



Swarm Intelligence

APPLICATION OF ANT COLONY OPTIMIZATION

1 Table of Contents

2	Introduction	2
3	Interpretation of the specifications.....	2
4	Requirements capture	3
4.1	<i>Functional requirements</i>	4
4.2	<i>Non-functional requirements</i>	4
5	Use Case View	5
5.1	<i>Brain storm</i>	5
5.2	<i>Scenario descriptions</i>	7
5.3	<i>Class identification</i>	8
5.4	<i>crc cards</i>	9
5.5	<i>initial statechart</i>	10
5.6	<i>Sequence diagram</i>	11
6	Analysis Model	12
6.1	<i>Attributes (all private)</i>	12
6.2	<i>methods</i>	13
6.3	<i>sequence diagram</i>	14
6.4	<i>Class diagram</i>	15
6.5	<i>Statechart diagram</i>	16
6.6	<i>non-functional requirements</i>	16
7	Design Modelling.....	17
7.1	<i>Revist use-case model</i>	17
7.2	<i>Sequence diagram</i>	17
7.3	<i>Textual description of object to object interaction</i>	17
7.4	<i>Subsystems</i>	17
7.5	<i>implementation of non-functional requirements</i>	17
7.6	<i>Deployment model</i>	17
7.7	<i>Legacy issues</i>	17
7.8	<i>Reconsider the attributes</i>	17
7.9	<i>Reconsider the associations</i>	17
7.10	<i>statechart</i>	18
7.11	<i>class diagram showing visibility</i>	18

8	Testing.....	19
8.1	<i>Aim</i>	19
8.1.1	<i>File loading and data extraction (together with distance calculation and result storage).....</i>	19
8.1.2	<i>Greedy algorithm and result accuracy.....</i>	20
8.1.3	<i>Basic aco algorithm and result accuracy.....</i>	20
8.1.4	<i>Improved aco algorithm(s) and result accuracy.....</i>	21
8.1.5	<i>Graphical user interface.....</i>	21
9	Conclusion	22
10	References.....	23
11	Appendix 1	24

1 Introduction

The aim of this project is to apply Swarm intelligence algorithms in an attempt to solve a discrete optimization problem. Through the use of UML object oriented designing tools and .NET C# (CSharp) object oriented programming, this report looks at the steps taken to gather information, analyzing the requirements, extracting the functional needs of the project, laying out a design for the software and finally, implementing the design. This report is based on four major sections namely: designing; program implementation; testing and conclusion.

The Ant Colony Optimization (ACO) is a type of Swarm Intelligence algorithm that imitates ant's capability to find the shortest path from their food source to their nest and eventually having an army of ants congregate on this path. This behavior is brought about by the ant's ability to deposit pheromones and make probabilistic decisions based on the ones already around. Individual ants start off from their nest in search for food and go separate paths laying pheromones along the way. Once a food source is found the ant will make its way back to the nest while continuing to lay pheromone. The shorter the trip to the food source, the more pheromone left on that path with more ants following the same path. With time elapsing, pheromone laid starts to evaporate leaving the path congested with the pheromone from the high number of ants frequently patronizing the shortest route. Eventually, the rest of the ants abandon their initial paths to use this shorter route which, of course, has large amounts of pheromone deposited.

The Travelling Salesman Problem (TSP) is a real life problem that is brought about by the need to find the shortest path between a set number of locations or cities distributed within a wide space. The aim is to travel to all locations at least once and make it back to the starting point without backtracking to a previously visited location. As the number of cities in the space increase so does the variations of how many paths there are to the solution $[(\text{Number of Cities} - 1)! / 2]$. The enormous number of possible paths make it impossible to solve the problem through conventional mathematics. Due to the similarities in nature of the TSP and that of ants needing to find the shortest path to a food source, ACO algorithm can be applied not to solve the problem but to at least come up with an optimum solution.

2 Interpretation of the specifications

One way to obtain a solution for the problem without applying ACO is by using Nearest Neighbor Heuristic (Greedy Algorithm). The Greedy algorithm works by identifying the nearest city from the list of cities and picking it. The process is repeated until all the cities have been visited. Although it seems to work, the solution obtained is not optimum therefore a better algorithm need to be used.

The basic ACO is the process whereby an artificial ant with capabilities of remembering where it had been (memory) and making a decision (probabilistic) of which city to go to next based on an artificial pheromone trail. The intensity of the trails are directly proportional to the estimations made by ants to build a good solution, and the trails are updated by the ants [14].

A number of artificial ants determined by the number of cities (N-Cities) available is generated and each ant is initialized at a unique node within the problem space. Each ant uses the probabilistic function [15], a User-set initial pheromone amount and inverse of the distances between nodes to decide which node to go to next. This process is repeated as the ants carry out their individual tours for a set number of times. Once all ants are done, the local pheromone deposit is updated by all

ants using the summation of individual arc deposits $\Delta\tau_{ij}$ and evaporation constant set between 0 and 1, expressed by:

$$\tau_{ij}(t) = (1 - \text{evaporation constant}) * \tau_{ij}(t) + \sum \Delta\tau_{ij}(t) \text{ for all ants generated [15].}$$

The above process is repeated 3000 times while using the updated pheromone deposit. Update is only done at the end of each iteration. Improvements are made to the above algorithm to lead to improved probabilistic decision making and a different local pheromone deposit method where only the shortest tour is considered.

3 Requirements capture

A specification sheet is provided for the project with brief descriptions of the Travelling Salesman Problem, the Ant Colony Optimization, the probabilistic decisions, individual deposits $\Delta\tau_{ij}$, local pheromone deposit functions and a list of characteristics of the algorithms [15]. The following points are features identified from the documentation:

- Pheromone is updated after a solution is found
- Ants do not exchange
- Pheromone evaporates over a time period less than the time taken for the swarm to find the shortest path
- Pheromone evaporation is triggered after all ants have arrived at a solution
- Ants make probabilistic decisions using a function
- Concurrent solutions are found by m ants and they all die.
- After an iteration, t is incremented and another set of ants are created
- Each ant finds its own solution
- Global daemon process to observe all Ant behavior
- Amount deposited by an individual ant is added to τ_{ij} and is proportional to the quality of the solution
- Pheromone quantity (local) is updated after all the ants have completed their tour

3.1 *Functional requirements*

C# classes to represent the TSP grid which incorporates information about the intercity distances [15].

- Accept input or loading of data files i.e. *.tsp files
- Pick (x,y) coordinates from the data files and pick data structure for storage
- Represent coordinate data graphically
- Calculate and store distances between points / nodes based on N-City value
- Associate *.tsp files with N-City value, optimum tour length value and dimension value

Solver Classes for solvers (Greedy and ACO + Improvements).

- Calculate Greedy algorithm based on loaded data
- Represent greedy algorithm solution on the graphical user interface
- Implement daemon process for monitoring and parameter change
- Calculate basic ACO algorithm based on loaded data
- Restrict N-City < 100 (without improvements)
- Apply 2 Opt local search heuristic

Graphical user interface.

- Display graphic representation of solution
- Select .tsp data file
- Select calculation to perform

3.2 *Non-functional requirements*

- Display matrix format of node distances
- Restrict node neighborhood to a small number greater than 1 to improve computational performance
- ACO version 2 through the implemented daemon process update the local pheromone with the shortest tour of the iteration.
- Probabilistic decisions by a pseudo-random-proportional rule
- 2 opt local search heuristic by switching the order of two edges
- Restrict *.tsp file size to a reasonable dimension size

4 Use case view

4.1 Brain storm

Table 4.1 Prioritization of Potential Requirements

Concept	Necessity	Risk	Cost	Priority
Accept loading of data files	High	Med	Low	Med
Pick x ,y coordinates from the data files and pick data structure for storage	High	Med	Low	Med
Represent coordinate data graphically	Med	Low	Low	Med
Calculate and store distances between points	High	High	Med	High
Associate *.tsp files with N-City value, optimum tour length value and dimension value	Med	Med	Low	Med
Calculate Greedy algorithm based on loaded data	High	Med	Med	Med
Represent greedy algorithm solution on the graphical user interface	High	Med	Low	Low
Implement daemon process for monitoring and parameter change	Med	Med	Low	Med
Calculate basic ACO algorithm based on loaded data	High	High	High	High
Represent basic ACO algorithm solution on the graphical user interface	Med	Med		Med
Restrict N-City < 100 (without improvements)	Med	Med	Low	Low
Apply 2 Opt local search heuristic	Low	Low	Low	Low
Graphical user interface Display graphic representation of solution	Med	Med	Med	Med
Graphical user interface Select .tsp data file	High	Med	Med	Med
Graphical user interface Select calculation to perform	High	High	Med	High

