Exploring the Performance of Optimization Algorithms When Applied to Real-World Travelling Salesman Problems

Robbie McNeil

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc Computing Science

School of Computing

April 2020

Authorship Declaration

I, Robbie McNeil, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project, I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed: Robbie McNeil

Date: 07/04/2020

Matriculation no: 40296075

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

Robbie McNeil

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

In this project we are exploring which optimisation techniques are the best when solving real-world travelling salesman problems, a famous combinatorial optimisation problem which requires finding the shortest distance around a set of cities, where each city must be visited exactly once. In total there has been six different optimisation algorithms implemented, nearest neighbour, 2-opt, hill climber, simulated annealing, tabu sear and evolutionary algorithm, each algorithm was implemented in its most basic form (i.e. there were no hybrid algorithms which utilise local search techniques to produce good initial solutions). These algorithms are used to solve four real world TSPs which take place within the UK, the TSPs range from 10-10,000 vertices, this wide range of vertices was chosen to show how each method copes with an increasing problem size. Each algorithm is able to use the Haversine formula to calculate the distance between vertices, after solving a TSP, each algorithm outputs the completed tour to a file so that the final route can be calculated by GraphHopper. GraphHopper is a routing software that is able to convert road networks into edges and calculate the shortest distance between two or more vertices by using roads at edges, this software produces more precise solutions which more closely resembles a solution which would be used in a real-world situation. By evaluating each algorithm's average distance and runtime against each other the conclusion was reached that the simpler algorithms such as nearest neighbour and 2-opt were able to produce far better results when compared to more complex algorithms such as tabu search and evolutionary algorithm.

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr Brian Davison, for his continued support and guidance over the duration of this dissertation. His help was critical to the success of the project.

I would also like to thank my parents, Lisa and Scott, my brother Tom and all of the friends that I've made during my time at Edinburgh Napier university. Although I cannot thank everyone individually, the unwavering encouragement and reassurance my friends and family have provided over the past four years has helped me make it this far.

1 Introduction

1.1 Background

Optimization is a technique of refining a process to make it as efficient as possible. This may include minimising the number of resources used (such as money or fuel), reducing the total time that a process takes or maximising the output of a process. There are many applications that optimization can be applied to in order to make more efficient, because of this optimization techniques have been adopted in many areas of the real world, from engineering to economics. For many real-world problems there are an extremely large amount of possible solutions available, some good and some bad, and generating and comparing all solutions is usually infeasible. There are a large number of optimization techniques available today which aid in finding good solutions to these problems, some are problem specific and only work for one application, while others have the ability to be modified to suit many different needs.

Problem-dependent techniques are known as heuristics, these can be thought of as a ‘rule of thumb’ which can take advantage of having knowledge about the problem in order to produce solutions. Heuristics typically benefit from being simple and fast, because of this they can be a good first choice when choosing an optimization technique. They can be relatively easy to implement, especially when compared to other optimization techniques. The trade-off is that heuristics are not as flexible as other methods, they can only be used in the context that they were designed for. Heuristics are typically less accurate than other methods, often producing solutions which are not as optimal as other optimization techniques.

Problem-independent techniques are known as metaheuristics, unlike heuristics, these techniques have no knowledge about the problem they are solving. Metaheuristics take an instance of a problem as an input and produces a solution as an output, with no indication as to how that solution was formulated, it is for this reason that metaheuristics can be viewed as black box optimizers. Metaheuristics are much more complex than heuristics but are also much more flexible, this means although they are more difficult to understand and to implement, they can be modified to solve a much wider range of problems. They are also more likely to produce better results than

heuristics as many metaheuristic optimizers utilise methods which allow them to explore much more of the search space, such as the ability to escape from local optima. Escaping from local optima means that the optimizer must first accept a worse solution in an attempt to find a better solution later on.

