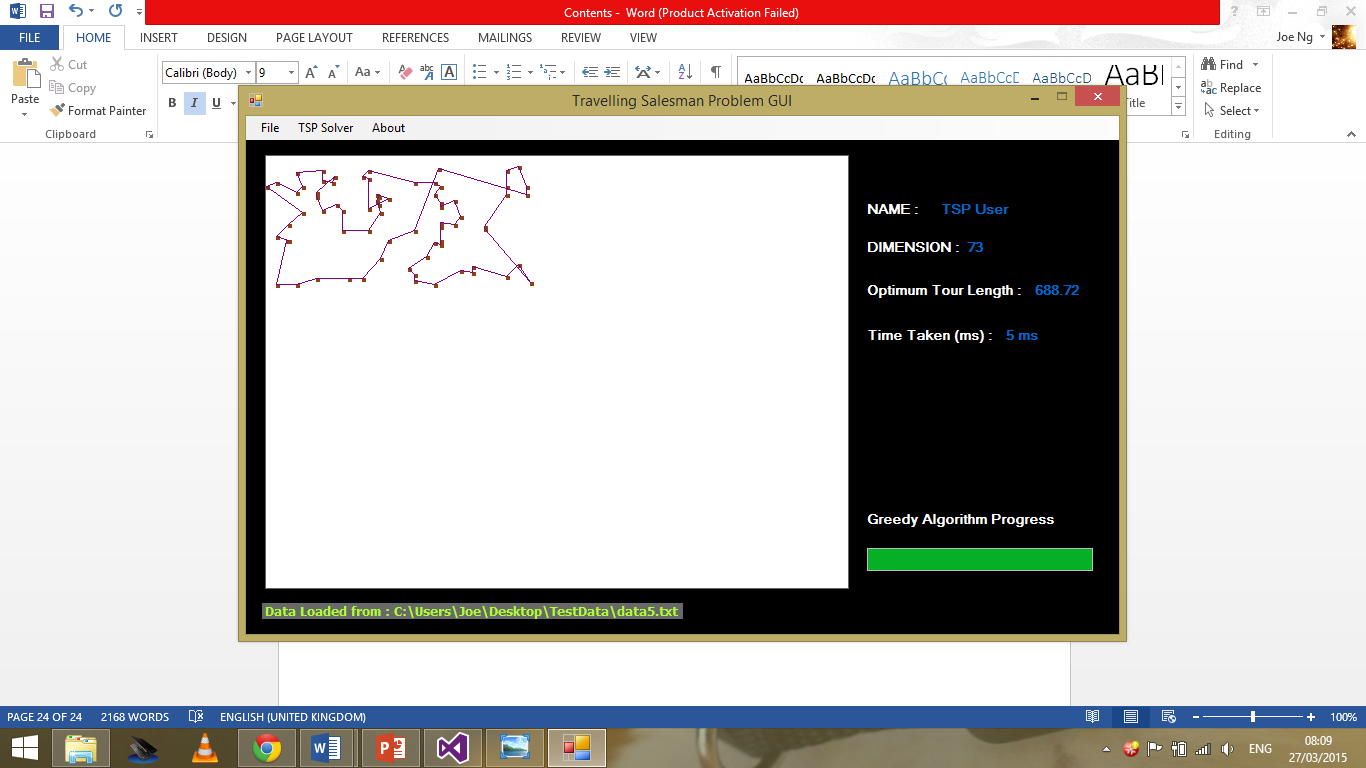


Figure 15 : Testing data with more cities with Greedy approach

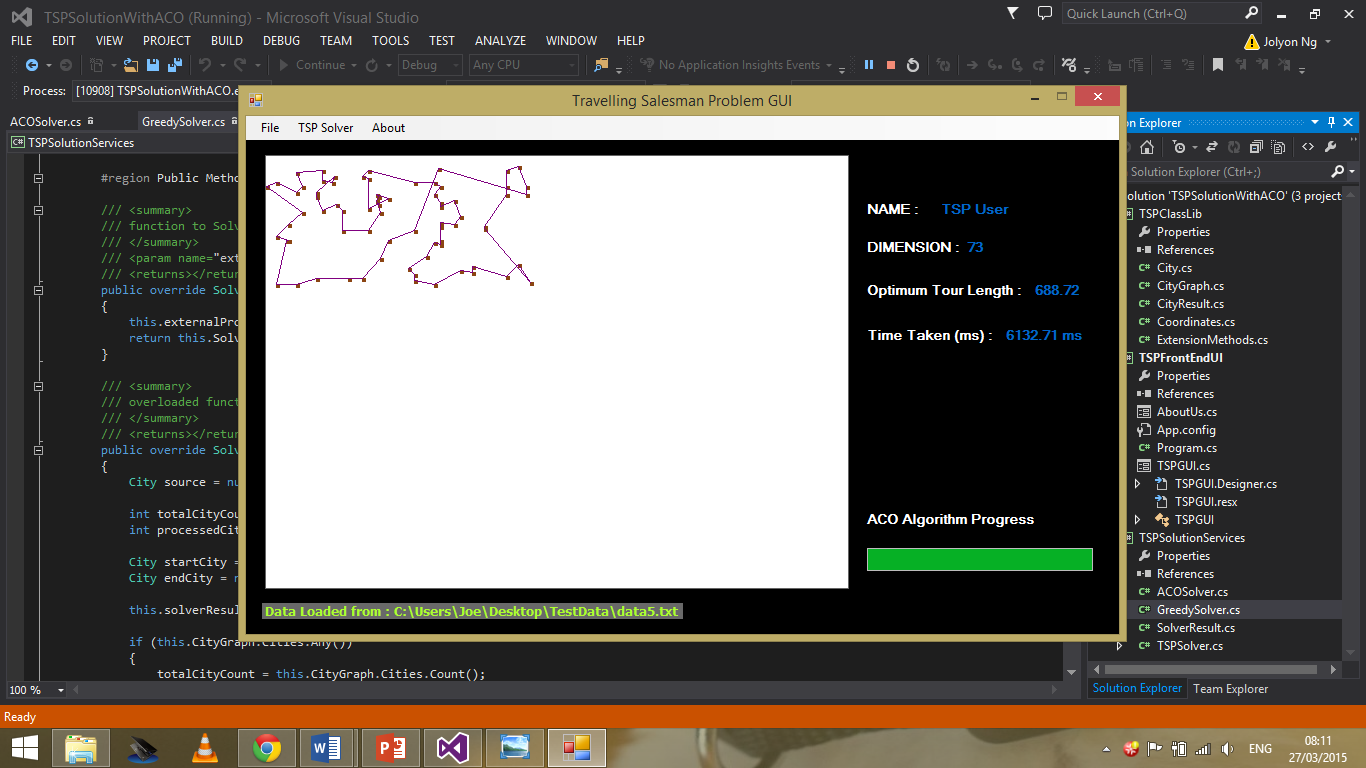


Figure 16: Testing data with more cities with ACO Implementation

# Conclusion

When undergoing the UML process for this assignment a good amount of thought had been invested into the design procedure. The Use Case diagram and scenario description was fairly straight forward as it had to incorporate the details within the brainstorm table, but when I progressed towards the Class Identification, this was when issues started to occur. Although gathering the nouns from the scenario description might sound like primary school work, it was in-fact very tricky as the nouns gathered had to be considered with thought, because it can directly affect the future design of the UML. This was where the problem appeared when implementing the code. Within the UML design, I had three Classes, this caused a lot of misperception during the implementation stage as I tried to include as many functionalities within the three individual classes, as a result, this was not a well organised or efficient piece of coding, consequently, I created a TSPClass library that consisted of four important classes dealing with the city graph implementation, distance between cities, coordinates of the cities and lastly the data that will be loaded onto the GUI. On-top of this implementation, I also created four other classes named TSPSolver, GreedySolver, ACOSolver and SolverResult, this implementation causes an inheritance from the TSPSolver for both the GreedySolver and the ACOSolver.

Having implemented the code in a way that somewhat follows the UML design, the program does meet the minimum specification and requirement set by the assignment. The graphical user interface displays a graphical illustration of the cities in addition, it also performs the greedy and ACO approach to the travelling salesman problem, however, I was unable to implement the improved ACO version of the code due to the hiccup faced with the classes and briefly trying to understand the concept behind the ACO algorithm, these drawbacks minimised the time I had to finalise the assignment.

To conclude, I personally feel that the UML design is to a reasonable, although certain aspects of the design could have been considered in a little bit more depth for example the Class Identification section. This may have been a setback in its own sense during the early stages, the UML design was still used as a foundation during the coding procedure, although the addition of classes afterwards may have ignored the reason behind a UML design, I still found the assignment very informative and a huge learning curve as I gathered a significant amount of knowledge and understanding behind a UML design.

# References

* D.J. Deitel & H.M. Deitel (2009). *Visual C# How to Program 2008*. Upper Saddle River, New Jersey: Pearson Education, Inc.. N/A.
* James McCaffrey. (2011). *Use Bee Colony Algorithms to Solve Impossible Problems.* Available: https://msdn.microsoft.com/en-us/magazine/gg983491.aspx. Last accessed N/A.