**Object Oriented Programming Using C#**

**Travelling Salesman Problem by Ant Colony Optimization**

**Shanthini Muthusamy**

Contents

[1. Introduction 3](#_Toc405154705)

[2. Use-Case Model: 5](#_Toc405154706)

[2.1 Brainstorm 5](#_Toc405154707)

[2.3 Scenario Description: 7](#_Toc405154708)

[2.4 Class Identification: 8](#_Toc405154709)

[2.5 CRC Cards: 10](#_Toc405154710)

[2.6 Interaction Diagram 11](#_Toc405154711)

[2.6.1 Collaboration diagram 11](#_Toc405154712)

[2.6.2 Sequence diagram 12](#_Toc405154713)

[2.7 Statechart diagram: 12](#_Toc405154714)

[3. Analysis Model 13](#_Toc405154715)

[3.1 Class Identification: 13](#_Toc405154716)

[3.2 Class Diagram: 13](#_Toc405154717)

[3.3 Attribute identification: 14](#_Toc405154718)

[3.4 Interaction Diagrams: 14](#_Toc405154719)

[3.4.1 Collaboration diagram: 14](#_Toc405154720)

[3.4.2 Sequence diagram: 15](#_Toc405154721)

[3.5 Statechart diagram: 15](#_Toc405154722)

[3.6 Identification of Associations: 16](#_Toc405154723)

[3.7 Non-Functional Requirements: 16](#_Toc405154724)

[3.8 Packages: 16](#_Toc405154725)

[4. Design Model: 17](#_Toc405154726)

[4.1 Revisit Use-Case description: 17](#_Toc405154727)

[4.2 Refine Class diagram: 17](#_Toc405154728)

[4.3 Behavioural description of the interaction diagram: 18](#_Toc405154729)

[4.4 Object-Object and Object-Subsystems interactions: 19](#_Toc405154730)

[4.5 Design Subsystems: 19](#_Toc405154731)

[4.6 Identify interfaces: 19](#_Toc405154732)

[4.7 Realise Non-functional Requirements: 19](#_Toc405154733)

[4.8 Reconsider the attributes and associations: 20](#_Toc405154734)

[4.9 Statechart: 20](#_Toc405154735)

[4.10 Method design: 20](#_Toc405154736)

[5. Program Testing and Results: 22](#_Toc405154737)

[5.1 Testing method / flow chart: 22](#_Toc405154738)

[5.2 Test Results: 23](#_Toc405154739)

[6. Discussions & Conclusions: 24](#_Toc405154740)

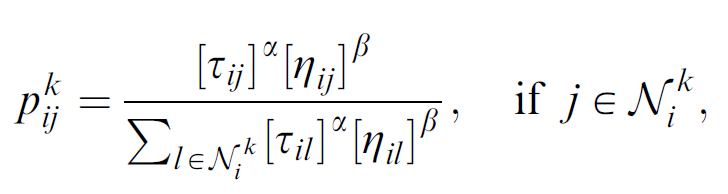
[7. References: 25](#_Toc405154741)

# Introduction

The Travelling Salesman Problem is more extensively studied problem in finding the optimal route in a tour. In TSP, the salesman is to find an optimal tour starting from his home city, visiting all unvisited cities in a tour exactly only once and return back to his home city. To specify the problem, TSP can be represented in a graph G(N,E) where N is the set of nodes representing the cities and E is the set of arcs connecting the nodes ( i, j) ϵ E representing the distance between adjacent cities (dij) In asymmetric TSP, there will be at least one arc in a graph where dij ≠ dji. In symmetric TSP, where dij = dji will be the same for all arc in a graph. The goal in TSP is to find the optimal solution in a closed graph visiting all cities exactly only once in a tour. In Symmetric TSP, a travelling salesman taking a tour in a closed graph will always have (|N| -1) Ị/ 2 possible tours to carry on. Finding the best optimal tour and optimal tour length out of possible tours is going to be NP-hard problem as there is no polynomial time solution.