1.2 Aims

The first aim of this project is to implement optimization algorithms and use them to solve real-world TSPs. Several different algorithms will have to implemented as there will be no benchmarks available for the specific real-world TSPs included in this project, therefore the algorithms will need to be compared against themselves when it comes to the evaluation. Each algorithm will solve four different TSPs, which will be represented as datasets that are inputted to the algorithms. These datasets will vary in size, having between 10 and 10,000 locations, with each subsequent dataset scaling by a factor of 10. By scaling the problem size by a factor of 10 we will be able to see how each algorithm's performance scales with the problem size. There will be a mix of algorithms implemented, some being heuristics and others being metaheuristics, this will allow us to see how a heuristic with problem specific knowledge is able to compare to non-deterministic metaheuristics which make changes to solutions randomly. Although simple in nature, the ability to utilise problem specific knowledge may give heuristics an advantage. In order to achieve the best possible performance, some of the metaheuristics will have to undergo parameter tuning where appropriate.

Next, I need to utilise pre-existing routing software that is capable of taking a large list of vertices and optimising the tour by finding the shortest route between all vertices. This software must not change the order in which the vertices are visited, as the algorithms discussed above will be generating the tours and will be deciding in which order the vertices are visited, instead the routing software will simply give us a more precise and realistic distance value. This can be achieved by forgoing the 'straight line' edges produced from the optimisation algorithms and instead finding the shortest distance between two neighbouring vertices via road. By using roads to connect all of the vertices we can make the answer to our problem less abstract, this allows us to find a real-world answer to our real-world problems.

For each algorithm two values will be recorded. The distance of each tour, which will be generated by the routing software, will be recorded as the main objective of the

TSP is to find the shortest route around a set of locations. The runtime of the algorithms will also be recorded, this will be achieved by timing how long each algorithm takes to run for certain amount of iterations. We are recording this value as real-world problems may require a TSP to be able to be solved in as little time possible. With these values we can then evaluate the performance of each algorithm and compare them against each other. As this project is researching the performance of optimization algorithms when solving real-world TSPs, this is the most important aim as it will allow us to see how the algorithms compare to each other as well as being able to assess how well each algorithm copes with an increasing problem size.

1.3 Scope

As expected, the vertices held within the datasets will consist of longitude and latitude pairs that correspond to locations on Earth. This is an important step in making the TSPs represent real-world problems, as an abstract TSP which uses points on a grid cannot be easily translated into longitude and latitude points as doing so would require turning a 2-D grid into a 3-D sphere. To find the distance between two coordinates on the sphere the algorithms will find the distance it would take a person to travel in a straight line from one location to the next. This is obviously something that could not realistically be done in the real world, hence the need for external routing software which will take the tours and translate them into real-world routes.

The algorithms that will be implemented will not hybridized with any other algorithm. Although it is quite common to implement additional search features into optimisation algorithms, I have decided against this, the reason being that I would like to explore how well each algorithm performs on its own without having to rely on a different algorithm for assistance.

In projects such as this it is often common to use benchmarks; these allow developers to compare the performance of their algorithms against other algorithms as well as compare against the optimal solution. This will not be done in this project as the online libraries which store benchmarks, such as the TSPLIB[1], do not contain real-world TSPs, instead, I will simply be comparing the performance of the algorithms against each other.

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Although I would like to allow each algorithm enough time in order to fully evolve an answer and converge towards an optimum, this will unlikely not be possible, due to the large amount of runs each algorithm will have to perform. The larger the problems are, the longer the algorithms will take to run. While also taking parameter experimentation and having to run each algorithm numerous times into account, allowing each algorithm enough time to converge is not a possibility as it would take far too much time to find an optimum as the size of the problem reaches thousands of locations to be visited. Therefore, the algorithms will be allowed a 5-minute time limit to generate solutions when gathering distance values and will be allowed 1,000 iterations when gathering runtime values.