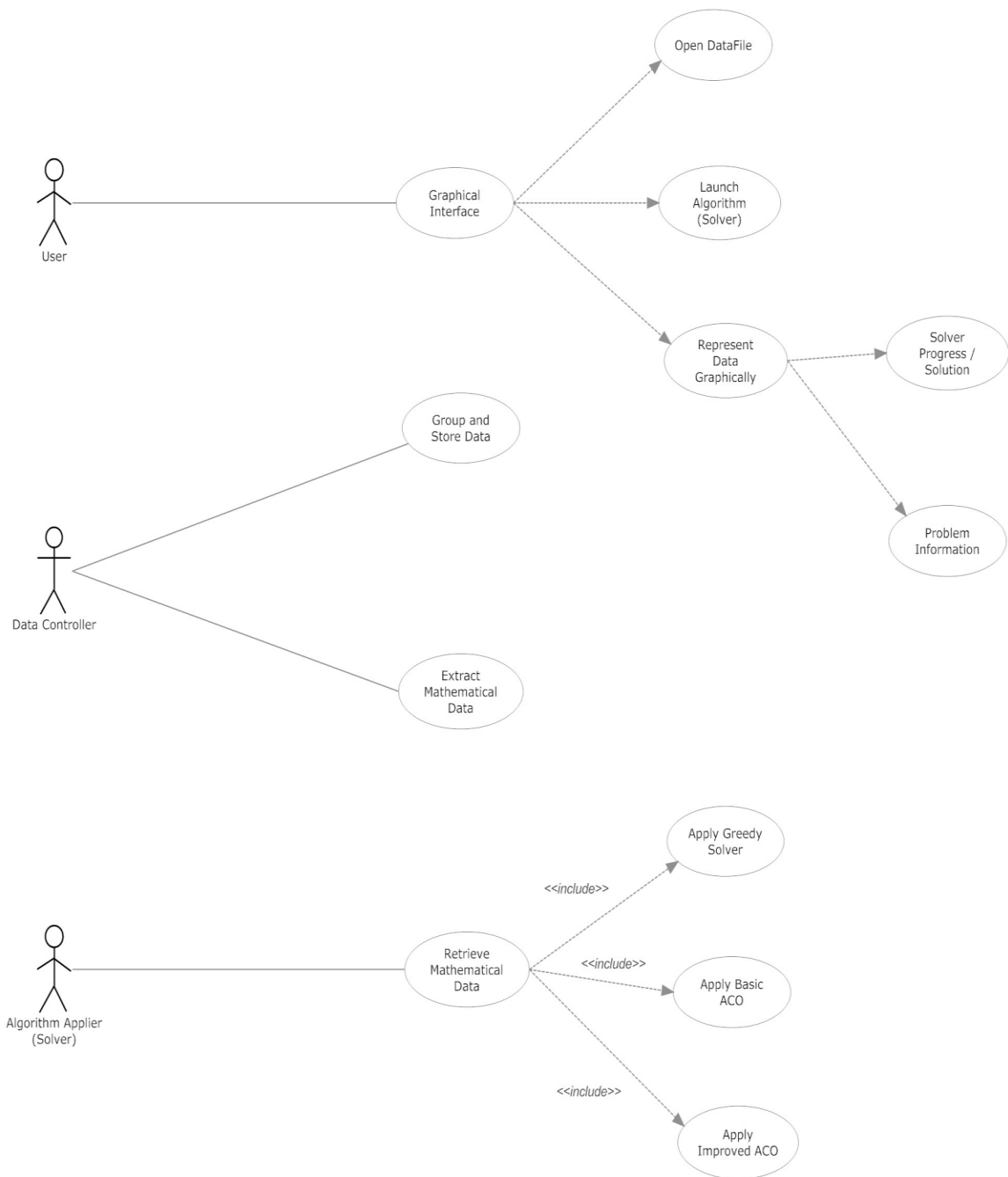


Figure 4.1 Use-Case Diagram

4.2 Scenario descriptions

User:

The user selects a *.tsp file to upload from a collection of data files a well suited data structure. A graphical user interface represents the relevant information on the scope of the problem as in; dimension; optimum tour length. This is displayed back to the user selects the solver to use on the problem including setting initial parameters and variables before pressing the start button to obtain a solution. Progress of the solution is displayed back to the user.

Data Controller:

Node x and y coordinates are obtained from the *.tsp files uploaded by the user. Relevant information associated to the particular problem is passed on to the graphical user interface. Distance between extracted coordinates is worked out and stored in storage structures that can be easily indexed and passed on to the Solver / Algorithm applier. The distance list sorted in descending order to easily determine short distances between the nodes.

Note: A node is a pair of x and y coordinates corresponding to a point in the problem space

Algorithm Applier/ Solver:

Algorithm applier retrieves coordinate and problem information from Data Controller. Algorithm applier is split into two subsections that carry out two separate functions on the distance data.

Greedy Algorithm:

- *Access from the list of distances the shortest distance between current node and the rest of the nodes*
- *Add that distance to the tour length*
- *Move to that Node and put the node just departed from to the visited list*
- *Repeat from the first point until the list is empty*
- *Once the list is empty go back to the first node*

Basic ACO Algorithm:

- *User sets the initial pheromone value for all the arcs*
- *User sets the arc influencing parameter*
- *The amount of ants to be initialized is equal to the dimension of the file*
- *Start iteration*
- *For each ant belonging to 'm'*
 - a) *a node is randomly selected from the list*
 - b) *The next node is determined through the probability function (probability against all the other unvisited nodes)*
 - c) *Pheromone for that arc is recorded*

- d) (b) is repeated until the tour is completed and a solution is found
- e) Once all m ants find a solution local pheromone for every arc is updated and the iteration count is incremented
- f) Process is repeated until 3000 iterations are done.

4.3 Class identification

Nouns

User scenario: tsp file, upload, solver, parameters, solution, GUI, storage, problem information, start button, display

Data Controller Scenario: node, x and y coordinates (Vectors), tsp files, uploaded, GUI, distance list, storage, solver

Algorithm Applier / Solver scenario: distance list, x and y coordinates (Vectors), current tour length, visited list, initial pheromone, iteration, arc length parameter, ants, dimension, next node, unvisited node, probability function, arc pheromone, solution, solver

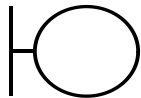
Potential Attributes

Parameters, Node, X and y coordinates, Current tour length, Previous, node list, Visited list, Arc length parameter, Ants, Dimension, Next node, Unvisited node, Arc pheromone

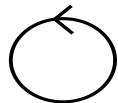
Potential Classes

Distance list, Solver

Stereotypical Classes



Boundary: start button, GUI, solution, display, upload



Control: controller

After reconsideration from the list of nouns and stereotypical classes the following are identified as a minimum set of classes: Controller, Solver and Vectors (x and y coordinates).

4.4 Crc cards

Table 4.1 Class Responsibility Collaboration Cards

Class: Vectors	
Responsibilities	Collaborators
The vectors class is responsible for defining the different data structures that will be used for the solver and controller.	Controller, Solver

Class: Controller	
Responsibilities	Collaborators
The controller class is responsible for extracting mathematical problem information uploaded by the user from a tsp file. It provides the information to the vectors class to sort and assign to the correct data structures. The controller will also provide information to the solver regarding the problem. The controller Displays information to the user	Controller, Solver

Class: Solver	
Responsibilities	Collaborators
The solver class holds the bases for the mathematical calculations to be carried out for the problems. The solver class uses the data structures from the vector class to perform its mathematical functions. It also gathers problem information for the controller	Controller, vectors