Ant colony optimization (ACO) (Dorigo and Stutzle 2004) has become successfully used swarm intelligence method used in TSP to solve hard optimization problems. Application of ACO in TSP is inspired from the Ant system (AS) (Dorigo 1999). In Ant System, ants will be able to find an optimal path between the nest and food source in an environment using pheromone deposit. ACO is a detailed treatment of Ant System in solving discrete optimization problems.

ACO is an iterative algorithm applied to the TSP to find the feasible tour from the set of all possible partial tours. Each ant in a tour represents a potential solution. For each iteration, some number of ants k = (1, 2...m) start a tour at random city i and chooses next city to visit at each stage based on the probabilistic rule (1). That is probability of choosing next city is based on the associated pheromone trail on each arc (τij) and heuristic information value ηij=1/dij until all cities are visited in a tour.



1

determines the feasible neighbourhood city for ant k. If parameter α = 0, it will select the closest city to visit according to nearest neighbourhood heuristic. If parameters α > 1, β = 0, only pheromone trail is used to find the next city without the heuristic value will lead to stagnation and poor results as ants will follow the same route to construct the tour. If β > 1, this will yield a better solution in finding the next city. Each ant has an internal memory to store the list of cities visited in sequence, which enables to select the next feasible neighbourhood. After all the ants completed their tour in iteration, the pheromone evaporation is triggered and adding the pheromones on the paths they have traversed in a tour. Pheromone evaporation and pheromone update is carried out as in the equation (2).



2

Where  is the quantity of pheromone on the paths they traversed in a tour. It is defined as in (3).



Where *C*k is the tour length and *T*k is the tour by ant k. The more pheromone on the arcs, the better the tour. The ants in future iteration are more likely to choose the shortest tour.

To implement the basic ACO and other improvements into the Travelling salesman problem, first analyse the requirements of the project specification and then model a design. It is a good approach to have a clear thought process of what actually a system wants. The Unified Modelling language (UML) is a visual modelling language that helps to visualize what the software system really wants. Once the requirements have been modelled using UML, they can be implemented using object oriented programming, C# in the TSP problem. This report explains detailed modelling method using UML and how it was implemented.

# 2. Use-Case Model:

## 2.1 Brainstorm

Potential requirements for the TSP domain model are list in below table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Concept | Necessity | Risk | Cost | Priority |
| Find the optimal tour and optimal tour length in a TSP tour | High | Med | Med | High |
| Start a tour at random city and visit all cities only once and return back to start city | High | Med | Med | High |
| Chose next city at each stage of a tour based on pheromone deposit and distance between cities. | High | Med | Med | Med |
| Memory to store the list of cities visited. | High | Low | Low | Med |
| Nearest next city uses nearest neighbourhood heuristic (Greedy Algorithm) for fast computational performance | Med | Low | Low | Med |
| Pheromone evaporation and update is triggered after completing each tour | High | Med | Low | Med |
| Develop a graphical user interface to represent the cities configuration , optimal tour and optimal tour length for a particular tour | Med | Low | Low | Med |
| City data sets is required to solve the TSP in order to find an optimal solution | High | Low | Low | High |

**Table 1 Potential requirements for TSP**

**Use case view diagram:**



## 2.3 Scenario Description:

## 

|  |  |
| --- | --- |
| Travelling salesman using ACO algorithm: | The travelling salesman will find the shortest minimum tour distance and shortest tour for a given city list in a closed graph using a basic Ant Colony Optimization algorithm. For a given city list, assume the distances are symmetric between the cities in a closed graph. For each iteration, generate some number of ants which acts as travel agents to start the tour in the TSP. Each ant in a group will start a tour at some random city, picks next city to visit at each step in a tour based on two main components: i) pheromone trail on the edges of the cities and ii) heuristic value, and returns back to the start city. At the start of each tour, some initial pheromone is deposited on all routes and the memory is emptied to store the new tour. The travelling salesman will make sure to visit all cities in a tour only once and return back to the start city. When the travelling salesman visits each city, the memory is updated with the new visited cities in sequence that will help to determine the list of unvisited cities yet to visit. After completing a tour, pheromone evaporation is triggered, pheromone update will take place on the traversed route and the ants are wiped out. The traversed route may be partial optimal tour in each iteration. Memory helps to retrace the traversed tour to deposit the pheromone. The shorter the route, the more pheromone deposited on the edges of the route. The search for optimal route continues for next set of ants for fixed number of times. |

