6CCS3PRJ Third Year Project

Visualisation of Travelling Salesman Problem

SeongHwan Kwak
1152707

Department of Informatics
King's College, London

Supervised by Prof Tomasz Radzik

Seonghwan.kwak@kcl.ac.uk

April 22, 2014

Originality avowal: I , Seong Hwan Kwak, verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words. (20/4/2014)

Abstract

The Travelling Sales Problem has been one of the most popular issues in both computer science and mathematics. It is not simply just because of a practical benefit from it's research; it is profoundly relevant to the P and NP problem that is validation of one question 'is P equal to NP' or the other. If the answer is yes, a NP problem can be solvable in polynomial time and once one NP problem is solved in polynomial time every other NP problem is solved in polynomial time because it is what the formal definition of NP-completeness indicates. Complexity theory is an area of study that deals with such problems. P and NP problem, the TSP or a study relevant to solving any NP problem in polynomial time has a huge implication various fields of study as complexity theory is quoted to many fields of study. Solving TSP is considered to be a typical way of validating such proof.

In this project, five algorithms have been implemented in order to yield optimal solutions for the Travelling Salesman Problem for random cities. The five algorithms are the following, Nearest algorithm, Insertion algorithm, 2-opt, Simulated Annealing and Genetic Algorithm. I visualise optimal solutions on Java Swing GUI and demonstrate intuitively to children. I will discuss in this report the efficiency of those algorithms and how I optimised them in the implementation.

Acknowledgements

I like to thank to Prof Tomasz Radzik for supervising my work. His knowledge and professionalism in his study field was the most adequate choice for the TSP project and I am thankful for his help.

Contents

1 Introduction

1.1 The definition of the problem

The Travelling Salesman problem is to find the shortest path for travelling to every city or node without visiting one place more than once, returning to the starting point. [Schrijver(2005), pp.38] Each edge has a different distance so it's a weighted graph, G=(V, E;w). G stands for Graph, V means the set of vertices and E is an edge and w means the edge is weighted, in another word, the graph has a vertex(or ces) and weighted edge(s). The whole path is closed therefore it is called a cycle and it does not visit a node already visited so that it is a Hamiltonian cycle.

The table below was produced by Van Rooji and the data was collected from 7-year-old, 12-year-old elementary school students and university students. The percentage denotes how each individual's work is far from the optimal solution.

Average tour quality found by participants. van Rooij et al. [485].

Number of Cities	7-Year-Olds	12-Year-Olds	Adult
5	3.8%	2.5%	1.7%
10	5.8%	3.4%	1.7%
15	9.4%	5.0%	2.7%

Table 1: cited from [David L. Applegate(2006), pp.35] and originally cited from [van Rooji I(2003)]

As described on Table 1, for a few nodes, it is a fairly easy problem since there are a limited number of routes. However as the number of nodes increases, the possible number of routes increases indefinitely. An optimal solution is hardly unlikely found as it is a NP-hard and NP-Complete problem. [Schrijver(2005), pp.38] This has been a popular and practical subject by mathematicians and computer scientists as it helps find an optimal solution for many practical issues, logistics and drilling holes into electrical circuits. There are ten cities and the possible routes are 10! = 3628800. If there are 100 holes on an electrical circuit there are 100! possible ways. Finding an optimal solution is significant cost effective.

1.2 The project : Visualisation of Travelling-Salesman Heuristics

Algorithms have been sought for long period of time. There are algorithms for absolute and heuristic solutions. Finding a guaranteed solution requires massive computation power for many nodes but heuristic solutions use much less computation resource, thus finding the best travelling cost by a heuristic algorithm is a good alternative to the absolute. [Schrijver(2005), pp.41]

Apart from solving the problem, this project has an educational purpose. An educational system needs to be created in order to present the problem solving visually to students. This also gave me a positive impression when I decided to do this project. The software should show the way visually and intuitively for educational purposes so that students can understand and gain an insight into each heuristic algorithm. The program should provide time-frame UI so that user can traverse nodes on the track of each heuristic method by scrolling or moving a bar on GUI.

2 Background

2.1 Practical Benefits

The TSP problem is finding the shortest path from one place to another. Think this as an abstract model and it has many applications and the TSP study benefited extraordinarily various subjects. Not only the travelling salesmen can visit places faster, salesmen could be replaced to cars or individuals. It can benefits logistics and travellers. Dr Willian Cook talks in his book several TSP applications which include logistics.

1 Software firms offer programs that use TSP heuristics to schools and bus routing scheduling efficiently. Postal deliveries is a multi-salesman problem, meaning that there are multiple entities moving around a given space and it becomes more complex to schedule. Dr Cook stated 'a component of the solution procedure is to divide the customers among the individual tours'. [David L. Applegate(2006), pp. 59]

2 Multi-salesman problem was furthermore discussed in the book with an example of visiting elderly people and carrying to caring them to a facility, 'Meal on Wheels' program was used in Atlanta, Georgia to resolve the problem. There are 200 places to visit and each driver is allocated for 30-40 locations. The solution is to 'divide the tour into segments of the appropriate lengths'. [David L. Applegate(2006), pp. 59]

3 The Genome project is one of the remarkable events in human history. Every human has DNA in each cell and DNA is a gene information with pairs of four nucleobases (G,A,T and C). There are 3 billion base pairs. The Genome project is to find the every pairs according to the genome sequence and map them. How The TSP helps this research is to 'estimate distance between adjacent markers' and to have 'accurate information on the order in which the markers appear on the genome.' [David L. Applegate(2006)]

4 Producing an integrated circuit is an extremely complex task and testing is essential in order to prevent faults. Scan chains method is used to test components. Ordering scan points and chain all electronic components on the board and send a special signal and tests if it works properly. Shorter chain means minimising valuable wiring space and also it reduces time to chain the components. [David L. Applegate(2006), pp. 67]

5 Every electronic device has circuit boards inside. Circuit board is essential in order to produce an electronic device. When the board is produced it should be drilled in order to mount components and integrated chip-sets. Drilling process is automated and moving between locations and making a hole is a TSP. There is interesting point suggested by Lin and Kernighan cited by Dr Cook. [Lin S(), pp. 498-516] [David L. Applegate(2006), pp. 69]

> ..but if drilling time outweights travel time, there is no particular advantage to any optimization...

Also Dr Cook mentioned, [David L. Applegate(2006), pp. 69]

> ..Although positioning time is just one component of the drilling process, the use of TSP algorithm in these studies led to improvements of approximately 10 percent in the overall throughput of the production time.

6 Observing distant stars or satellites is not an easy task. Telescope slewing takes time and moving one position to another is the TSP. Telescopes have computer driven motors and heuristics used to have an optimal slewing schedules. [David L. Applegate(2006), pp. 75] For example, Dr Cook's book introduces the Australia Telescope Array and mentioned The ATMOS program. Fortran language and simulated annealing heuristics used in the ATMOS program. [Array(2005)] Not only telescopes set on the earth, space based telescopes necessarily needs an optimal solution of the TSP as they have a limited amount of electricity generated by solar panels in space. The SIM PlanetQuest that was a cancelled space telescope project, the authoritative white paper mentioned about the telescope's slewing process, estimating that '12 percent of the flight time of the mission will be devoted to spacecraft slewing'. This proves the TSP is holding a key to successful response to research demands. [Array(2005)] [Edberg(2005)]

7 Distributed systems like Cloud, Super computer, Grid computing do data clustering. They collect data and categorise in order to analyse them. The TSP interestingly has its role in data clustering. A system can grab a data from a sort of information. For example, a user is registered to a system and provides the user's personal information. The system now then is able to estimate the user's salary from the address information. User's address information is linked to a third party property database and the system can estimate the user's salary. Dr Cook cited from ACM Computing surveys and introduces a technique. S(a, b), S is similarity between pairs. 'a' and 'b' are data points. s(user's name, salary value) is a representation of the previous example. [David L. Applegate(2006), pp. 77] [Jain A. K.(1999), pp. 264-323]

2.2 P and NP problems

The problem was first introduced by Stephen Cook in his paper 'The complexity of theorem providing procedures'. [**?**, pp. 151-158] It was informally discussed in Godel Kurt's letter sent to John Von Neumann in 1956. The mathematician Godel didn't mention the terms P and NP but his statements were directly pointing to the fundamentals of P versus NP problem, the following was stated in the letter.

> The question is how fast phi(n) grows for an optimal machine.
> ... the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search..

[Kurt(1956)] [Sipser(1992), pp. 603-618]

There are several things to mention before proceeding to the P and NP problem. Problem can be categorised into two groups, decision and function problems. A decision problem deals with cases with Boolean answers. A function problem has multiple possible answers. For example, deciding whether or not number N is a prime number is a decision problem, how many prime numbers N number has is a function problem. Many function problems include a set of decision problems and function problems can be converted to decision problem.

Problems are solved by an algorithm and there are two kinds of algorithms, Deterministic and non-deterministic algorithms. A deterministic algorithm deals with fixed cases so that there is only one answer for one condition. A non-deterministic algorithm has multiple states from one state and may not produce one unique answer from one input.

P class problems are solved by a deterministic algorithm in polynomial time. NP problems are solved by a non-deterministic algorithm in polynomial time. In other words, the solutions can be verified by deterministic machine in polynomial time. Although there tends to be a very large number of solutions, finding an optimal solution is not an easy subject. [Cook(2000)]

The 'NP hard' term is referenced to problems as hard as the hardest problems in NP but noting that is not inclusive to NP category, much challenging problems. For example, there could be an optimal solution but there is a way known to verify that this is the best answer. The TSP problem is therefore catalogued as NP hard class problem. NP complete problems are both NP and NP hard which are reducible from NP problems in polynomial time. [Cook(1971), pp. 151–158] [Cook(2000)]

A non-deterministic algorithm has more than one state from another state, the maximum number of cases in each state is defined as 'degree of non-determinism'. Deterministic algorithm has just value '1' for degree of non-determinism therefore NP problem is including a set of P problem. In another words, every P class problem belongs to NP class. The P and NP

problem is questing if the reverse is also valid; proving that if every NP problem could be also P problem. [Cook(2000)]

2.2.1 In another word : by Michael Sipser

> NP problems are to check answers easily, P problems are to solve problems easily. Are easily checkable problems the same as solvable problems?. Easily checkable problems done without searching. So What P versus NP problem means any problem without searching you can solve it without searching. search problems be solved easily without searching. Searching is always eliminated if P=NP. If P is not equal to NP, sometimes searching is necessary.

[Sipser(2011), 20:46] [Sipser(1996), pp.257] The TSP is a search problem. There is no formula giving an optimal solution without searching for every NP-complete problem.

2.3 NP-Completeness

The gap between P and NP problems is not something that can be representable easily or understandable. Mathematicians have hypothetically suggested an idea, called NP-Completeness. A problem is defined to be NP-complete if it is a NP problem that is a reducible form of every other NP-problem in polynomial time. Let there be problem A and B and there is a function f(x). Let's assume that substituting problem A to f(x) shows a same result of what problem B computes in polynomial time, then it is polynomial-time reduction.

If there is a deterministic algorithm that solves one of NP-complete problems in polynomial time, every NP-problem can be solved. This is why NP-complete problems are so frequently discussed exceptionally by mathematicians and computer scientists not only because of its practical usefulness. The Travelling Salesman Problem, Hamiltonian cycle, protein structure prediction and the Satisfiability problems are typical NP-complete problems. [Cook(2000)]

2.4 NP-hard problems

The travelling salesman problem is also a NP-hard problem because an optimal solution can not be verified to be the best path in polynomial time as every path needs to be found in order to be compared.

2.5 The P Versus NP problem

The implication of this problem is fundamental. It will first bring a huge impact on mathematics and the essential part of mathematics will be rede-

fined. This will cause chain reactions to every subject profoundly. Computer science is one of them. and the frame of computer science is also going to be rewritten. [Cook(2000)]

If P = NP, a NP-complete problem is solvable by deterministic algorithm in polynomial time, it implies every NP problem is solvable by solving one. The contemporary cryptography technology, public key, PGP and cryptographic hash function is based on computational hardness assumptions and decoding it is a NP or NP-complete problem. If P = NP, computational hardness is not reliable and the contemporary cryptography will be likely compromised. [Cook(2000)] Banks, governments and telecommunication systems probably need fundamental change. Nevertheless, solving P problem in polynomial time doesn't mean always producing a result in five minutes or one hour or perhaps couple of days. How long would it take if 99999 is substituted into 'n' in the following equation '$99999 * n^9 9999$'. Not only the negatives, logistics, car assemblers and circuit makers can be benefited significantly and are able to produce with much less costs.

If P is not NP, it is proven that there is impossibility to solve some problem in polynomial time. This might not bring a huge change as much as the P=NP case but mathematicians can provide theoretical proof that the contemporary encryption system is reliable. Computer scientists and mathematicians have a strong foundation to advance to next stage. [Cook(2000)]

2.6 Similar NP-complete Problems

Hamiltonian cycle is a path directed or undirected visiting each node exactly once. The Hamiltonian Cycle problem is a decision problem, to know whether there is a Hamiltonian cycle when a graph is given. This is a NP-complete problem. [Christos H. Papadimitriou(1998), pp.383] The TSP is to find the shortest Hamiltonian path.

Eulerian path is a path visiting every edge exactly once and finding an eulerian path in a graph is not a NP problem. It can be found in O(V+E) time that is polynomial time.

Clique is an web of every node connected to each other, each node in a graph is connected to every other nodes. For example, social media websites such as Linkedin and Facebook show mutual relationships, this is a clique problem. The clique decision problem is to know whether there is a clique in a graph larger than a given size. This takes exponential time and this is a NP-complete problem apart from planner graph. [Christos H. Papadimitriou(1998), pp.360]

The Knapsack problem and the bin packing problem converted into a form of decision problem is also a NP-complete problem.

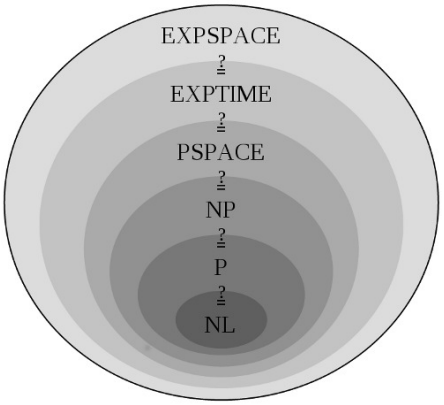


Figure 1: Complexity Classes [Qef(2012)]

2.7 Computational Complexity theory

Complexity theory is a field of study regarding how related each computational problem are and categorising computable problems into different subcategories depends on hardness of their solvability. It is required how much resources available in order to solve a problem, 'time' and 'space' are the main types of resources in this area. 'Time' is how much it is taken to solve a problem and 'space' is how much memory is required to solve a problem. This includes the study of P and NP problem. The complexity theory introduces few other terms, EXP and R. EXP problems are solvable in exponential time and R is solvable in finite time. R stands for Recursive Languages and the class of decision problem that is solvable in a Turing machine. A set of R problems is equal to the set of all recursive problems and in other words, all computable functions. For example, n X n chess problem is solvable in exponential time that does not belong to P problem. [Scott Aaronson(2014)] The most problems are mathematically uncomputable and these problems are non-deterministic.

2.8 Combinatorial Optimisation Problem

There are two kinds of optimisation problems, one is with continuous variables and the other is with discrete variables. The one with discrete variables is called combinatorial optimisation. An object is to be found from a finite or an infinite set. [Christos H. Papadimitriou(1998), pp.3] Every leaf is a solution and a least cost goal should be found. The maximum depth is equal to the number of variables in a search problem. The TSP and max-CSP are typical examples of combinatorial optimisation. [Bertrand Neveu(2003), pp.910]

2.9 Algorithms for the TSP : a brief introduction

There are two kinds of algorithms as said earlier, one is absolute and the other one is heuristic. Heuristic algorithms have two types; one is tour constructive and tour improvement heuristics. Tour constructive heuristic is to find one Hamiltonian cycle path and it doesn't require a further step once a cycle found so that a tour constructive heuristic usually requires less time than tour improvement heuristics but it's less optimal [1 M Gendreau(1992), pp. 1086-1094]. A solution found by a tour constructive heuristic could be used as an initial solution for a tour improvement heuristic [B Golden and Stewart(1980), pp. 694-711]. Typical examples of Tour constructive heuristics are Nearest Neighbour Search and Christfides. K-opt and Lin-Kernighan are examples of Tour improvement heuristics. A heuristic doesn't give a solution to other problems; modifications need to be made in each case. Finding an optimal solution, some become not progressive by getting into local optima. Meta heuristics were suggested in order to deal with the issues. It's a general direction or a master strategy to solve a set of problems. Well known meta heuristics are Simulated Annealing and Ant Colony optimisation [Gambardella(2009), pp. 239-287].