4.5 Initial statechart

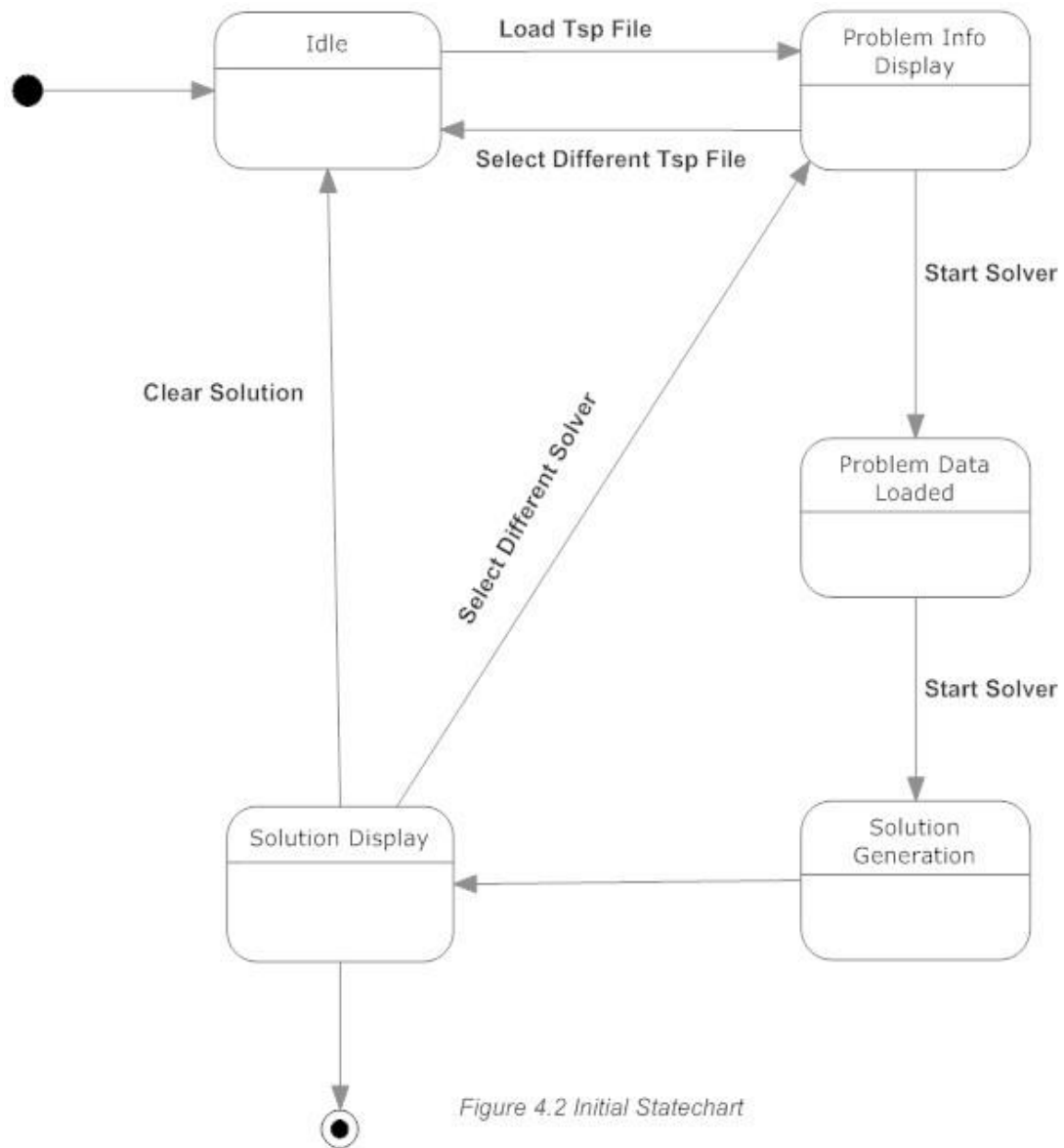


Figure 4.2 Initial Statechart

4.6 Sequence diagram

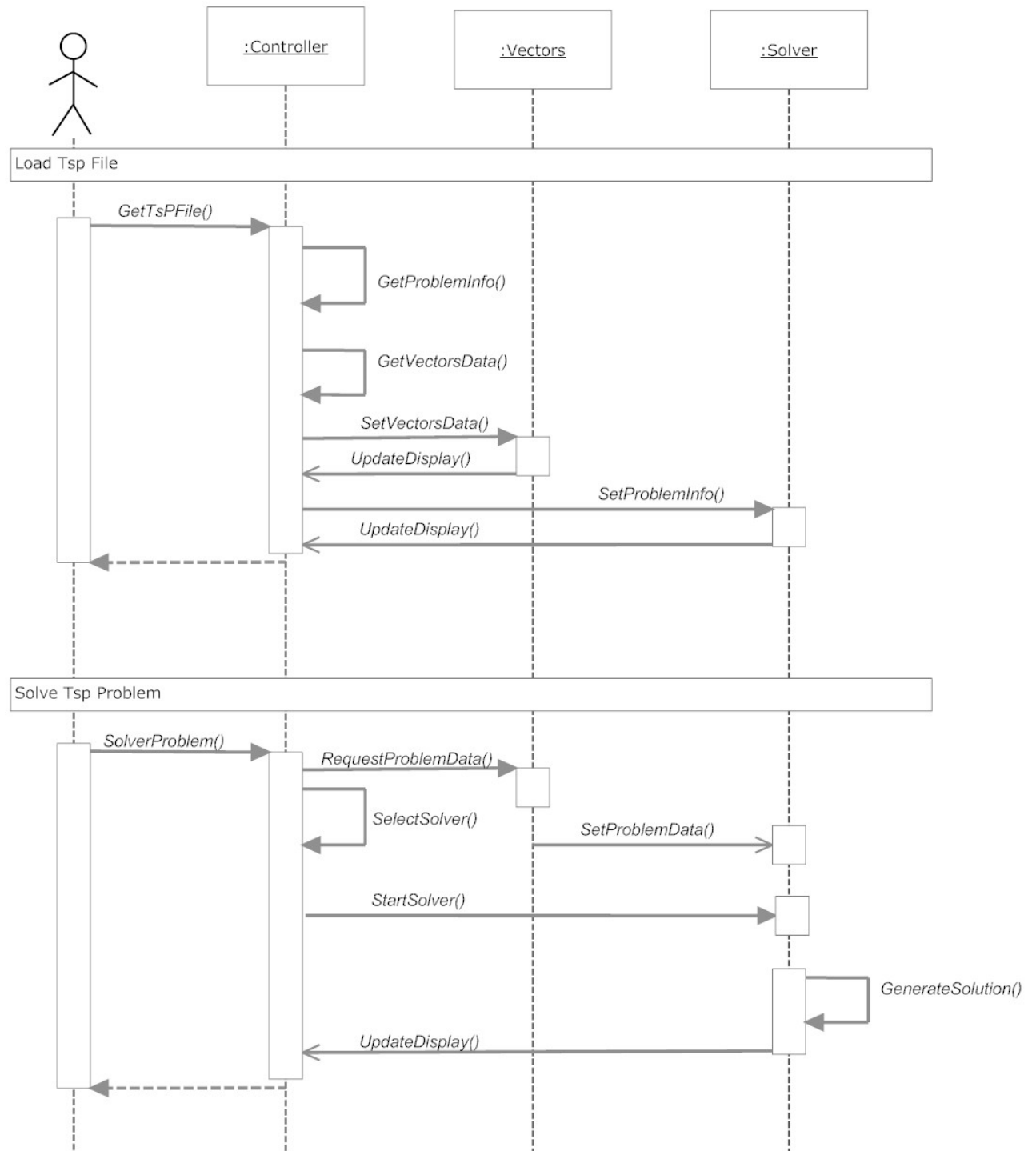


Figure 4.3 Sequence Diagram

5 Analysis model

From Vectors Class

Potential attributes: dataFile; dataFileArray; DistanceList; dimension size; arc_deposit; pheromone_deposit; arc;

Potential methods: SortDistanceList(); CalcDistance();

From Controller Class

Potential attributes: tspfile; dimension; optimum tour length;

Potential methods: LoadTspFiles(); StartSolution(); SelectAlgorithm(); InfoDisplay()

From Solver Class

Potential attributes: initialPheromone; iteration; beta; N_Cities; Pheromone

Potential methods: ProbDecision(); Greedy(); individDeposit(); PheromDeposit();

5.1 *Attributes (all private)*

Table 5.1 Attributes

Class	Attribute	Comment
Vectors	dataFile	file
	dataFileArray	Integer Array
	DistanceList	Integer Array
	DimensionSize	Integer
	arc_deposit	Decimal
	pheromone_deposit	Double
	arc	String
Controller	tspfile	File
	dimension	Integer
	optTourLength	Integer
Solver	initialPheromone	Double
	iteration	Integer
	beta	Double
	N_Cities	Integer

5.2 Methods

Table 5.2
Methods

Class	Method	Comment
Vectors		
	SortDistance()	Sort the distances obtained from CalcDistance() in descending order
	CalcDistance()	Calculate distances between all the nodes
Controller		
	LoadtspFile()	Load the tsp file and extract problem information and data for use with calculations
	StartSolution()	Start generating a solution after a tsp file has been loaded and an algorithm is selected
	SelectAlgorithm()	Select an algorithm to use to solve the problem based on the tsp file loaded
	Reset()	Reset the interface for further operations
	InfoDisplay()	Displays problem information and solution
Solver		
	ProbDecision()	Determine the next city to visit based on the probabilistic function and available DistanceList
	Greedy()	Performs the greedy algorithm of picking the next city to go by picking the shortest distance from the distance list
	IndividDeposit()	Calculates deposits done by individual ants per arc
	PheromDeposit()	Calculates the final pheromone deposit from the accumulated values from all ants at the end of an iteration
	BasicACO()	Performs the task of running and monitoring ProbDecision(), IndividDeposit(), PheromDeposit() and make sure they are run in a desired ordered to obtain results.