|  |  |
| --- | --- |
| Travelling salesman using Greedy algorithm: | The travelling salesman will find the short tour for list of cities in a closed graph. The traveller starts a tour at some random chosen city, visits next nearest unvisited city until all cities are visited exactly once and return back to the starting city. The tour cycle is to repeat for fixed number of iterations to return the final short tour. |

|  |  |
| --- | --- |
| Data Source: | Data Source provides several TSP instances which is a test data to construct the optimal tour to solve the TSP. Each TSP instance is TSP city data sets that provide the city co-ordinates. The intercity distances are calculated using the city co-ordinates. |

|  |  |
| --- | --- |
| User Interface: | User interface that will help to solve the TSP tour is Graphical user interface (GUI). It outputs the constructed tours along with the optimal tour and optimal tour length for user view. Initially, the interface will allow the user to load the TSP city list file and set up the location of all cities in a tour graph. The graphical user interface will provide various options for a user to select an algorithm to run on the TSP tour. All constructed tours will be displayed as the algorithm runs for fixed number of iterations and the tour with best route is highlighted at the end of the tour execution process. |

## 2.4 Class Identification:

Identification of potential classes for the TSP is as follows:

The following nouns are identified from the scenario description.

**Noun Identification:**

|  |  |
| --- | --- |
| Travelling salesman using ACO: | Travelling salesman, Tour, Distance, City List, Graph, Algorithm, Iteration, Pheromone, Cities, Value, Route, Memory, Number, Ants, Travel Agents, Step |

|  |  |
| --- | --- |
| Travelling salesman on Greedy algorithm: | Travelling salesman, Traveller, List, Tour, Graph, City, Cities, Cycle, Point |

|  |  |
| --- | --- |
| Data Source: | Instance, Co-ordinates, Data set, City |

|  |  |
| --- | --- |
| User Interface: | Graphical User Interface(GUI), Travelling salesman, Tour, Position, User, Algorithm, Iteration, Route, Process, Number |

|  |  |
| --- | --- |
| Travelling Salesman using Elitist algorithm: | Travelling salesman, Iteration, Tour, Length, City, Cities, Pheromone, Path, Memory, Stage |

The following nouns that are not part of the system under software control:

Traveller, User, Travelling Salesman

Nouns that lack attributes and methods associated to a class are:

Cost, Co-ordinates, Pheromone, Solution, Route, Length, Memory, Path, City List, Distance, Graph, Iteration

Nouns that describe vague names:

Sequence, Value, Number, Instance, Stage, Step, Cycle, Process, List

Stereotypical Classes:

Boundary: Graphical User interface

Entity: Distance

Control: Algorithm

Nouns left:

Ant, City, Cities

Comments:

1. Cost, Co-ordinates, Pheromone, Solution, Route, Length, Memory, Tour, Path, Distance, City List with very few attributes and methods.
2. Distance can be a helper method to calculate the distance between the cities.
3. Sequence, Value, Number, Instance, Stage, Step, List, Cycle and Process are vague names.

The following are identified as a minimal set of classes for TSP:

City, Ant, ACO, GUI

|  |  |
| --- | --- |
| Conclusion: | Control class Algorithm refers the general name of an algorithm. It is appropriate to be more specific about the algorithm name as each of them has different implementation to finding the optimal tour. The class City and Cities can be combined to make a single class called City. |

The revised set of classes at this stage:

City, GUI, Ant, ACO

**Class Diagram (Stereotype):**



## 2.5 CRC Cards:

CRC cards explain the responsibilities and collaborators of each class in detail.