2 Literature review

2.1 Travelling Salesman Problem

The Travelling salesman problem (TSP) is a well-known combinatorial optimization problem in mathematics and computing which was coined in 1934 by Hassler Whitney (Flood, 1956). The TSP consists of a salesman and a set of vertices and edges, the objective is to find the minimum cost route of passing through each vertex exactly once before returning to the starting position. There are two main types of TSP, symmetric (STSP) and asymmetric (ATSP). If the distance from vertex *i* to vertex *j* is the same as the distance from vertex *j* to vertex *i* then the TSP is symmetric, otherwise it is asymmetric (Deng, Liu, & Zhou, 2015). Figure 1 is an example of a STSP, in this example the vertices are represented as circles and the edges are represented as lines which connect vertices. Each edge has an associated number which tells us the distance of the edge. The objective is to start at the starting depot and then visit all of the vertices from A-L once before going back to the staring depot in the shortest distance possible. The distance is calculated by summing the values of each edge that has been travelled down.

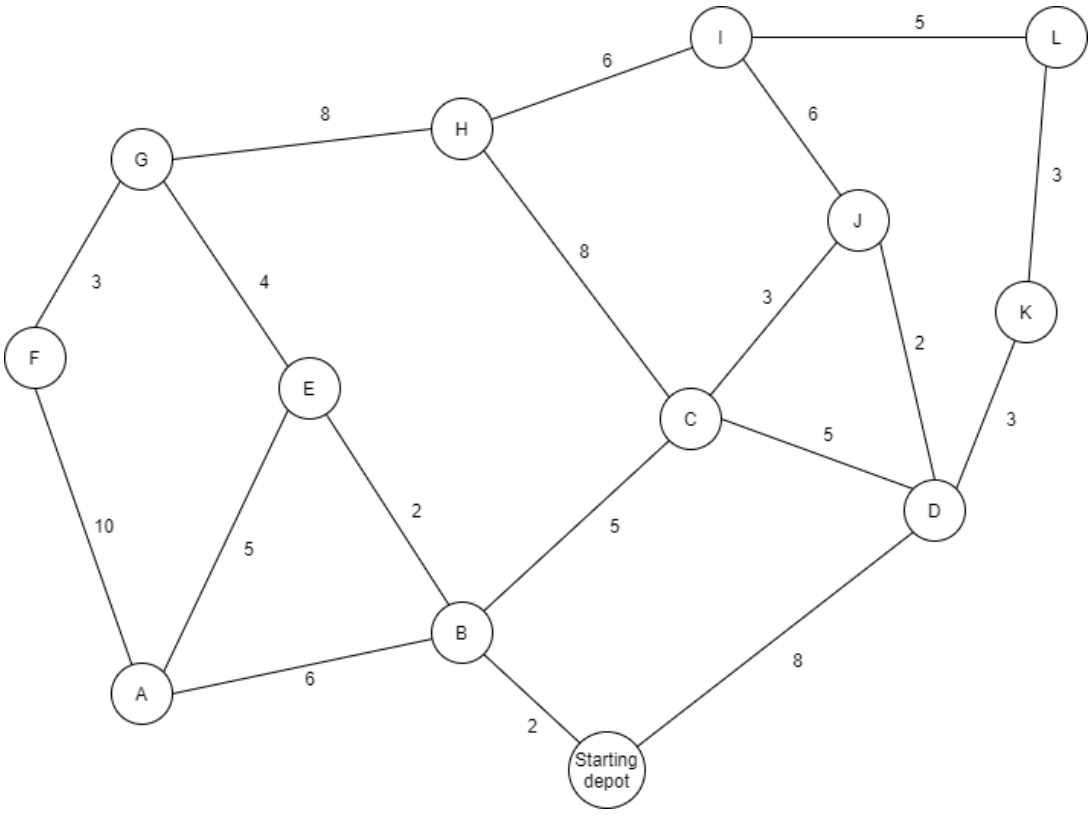


Figure 1 – Example TSP network

2.1.1 Evaluation Metrics

Typically, the objective of a TSP is to find a tour which minimises the total distance. A tour is a complete solution to the TSP, where every vertex has been visited only once and the salesman has returned to the starting depot. The distance between vertices depends on the type of TSP being solved, for abstract TSPs which take place on a 2D graph Euclidian distance is commonly used, which is the straight-line distance between two points. For TSPs which use real locations on Earth as vertices we must use a method which can calculate the distance between two points on a sphere, rather than two points on a 2D plane. The Haversine formula can be used to find the great circle distance between two points on a sphere, this is the shortest distance connecting two points that travels along the surface of a sphere, which in this case is Earth. It is important to note that in this situation the Haversine Formula will assume that the Earth is a perfect sphere, which it is not, because of this there will likely been some inaccuracies in the distance value returned. All that is required to calculate this distance is two pairs of longitudes and latitudes, as well as the radius of the circle. The formula (Piarsa, Hadi, & Wirdiani, 2015) is as follows:

$$a = sin^2\left(\Delta\varphi/_2\right) + \cos(\varphi_1).\cos(\varphi_2).sin^2\left(\Delta\lambda/_2\right)$$

$$c = 2.atan2(\sqrt{a}, \sqrt{1-a})$$

$$d = R.c$$

Equation 1 – Haversine formula