5.3 Sequence diagram

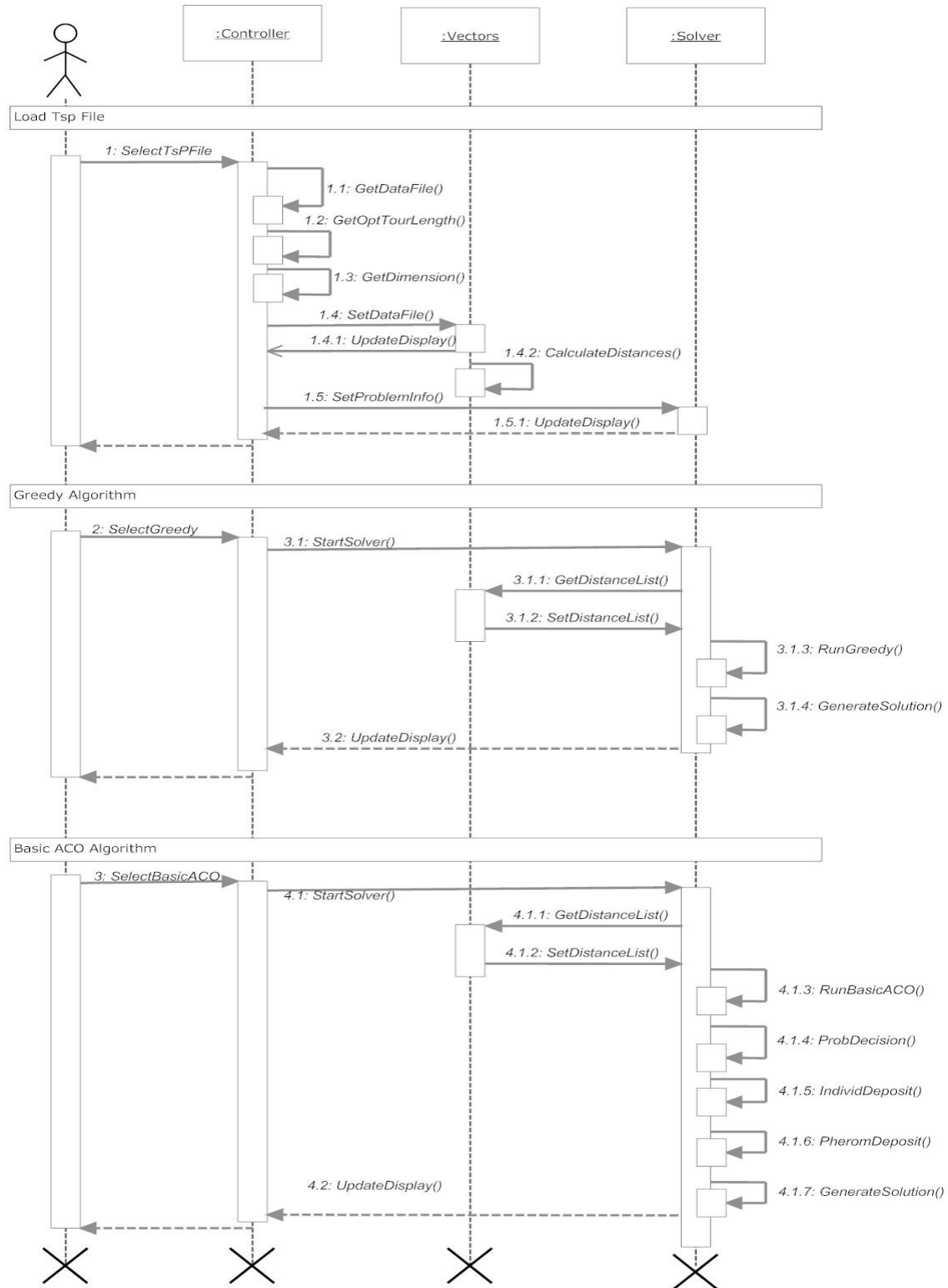


Figure 5.1 Analysis Model Sequence Diagram

5.4 Class diagram

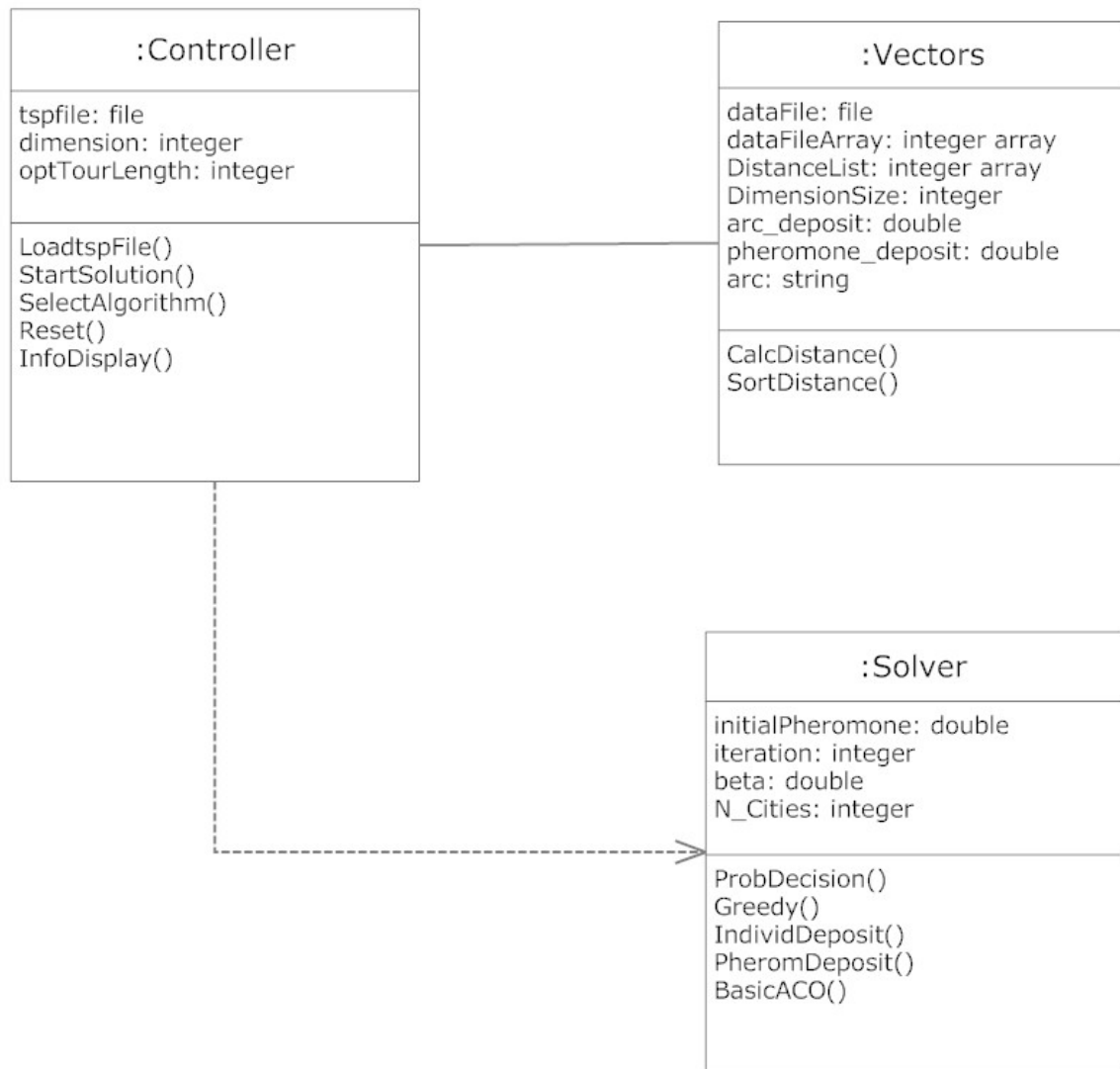


Figure 5.4 Analysis Model Class Diagram

5.5 Statechart diagram

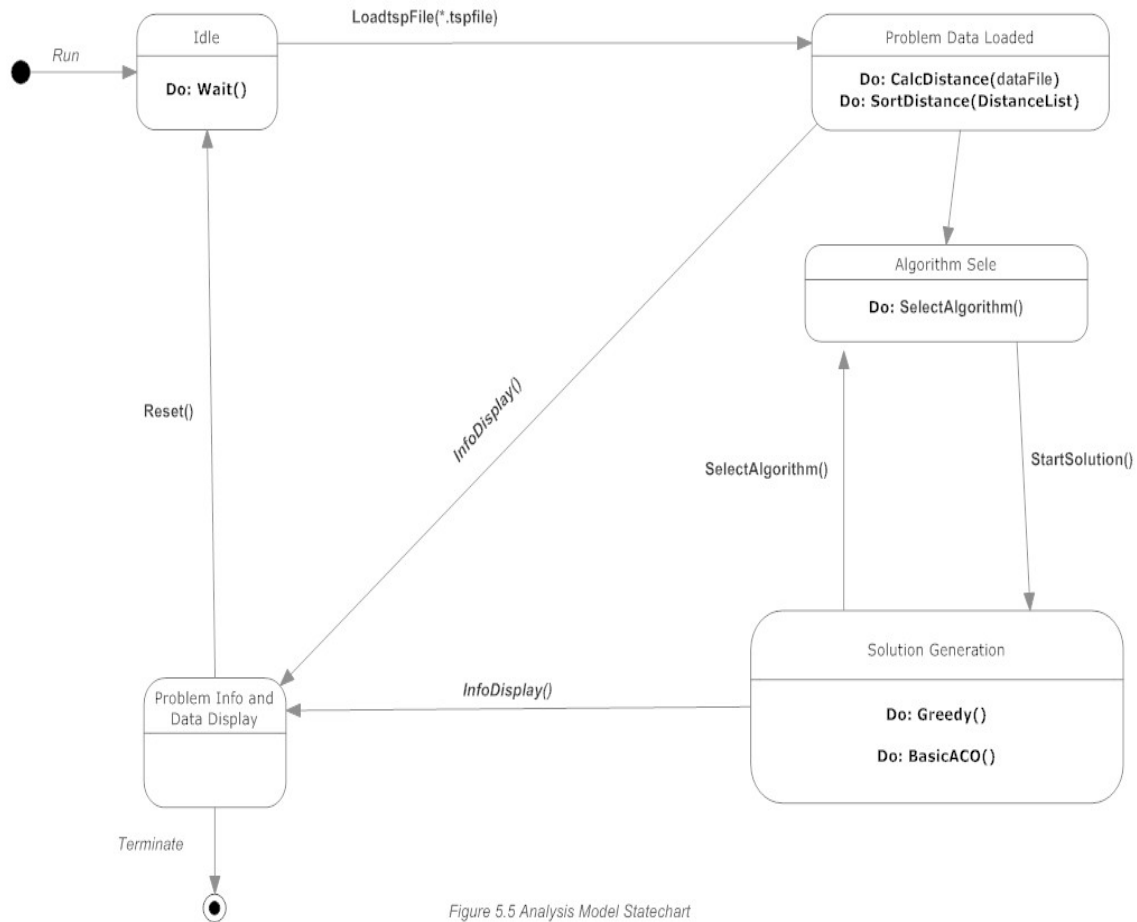


Figure 5.5 Analysis Model Statechart

5.6 Non-functional requirements

- Display matrix format of node distances
- Implement that the node subset (neighborhood) can be between the range of 4 and 10 to improve computational performance
- ACO version 2 should be implemented to update the local pheromone with the shortest tour of the iteration.

6 Design modelling

6.1 *Revisit use-case model*

The design meets the minimum requirements necessary to test and implement.

6.2 *Sequence diagram*

The sequence diagram has already had its qualifiers applied.

6.3 *Textual description of object to object interaction*

The controller which is part of the interface accepts input in the form of *.tsp files. The user selects an algorithm to perform a calculation. The interface displays a graphical representation of the solution with the optimum tour length

The vector class stores the x and y coordinates obtained from the *.tsp file, and stores the distances generated

The Solver class contains all the methods necessary for the calculations. Inheritance has been kept to a minimum to reduce complexity of implementation.

6.4 *Subsystems*

There are no subsystems required

6.5 *Implementation of non-functional requirements*

The ACO improvements will be applied later on in the testing stages through existing function alterations

6.6 *Deployment model*

Algorithms will be run sequentially due to the need to update a local file every iteration so iterations cannot be run in parallel. So the Controller class will be deployed on a single processor

6.7 *Legacy issues*

The software will be developed on a .Net Framework which is constantly updated so there will not be any legacy issues

6.8 *Reconsider the attributes*

They have been dealt with in the Revisited Class Diagram

6.9 *Reconsider the associations*

Relationship between Vectors and Controller class remain the same, however that of Solver and Controller have changed to a bi-directional association as well to keep things simple.

6.10 Statechart

No further revisions required

6.11 Class diagram showing visibility

Methods and attributes have been relocated within the classes to keep class dependency at a minimum.