Heuristic methods for TSP are the following, Simulated Annealing, Genetic Algorithms, tabu search, '2-opt and 3-opt' search [2 David S. Johnson(1995), pp. 1] [Mostafa Rahimi Azghadi(2008), pp. 11], Nearest Neighbour, Convex Hall [Iris Van Rooij(2003), 215-220] [2 Justin Chester()] and Christophides heuristics [2 David S. Johnson(1995), pp.9] [Lin(2005)]. There are also other combinational [2 David S. Johnson(1995), pp. 36] and nature-inspired meta-heuristic methods. [Yang(2010), pp. 127]

With Simulated Annealing, for example, assume there is a ball at the peak or a fairly high position on a mountain, finding an optimal solution is getting it to the ground position. The ball can get stuck at local minima position but if there is a little shake the ball could be moved by this to a lower position eventually. An optimal solution for a NP problem could be gained by giving a lot of shake, beginning to randomising at initial states, using lesser randomisation at later states until it reaches zero randomisation. [2 Dimitris Bertsimas(1993), pp. 10-15] Simulated Annealing, the name comes from Metallurgy and it indicates a way to gain a crystal from molten metal without a defect. In optimisation, the idea is adapted, the process mentioned above is another way to describe SA. It borrows terms from metallurgy, temperature, schedule and Halt. The randomisation mentioned above is named as Acceptance Probability and formulated as $exp(E/T)$. exp is e^x, x is (E/T). E denotes energy and T denotes temperature. Allowing worse solutions in a variable probability of $exp(E/T)$. Using exponential function and (E/T) is based on Metropolis-Hastings Algorithm.

'Genetic Algorithms were invented by John Holland and developed by him and his students in the 1960s at the University of Michigan' In Melanie

Mitchell's book 'An Introduction to Genetic Algorithms'. [Melanie(1998), pp. 7] Genetic Algorithms are said to be about finding an optimal solution by simulating adaptive solutions for a problem and finding the best solution among them. Traditional optimising algorithms differentiate functions and explorer solutions. Genetic Algorithms are different, they use selection, crossover and other operations. These don't require special mathematical operations and parallel exploration. However they are inefficient during iteration and variable conditions are limited [Melanie(1998), pp. 7]

Tabu search makes a list of recently visited nodes and those nodes are not selected as neighbours, in other words, it changes the structure of neighbourhood every-time it visits a node. [Glover(), pp. 74-94]

'Branch and bound' and 'dynamic programming' on the other hand are typical algorithms which always guarantee the best results. [Clausen(1999), pp. 2] [3 Sanjoy Dasgupta(2006), pp. 188] Hence Clausen stated in his thesis on the B and B that "Branch and Bound is by far the most widely used tool for solving large scale NP-hard combinatorial optimization problems". [Clausen(1999), pp. 2] Branch and Bound first makes range assumption of optimal answers and it prunes answers outside of this range. Repeating the process until it reaches every node. [Clausen(1999), pp. 2] Dynamic programming is like 'divide and conquer' method. It stores calculated values in a table for a divided problem and if other problems need to repeat the previous process it just loads the stored values [3 Sanjoy Dasgupta(2006), pp. 188].

Tabu search, branch and bound, dynamic programming, Christofedes and Lin-kerninghan are not going to be used for this project. I am going to demonstrate heuristic algorithms, Nearest algorithm, Insertion algorithm, 2-opt, Simulated Annealing and Genetic Algorithm. I will discuss its efficiency and optimisation.

3 Requirements

3.1 A TSP example : A website providing a JSP applet

The above website providing a JSP applet for the TSP problem. This is a program which satisfies basics of the requirements and specifications which I need to meet. Although The Graphical User Interface is not quite the same the prototype below it gave me a good idea what result I actually need to produce.

3.2 System Requirements

1. A user needs to be able to interact with the program graphically and the program is to be made for students so that an intuitive design is necessary.
1.1 Graphical User Interface takes user's input and showing a result on it,

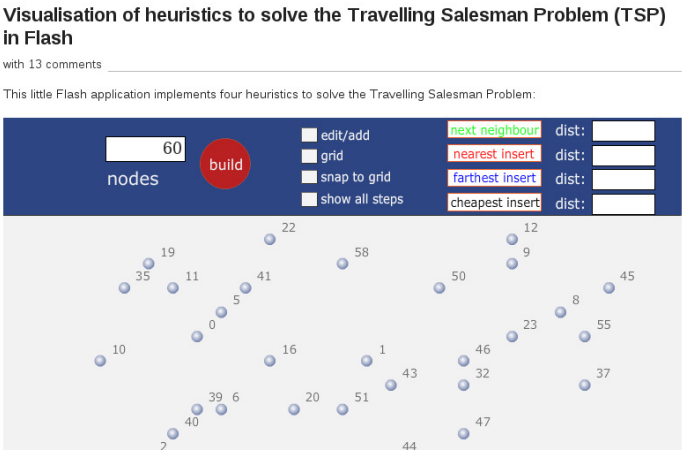


Figure 2: http://bjornson.inhb.de/?p=26

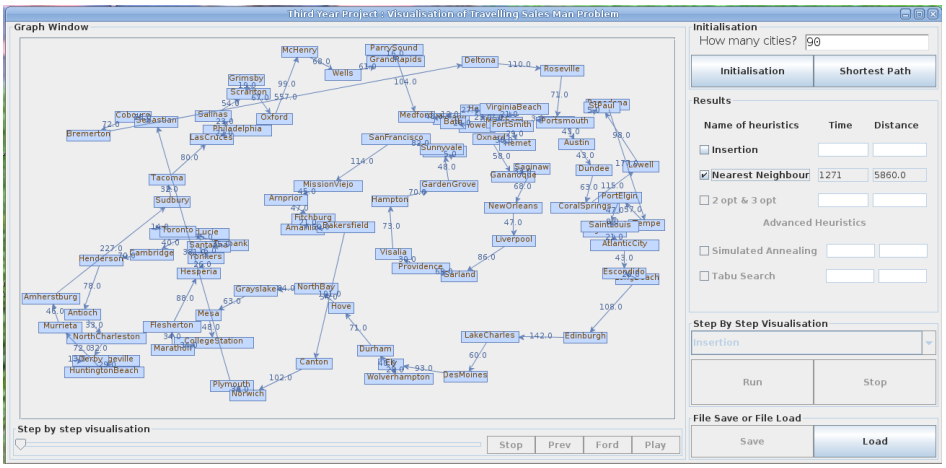


Figure 3: Prototype GUI and a generated sample path

for example, A user types a number of cities on GUI he or she wants to create and clicks a button. The program starts running.
2. Optimal paths need to be found by each heuristic. 2.1. Optimal solutions may be found by one heuristic method or several and to be improved or compared with solutions found by other methods if there is any. This implies there is not just one optimal solution. One heuristic method shows a competitive result particular situation but may be less effective in other circumstances. 2.2 For user's convenience, the program will try several algorithms at once if possible and select a best result according to time taken or distance.
3. Each path found by heuristics needs to be represented graphically. 3.1 Graph can be represented mathematically with symbols and it is not intu-

itive to students whom don't have knowledge to comprehend notations and it's meaning. Also numbers of nodes that students will create are likely more than ten or twenty. Three hundred will be set as a limit.
4. A user can go to previous or future states. 4.1. In order to make the design more intuitive, the program is to provide an interface that the user can see more clearly and directly paths starting from where and ending where.
5. The program may not require more than 10 seconds in order to find optimal paths. 5.1. The program is built for an educational purpose for students and they want to interact with the software as often as possible. Time that a user waits to see a result should be minimised in order for better interaction. 5.2. So that Students may learn better. 6. For insertion, Nearest Neighbour and 2-opt algorithms, a user can expect an answer within a short period of time and the program will indicate which algorithm provides the best answer once 'Shortest Path' button is clicked.

3.3 User Requirements

1. The software doesn't contain any nudity, violence and harsh language therefore no age limit is required.
2. A user should be able to understand algebraic notations and basic instruction and able to give basic commands.
3. A user should understand the purpose of the program.
4. A user should understand what advanced heuristic means and aware that it needs time to compute an answer and user should be patient until an answer is given by iteration process.

4 Specification

4.1 Information on Standard development environment

The development platform is Sun Java JDK (1.6) and thus will be operational on any operating system, Windows, Mac and Linux are typical choices but Sun Java installation is a prerequisite. I was using a third party library called JGraph for visualising graphs and it provides so many APIs and it required me to look in depth so I decided to use another library. mxGraph seemed to provide concise APIs for handling objects and representing Graph and I am delighted that I found it. I am building a Graphical User Interface with the help of JFormDesigner provided by FormDev and event handling is done manually.

4.2 Each Algorithm runs in separate threads

There are two reasons for using multi-threading programming.
1. Running the algorithms mentioned above requires quite burdensome com-

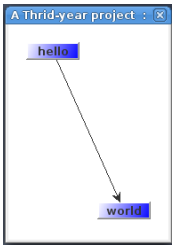


Figure 4: Jgraph example

putational resources. Using threads maximises CPU resource utilisation and enable a program to finish a task within a shorter period of time.
2. While more than one algorithm running additional algorithm can be executed by a user. A user can run multiple instances at the same time.

4.3 How is the direct distance between two given cities determined?

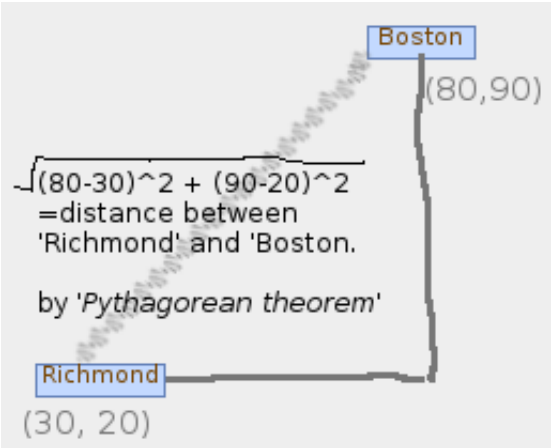


Figure 5: Finding distance

Although The GUI doesn't show Cartesian information, each node has x and y coordinate value. This information will be generated randomly when nodes become initialised. In other words, I am following the Cartesian coordinate system. For example, Richmond has (30, 20) coordinate value and 30 is x and 20 is y coordinate values. Boston has (80, 90). Virtual Richmond and Boston cities from the diagram vertically 70 distance apart (90 - 20) and horizontally 50 distance apart (80-30). Once the vertical and horizontal information is known, direct distance can be found by Pythagorean Theorem. Dr Sally states in his book 'Roots to research: a vertical development of mathematical problems' that

'For ABC is a triangle with sides of length a, b and c, then x =a, y=b and z=c is a solution of the algebraic equation.

$$x^2 + y^2 = z^2.$$

, if and only if triangle ABC is a right-angle triangle with legs of length a, b and hypotenuse of length c. The algebraic equation

$$x^2 + y^2 = z^2.$$

is often called the Pythagorean equation'. [Sally(2007), pp. 63].

4.4 Graph, node and edge

There is a direction for a path so it is a directed graph. In order to make a Hamiltonian cycle, each node must have two edges and there should be only one edge between two nodes. An edge has source node and target node. Each node is stored in 'graphNodeArray' object array in 'GraphGenerator' class, in 'createGraph' method and Array's index is acting as ID for each node . Array data structure's time complexity is the follow. Array has the

	Linked list	Array	Dynamic array	Balanced tree	Random access list
Indexing	Θ(*n*)	Θ(1)	Θ(1)	Θ(log n)	Θ(log n)
Insert/delete at beginning	Θ(1)	N/A	Θ(*n*)	Θ(log n)	Θ(1)
Insert/delete at end	Θ(*n*) last element is unknown Θ(1) last element is known	N/A	Θ(1) amortized	Θ(log *n*)	Θ(log *n*) updating
Insert/delete in middle	search time + Θ(1)[10][11][12]	N/A	Θ(*n*)	Θ(log *n*)	Θ(log *n*) updating
Wasted space (average)	Θ(*n*)	0	Θ(*n*)[13]	Θ(*n*)	Θ(*n*)

Figure 6: Time complexity of various data structures [Wikipedia(2014a)]

fastest indexing speed $\theta(1)$ and no wasted space.

4.5 Third party graph library : mxGraph

mxGraph has model and view implementations and provides the following key APIs on JGraph user manual. [JGraph(2014)]

> Key API Methods:
> mxGraphModel.beginUpdate() - starts a new transaction or a sub-transaction.
> mxGraphModel.endUpdate() - completes a transaction or a sub-transaction.

mxGraph.addVertex() - Adds a new vertex to the specified parent cell.
mxGraph.addEdge() - Adds a new edge to the specified parent cell.
mxGraph.insertVertex(parent, id, value, x, y, width, height, style) – creates and inserts a new vertex into the model, within a begin/end update call.
mxGraph.insertEdge(parent, id, value, source, target, style) – creates and inserts a new edge into the model, within a begin/end update call.

For insertVertex() method in this project, 'value' represents as a name of city. 'height' is fixed as fifteen. 'Width' is not fixed, it varies according to the length of a city name it uses. style is "Rounded" in other words, it's a square. If you look at my code, in 'GraphGenerator' class, in 'createGraph' method, these APIs are used. For insertEdge() method, that is used in the same class but in different method, 'edgeDrawerFromNodeItoJ()', 'value' is the length of the edge and 'source' is which node the edge starts from. 'target' is destination node of the edge. 'style' defines colour of the edge and thickness of it. Each algorithm has different configure value for style so that it has different colour and thickness in order for user to identify. Lines of codes that contain examples of the APIs are also provided on their tutorial. This will be further discussed in implementation section.

```
// Adds cells to the model in a single step
graph.getModel().beginUpdate();
try
{
  Object v1 = graph.addVertex(parent, null, "Hello,", 20, 20, 80, 30);
  Object v2 = graph.addVertex(parent, null, "World!", 200, 150, 80, 30);
  Object e1 = graph.addEdge(parent, null, "", v1, v2);
}
finally
{
  // Updates the display
  graph.getModel().endUpdate();
}
```

Figure 7: mxGraph example, cite from [JGraph(2014)]

4.6 Graph ADT

There are three typical ADTs for Graph ADT. Adjacency Matrix, Adjacency List and Adjacency Set. Adjacency list can have several different expression for the same graph. I compared the time complexity with the other graph ADTs showns on figure 11. Adjacency Matrix needs O(V*V) time and space for initialisation. The Adjacency List does not need much space but it requires complex procedures in order to do some specific operation. For example, when an edge is removed, others nodes connected to

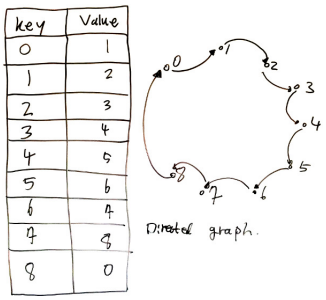
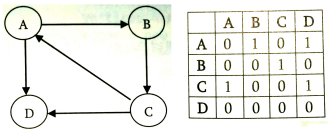


Figure 8: Hash map as an edge list



	A	B	C	D
A	0	1	0	1
B	0	0	1	0
C	1	0	0	1
D	0	0	0	0

Figure 9: Adjacency Matrix, cite from [Karumanchi(2012)]

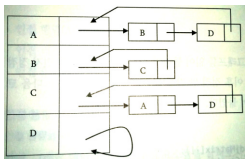


Figure 10: Adjacency List, cite from [Karumanchi(2012)]

Graph ADTs	Space	is an edge between 'v' and 'w'	Find an edge belongs to 'V'
Edge list	E	E	E
Adjacency Matrix	V^2	1	V
Adjacency List	$E+V$	Degree(v)	Degree(v)
Adjacency Set	$E+V$	log(Degree(v))	Degree(v)

Figure 11: Graph ADT comparisons, cite from [Karumanchi(2012)]

the edge need to be found. Hash map is used for edge representation. Hash map or Hash table is a data structure consisting of key and value. Key represent source node and value represents as target node. Although Hash map shows slow performance, overall it provides a simple interface to handle data. Hash map also does not allow same keys so it prevents a user or a program adding more edges that has a different direction to a node. A path is not made if same keys are allowed as on figure 13. Hash map is not thread safe and 'Concurrent Hash Map' data structure was used in 'GeneticAlgo',

	Average	Worst case
Space	O(*n*)[1]	O(*n*)
Search	O(1)	O(*n*)
Insert	O(1)	O(*n*)
Delete	O(1)	O(*n*)

Figure 12: Hash map time complexity in Big O notation [Wikipedia(2014b)]

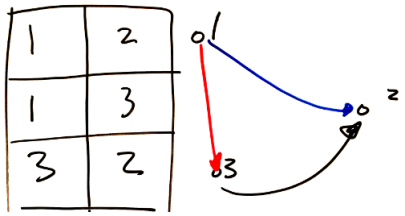


Figure 13: If same keys are allowed

'PathDrawerIfSubgraphExist' and 'GraphGenerator' classes in order to prevent thread safe issue. Tree map is often used in 'GeneticAlgo' class in order for myself to debug easier as it sort data in descending order.

4.7 Flow Chart : Specification

There are two flow charts provided. One is for prerequisite settings for running the program that is figure 14 and the other one is an abstract version of actually running it that is figure 15. Therefore figure 14 is prior to figure 15. 'Sun Java' on the flow charts denotes Sun Microsystems, Inc. Java. All button names are symbolic, "Initialisation" button is to make nodes with random city names. "Run Algorithms" button is to run algorithms. Traverse simulation means a user initiates an icon on the screen and let the icon move around a selected path, it is a visualisation for the user.