**Table 2 Class Responsibility Collaboration Cards**

|  |  |
| --- | --- |
| **Class: City** |  |
| **Responsibilities** | **Collaborators** |
| The City class reads and stores the values of X and Y co-ordinates of city from city list xml file. The City class works in conjunction with graphical user interface. Class to plot the location of cities in a graph to implement the TSP tour. The City class calculates the distance between adjacent cities with the use of city coordinates and finds the closest cities. | GUI |

|  |  |
| --- | --- |
| **Class: GUI** |  |
| **Responsibilities** | **Collaborators** |
| The GUI class displays graphically the constructed tours, optimal tour and optimal tour length with respect to the user inputs. The class contains GUI features that build the front end for the user to interact with the application. Initially, the class GUI reads the city list from xml file and the city locations are plotted in the GUI. Arc starts connecting cities displayed when selected algorithm builds the tour in GUI. The GUI must allow user to: (1) select and load the city list file, (2) select an algorithm to run and build the tour. The GUI class displays the tour information as algorithm progresses for specified number of iterations. The GUI class interacts with algorithm classes such as Ant Colony Optimization (ACO and Greedy to implement on TSP tour to display the optimal tour and optimal tour length. | City, ACO, |

|  |  |
| --- | --- |
| **Class: AntColonyOptimization(ACO)** |  |
| **Responsibilities** | **Collaborators** |
| The ACO class implements the ACO algorithm on TSP tour. The ACO class initialises the pheromone trails and algorithm parameters. The ACO Class initialises some number of ants to build the tour in each iteration, selects next city in a tour based on probabilistic decision rule which is: i) pheromone trail and heuristic value using nearest neighbourhood strategy and ii) a private memory that stores the past information about the paths it visited once. The memory will be updated in each step of the tour to compute the tour. Pheromone update and pheromone evaporation will be performed after completing each iteration. Ants will be updated for each trail. Each iteration provides best tour and updates the best so far tour comparing the current best tour with the best so far found tour for every iteration. After completing the tour, it returns the best optimal tour and tour length. The ACO class interacts with GUI class to implement the algorithm graphically and display the output as optimal tour and optimal tour length in GUI. The ACO class works in conjunction with Ant class in setting the tour, tour length, number of nodes, visited cities. | GUI, Ant |

|  |  |
| --- | --- |
| **Class: Ant** |  |
| **Responsibilities** | **Collaborators** |
| The Ant class helps ACO class in every step of the tour to set up the tour, tour length, visited cities and number of cities in a tour for each iteration. | ACO |

## 2.6 Interaction Diagram

Using the Class responsibilities and collaborators in CRC Cards, the following simple collaboration and sequence diagram are drawn.

### 2.6.1 Collaboration diagram



### 2.6.2 Sequence diagram



## 2.7 Statechart diagram:



# Analysis Model

An analysis model provides the technical specification of the model that is used to implement the design. Revisiting the use case view to comment any additional requirements with respect to the application specification is necessary.

## 3.1 Class Identification:

Revisit the use case class identification part to ensure all necessary classes have been identified.

Comments

The main program is always needed for any GUI class to execute the application.

XML file is used to read the city coordinates.

## 3.2 Class Diagram:



## 3.3 Attribute identification:

The attributes are identified from class responsibilities. Most of those listed are explicitly mentioned in CRC and some are added for general understanding.

City: Location, Distance

ACO: Distance, Pheromone, ChoiceInfo, NoAnts, NoNodes, Memory,

Algorithm Parameters, Iteration, BestSoFarTour

Ant: NoNodes, TourLength, Tour, Visited

GUI: NoAnts, NoNodes, Algorithm Parameters, Iteration

## 3.4 Interaction Diagrams:

### 3.4.1 Collaboration diagram:



### 3.4.2 Sequence diagram:



## 3.5 Statechart diagram:



## 3.6 Identification of Associations:

The association between classes can be identified using Collaboration diagrams, sequence diagrams and attribute identification. Identification of associations helps to identify the relationship between classes.