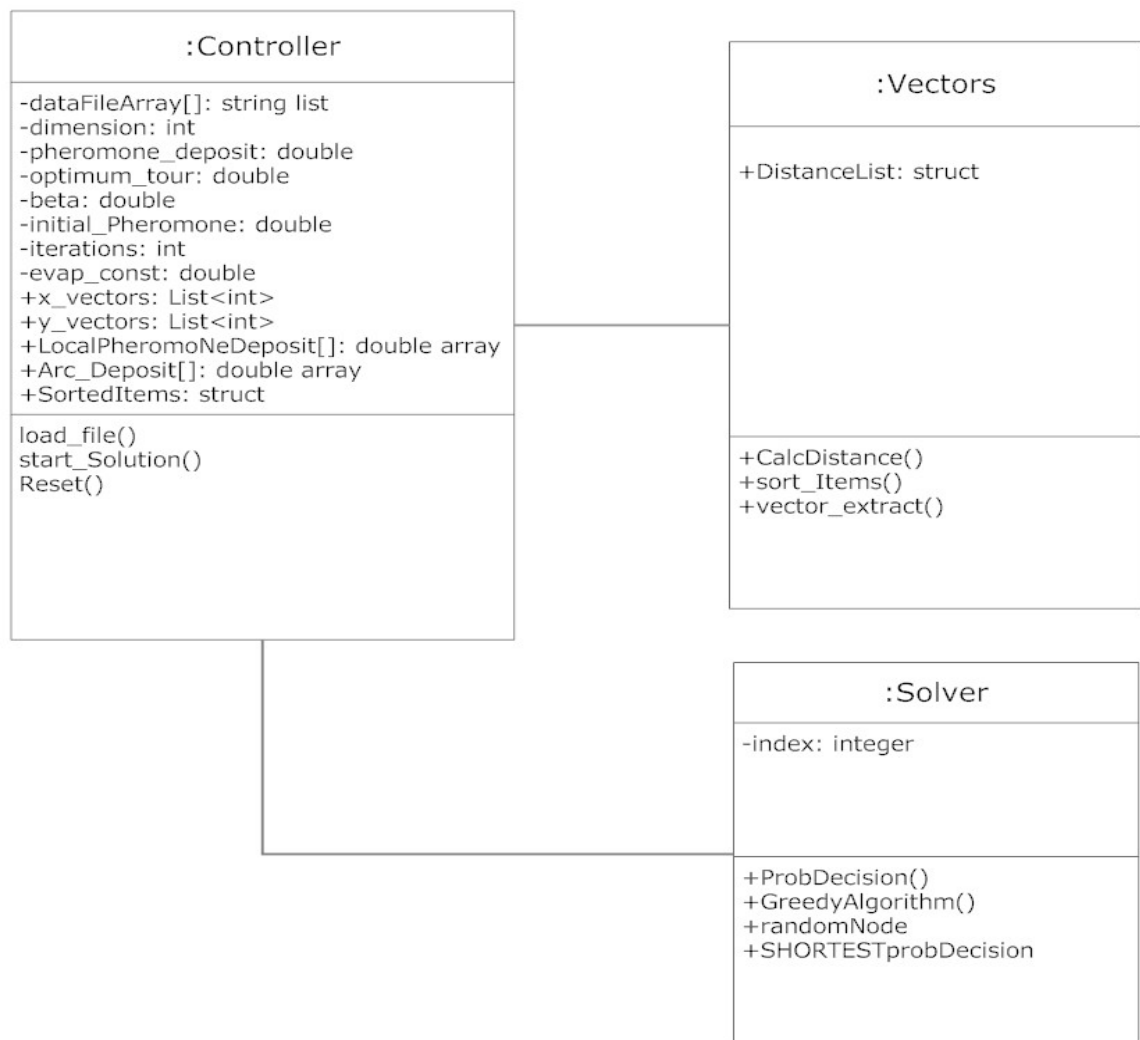


Figure 6.1 Design Model Class Diagram

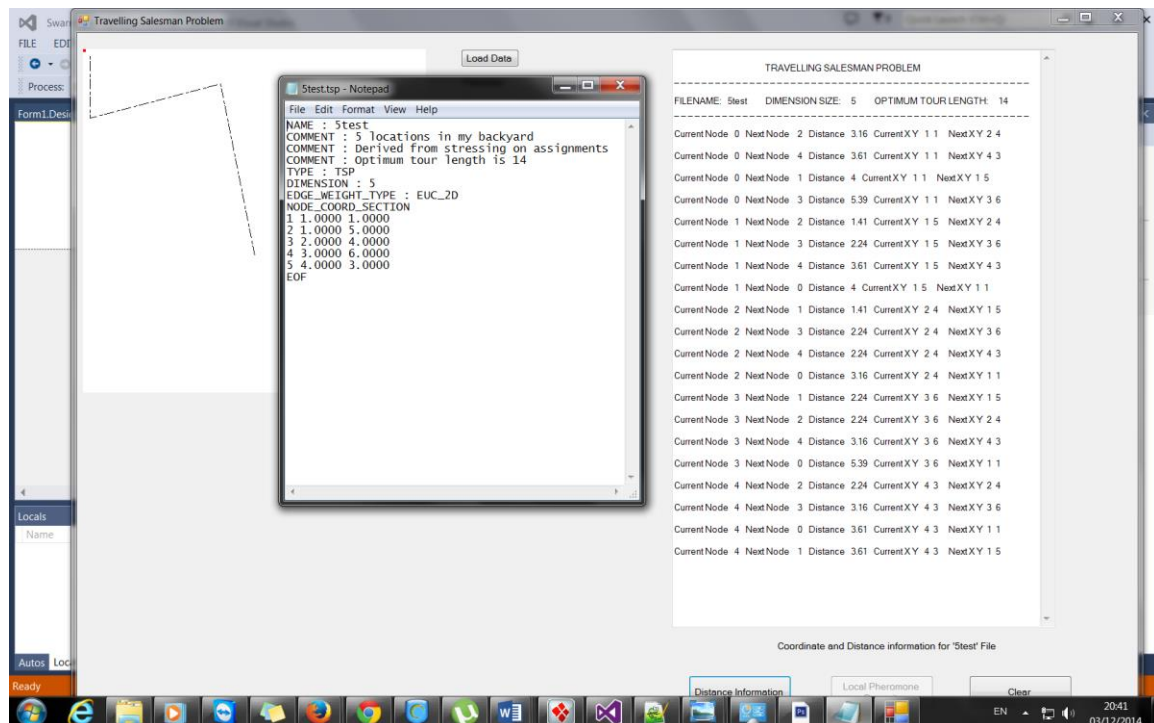
7 Testing

7.1 Aim

Software has been implemented and now needs to be checked for proper operation. Unit test is performed to ensure that all the subsystems that make up the software as a whole are working properly. The subsystems of this software can be named as: File Loading and Data Extraction; Distance Calculation and Result storage; Greedy Algorithm and Result Accuracy; Basic ACO Algorithm and Result Accuracy; Graphical User Interface Functionality. To make testing easier and quicker a custom *.tsp of Dimension 5 and Optimum tour length rounded to 14 has been created with 5 x coordinates and corresponding y coordinates. Calculations of the Optimum tour length for the file have been done on paper and are attached to Appendix 1.