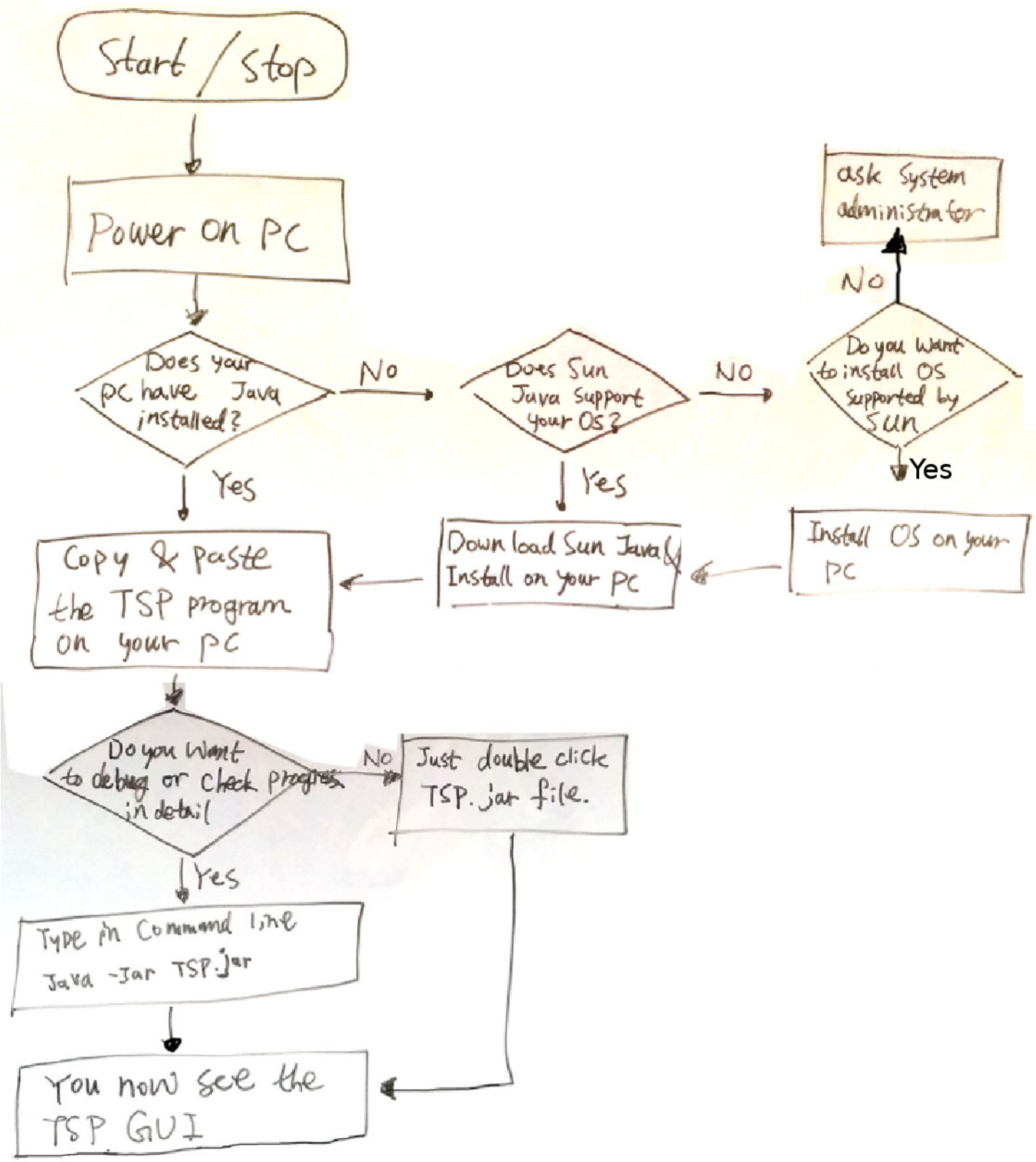


Figure 14: Basic requirements for running the program.

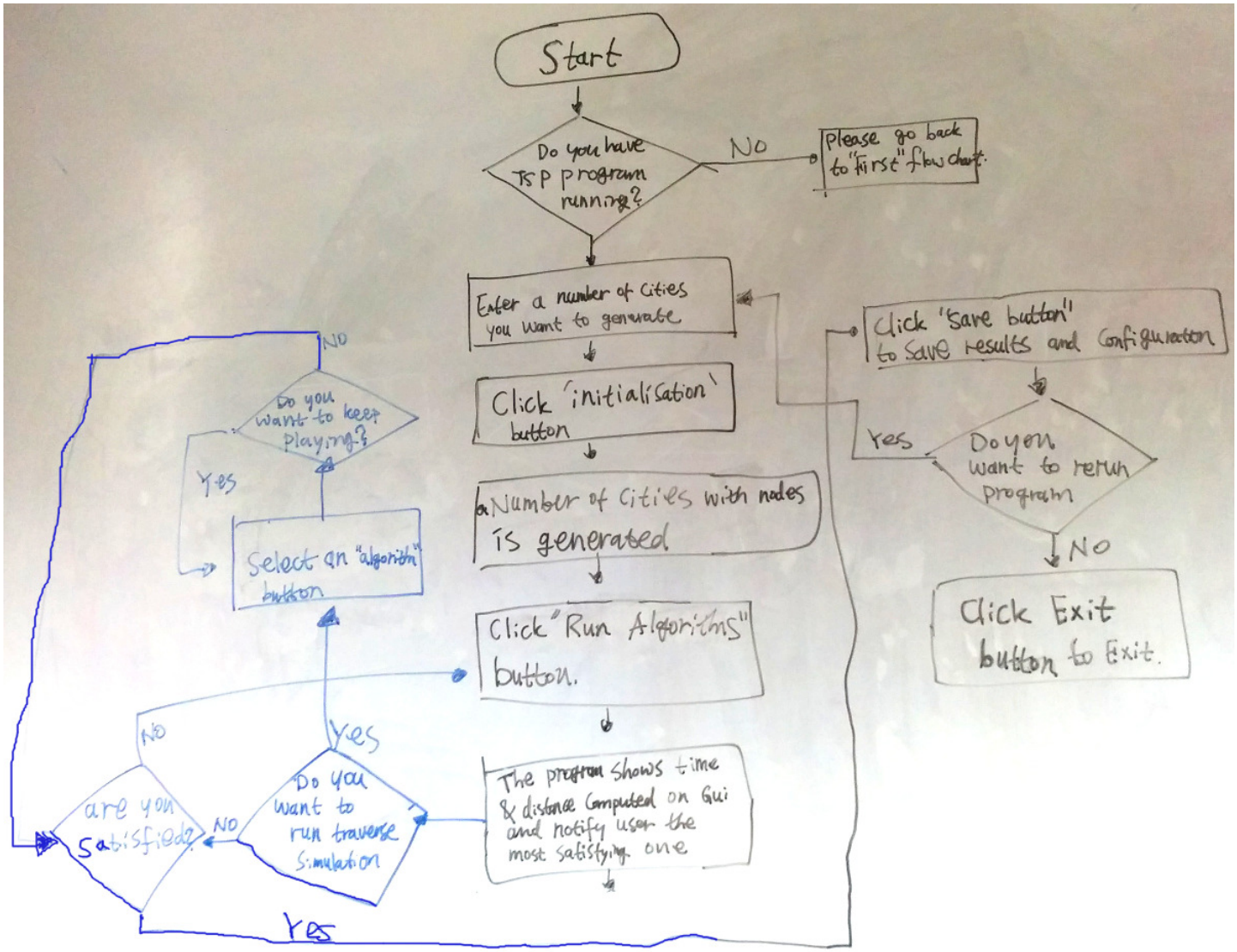


Figure 15: Abstract version of running the TSP program

5 Design

5.1 Use Case

5.2 Sequence Diagram

5.3 Functionality of the system

When a user first runs the program, a graphical user interface is loaded. User specifies into a text box how many nodes should be created and he or she then presses the 'Start' button and nodes will be presented with randomly distributed names on GUI and the system will try algorithms and heuristic systems as mentioned above and find out the best result. It will then show the best route on the GUI.

I have two designs, the first is that the user specifies which algorithms he or she wants to use. User clicks one or several check boxes in order to specify what algorithms need to be run. Once user clicks the 'Finding Shortest Path' button, the program computes results.

The second design is the following. The program will compute every algorithm in order to make the program more intuitive with less input from user and shows results on GUI with different colour lines. If user clicks a check box, one of the computed paths will appear or disappear so that the program provides to users graphical comparisons.

If the number of nodes is less than 200 the program will try either 'branch or bound' or 'dynamic programming' algorithms depending on the specifics of the problem. If the number is larger than 200 it will use heuristic methods.

The Number of nodes created is specified by the user but the position of nodes is randomised so that different results can be produced and are to be are expected.

If there is still sufficient time available completing this task, there may be additional specifications that a user may use e.g. Google map. Once user specifies an area on a Google map by clicking and dragging a mouse, a graph will be generated from simplified version of a map provided by Google. This can be done by image recognition or I will try to use a Third party library that will make process easier. An optimal solution will be found and a best path will be represented on Google map with a separate layer.

5.4 Graphical User Interface

Below is a prototype GUI for the project. User first inputs a number of nodes he or she wants to create into the text box in the initialisation section and clicks 'Run' button will be enabled and the user clicks it in order to run the program. The program will run every heuristics and it will show results in the Result section. A distance of total path will appear on each box in the section with time taken to calculate. The user can check or uncheck

the small boxes and the paths will be coloured. If the user is interested in seeing a step by step tour, he or she can choose a heuristic in the JCombo box and run a path animation by clicking 'Run' button in the 'Step by Step Visualization section'. The user can go back to a previous state or future state by clicking the sliding bar or 'prev' or 'ford' buttons. The result can be saved and loaded at any time when a user wants to see it again and wants to play with it. In order for the GUI to work interactively with the program, a design pattern is going to be used when coding, called 'Mediator Pattern'.

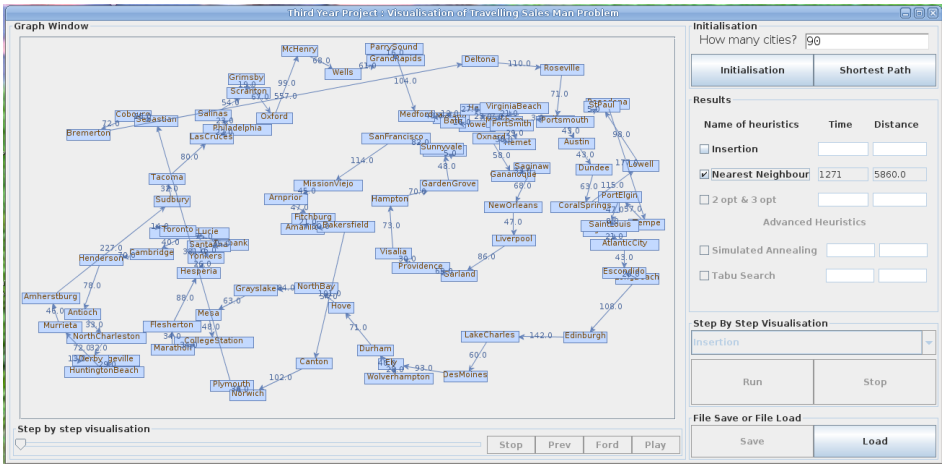


Figure 16: Prototype GUI and a generated sample path

6 Implementation

There are 27 Java classes in the implementation and two of them, EdgeLayers and Observer classes are not used. IntelliJ IDE is used for development. 'Lloydp' appears on JavaDoc that is my user name used on my personal PC. The code will be provided in Appendix as a pdf file and also an executable jar file. There are three packages, GUI, Graph and Algorithm packages and Main class as shown on figure 6. Git version control is used and committed to my Github private repository. The implementation has started from the 11th of October in 2013. Java library 1.7.0.51 version is used and compatible with contemporary Java. JGraphx library is used and it include mxGraph.

6.1 UML diagrams

I provide UML diagrams of each package. The UML diagrams were created by eUML2 tool integrated with Eclipse IDE. The tool can be downloaded from 'http://www.soyatec.com/euml2/'. The documentation can be found on 'http://www.soyatec.com/euml2/documentation/com.soyatec.euml2.doc/'.

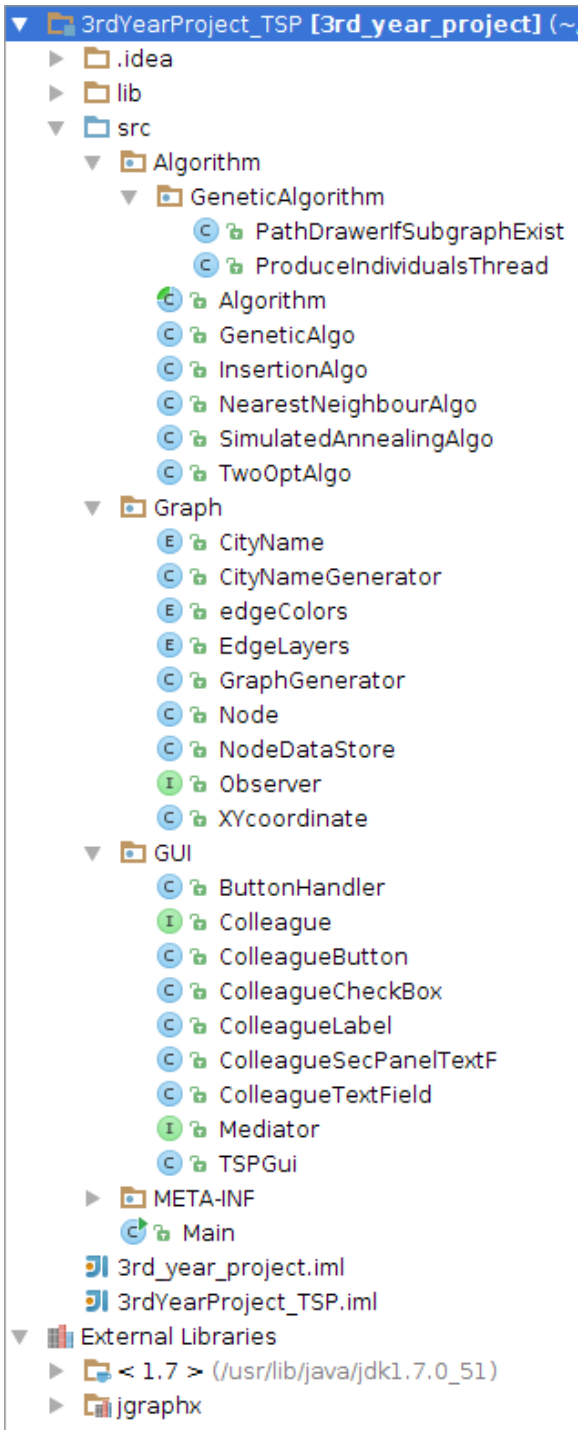


Figure 17: Java classes consists of the TSP project, Notice that there are three packages and Main class.

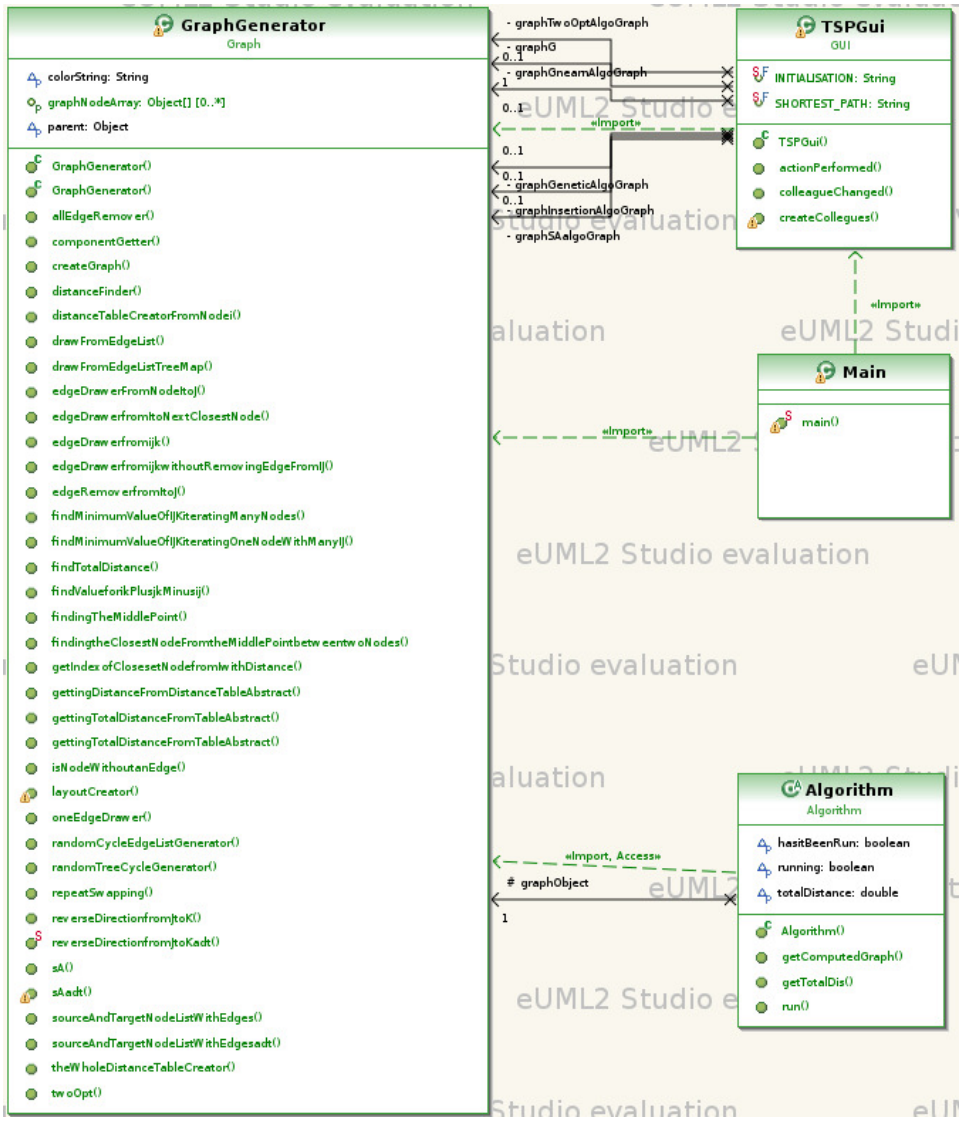


Figure 18: Big picture of the whole implementation

Main class instantiate objects of GraphGenerator class and TSPGui. Algorithm abstract class directly is associated with GraphGenerator class, indicating basic functions needed by all other algorithm classes defined in either GraphGenerator or Algorithm class.