Where R is the radius of the sphere, λ and φ are longitude and latitude respectively and d is the distance. For this formula both longitude and latitude must be represented in radians rather than degrees, if they are in degrees then they can easily be converted to radians by multiplying the value by π/180. This formula has been implemented into numerous studies about routing and navigation - Nordin et al. (2012) used this method for their ambulance routing project. They state that the Haversine formula gives the exact distance, which is why it is commonly used for routing projects. This method may find the exact distance of a perfect sphere, however, when this model is applied to the Earth some inaccuracies may occur, this due to the Earth being an ellipsoid rather than perfectly spherical (Piarsa et al., 2015). Other means of calculating great circle distance can also be used to calculate the distance between two vertices, such as the spherical law of cosines, although these other methods are not often used in the

literature. In a real-world situation a route between two vertices would not follow the line of shortest distance as given by the great circle distance, instead roads must be used. This will cause more inaccuracies in the distance values produced by the Haversine formula.

The travelling salesman problem with time windows (TSPTW) is a variation of the TSP where each vertex has a service time and a time window that define which time the salesman can arrive, if the salesman arrives too late at a given vertex the tour is infeasible, if they arrive too early the salesman must wait for the customer to become available. This instance of the TSP is useful for simulating real-world delivery systems as customers can oftentimes pick a delivery window for when they are available (Karabulut & Fatih Tasgetiren, 2014). The 'makespan' or the time to complete a tour of all the vertices is commonly used as the evaluation criterion rather than the total distance for this type of problem.

The use of CO_2 emissions as an evaluation criterion has been explored where the objective is to find the tour which produces the least amount of CO_2. Urquhart, Scott, & Hart (2010) implemented a fuel consumption model into a TSP, the model took variables such as distance, type of vehicle and acceleration into account to find accurate emission data for multiple tours. Using this model, they were able to find tours which decreased the amount of CO_2 emitted.

In some cases, it may be necessary to optimise several objectives simultaneously, this is known as a multi-objective travelling salesman problem (MOTSP). The MOTSP is a more complex problem as many times objectives will conflict with one another and a solution which optimises one solution will decrease the fitness of another objective, solutions with this property are called 'Pareto optimal solutions' (Delgado-Antequera, Laguna, Pacheco, & Caballero, 2019). If the objectives of a MOTSP do not conflict with one another then an optimal solution which optimises all objectives can be found, otherwise no one optimal solution can exist and instead a set of Pareto optimal solutions are used instead (Coello, 2005). If the solution to a MOTSP is better than another solution in all areas and better in at least one then we say that the better solution dominates the poorer solution, a set of pareto optimal results is a set of non-dominated solutions, the user must take all the solutions into account and decide which one is most appropriate for the given problem.