In the travelling salesman problem, the associations identified as follows

1. GUI uses City, ACO
2. ACO uses Ant

Identification of Messages:

initAnts, getAnt, getDiatance, setDistance, getHeuristic, setHeuristic, initPheromone, getPheromone,setPheromone,depositPheromone,computePheromone,updatePheromones,constructSolutions,computeHeuristic,globalPheromoneDeposit,UpdateBestSoFarTour,EvaporationRate,initData,PheromoneUpdate,decisionRule,ComputeTourLength,getVisted, setVisited, Ant, getTour, setTour, drawBestTour, updateCityCount, DrawCityList, drawGraphPlain, drawGraphPheromone, openCityListBUtton\_Click, Step, InitializeComponent, CityTourDiagram, Label\_Click, Stop\_Click, Step\_Click, ResetClick, StartClick, NewButton\_Click, InitButton\_Click, CheckBox, SelectFileButton\_Click, OpenCityList, FindClosesetCities, getIntegerDistance

## 3.7 Non-Functional Requirements:

Multi-platform usability, Memory allocation, Power supply constraints for computer systems

## 3.8 Packages:

**Package Diagram**



# Design Model:

## 4.1 Revisit Use-Case description:

The Use-Case description is not complete to carry out the design process because it gives the generalised procedure of ACO and failed to mention the importance of algorithm parameters set. The algorithm parameters such as alfa, beta controls the behaviour of ACO. If parameter α = 0, the ant will select the closest city to visit according to nearest neighbourhood heuristic. If parameters α > 1, β = 0, only pheromone trail is used to find the next city without the heuristic value will lead to stagnation and poor results as ants will follow the same route to construct the tour. If β > 1, this will yield a better solution in finding the next city. So it might be appropriate to clearly set those parameters in the design stage.

Each use case realises the behaviour of the system. The GUI use-case realises interaction between users and various internal subsystems. The City use-case realises the city coordinates and distance between the cities. The AntSys use-case realises the necessary algorithm to find optimal tour length.

## 4.2 Refine Class diagram:

Class diagram is refined to show all methods and attributes. All methods and attribute visibility are identified. Also methods passing the arguments and return data types are identified. Class Main is used in the class diagram to run the GUI process. Also a thread is used to run the application.



## 4.3 Behavioural description of the interaction diagram:

Detailed sequence diagram with added qualifiers are drawn as follows.





## 4.4 Object-Object and Object-Subsystems interactions:

In TSP, an object communicates with other objects to execute an application. An object GUI interacts with the object City to create a base platform for executing the TSP. The objects City and GUI identify the relationship between them. object GUI also interact with object Antsys to implement the application. As a result of this interaction, Antsys communicates with Ant object to start the implementation process.

The three subsytems of TSP application is TSPCity, TSPSolver, TSPGUI understands the overall system behaviour. The object in each subsystem interacts with other subsystems to execute the application. For instance The object GUI interacts with the TSPCity subsystem to carry out the activity of plotting the city layout. Each object in subsystem is aware of the relationships exists between them.

## 4.5 Design Subsystems:

Designing subsystems involves subsystem specification and subsystem realization. The subsystem TSPGUI collaborates with other model elements of the system such as TSPSolver and TSPCity to show the output graphically. TSPGUI have their model elements that realises the interfaces and behaviour of outputting the graph in GUI screen.

## 4.6 Identify interfaces:

A design class GUI provides a graphical user interface between sub systems and user.



## 4.7 Realise Non-functional Requirements:

In this exercise following are identified as non functional requirements.

Multi-platform usability, Memory allocation, Power supply constraints for computer systems

The software package restricted to use in windows platform mostly but there are many operating systems available in the market like Mac, Linux etc.

Developing codes and back up is user friendly therefore reusability of completed codes is much simpler than other packages.

## 4.8 Reconsider the attributes and associations:

Covered earlier sections

## 4.9 Statechart:

Statechart is complete for design model and will still applicable for design model.