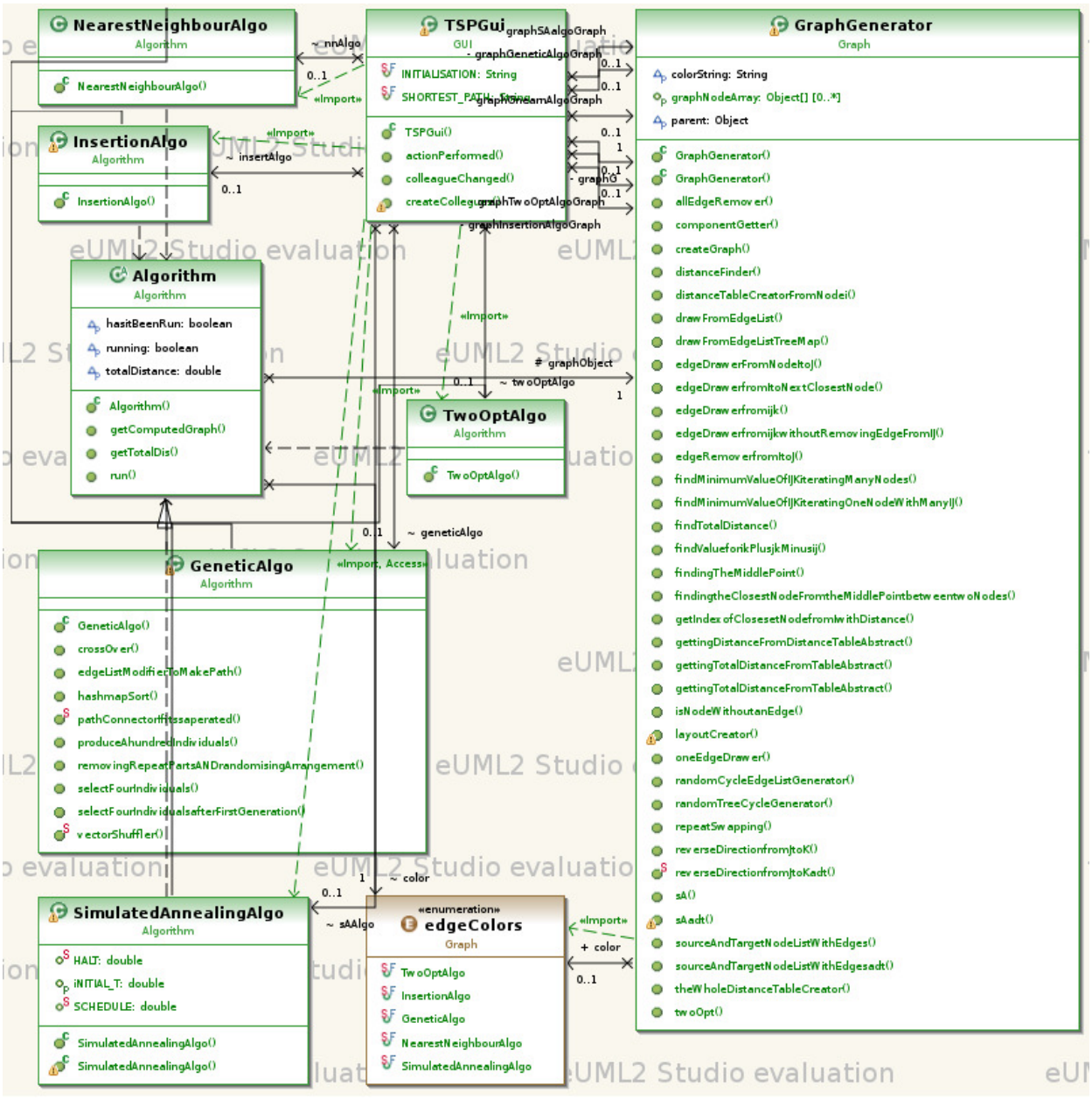


Figure 19: UML : Algotithms

TSPGui class instantiates objects of each algorithm class. Each algorithm class is based on Algorithm abstract class and I used 'Command' pattern so that the algorithm classes have a simple way to run their tasks without switch or conditional statement. 'edgeColors' class is to identify path of each algorithm on GUI. Nearest Neighbour path is made blue. Insertion Algorithm is coloured as green. 2-opt as red and Simulated Annealing becomes black. Genetic algorithm is purple. Algorithms either use methods defined in them or GraphGenerator class. What methods it uses it will be discussed in each algorithm's Pseudo code section.

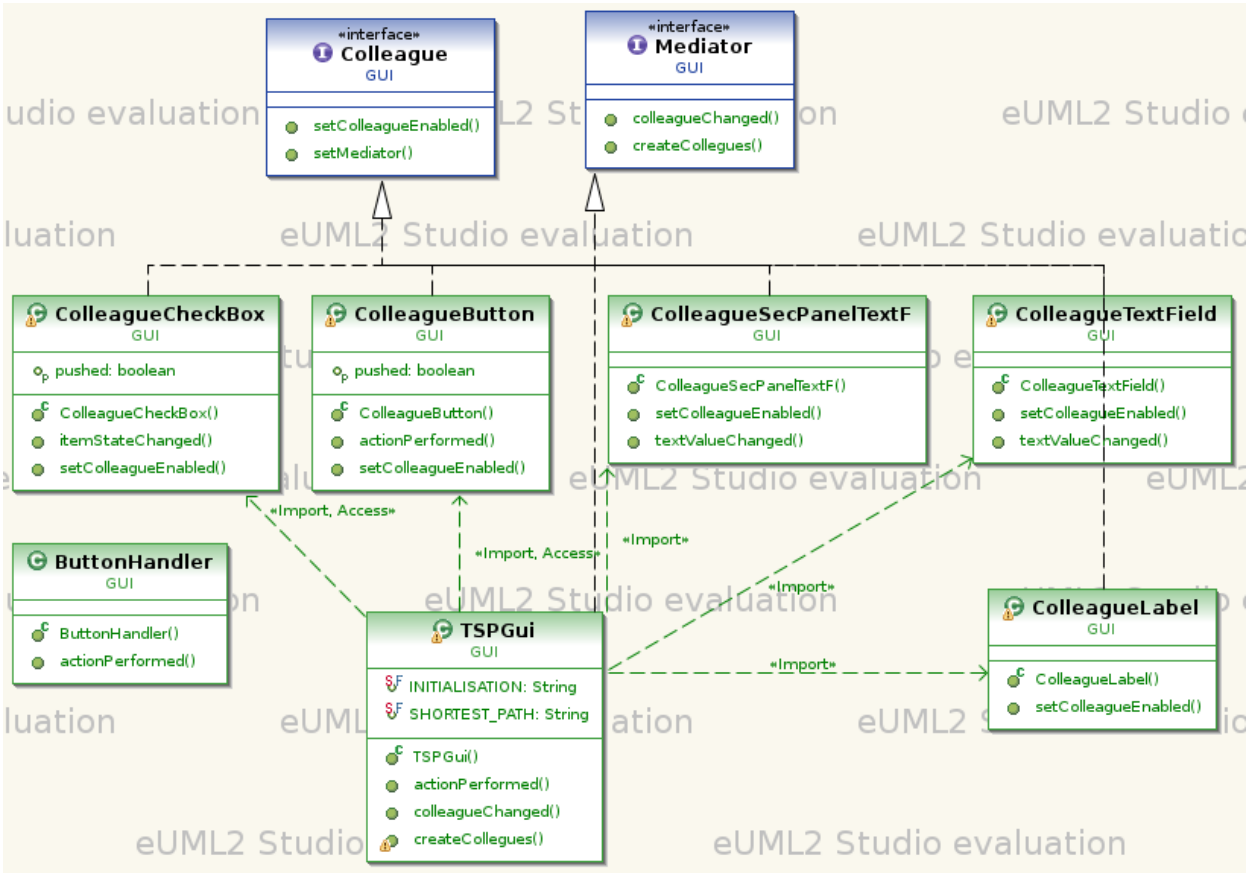


Figure 20: UML : GUI, Mediator design pattern

Mediator pattern was used in order to manage GUI activity more interactively and efficiently rather than defining an EventListenner for each activity. According to Erich Gamma [Erich Gamma(1994), pp.306]

> ..Different dialog boxes will have different dependencies between widgets....Customizing them individually by sub-classing will be tedious, since many classes are involved... You can avoid these problems by encapsulating collective behavior in a separate mediator object...The mediator serves as an intermediary that keeps objects in the group from referring to each other explicitly. The objects only know the mediator, thereby reducing the number of interconnections...

Various activities can be managed as follow in my code.
@Override; public void colleagueChanged()
{$cityTextChanged(); startButtonPushed(); shortestButtonPushed();$} Mediator is sort of an activity observer, the above statements means when texts in 'number of city input' changes do 'cityTextChanged()' method. When 'Initialisation' button is pressed run startButtonPushed();

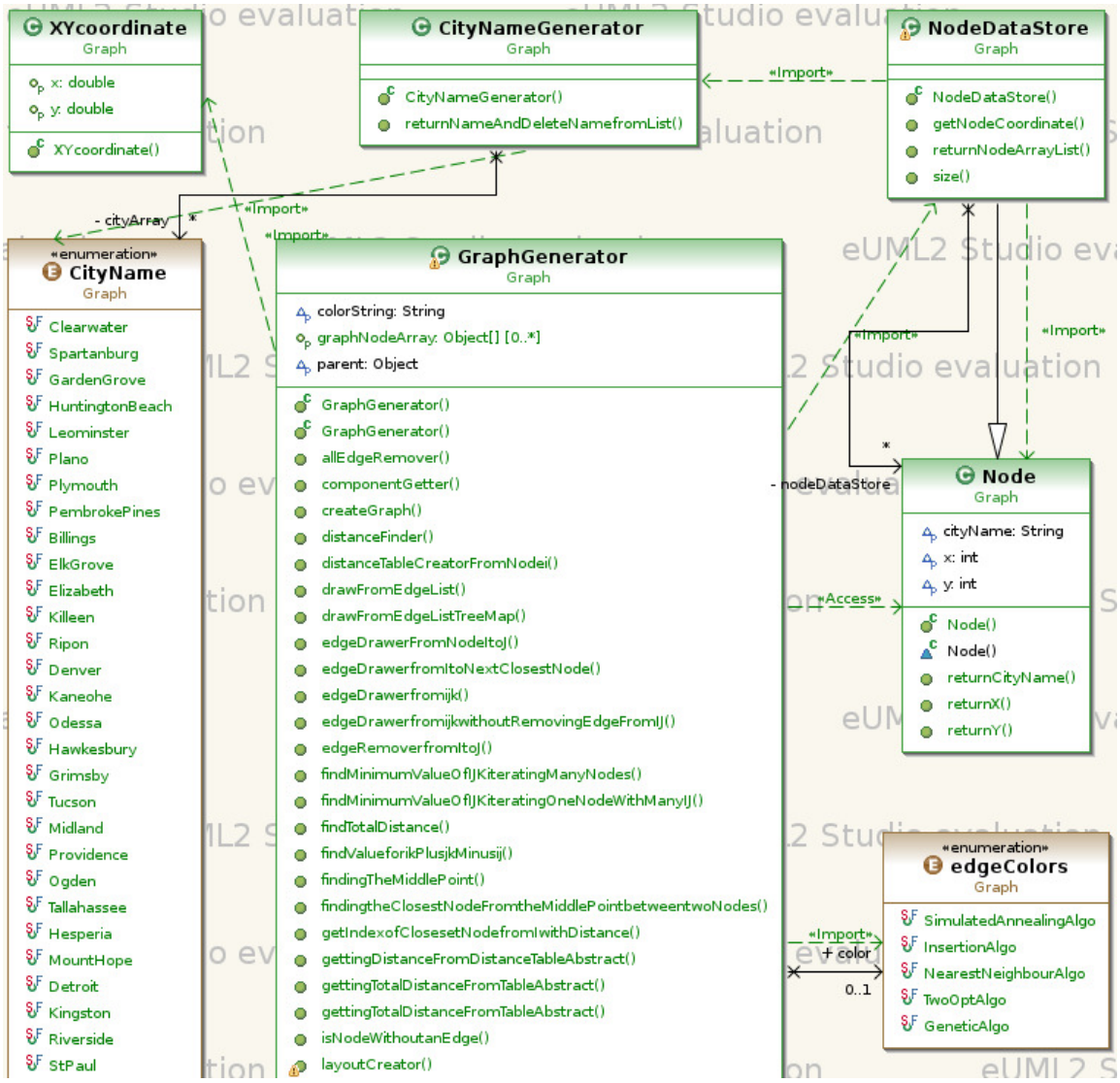


Figure 21: UML : Graph Generator

When random cities are generated on GUI NodeDataStore class initialise an arraylist that will contain node objects with randomly generated x y coordinates and a name of node (city name). The array index is working as ID of each node. 'CityName' class pick a city name from 'CityName' enum class and pass it to 'NodeDataStore' class. There are two Enumeration classes, one is CityName and edgeColors. Enumeration can be used as a switch as it is stated on 'Java How to program book' [Paul Deitel(2010), pp.343]

> The enum constants can be used anywhere constants can be used, such as in the case labels of switch statements and to control enhanced for statements.

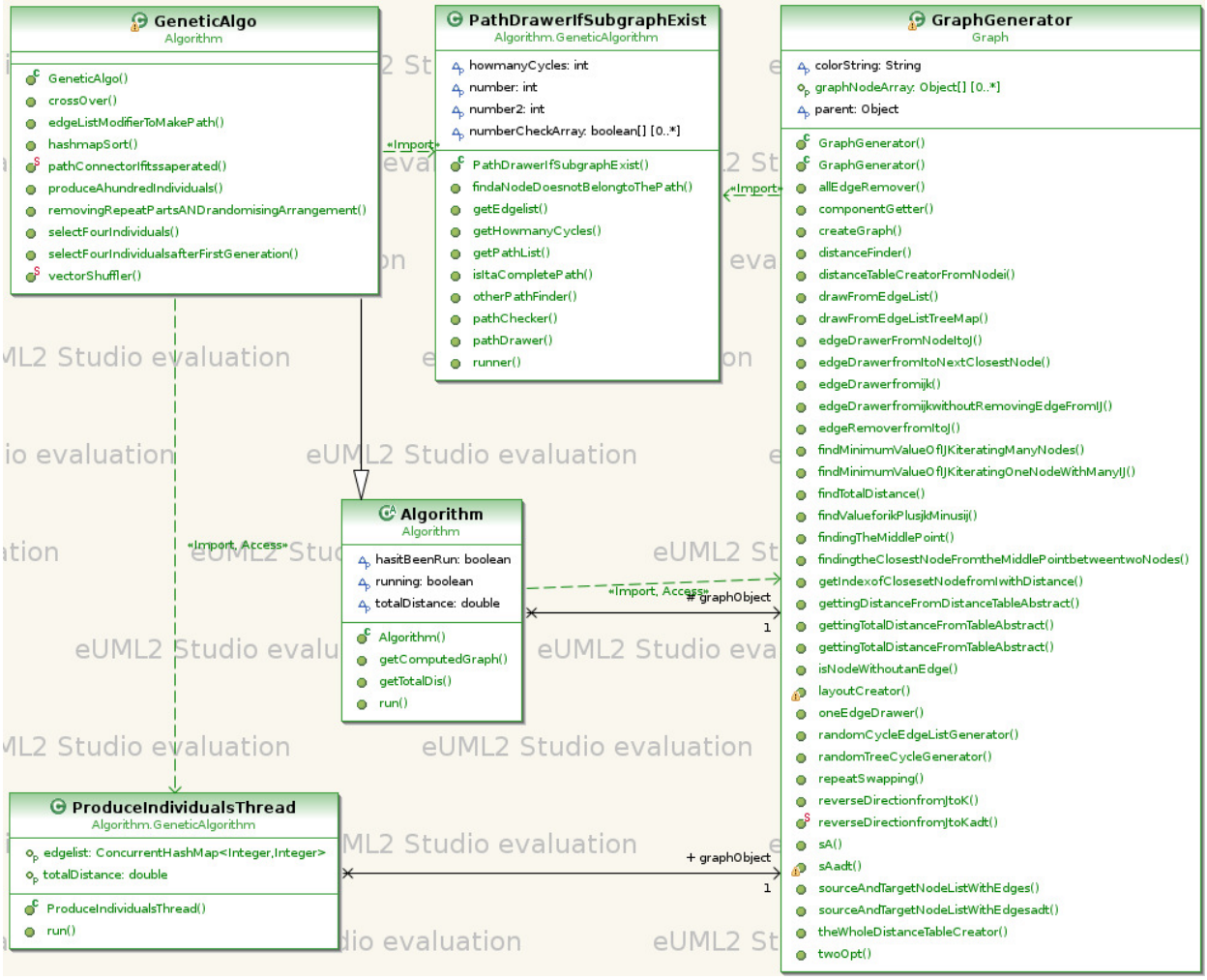


Figure 22: UML : Genetic Algorithm

'GeneticAlgo' class instantiates an object of 'ProduceIndividualsThread' class and produces 100 random individuals. It selects the four shortest individuals and crossover them. The path might not complete because of the randomness of crossover operation. 'PathDrawerIfSubgraphExist' class modifies a broken path and gives a complete one. Both 'PathDrawerIfSubgraphExist' and 'ProduceIndividualsThread' class have a GraphGenerator object (graphObject) for basic graph manipulation functions. The details of this process are explained in Genetic Algorithm, Implementation section, the section number 6.8.