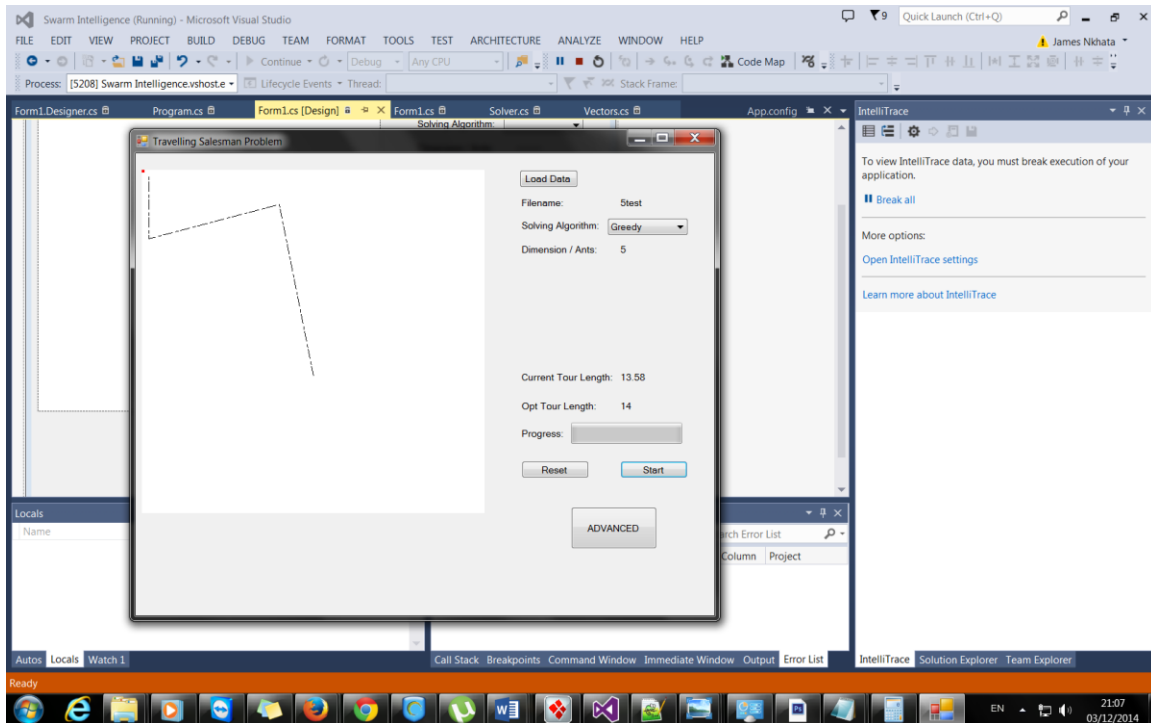
7.1.1 File loading and data extraction (together with distance calculation and result storage)

The file loading and data extraction is tested by opening the source *.tsp file using notepad and comparing its contents to those of the same file opened by our software being tested.



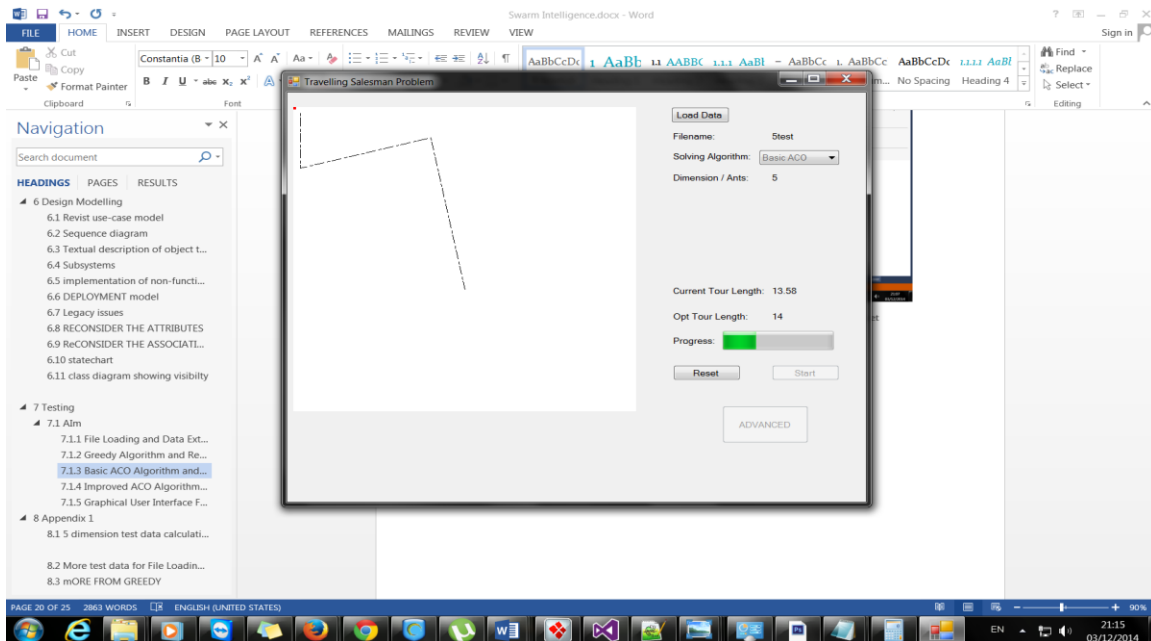
The screens are compared side by side. And the results are satisfactory for this file (more test attached to Appendix 1)

7.1.2 Greedy algorithm and result accuracy



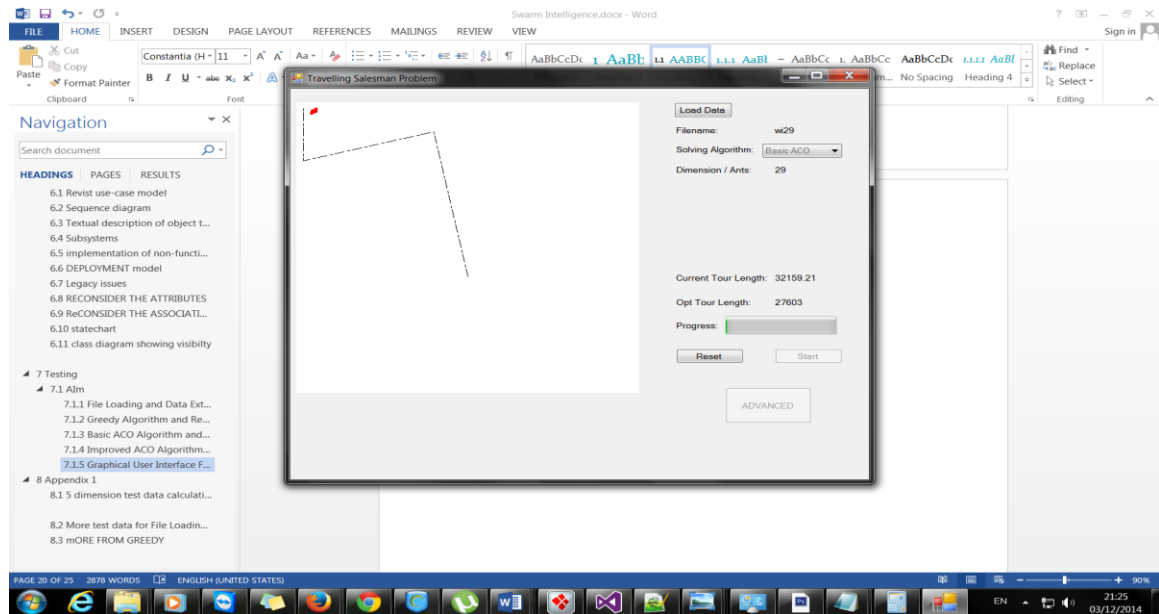
Current Tour Length is the result returned which is identical to that of the worked out sheet