## 4.10 Method design:

Initialise Data

Begin Loop

Distance : = 0

Pheromone := 0

ChoiceInformation := 0

End Loop

ComputeHeuristics ()

Begin Loop

If(Distance > 0)

Then

HeristicsValue = 1/Distance

End

Choiceinformation = (Pheromone)Alfa x( HeuristicValue)beta

EvaporationRate

Begin Loop

Pheromone = (1-ro)\*Pheromone

Compute Tour Length

Begin Loop

Length = Length+Distance between cities

Return Length

End

Psudeo code:

Begin procedure Antsystem

Initialize algorithm parameters

Initialize pheromone values

Initialize distance, choiceinformation

For iteration equal to maximum iteration do

Construct solution

For k = 1 equal to no of Nodes do

Initialize ants for tour

Each ant start at Random City

Apply DecisionRule

Chose next City

Store visited cities and length of the tour in memory

Deposit pheromone on the route visited

Compute tourlength

Update bestsofartour

If (current tour< bestsofartour)

store current tour

Evaporate Pheromone

Update Pheromone

End For

End For

Return Best Tour

End Begin

For implementation purpose and for users clear identification, we prefix the class with name TSP\_Px ( where x is project number)

# Program Testing and Results:

## 5.1 Testing method / flow chart:



## 5.2 Test Results:

**Constants:**

Alfa = 1

Beta = 7

Evaporation Rate = 0.5

**Method:**

Ant System with Greedy optimization

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | No of Cities | Max Iteration | No of Ants | Initial Tour Length | Optimal Tour Length by ACO |
| Test 1 | 40 | 100 | 40 | 2696 | 2197 |
| Test 2 | 40 | 1000 | 40 | 2606 | 2172 |
| Test 3 | 70 | 3000 | 70 | 3815 | 3731 |
| Test 4 | 70 | 1000 | 70 | 3815 | 3731 |

|  |  |
| --- | --- |
| **Test 1** | **Test 2** |
| **Test 3** | **Test 4** |

# Discussions & Conclusions:

The implementation was tested by various data sets and performance analysis was undertaken. In addition to these, variance analysis was done in order to identify the effect values of the parameters on the system.

The key part of this assignment work is to apply ACO concept to define and work out optimal tour length for travelling salesman. This project work done in line with provided specification.

The specification was thoroughly analysed and following core concept have been designed and completed.

1. Use case model
2. Analysis model
3. Design model

These models have been applied to in order to determine the best optimum route length. The main concept of Ant System has been implemented as part of this project along with Greedy Algorithm as an improvement.

Other improvement methods could be included in this Algorithm on separate classes. The different algorithm could be linked to main windows form using interface class. Selection of various improvement techniques could be added into the GUI. Users can select any one of the algorithm and associated parameters to compute optimal tour length.

During this project the no of cities selected with minimum numbers to test the basic Ant System Algorithm. By using various improved methods, the no of cities involving this exercise could be increased.

As you can refer in section 5.2, although the results are satisfactory there are several ways to improve the results such as

1. Ant Colony System ( which could include local optimization heuristics)
2. Min Max Ant system
3. Elitist Ant System
4. Non-probabilistic transition.
5. Rank order algorithm

# References:

Marco Dorigo (2012) *Ant Colony Optimization,*Available at: *http://www.aco-metaheuristic.org/* (Accessed: 21st November 2014

Marco Dorigo, Luca Maria Gambardella (1997) **'Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem'**, *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,*1(1), pp. 53-54

Marco Dorigo and Thomas Stützle (2004) ***Ant Colony Optimization***; A Bradford Book The MIT Press Cambridge, Massachusetts London, England. 67-76

Paul J. Deitel, Harvey M. Deitel (2009) ***Visual C# 2008: How to Program***, : Pearson Education, Limited.

Perdita Stevens (2006) ***Using UML: Software Engineering with Objects and Components*** *(2nd Edition) (Addison-Wesley Object Technology)*,: Addison Wesley.