6.2 Lines of codes

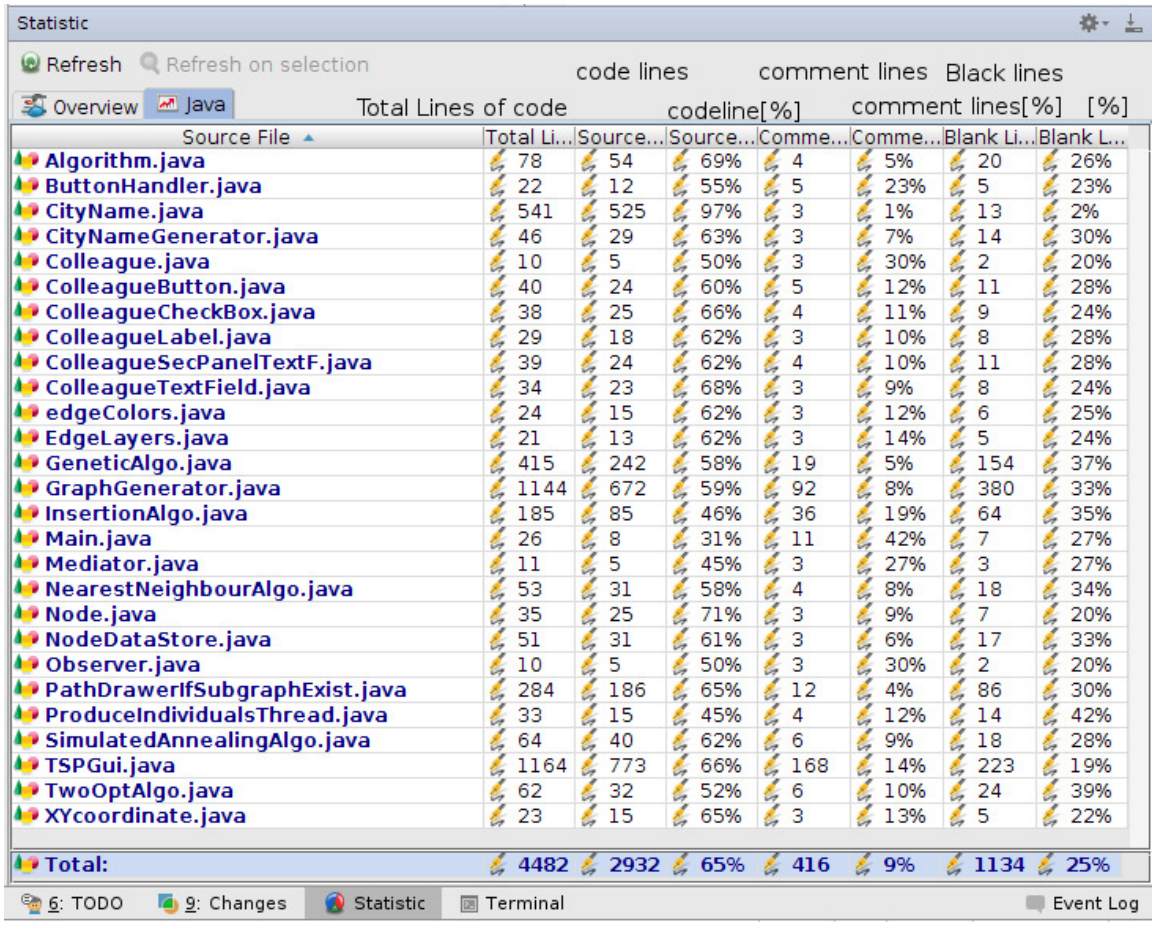


Figure 23: IntelliJ IDE Plug-in : Statistics developed by Ing. Tomas Topinka

Jetbrain IntelliJ IDE provides a plug-in called statistics and a table was created. According to this table. The total lines of code is 4482.

6.3 Regarding Pseudo Code Generation

The below pseudo codes are a summarised form of the code implemented and examples of Java classes and methods will be given. The following pseudo codes are not a form of replica of the standard. Each algorithm has an override method 'drawer()' and things written in that method will run when the program runs. 'graphObject' is an object of GraphGenerator class. 'graphObject.edgeDrawerfromItoNextClosestNode(int i)' draw an edge from a node 'i' to the closest node and returns an index of the closest node. 'graphObject.edgeDrawerFromNodeItoJ(int i, int j)' method draws an edge from node 'i' to 'j'.

6.4 Nearest Neighbour Algorithm

Algorithm 1: Nearest Neighbour

input : Node i, Node j ... Node N-1

```
1  initialization;
2  start from node i.;
3  pick node j ;
4  distance1 ← Find distance between i and j;
5  make a resizable array p;
6  put j to array p;
7  while Iterate until the last node, N-1 do
8  |  Pick node k that is not in array p and without an edge;
9  |  distance2 ← distance between i and k;
10 |  put k to array p;
11 |  if distance1 < distance2 then
12 |  |  distance1 ← distance2 and remember node k;
13 draw edge between i to k go to line 3 and repeat the same
   procedure Until there is no node without an edges if node k can
   not find any node without an edge then
14 |  connect node k with node i;
15 else
```

This algorithm is named as 'nearestNeighbour();' method in 'Nearest-NeighbourAlgo' class. The method calls a method 'drawerIterator(int i)', it is a recursive function starting from the last node and connect the closest node and the next one and so one.

6.4.1 Time Complexity : Nearest Neighbour Algorithm

It takes O(n) from step 3 to 13. The step 13 takes O(n). The total time complexity is O(n*n) that is $O(n^2)$. This is time complexity of Nearest neighbour algorithm.

6.5 Insertion Algorithm

Algorithm 2: Insertion

input : Node i, Node j ... Node N-1

```
1  initialization;
2  start from node i.;
3  find the closest node unvisited j draw an edge from i to j;
4  distance1 ← Find distance between i and j;
5  make a resizable array p;
6  put j to array p;
7  Find the closest unvisited node from i other than j and make it k;
8  distance2 ← distance from node i to k;
9  distance3 ← distance from node j to k;
10 distance4 ← (distance2 + distance3) - distance1;
11 while Pick different nodes for i and j trying every pair of nodes do
12 |  Interate the process 7 ∼ 10
13 Find the best distance4 value and corresponding node i and j.
   Draw edges from node j to k and node k to i while Until there is
   no node without an edge do
14 |  pick two connected nodes within the path;
15 |  let them be node i and j;
16 |  repeat the process 4 ∼ 11;
17 |  remove an edge between node i and j;
```

In 'InsertionAlgo class', There are two stages, the first stage is makingA-triangle(); method makes a triangle path connecting node i,j and k, this is until the step 11. 'newSetting()' method is an iteration of the steps 13 ∼ 17. 'previousSetting()' method works but it accesses graphObject and draws and removes lines there every time each calculation is done and this brought a performance issue so Hash map data structure is used in 'newSetting()' method.

6.5.1 Time Complexity : Insertion Algorithm

Step 3 takes O(n). Step 7 could be O(n) but the calculation is already done on step 3. It does not count. Step 7 to 13 is O(n), it searches every pair of nodes and storing distance4 value into data structure. From step 13 to step 17 is another O(n). The total time complexity is $O(n^3)$.

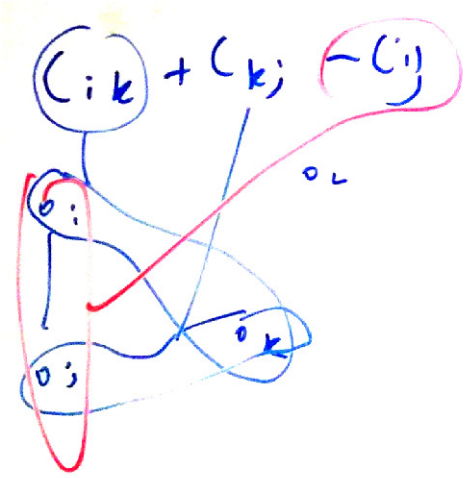


Figure 24: The pseudo code, line 10

Figure 25: The pseudo code, line 17

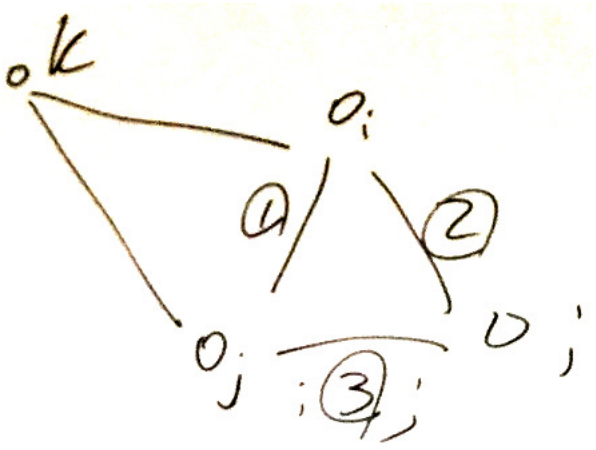


Figure 26: The pseudo code, line 11

6.6 2-opt

Algorithm 3: 2-opt

input : A Hamiltonian path is given

start from node i; select a node connected from i and let it be j; find two nodes connected each other apart from node i and j and not connected to i or j ;

let the two node be l and k; k is a source node and l is a target node;

initialdistance ← edge length between i and j + edge length between k and l;

laterdistance ← distance between node i and k + distance between node j and l;

if *initialdistance > laterdistance* **then**

    node g ← the node connected from source node j;

    remove two edges, one between ij and the another is between kl;

    draw two edges between node i and k and node j and l;

    node h ← Find the source node connected to k;

    while *until it reaches to the node g* **do**

        reverse the direction of edges connected from node h and k;

    draw an edge from node g to j;

while *until you reach to the every node not connected to node i and j* **do**

    go back to the step 1 iterate the same process

let nodes i and j be the next nodes from the edge list introduced in section 4.6 ;

repeat the steps from 1 $\sim$ 13 ;

while *until it reaches to the whole edge list set introduced in 4.6* **do**

    repeat from the steps 15 $\sim$ 16;

if *there is no edge satisfying the whole process* **then**

    it is done

else

    repeat the step 17

Nearest neighbour algorithm gives a Hamiltonian path. 'graphObject .drawFromEdgeList(graphObject.randomCycleEdgeListGenerator())' method creates a random cycle and it was used previously for 2-opt and simulated annealing. 2-opt and simulated annealing may not be compared correctly since it uses a random path as a base. The base should be the same for any

algorithm for comparison. Nearest neighbour algorithm is used as a base path, it selects the same node for a start point and it gives a deterministic answer.
Inside of TwoOpt class, there is 'graphObject.twoOpt()' method and it leads to sAadt(0, 0, 0, false) method in GraphGenerator class. sAadt() method has four parameters and 'SimulatedAnnealingAlgo' class uses sAadt() methods with values. If the last Boolean value is false it is just 2-opt algorithm. If it is, the value is true it works as a simulated annealing method. The details will be discussed in Simulated Annealing section.

6.6.1 Time Complexity : 2-Opt

Steps 15 and 16 are a name tag for step 1 to 13. Step 1 to 13 contains two big steps. One is to find a replacement a edge between i and j, this is O(n). The second step is to reverse the direction of edges as on figure 27. The direction of the edges from node 2 to node 12 should be reversed otherwise it won't be one way path. This is also O(n). Therefore step 1 to 13 takes $O(n^2)$. Step 17 takes O(n). The total time complexity is $O(n^2) * O(n) = O(n^3)$

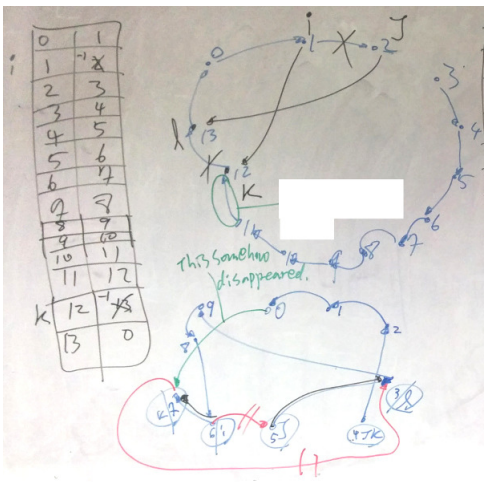


Figure 27: The pseudo code demonstration

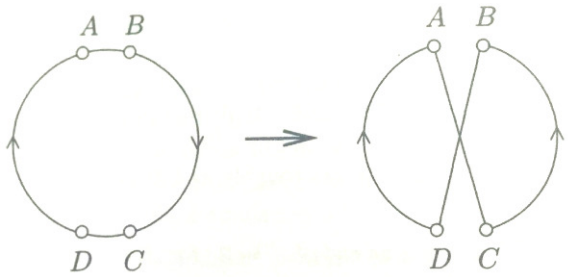


Figure 28: A 2-opt move, cited from [David L. Applegate(2006), pp.104]

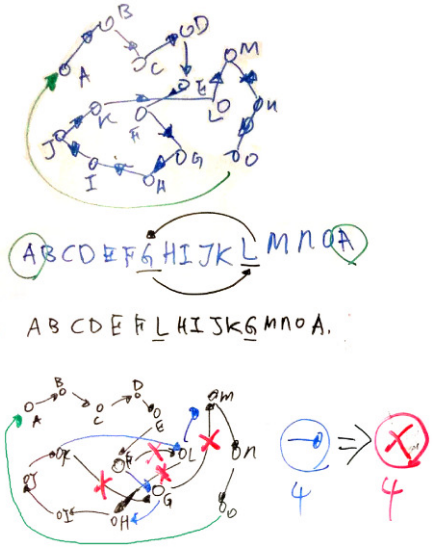


Figure 29: A 2-opt change, before and after, notice that 4 edges removed and added

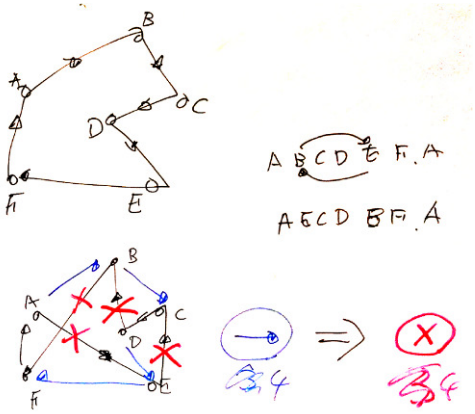


Figure 30: A 2-opt move, a different example, notice that 4 edges removed and added

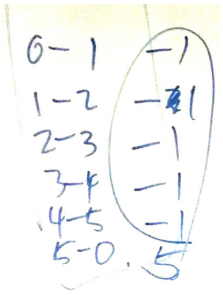


Figure 31: The sum of source nodes minus target nodes is always 0, if every nodes are connected

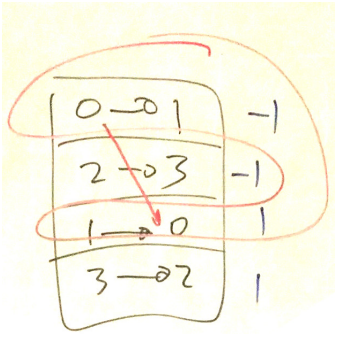


Figure 32: The method on figure 31 could cause multiple sub tours

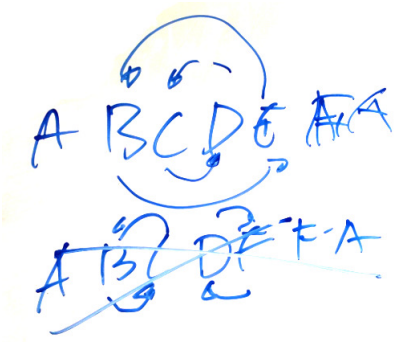


Figure 33: This is a normal 2-opt move.

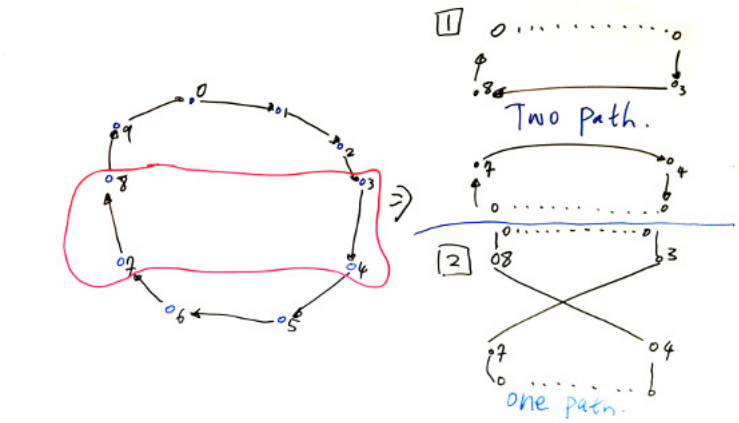


Figure 34: Multiple sub tours could happen.