2.1.2 Complexity

The TSP and its numerous variations are considered very computationally complex as the number of possible solutions increases exponentially with the number of vertices and there currently is no algorithm which can find the optimal solution in polynomial time (Juneja et al., 2019). Because of this, combinatorial optimisation problems fall under the category of being NP-hard, this is in reference to that fact that the problems are solvable in nondeterministic polynomial time. This means that as NP-hard problems get more complex (i.e. increasing the number of vertices in a TSP) the time take to solve those problems increases. However, once a solution to the TSP has been found you can quickly verify whether it is a feasible solution. Solutions to the TSP can be evaluated by their fitness. The fitness of a solution is a numerical value that represents how good the solution is and is produced via a fitness function. Solutions which have a better fitness are more desirable than solutions with a worse fitness, by assigning solutions a fitness we can quantify the quality of the solution and compare it to other solutions. The fitness function evaluates solutions based on criteria which determines how close it is to the optimum solution. For a standard TSP an appropriate fitness function would calculate fitness based of a solution's distance, with shorter distances being rewarded with a higher fitness. However, depending on the problem a fitness function may incorporate more than one type of criterion to help evaluate solutions, for example a TSPTW fitness function may evaluate distance as well as number of missed delivery windows. Typically, solutions for a TSP are represented as a permutation of the set of vertices which needs to be visited, the formula for the number of possible solutions for a typical STSP is as follows:

$$\frac{n!}{2n}$$

Equation 2 – Number of permutations for STSP

Where *n* is the number of vertices. We get this formula as the possible number of permutations is *n!*, however each permutation can be represented in *2n* ways. For example, if we had the following permutations:

p_1 = (1-2-3-4-5)

p_2 = (2-3-4-5-1)

These are two ways of representing the same tour, although they look different the order of the vertices is same in both permutations. This can only be the case for a STSP, as the edges which connect vertices in an ATSP may be one-way or have different values depending on the direction of travel. Therefore, there is no guarantee that each permutation can be represented in *2n* ways for an ATSP.

2.2 Applications

Though often implemented as abstract problems, TSPs have practical uses in real-world applications, the most obvious of which is vehicle routing. TSPs can be used to model routing problems for a single vehicle, such as finding an optimal tour for a bus around a set of stops or finding the most efficient route for a home delivery driver. Algorithms which are used to solve these problems can be useful for satellite navigation systems.

There are also applications outside of vehicle routing that these problems can be applied to. An example of this is x-ray crystallography, when x-rays are fired at a crystal sample they are scattered by the crystal due to diffraction, by taking measurements of the angles and intensity of the scattered x-rays a crystallographer can determine the electron density of the crystal. These experiments can have 5,000 to 30,000 readings and thus can be highly time consuming. By translating the problem of sequencing the x-ray readings into a large-scale TSP Bland & Shallcross (1989) were able to implement TSP heuristics and found that by doing so they made considerable improvements over the previous method of sequencing the readings.

Another use of the TSP is drilling and component placement sequencing on printed circuit boards (PCB). Component placement sequencing is the order which components are placed onto the circuit board, for a majority of machine architectures a TSP is used to solve the placement sequencing problem (Duman & Or, 2004). Finding an optimal sequence is an important job as some sequences may result in the components being damaged which will result in a faulty circuit board. The drilling problem is concerned with finding the optimum way to drill through the PCB to reduce cost and maximise productivity, this may include drilling through multiple stacked PCBs at once or reducing the length of the drill path between holes by applying TSP heuristics (Ancău, 2008). Both these applications of the TSP have seen successful results when compared to conventional methods, with the former showing a 97.5% decrease in damaged circuit boards and the latter showing that TSP heuristics were

able to improve upon conventional methods when trying to find the shortest drill path length between holes in a circuit board.

2.3 Approaches

In early studies of the TSP, permutation matrices were used to find solutions as no algorithms existed which could solve the problem (Flood, 1956). Since then numerous methods have been developed which can find solutions to the TSP without the use of matrices. These methods can be separated into two distinct categories: local search methods and global search methods.

Local search methods start with a candidate solution or set of solutions and then makes random local changes to those solutions. Because local searches utilise random changes to the candidate solutions these methods are not guaranteed to find the global optimum, however local search methods are commonly used to solve hard combinatorial problems such as the TSP and are well documented by the literature (Jadon & Datta, 2013; Nguyen, Yoshihara, Yamamori, & Yasunaga, 2007; Osaba & Diaz, 2012; Ye & Rui, 2013).