7.1.3 Basic Aco algorithm and result accuracy



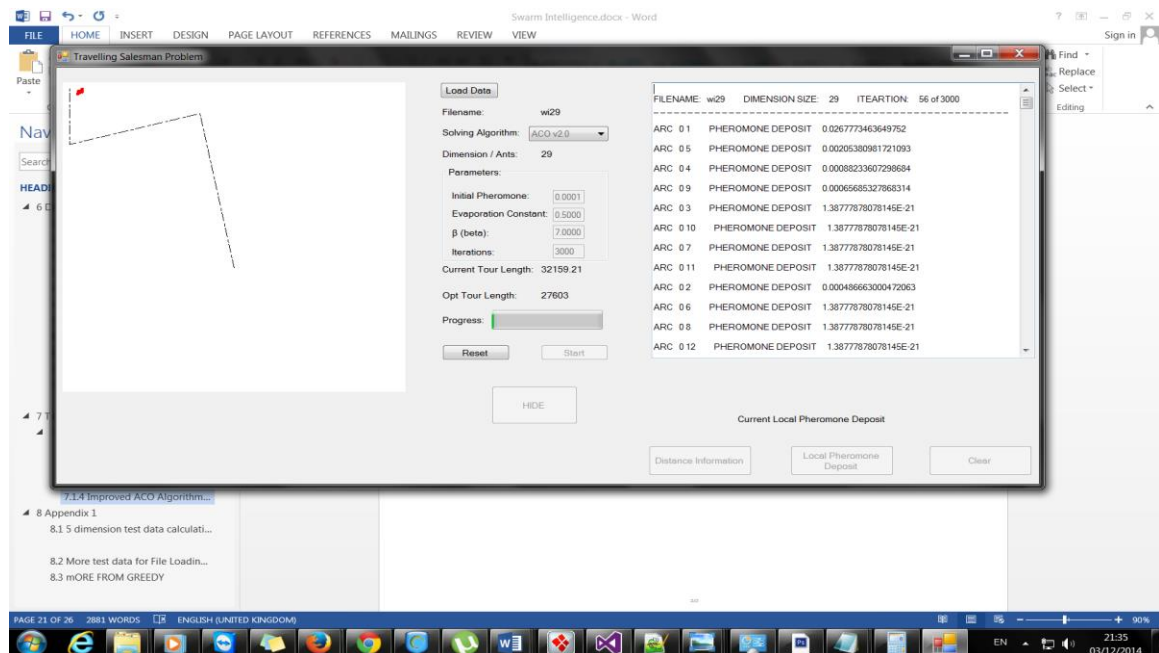
BasicACO returns the same value as greedy

BasicACO on bigger file with larger dimension



The results improved so far from greedy

7.1.4 Improved Aco algorithm(s) and result accuracy



7.1.5 Graphical user interface

The graphical user interface is tested concurrently with the various algorithms and data loading and storing tests.

8 Conclusion

The file loading of the software and sorting returned results that met the user requirements with the exception of files with coordinates of more than four decimal places. Limitations also include files with dimensions larger than 3496.

Distance calculation and result storage satisfied user requirements providing a storage and indexing system that is easy to use, holds multiple lines of a node's information such as distance, node number, next node, and x and y coordinates for both current and next in the same indexed location.

The greedy algorithm performs as expected (sub optimal) and returns results that are far from optimal but still within an acceptable range due to the nature of the algorithm. This proves that the algorithm has been implemented correctly but does not change the fact that the solution is not ideal.

Basic Aco algorithm initializes and returns optimal length for the 5 dimension test file giving the impression that its implementation was correct but however if a larger file of about 29 dimension size is loaded and solution is initialized the amount of time taken to generate results are very long with the result being better than greedy but not close to optimal. This behavior of slow solution generation has been experienced throughout the rest of the Aco improvement versions making it extremely hard to determine if the implementation was done correctly and an optimum solution is achieved. This slow performance to run through the 3000 iterations shows a flaw in the implementation of the algorithms. A factor that was considered was the arcs upon which individual ants deposit pheromone is treated as a uni-directional deposit and the reverse of it is a completely different arc. An adjustment was done to add them both together and then update based on the local pheromone but the results were unpredictable and pheromone deposits rapidly lead to infinite values. This proves that the implementation has to be revisited and adjusted to get accurate results and less processor hungry solutions.

The graphical user interface was not complete with a lot of software lagging bugs that were not addressed due to time constraints. Overall besides the requirements being met in the earlier stages the overall results suggest the Aco algorithm should be revisited and re-implemented to have it working correctly and obtain better results.

9 References

[1] - Reading a text file using OpenFileDialog

<http://stackoverflow.com/questions/16136383/reading-a-text-file-using-openfiledialog-in-windows-forms>

[2] - C# 2010 for Programmers (4th Edition) (Deitel Developer Series) by Paul Deitel, Harvey M. Deitel

[2.1] – Chapter 17 Files and Streams

[3] - <http://csharp.net-informations.com/collection/list.htm>

[4] - <http://stackoverflow.com/questions/7318284/delete-list-of-structures-by-field> (List of Structs)

[5] - <http://stackoverflow.com/questions/18890152/how-to-return-struct-from-function-in-c>

[6] - <http://stackoverflow.com/questions/3360555/how-to-pass-parameters-to-threadstart-method-in-thread> (Passing parameters to threads through functions)

[7] - <http://www.dotnetperls.com/concurrentdictionary> (ConcurrentDictionary)

[8] - <http://www.dreamincode.net/forums/topic/73035-comparing-4-numbers-and-finding-the-large-one/> (Number comparison)

[9] - <http://stackoverflow.com/questions/9570110/concurrentdictionary-addorupdate-by-predicate> (ConcurrentDictionary AddOrUpdate by predicate)

[10] - <http://stackoverflow.com/questions/282118/togglebutton-in-c-sharp-winform>
(**Simple** toggle button)

[11] - https://www.youtube.com/watch?v=r1FbKiHYHcw&list=PLkJzKYGRU_tL-QtyeV9Y_8AzTXw2tVPdC&index=11 (Parallel Programming)

<http://stackoverflow.com/questions/10120693/c-sharp-search-text-file-return-all-lines-containing-a-word>

[12] - <http://stackoverflow.com/questions/11322841/button-tip-when-mouse-is-over>
(Button tip on mouse over)

[13] - <http://www.albahari.com/threading/> (Threads)

[14] - (IMAGE EDGE DETECTION USING ANT COLONY
OPTIMIZATIONALGORITHM, Section 2.2)

[15] – Object Oriented Programming Using C#: Swarm Intelligence Assignment 2014
– 2015 - Dr M. Spann

[16] - <http://msdn.microsoft.com/en-us/library/w34xb12c%28v=vs.110%29.aspx>

(Draw custom dotted lines)

[17] - <http://stackoverflow.com/questions/5730828/c-sharp-drawing-circles-in-a-panel>

(Draw dots)

10 Appendix

5 dimension test data calculation

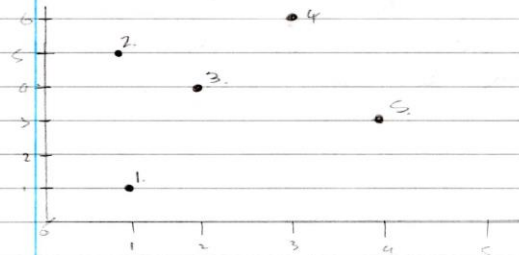
Testing

TESTING

Pythagorean theorem

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

5 random points



1. (1, 1)
2. (1, 5)
3. (2, 4)
4. (3, 6)
5. (4, 3)

Starting from point 1. (Node 1.)