6.7 Simulated Annealing

Simulated annealing heuristic is shown below using 2-opt as a base algorithm.

Algorithm 4: Simulated Annealing based on 2-opt algorithm

```
   input : A Hamiltonian path is given
 1 initialise temperature, schedule and halt variables;
 2 Let temperature be 20000 and schedule be 0.99 and halt be 0.03;
 3 while until temperature > halt do
 4 |   temperature ← temperature * schedule;
 5 |   initialise initialdistance and laterdistance variables as
   |   mentioned in the 2-opt algorithm;
 6 |   E ← initialdistance − laterdistance;
 7 |   if E>0 then
 8 |   |   do 2-opt swap;
 9 |   else
10 |   |   do 2-opt swap in probability of ;
11 |   |   (exp power to(E/temperture));
12 done;
```

In 'SimulatedAnnealingAlgo' class, temprature is INITIALT variable and set to 2000000. schedule is SCHEDULE set to 0.9995 and halt is HALT 0.03. The class uses nearest neighbour as a base path. 'graphObject.sAadt(INITIALT, SCHEDULE, HALT, true)' leads to sAadt(double INITIALT, double SCHEDULE, double halt, boolean onoff) method in GraphGenerator. Since the last parameter is true it is a simulated annealing function not 2-opt.
2-opt is occurring until the while loop terminates in the java class, 'while(!noImprovementcantbemade)'. That terminates when temperature becomes lower than the value of halt variable, 'if(T < halt) noImprovementcantbemade = true'. The step 3 on the pseudo code shows it. When it gets the shortest path it draws a path on GUI, 'drawFromEdgeList(edgesWithSourceAndTargetNodes)'. The step 9 was implemented as
'double prob = Math.exp(E / T); double prob2 = Math.random();
if(E < 0) if(prob2 <= prob) $\{2 - optswap\}$'. This is called 'Acceptance Probability', meaning that random things happen within a probability of prob, this essentially is the same as things happen in a probability of prob. This is based on Metropolis-Hastings Algorithm.

6.7.1 Time Complexity : Simulated Annealing

This SA uses 2-opt implementation mentioned in the previous section but it does not search every combination at each iteration. When a change is

made in the 2-opt implementation and it iterates the whole process again. If there is no possible candidate and then it terminates and returns an optimal path. This SA implementation works fine but if a swap happens in the last iteration by probability of (Math.exp(E / T). It just terminates because 'Temperature' value becomes lower than 'halt'. Nevertheless, this staggeringly is rare because $exp(negativevalue/T)$ gives infinitely small number T value is close to zero. For example, $exp(-1/0.03) = 3.338 * 10^{-15}$ $exp(-1/0.1) = 0.0000453$. This is why halt value should be low in order to prevent such error or let it do the 2-opt process until there is no candidate at the last iteration.

The time complexity would be, 'a number of iteration of the SA implementation' * '2-opt time complexity $O(n^3) = O(m) * O(n^3)$'.

6.8 Genetic Algorithm

Algorithm 5: Genetic Algorithm

```
   input : A Hamiltonian path is given
 1 initialisation;
 2 Produce a hundred of individuals ;      /* create random path,
   first generation */
 3 select the four shortest path from the hundred individuals ;
   /* selection */
 4 while Until it iteration 6000 times do
       ;                                    /* 6000 generation */
 5     initilise a list Q or P;
 6     put them into a list Q ;
 7     for i = 0 to Qlistsize do
 8         for j = 0 to 24 do
               ;      /* offspring = 4 selected indis + random
               indis, this repeats 24 times in order to make
               24 individuals */
 9             Iedglist ← load ith path edgelist from list Q ;
10             remove the half of the Iedglist add random path to the
               emptied space ;                    /* crossover */
11             Iedglist might not have a Hamiltonian path modify it
               in order to make a path ;           /* mutation */
12             put the new path into list P;
13     for i = 0 to Qlistsize do
14         for u = 0 to Qlistsize do
15             if i is not equal to u then
                   ; /* Crossover the four selected individuals
                   */
16                 Yedgelist ← load ith path edgelist from list Q;
17                 Oedgelist ← load uth path edgelist from list Q;
18                 remove the first half of Yedgelist;
19                 remove the second half of Oedgelist;
20                 add Oedgelist into Yedgelist ;     /* crossover */
21                 Yedgelist might not have a Hamiltonian path
                   modify Yedgelist in order to have one ;
                   /* mutation */
22                 add Yedgelist into list P;
23     select the four shortest path from the list P put them into list
       Q;
24 return the shortest one from list Q;
```

'GeneticAlgo' class first uses 'produceAhundredIndividuals()' method in order to create first generation 100 individuals (the step 2). Inside of the method, 'ProduceIndividualsThread' class' instances are created and stored in an array 'threadArray[]'. 'produceIndividualsThread' class draws a random path on 'graphObject' and the total distance is together stored. Each object's edge list and total distance in the array becomes stored in a tree map as on figure 40. In a tree map 'firstGeneration', edge lists become sorted in an ascending order according to distance. 'selectFourIndividuals' method takes four shortest path from 'firstGeneration' object and stores them in to a Vector data structure (the step 3). The while loop in 'GeneticAlgo' class determines how many generation there be for an optimal solution (from step 4 to 22). From second generation to the last one, it is done inside of the while loop. 'crossoverIndividuals()' method takes the vector containing the four selected individuals and mix(crossover) them together and produce offspring path(the steps 4 to 12 and from 16 to 22). 'crossover()' method is to crossover two edge lists(path) and produce one new path (step 9 to 10 and 16 to 20). Just mixing does not produce a good offspring because crossovered path might not be a complete cycle as on figure 41. This will be discussed in the following chapter.

6.8.1 Time Complexity : Genetic Algorithm

Step 2 is to make random path. It takes $O(n^2)$ to make a random path because it uses double for loops in 'GraphGenrator' class in 'randomCycleEdgeListGenerator()' method and this is repeated 100 times in 'produceAhundredIndividuals()' in 'GeneticAlgo' class. $O(n^2) * 100$ is $O(n^2)$. Steps 10 and 10 takes O(n), it is the same time complexity as making a new array. Step 11 is about 'PathDrawerIfSubgraphExist()' method, which has $O(n^2)$ time complexity. An edge list of this path is stored in list P, this is another O(n). It is $O(n) * O(n^2) * O(n) => O(n^4)$. Step 20, 21 and 22 are almost same as steps 10, 10 and 11, another $O(n^4)$ is here. Step 23, the tree sorting has time complexity $O(nlogn)$. $= O(n^2) + (O(n) * O(n^2) * O(n) + O(n) * O(n^2) * O(n)) * O(nlogn) * iteration$
$= O(n^2) + 2 * O(n^4) * O(nlogn)$
$= O(n^5)$

The complexity of this implementation exponential.

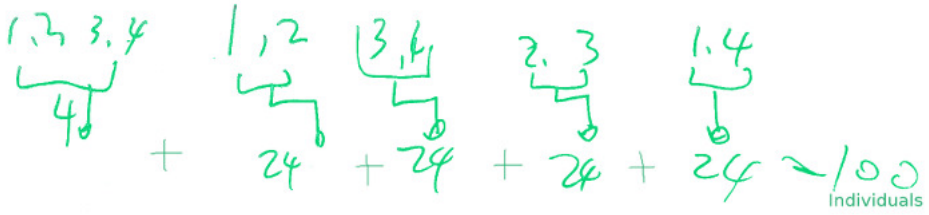


Figure 35: 1234, indicates the shortest selected individuals for next generation, Total 100 individuals produceds at each generation

$$TATCGGATCG|GTATATCCGA$$
$$+$$
$$GCTATTCGAG|CTTAAAGCTA.$$
$$=$$
$$GTATATCCGA|GCTATTCGAG.$$

Figure 36: Crossover for two path, Cited from [David L. Applegate(2006), pp.44]

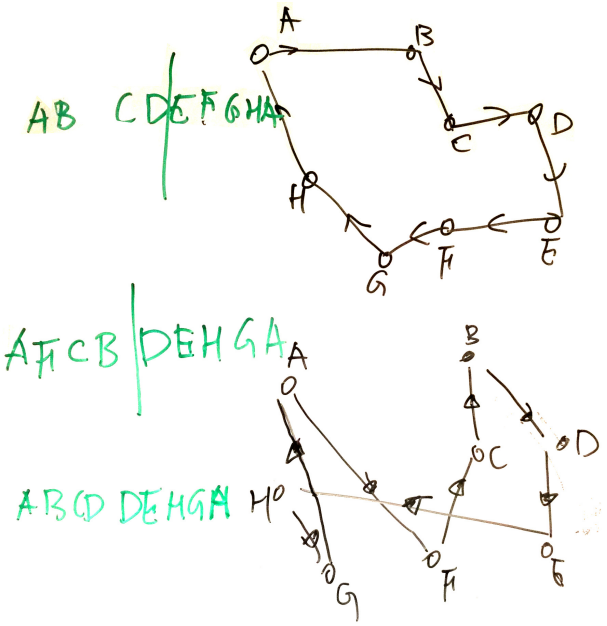


Figure 37: Successful crossover without an error, don't need any modification for a Hamiltonian cycle

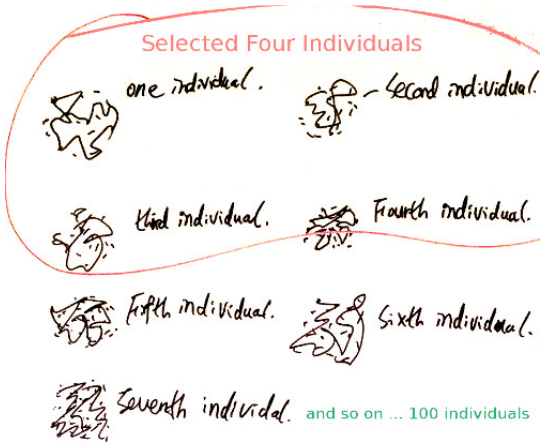


Figure 38: Visual representation of the four shortest path (individual) selection

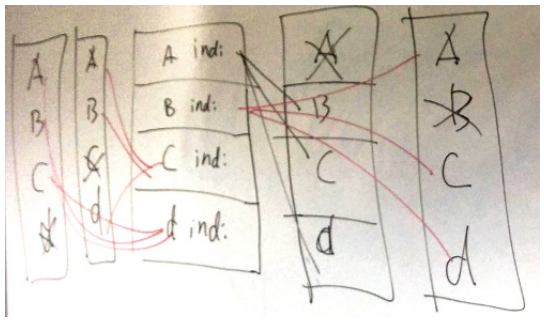


Figure 39: Crossover should be done by two individuals

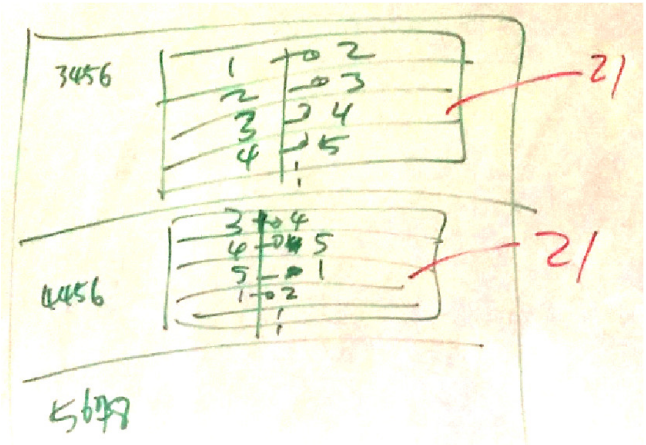


Figure 40: Edge list(path) is stored in a tree map with the total distance. The distance is a key and the edge list is a value

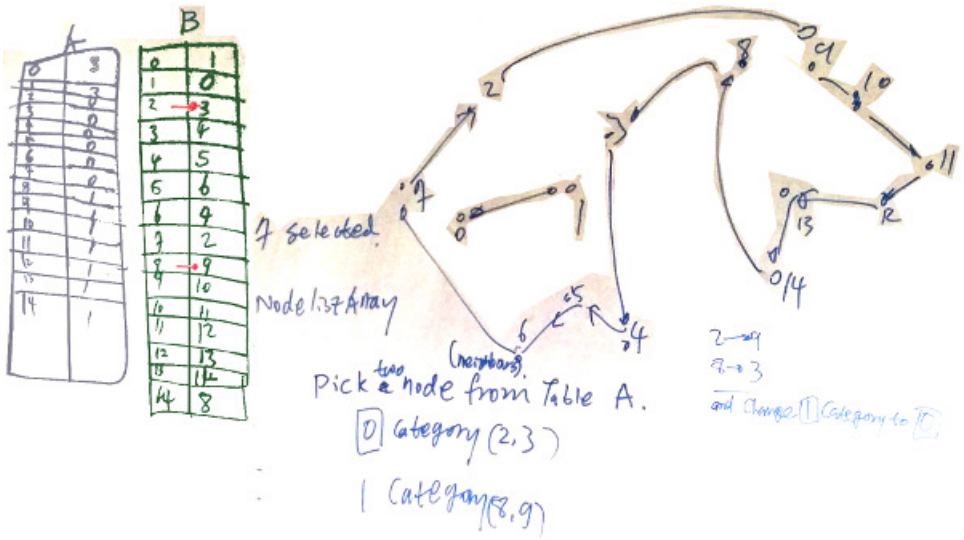


Figure 41: This is why we need 'path modification', only one Hamiltonian cycle should be made

6.9 Genetic Algorithm : One Hamiltonian cycle drawer

Inside of 'crossover()' method, two important methods are there, 'removingRepeatPartsANDrandomisingArrangement()' and 'pathConnectorIfitssaperated()'. 'removing ()' method minimises repeated edges in order to make it less complex for next step, it has an arraylist called 'nonExistiveValueList' that is shown on figure 6.9.'pathConnect..()' method fixes a broken path to a complete cycle. This is step 11 and 21. 'removeing..()' method passes an edge list to 'pathConnector..()' method. 'pathConnector..()' method instantiate an object of 'PathDrawerIfSubgraphExist()' and test if the path is a complete Hamiltonian cycle (steps 11 and 21). If not, it fixes and produces a complete cycle.

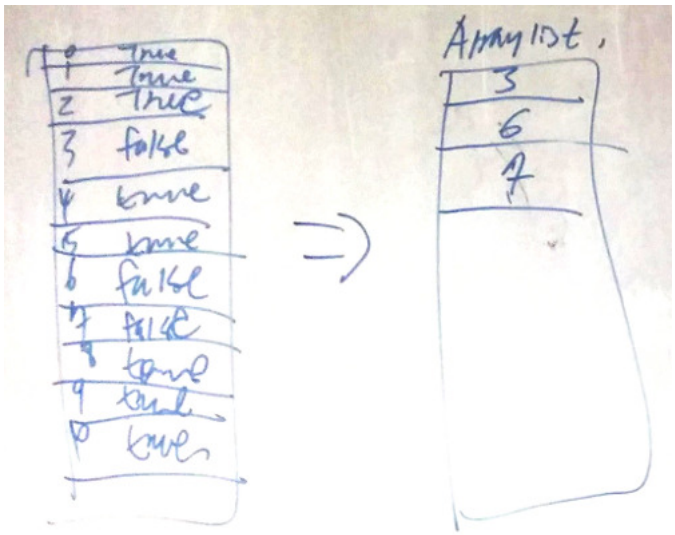


Figure 42: removingRepeat..() method, In an edge list, some nodes may not have an edge and are not there. The nodes are not there identified as 'false' and blacklisted to an arraylist in order to yield a better chance to make a complete path.

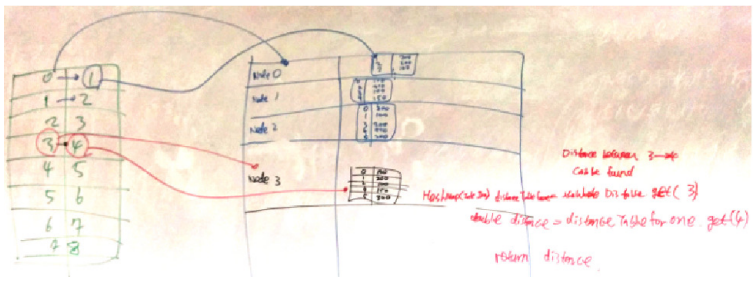


Figure 43: 54

7 Professional and Ethical Issues

On British Computer Society, There is a section called BCS Code of Conduct, http://www.bcs.org/category/6030. Here is a check list.

1. public interest.
a) The program does not use network resource or collect system information. Therefore there is no privacy or security issue.
b) Use of Third party libraries always explicitly mentioned on this report.
c) It does not contain any material related to sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability.

2. Professional Competence and Integrity
a) Yes, everything I do should be own my own.
b) Competencies that does not belong to me always to be explicitly stated on this report.
c) My Java coding and Latex skills have been developing throughout the whole process.
d) I used free third party libraries.
e) I attended many of Prof Tomasz Radzik's tutoring sessions for his advice regarding this project.
f) There is no such chance to commit such malicious actions.
g) I rejected any unethical inducement.

3. Duty to Relevant Authority.
a) I have been following the college's guide to deliverables on 6CCS3PRJ 13 14 1 FINAL YEAR INDI on Keats.
b) For the same reason on a), I have avoided any conflicting situation between myself and the college.
c) There is no colleague working with me for this project.
d) I have not published the college's material to internet.
e) This does not require any network resource and does not invade user's privacy.

4. Duty to the profession
a) The program has never been sold or disclosed to the public and there is no reputation issue.
b) I have been following conventional design patterns for software architecture in order to meet the professional standards.
c) My BCS membership expired last year and I am not representing BCS any more.
d) My BCS membership expired last year and I no longer connected to the BCS society.
e) I have not established any firm or business.

f) This is a personal project and I have no working colleagues for this project.

8 Results and Evaluation

8.1 Evaluation of each algorithms pseudo code

There are pseudo codes on the Implementation section, section numbers are 6.4, 6.5, 6.6, 6.7, 6.8.
Nearest Neighbour has $O(n^2)$. Insertion and 2-Opt have $O(n^3)$. Simulated Annealing has $O(n^3)$. Genetic Algorithm has $O(n^5)$. They all take non-linear time.

Values for n^3 and $n^2 2^n$.

n	5	10	20	40
n^3	125	1,000	8,000	64,000
$n^2 2^n$	800	102,400	419,430,400	1,759,218,604,441,600

Table 2: cited from [David L. Applegate(2006), pp. 48]

8.2 Evaluation of computational results : A introduction

I have researched heuristics and meta-heuristics. I am aiming to implements four heuristics algorithms, Insertion, Nearest Neighbour, 2 opt-3-opt and Branch-and-Bound algorithms. I am also planning to adapt meta-heuristics, Simulated Annealing and Genetic algorithm. I am going to use the GUI in order to test each algorithm. Two things are to be monitored, computation time in milliseconds, the total traverse distance. They will be shown on the GUI. I will make a table and run tests, by increasing a number of nodes and gathering computation time and total distance information and input into the table.

8.2.1 Algorithm comparison by what standard?

There are three ways to compare algorithms, execution time, a number of instructions and time complexity. Execution time varies depends on the machine that you are running a program and other processes might be running at the same time. Nonetheless, it's still a physical representation of algorithm efficiency. As long as, it runs on the same machine without extra processes running, it's still an empirical evidence to judge although it is not precise. A number of instructions could be a candidate but it becomes different on various programming languages and also developers have a different coding style so that it is not accurate. Time complexity is theoretical

however an ideal one to judge efficiency of algorithms since it is not bound to a specific machine or developer's preference.

8.2.2 Test Environments

The algorithms were tested by an Intel Core i3 laptop otherwise explicitly stated. The time was measured in milliseconds and can be significantly varied on different machines. Using multiple thread does not mean each algorithm uses the same amount of computational resource thus it is not a fair experiment. The thread.join() method is used in order to deal with such issues. The tour distance is not affected by computational performance and it is merely a logic of the code that it influences it.

8.3 Comparisons of three algorithms : Tour distance

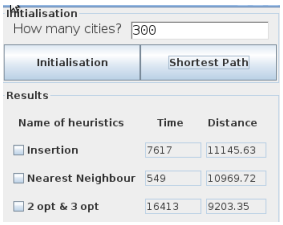


Figure 44: The TSP program GUI shows a result for each heuristic

8.3.1 How was it tested

The test was done by running the three algorithms on the same nodes. Clicking 'Shortest Path' executes the three algorithms simultaneously and shows time taken and the total tour distance. 42 tests was done and the results were input into a spreadsheet as on the table 3. A chart (graph) was produced from the table, generated by OpenOffice Calc and figure 45 is the graph. The three algorithms ran on the same node arrangement at each iteration in order to test in the same condition.

8.3.2 Evaluation of the result

2-opt always showed a good result and insertion algorithm showed slightly better performance than the Nearest Neighbour. The three algorithms showed almost the same performance until it reached to 60 nodes. After 60 nodes, the gap between 2-opt and the other algorithms becomes larger and larger.
Insertion algorithm and 2-opt shows a steady line. Nearest neighbour has a small and big fluctuations often because of the big X line highlighted on figure 54.

	insertion	nearest	2opt
3	814	814	814
5	1318	1399.49	1318
10	1950	2274	1950
20	3125	3223	2882
30	3669	3369	3101
40	3843	4287	3693
50	4333	4349	3533
60	4705	5162	4329
70	5463	6142	4580
80	5462	5794	4767
90	6180	6397	5206
100	6806	6448	5795
110	6902	7532	5837
120	7279	7427	6105
130	7747	7039	6389
140	7502	7223	6417
150	7663	7171	6301
160	8248	8762	6998
170	8583	7780	6802
180	8628	8534	7298
190	8556	9085	7535
200	9072	9174	7663
210	8955	8735	7510
220	9499	9365	7600
230	9773	9839	8191
240	10092	9105	7735
250	10184	9624	8233
260	10273	10265	8782
270	10373	10140	8932
280	10638	9834	8601
290	10946	10185	8935
300	11101	10544	9087
330	11139	11444	9608
350	11682	11544	9960
370	12173	12176	10021
400	12617	12695	10330
430	12913	12574	10677
450	12767	13772	10794
470	13650	13458	11323
490	13968	13786	11503
500	13950	13613	11276

Table 3: Algorithm Comparisons : Nearest Neighbour, Insertion and 2-opt

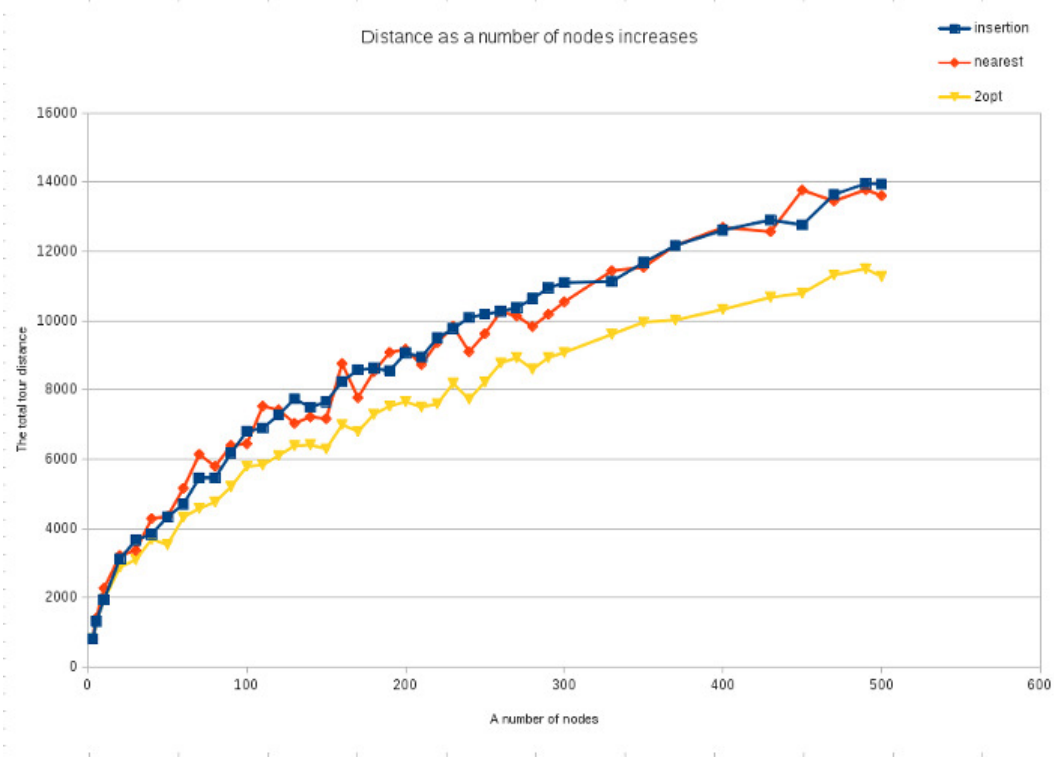


Figure 45: Three algorithm comparisons

8.4 Comparisons of three algorithms : Time

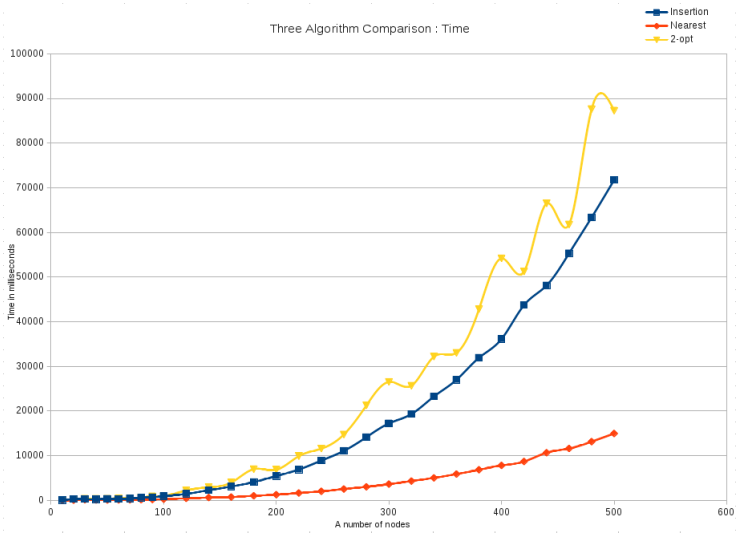


Figure 46: Algorithm Comparisons In Terms Of Time Taken.

Nearest Neighbour shows a stead line, it requires linear time to compute a result. On the other hand, Insertion and 2-opt shows a exponentially increase. 2-opt requires more time than insertion and it shows a strange tendency that keep repeating fall and jump. The results were observed as time complexity values were calculated for each of algorithm. Nearest Neighbour has $O(n^2)$. Insertion and 2-Opt have $O(n^3)$, this explains the dramatic increases of insertion and 2-Opt and the gaps from nearest neighbour graph.

8.5 2-opt and Simulated Annealing comparison

8.5.1 Comparison at 100 nodes

A test was done for Simulated Annealing heuristic for 100 cities as on figure 47. The lowest distance value found was 5121 for Simulated Annealing shown on the table 4. 2-opt showed 5308 distance and the time was 1137 on the same nodes as on figure 47. The following calculation on figure 48 is done on 'http://www.calculatorsoup.com/
calculators/algebra/percent-difference-calculator.php'. 3.5862 % improvement was made by Simulated Annealing from 2-opt. In terms of time, there is drawback. Simulated annealing took 712118 and 2-opt took 1137 milliseconds.

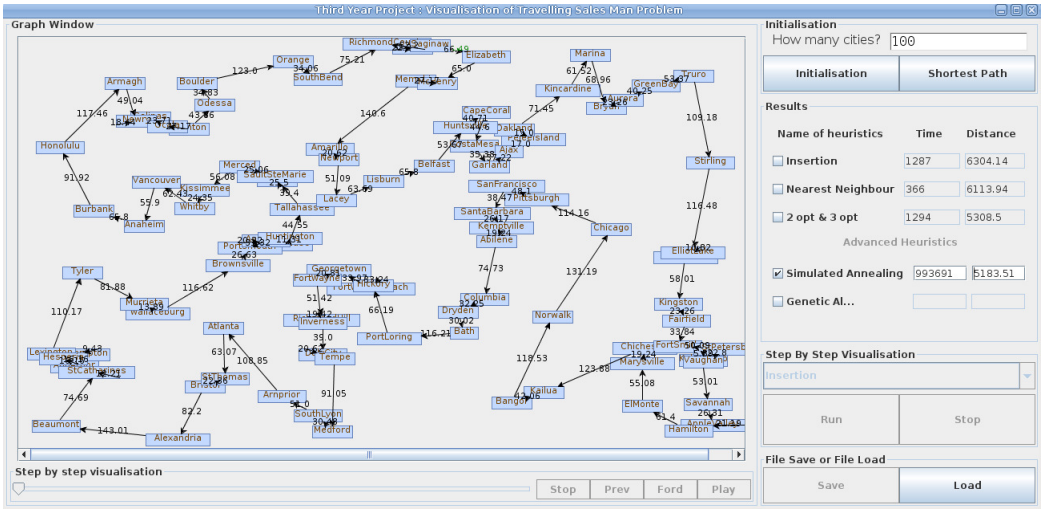


Figure 47: 2-opt and Simulated Annealing

```
Calculate percentage difference,| between V1 = 5121 and V2 = 5308
( | V1 - V2 | / ((V1 + V2)/2) ) * 100
= ( | 5121 - 5308 | / ((5121 + 5308)/2) ) * 100
= ( | -187 | / (10429/2) ) * 100
= ( 187 / 5214.5 ) * 100
= 0.035862 * 100
= 3.5862% difference
```

Figure 48: Simulated Annealing Improvement

8.5.2 Find good schedule and halt value

A test was done and the result is shown on the table 4. The lowest value 5121 and 'halt' was 0.001 and 'schedule' was 0.999. However it took 712118 milliseconds and it was recorded at 712 seconds, this took more than 10 minutes. The second lowest value with different halt and schedule numbers was the highlighted one, 'schedule' was 0.995 and 'halt' was 0.05 and it produced a tour with 5171 distance. It took 59711 millisecond, and was recorded at 60 seconds. 'halt' and 'schedule' values are now fixed as 0.995 and 0.05.
The 3D graphs were generated on 'http://almende.github.io/chap-links-library/js/graph3d/playground/'.

node	schedule	halt	time	distance	iteration
100	0.999	0.001	712118	5121	21406
100	0.999	0.001	762492	5145	21406
100	0.999	0.0005	883168	5154	22099
100	0.995	0.05	59711	5171.53	3493
100	0.9992	0.001	993691	5183	26760
100	0.99	0.001	81966	5242	2131
100	0.995	0.001	148974	5248	4273
100	0.99	0.0005	87724	5252	2200
100	0.99	0.005	46655	5277	1971
100	0.99	0.005	54046	5278	1971
100	0.995	0.03	75907	5308	3595
100	0.99	0.0001	108285	5308	2361
100	0.99	0.0009	81465	5308	2142
100	0.99	0.0009	81465	5308	2142
100	0.95	0.00001	25254	5310	508
100	0.99	0.03	36426	5311	1793
100	0.99	0.003	61474	5316	2022
100	0.995	0.001	153762	5318	4273
100	0.999	0.1	164072	5329	16803
100	0.99	0.03	32713	5468	1793
100	0.995	0.01	105907	5477	3814
100	0.995	0.05	43994	5484	3493
100	0.99	0.001	69894	5484	2131
100	0.995	0.05	43130	5506	3493
100	0.95	0.03	4574	5547	352
100	0.95	0.0001	20681	5554	463

Table 4: SA tests, sorted in accesnding order by distance

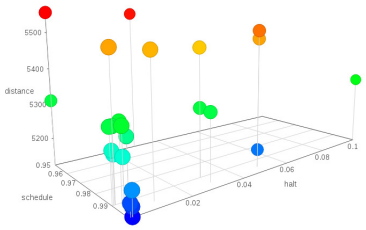


Figure 49: Distance, the lower is better

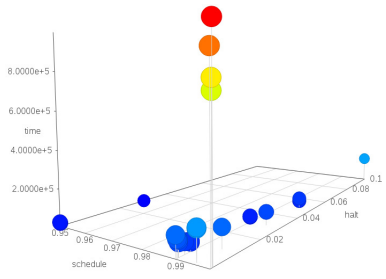


Figure 50: Time, the lower takes shorter time

8.5.3 SA and 2-opt comparisons at different nodes

no. of nodes	2-opt : time	SA : time	2-opt : dis	SA : dis	Improvement
10	99	2760	1937	1937	0
20	528	6012	2223	2178	45
30	574	8237	2773	2773	0
40	236	3266	3626	3602	24
50	180	25677	3974	3903	71
60	310	30892	4232	4101	131
70	682	35806	4360	4191	169
80	581	42721	4883	5012	-129
100	1041	65278	5196	5307	-111
120	1818	67234	5468	5563	-95
140	1882	48560	6641	6688	-47
200	4355	225998	7731	7435	296
300	24250	359786	8899	8946	-47
400	30220	327561	10229	10122	107
500	62961	296263	11840	11858	-18

Table 5: 2-opt and SA : + number means improvement, -number means drawback

Figure 51: 2-opt and SA : Time Comparisons

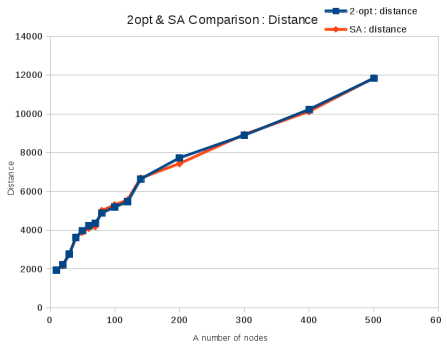


Figure 52: 2-opt and SA : Distance Comparisons

As on table 5, it showed improvements more than the half of the chances but not always. The figure 51 shows that SA takes a significant time to compute results after 140 nodes. At 300 nodes, it somehow takes the longest time and the time decreases afterwards. SA also yields longer distance than 2-opt after 80. The 'schedule' and 'halt' value was fixed based on the experiment done at node 100. This suggests a need to implement a system

that varies 'Schedule' and 'Halt' values according to input it receives in order to expect a better result.