$$1. \rightarrow 2 = \sqrt{(0)^2 + (4)^2}$$

From (Distance) TO

$$1. - 4 - 2$$

$$1. - 3.16 - 3$$

$$1. - 5.38 - 4$$

$$1. - 3.60 - 5$$

$$2. - 1.41 - 3$$

$$2. - 2.23 - 4$$

$$2. - 3.60 - 5$$

$$3. - 2.23 - 4$$

$$3. - 2.23 - 5$$

$$4. - 3.16 - 5$$

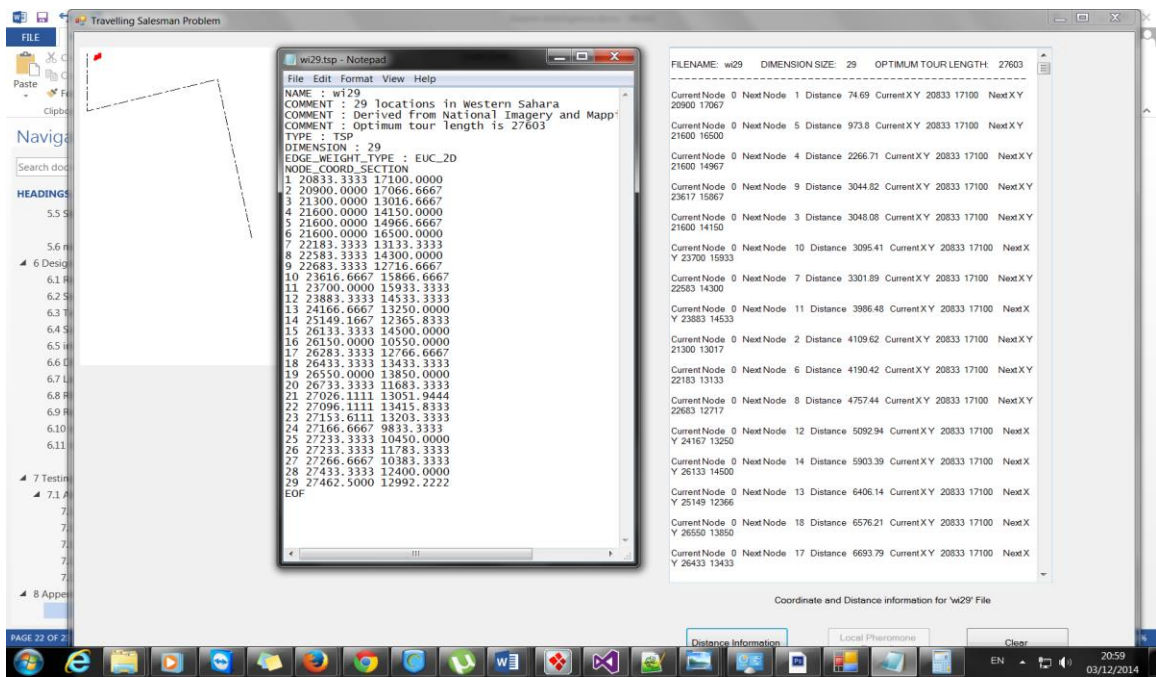
5.

$$1. \frac{3.16}{3.16} \quad 3. \frac{1.41}{1.41} \quad 2. \frac{2.23}{2.23} \quad 4. \frac{3.16}{3.16} \quad 5. \frac{3.60}{3.60} \quad 1.$$

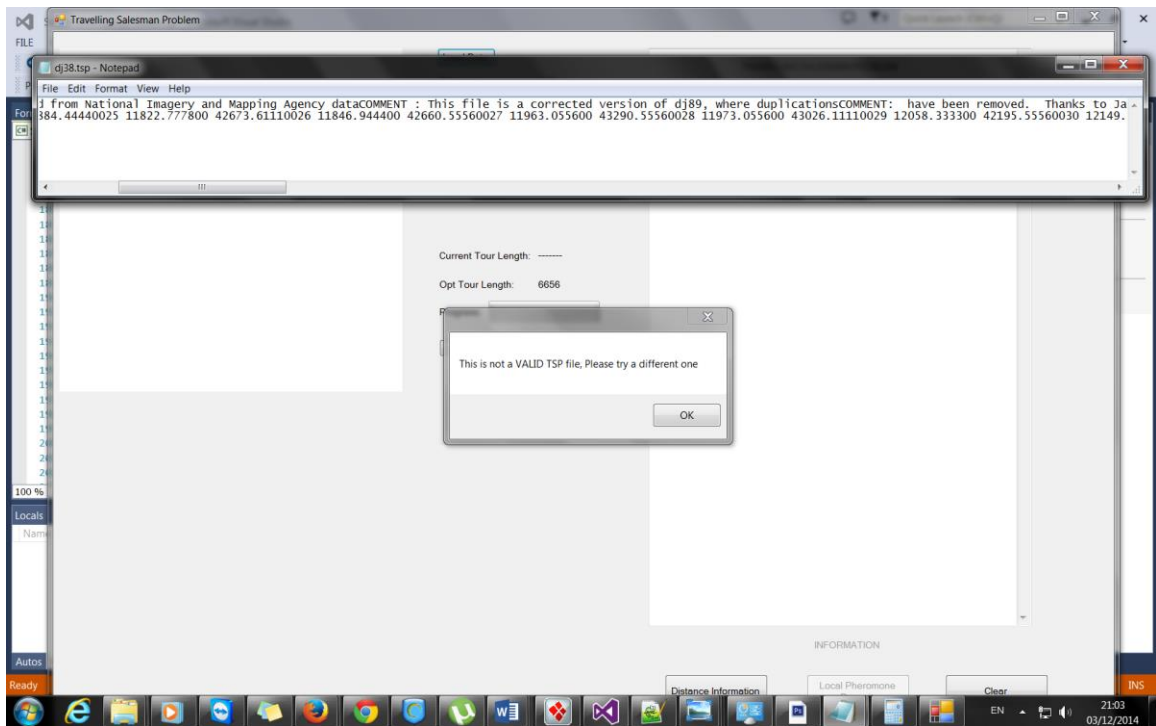
$$3.16 + 1.41 + 2.23 + 3.16 + 3.60$$

$$\text{Current tour length} = 13.56$$

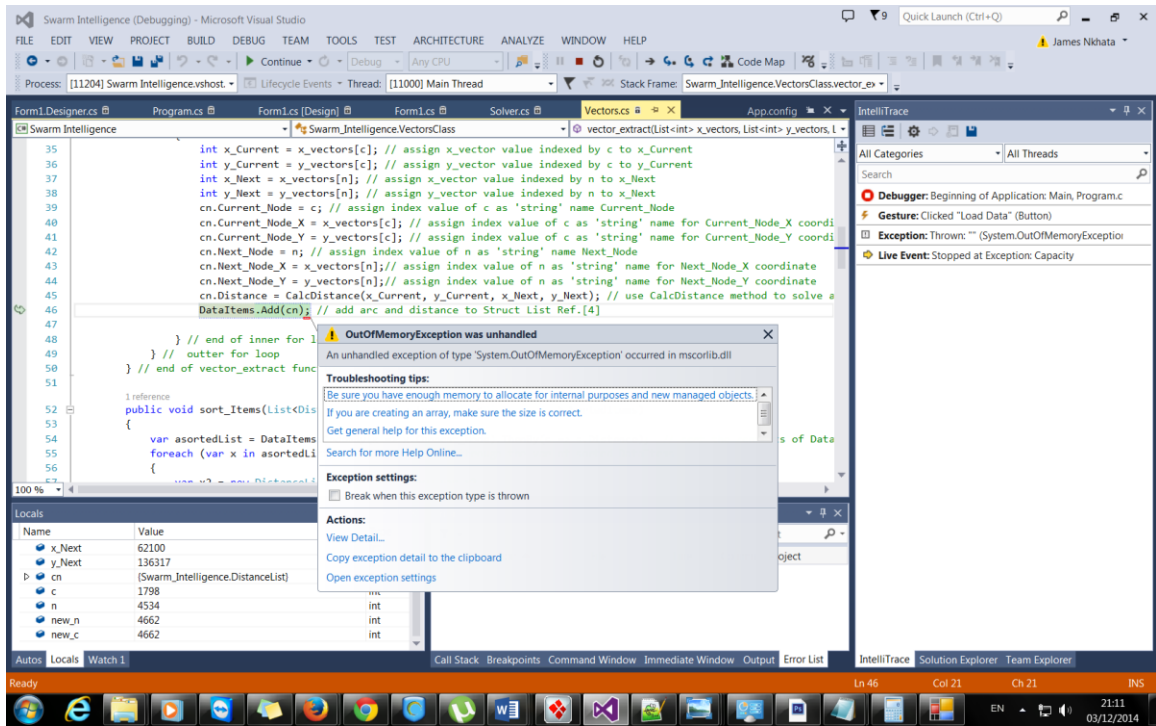
More test data for File Loading and Data Extraction (together with Distance Calculation and Result storage)



ERROR FROM dj38.tsp



FILES BIGGER THAN 3496



MORE FROM GREEDY