8.6 Genetic Algorithm

Unfortunately, the Genetic algorithm implementation does not promise a good performance as the time complexity was found $O(n^5)$. Between the 11th and 12th, there was 11.11% improvement observed. The following day 0.49% improvement was made. The third day, 0.51% improvement was seen. There was no improvement on the fourth day. It is total 12.00% from a random path. However it is much slower than the time complexity. There could be two reasons, one is that there were too much mutations. The second reason is that each selected individuals are crossovered with a random path. As said in section 2.9, genetic algorithm needs an attention.

Date	Distance	Generation
11/04/14	32824	0
12/04/14	29175	106593
13/04/14	29032	139406
14/04/14	28882	408702
15/04/14	28882	577273

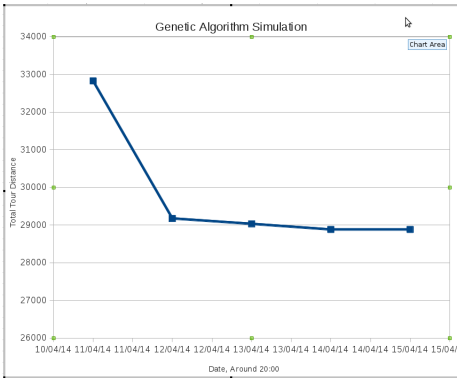


Figure 53: Genetic Algorithm Simulation

> However they are inefficient during iteration and variable conditions are limited [Melanie(1998), pp. 7]

There should be modifications made mentioned above in order to make it more efficient. 'GeneticAlgo' class' lines of codes are 415, 'PathDrawerifSubgraphExist' 284 and 'ProduceIndividualsThread' is 33. The sum is 732. The total lines of code is 4482. It is 16 % of the code. In order to test it until it's fullest termination, time of a week is at least needed. There should be logical optimisations, less mutation and leaving offsprings to next generation rather than dismissing them.

8.7 Extra information of Each Algorithm

8.7.1 Nearest Neighbour

Nearest neighbour does not promise a good result as the last node and the first connected and they are usually located far from each other, so there

often be big X path as on figure 54 and it costs a lot.

8.7.2 Insertion

8.7.3 2-opt Evaluation

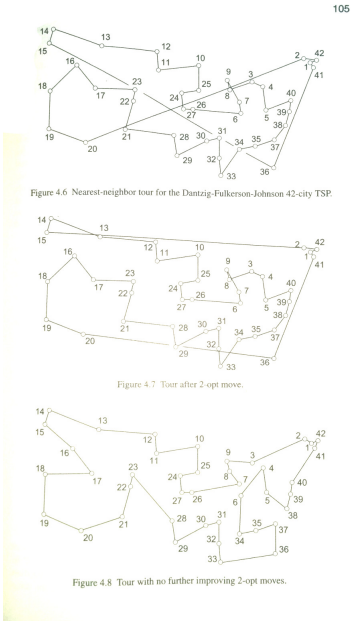


Figure 54: A traditional example of 2-opt move, cited from [David L. Applegate(2006), pp.105]

There is a local minima and sa is used in order to avoid this problem.

8.7.4 Simulated Annealing

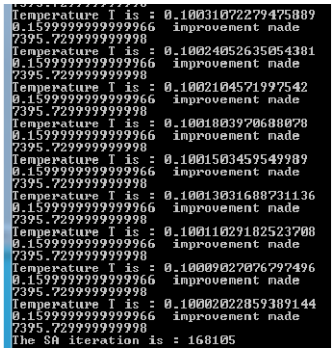


Figure 55: Internal Messages. Temperature and improvement are specified at each iteration. An iteration number is given to a user at it's termination.

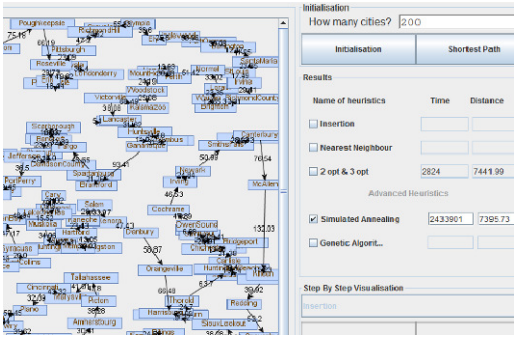


Figure 56: This will be shown to users

8.7.5 Genetic Algorithm

8.8 What was optimised

In TwoOptAlgo class, there is 'repeatSwapping()' method commented. It basically did the same job but hashmap data structure was not used for edge listing and it directly access to graphObject when it add an edge and remove an edge, an instance of GraphGenerator class. For two hundred nodes, it took 4 hours as it takes significant time to do a graph addition and subtraction by mxGraph library. After I used hash map data structure for edge listing, it took 30 seconds for 200 nodes.

8.9 What can be fixed

1. Rather than using JCheck-box for selecting a path generated of each algorithm, JRadio-button is a good alternative. 2. I have been under time pressure and many of the diagrams on this report is hand written and not much neat. That could be fixed by re-drawing electronically. 3. Use of Latex. This is the first time I use latex and it becomes not formal sometime.

8.10 Concorde Project and the traditional way to deal with the TSP

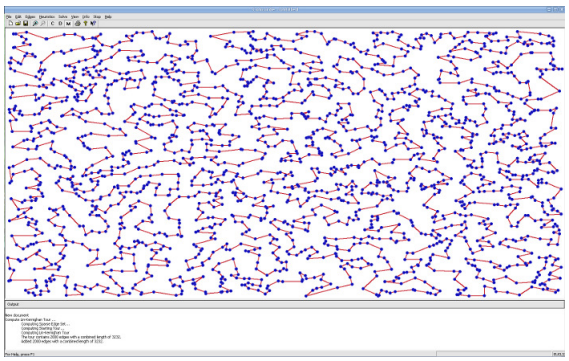


Figure 57: The Concorde program running Lin-Kerninghan Algorithm for 2000 nodes

Apart from the five algorithms, there are several ways to deal with the TSP. The traditional methods are, the Lin-Kerninghan Heuristics, TSP cuts, Branch and cut method, Linear Programming and Bound and Cutting planes. Those were not introduced in this report. David, Cook, Bixby and Chvatal had a project called 'Concorde' for the TSP. They mentioned about the project on their book, stating that

> The developement of Concorde is the center of our TSP research... The software contains approximately 130,000 lines of

> code in the C programming language. This could be downloaded at www.tsp.gatech.edu [David L. Applegate(2006), pp.489]

It is also stated in their book that the Concorde project is for larger and more difficult instances with distributed parallel processing and shared-memory parallel processing. [David L. Applegate(2006), pp.489]

> ..using several years of total CPU time per problem. These instances can now be readily solved with Concorde on a commodity workstation... At this current time, out notion of "very large" refers to problems having over 10,000 cities. [David L. Applegate(2006), pp.506] '

8.11 For A Very Large Number Of Cities

The program was limited not to exceed over 500 for the number of cities chosen by user in order for an user not to wait and become bored. What if the number could be over 500, what would happen?. Tour of Sweden could be an example how to deal with such problem. According to David and Cook in their book, there was a project finding the shortest Hamiltonian tour of 24978 cities. The computation started in the year of 2003 and finished in January 2004. This problem was solved by four branch and Cut. The total computational power was 84.8 CPU years on the clusters of 2.66Ghz Intel Zeon CPUs. This staggering task was quickened by the accurate upper bound found by Helsgaun's tour. [David L. Applegate(2006), pp.516] The Concorde is a good solution to very large instance as described section 8.10

9 Conclusion and Future Work

I have achieved the project's aim that is to an educational software with an intuitive design. It receives a number how many cities a user wants to generate and generate random cities and find shortest path by trying heuristic algorithms. However I have not provided a user virtual tour on the project description stated by Prof Tomasz Radzik, named as 'step by step visualisation' meaning that an icon appears on the screen after computation of an algorithm, it travels around the path generated and the user can step forward and backward pause the tour.
In terms of intuitivity, it is less appealing to a user. I provided actual values of time taken and distance on GUI but the numbers are counted as milliseconds and so it is represented as a huge number. The distance does not use a unit and it may be quite confusing to users. However a user can compare algorithms with the numbers if they are unable to compare graphically with path drawn on GUI. Rather than letting user observe and know which one is the shortest, a green icon can appears next to the name of algorithm, indicating that it is the shortest one.

Although the program is operational, it is not polished in terms of user interface. It does not show a notification dialogue when an algorithm is running and it seems frozen. It also doesn't fully support multi-threading operations. Each algorithm run as a thread but the design is not fully modified for multi-threading. The program has only one big panel for path representation and it was meant to store additional path in memory, unseen place from user. Algorithms could run by either pressing one of check-boxes or 'Shortest Path' button. After the user clicks it and if it is not finished then it prevents user from clicking check-box again and various GUI configuration is automatically made. Alternatively, it can be resolved by generating tabs and stores path in each tab. This has not been fully organised and implemented. Also it provides indicative console messages, the GUI does not show it on the screen and user does not receive enough information unless he runs the program with GUI interface on terminal.
A user might want to resize the screen and zoom in and out. This function has not been implemented.
From October 2013 to February 2014, I have been concentrating on software architecture as it has many components on GUI and it uses five algorithms, implying that the software design is a necessary topic to be thought of and it is fundamentally important as a base of the whole project. 'Command' design pattern for organising algorithms and 'Mediator' pattern for GUI activity management. From February to April, my main focus was to implement algorithms and five algorithms were implemented; Nearest Neighbour, insertion, 2-opt, Simulated Annealing and Genetic Algorithm. I have been writing my report since that time rather than polishing the final stage of the software.
The Genetic Algorithm implementation takes too long time and there should be modifications made in optimisation. In order to reduce $O(n^5)$. The GUI design also should be modified for computation of Simulated Annealing and Genetic algorithm as it requires significant time. For example, the current design does not provide 'Stop' button and the user has wait until it finishes. The user might want to graphically tweak the mutation and the generation value in order to expect a more optimal value.
If I have more concrete software design, a concurrent system and more proper GUI management, I would like to extend the number of city that user can generate from 500 to 3000.

On section 8.10, 'Concorde Proejct' was introduced. That is the standard way to approach the TSP problem. It was developed in the C programming language and open sourced. The Concorde program is a sublime example with distributed parallel processing and shared-memory parallel processing. [David L. Applegate(2006), pp.489] The Concorde project was seen by me in April 2014. As a person accustomed to the C programming language and the Concorde should have been seen and studied. I wish I studied it before implementing the core features of my program. Throughout the

whole year, I realised how important data structure and algorithm are in software development as I discovered a significant performance difference discussed in section 8.8. I also realised what 'Searching' means in computer science and mathematics. If there is a formula like a magnet that attract a needle in haystack that means P = NP. Searching is the essential topic. The P versus NP problem is one question out of ten millennium problems, that which the list was made by the Clay institute with 1000000 US dollar prize. Nevertheless, the P versus NP problem is worth more than 1000,000 US dollars as mentioned in section 2.5.

Figure 58: Cited from [giantbomb(2013)]

References

[1 M Gendreau(1992)] . G. L. 1 M Gendreau, 2 A Hertz. *New Insertion and Post Optimiation Procedures for the Traveling Salesman Problem.* 1992.

[2 David S. Johnson(1995)] . L. A. M. 2 David S. Johnson. *The Traveling Salesman Problem: A Case Study in Local Optimization.* PhD thesis, Amherst College, 1995.

[2 Dimitris Bertsimas(1993)] . J. T. 2 Dimitris Bertsimas. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[2 Justin Chester()] . A. W. 2 Justin Chester. Tour planning for uav data mules in border wireless multimedia sensor network. http://www.cs.utsa.edu/ korkmaz/grant/dod/posters/TourPlanning.pdf.

[3 Sanjoy Dasgupta(2006)] . U. V. 3 Sanjoy Dasgupta, 2 Christos Papadimitriou. Algorithms. University Lecture notes, `http://www.cs.berkeley.edu/~vazirani/algorithms/all.pdf`, 2006.

[Array(2005)] A. T. C. Array. New user information. available at, `www.narrabri.atnf.csiro.au`, 2005.

[B Golden and Stewart(1980)] T. D. B Golden, L Bodin and W. Stewart. *Approximate Traveling Salesman Algorithms.* 1980.

[Bertrand Neveu(2003)] G. T. Bertrand Neveu. *INCOP: An Open Library for INcomplete Combinatorial OPtimization.* PhD thesis, Projet COPRIN, CERMICS-I3S-INRIA Route des lucioles, BP 93, 06902 Sophia Antipolis France Bertrand.Neveu,Gilles.Trombettoni@sophia.inria.fr, 2003.

[Christos H. Papadimitriou(1998)] K. S. Christos H. Papadimitriou. *Combinatorial Optimization : Algorithms and Complexity.* 1998.

[Clausen(1999)] J. Clausen. *Branch and Bound Algorithms - Principles and Examples.* PhD thesis, University of Copenhagen, 1999.

[Cook(1971)] S. Cook. The complexity of theorem proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, page 151–158, 1971.

[Cook(2000)] S. Cook. The p versus np problem. The official problem description, `http://www.claymath.org/sites/default/files/pvsnp.pdf`, 2000.

[David L. Applegate(2006)] W. C. David L. Applegate. *The Traveling Salesman Problem A Computational Study.* 2006.

[Edberg(2005)] C. A. B. Edberg, M. Shao. Sim planetquest: A mission for astrophysics and planet-finding. white paper, may 2., 2005.

[Erich Gamma(1994)] R. J. J. V. Erich Gamma, Richard Helm. *Design Patterns Elements of Reusable Object-Oriented Software.* 1994.

[Gambardella(2009)] B. L. M. D. L. M. Gambardella. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing an international journal 8 2*, 2009.

[giantbomb(2013)] giantbomb. needle. available on Youtube, `http://static.giantbomb.com/uploads/original/1/17172/1074014-haystack.jpg`, 2013.

[Glover()] F. Glover. Tabu search: A tutorial.

[Iris Van Rooij(2003)] A. S. Iris Van Rooij, Ulrike Stege. Convex hull and tour crossings in the euclidean traveling salesperson problem: Implications for human performance studies. *Memory and Cognition*, (31(2)): 215–220, 2003.

[Jain A. K.(1999)] P. J. F. Jain A. K., M. N. Murty. *Data Clustering: A review.* 1999.

[JGraph(2014)] JGraph. Jgraphx (jgraph 6) user manual. available at, `http://jgraph.github.io/mxgraph/docs/manual_javavis.html` on the 13th of April, 2014.

[Karumanchi(2012)] N. Karumanchi. *Data Structures And Algorithms Made Easy In Java.* 2012.

[Kurt(1956)] G. Kurt. The gödel's letter. Translated and publised by Michel Sipler in his book, `http://rjlipton.wordpress.com/the-gdel-letter/`, 1956.

[Lin(2005)] J.-S. Lin. Christofides' heuristic. University Lecture notes, `http://ieor.berkeley.edu/~kaminsky/ieor251/notes/2-16-05.pdf`, 2005.

[Lin S()] B. W. K. Lin S. *An effective heuristic algorithm for the traveling-salesman problem.*

[Melanie(1998)] M. Melanie. *An Introduction to Genetic Algorithms.* MIT Press, 1998 edition, 1998.

[Mostafa Rahimi Azghadi(2008)] M. R. B. Mostafa Rahimi Azghadi. Travelling salesman problem. In F. Greco, editor, *Population-Based Optimization Algorithms for Solving the Travelling Salesman Problem*, page 11. I-Tech, 2008.

[Paul Deitel(2010)] H. D. Paul Deitel. *Java How to program, Eighth Edition.* 2010.

[Qef(2012)] Qef. Complexity subsets pspace. available at, `http://en.wikipedia.org/wiki/File:Complexity_subsets_pspace.svg` on the 4th of December, 2012.

[Sally(2007)] J. D. Sally. *Roots to Research: A Vertical Development of Mathematical Problems.* 2007.

[Schrijver(2005)] A. Schrijver. *On the history of combinatorial optimization (till 1960).* PhD thesis, University Of Amsterdam, 2005.

[Scott Aaronson(2014)] G. K. Scott Aaronson. Complexity zoo. available at, `www.narrabri.atnf.csiro.au` on the 22nd of March at 11:41, 2014.

[Sipser(1992)] M. Sipser. *The History and the Status of the P versus NP Question.* 1992.

[Sipser(1996)] M. Sipser. *Introduction to the Theory of Computation.* 1996.

[Sipser(2011)] M. Sipser. Beyond computation: The p vs np problem. available on Youtube, `http://youtu.be/msp2y_Y5MLE` on the 25th of November, 2011.

[van Rooji I(2003)] H. K. van Rooji I, A. Schactman. *Chilerens performance on the Euclideantraveling salesperson problem. Technical report.* 2003.

[Wikipedia(2014a)] Wikipedia. Efficiency comparison with other data structures. available at, `http://en.wikipedia.org/wiki/Array_data_structure` on the 16th of April, 2014a.

[Wikipedia(2014b)] Wikipedia. Hash table. available at, `http://en.wikipedia.org/wiki/Hash_table#cite_note-Cormen_et_al-1` on the 16th of April, 2014b.

[Yang(2010)] X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms.* Luniver Press, second edition edition, 2010.