Global search methods attempt to find either the global maximum or minimum. Although the main objective of optimisation is to find the most optimal solution, for situations where we have a large search space global search methods may be unsuitable due to time complexity of certain algorithms (Xu, Pei, & Zhu, 2018). An example of this would be a brute-force algorithm, which always finds the optimal solution by searching through each possible solution and picking the solution which has the best fitness. Despite always finding the optimal solution, brute-force algorithms are only useful for solving TSPs with small amounts of nodes, as shown by Singh & Chopra (2012) a brute-force algorithm used for a TSP with 16 nodes would take an estimated 2 years to solve, when the number of nodes is increased to 20 the estimated time becomes 193,000 years. This is due to the extremely large number of calculations that the algorithm would have to perform, if we use Equation 2 from section 2.1.2 we can see that a 16 vertex TSP has 6.538×10^{11} possible permutations, while a 20 vertex TSP has 6.082×10^{16} possible permutations. This demonstrates the exponential complexity of the TSP discussed earlier and why it is infeasible to calculate every possible solution for most TSPs.

Once a search method has been implemented, we can perform tests to show how well it performs. Many researchers use the TSPLIB[2] for their studies which is an online library that consists of benchmark TSP instances, this is a useful resource as the optimum solutions are often documented, meaning that developers can assess how their algorithm's results compare to the optimum as well as other users' results.

2.3.1 Dijksta's Algorithm

Dijksta's algorithm, also known as the single source shortest path algorithm, is an optimisation algorithm that is used to find the shortest path around directed and undirected graphs. This algorithm starts at an initial position, the initial position is given a distance value of 0, while all unvisited vertices get assigned a distance of infinity. The algorithm then considers all of the neighbouring vertices and updates their distance values, a neighbours' distance is equal to the current vertex's distance plus the weight of the edge connecting the two vertices. Once all neighbours have been considered the algorithm moves to the node with the shortest distance, from here the algorithm repeats the previous step, calculating the new distances for all neighbours. If the algorithm calculates the distance of a neighbour and its new distance is shorter than its current distance then the current distance is replaced, otherwise the new distance is discarded. The algorithm repeats this process until all of the nodes have been visited, once this has happened the algorithm will be able to determine the shortest path from the start node to the goal node.

In the literature, Dijkstra's algorithm has been effectively applied to real world TSPs, such as home delivery services (Ginting, Osmond, & Aditsania, 2019; Sahputra, Devita, Siregar, & Kirana, 2016). Both of these studies found that Dijkstra's algorithm was able to find the optimal route for their respective problems, with Sahputra, Devita, Siregar, & Kirana (2016) finding that their Dijksta's algorithm was on par with Google Maps in terms of route distance.

2.3.2 A* Algorithm

The A* algorithm is an extension of Dijksta's algorithm, the two algorithms both work in the same way, starting from one position and then assigning distances to all of the current vertex's neighbours. Both algorithms then move to the neighbour which has the smallest distance and the process is repeated until the shortest path has been

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

found. A* has an extra heuristic which allows it to make better decisions, this heuristic is able to determine how far each node is from the goal states. When assigning the neighbouring vertices a distance value, A* will determine the value by adding the distance from the starting position to the distance to the end position. This means that vertices which are further away from the goal will be weighted higher, which will discourage the algorithm from travelling down that path. Depending on the implementation, the A* algorithm may even be able to prune vertices that would lead to bad solutions that Dijkstra's algorithm would otherwise expand. Because A* is an improvement upon Dijkstra's algorithm, it too has been used to solve real-world TSPs such as home delivery services (Jalaluddin, Kamaru-zaman, Abdul-Rahman, & Mutalib, 2017). Jia et al. (2016) claims that when applied to pathfinding/graph traversal problems A* is often outperformed by algorithms which are able to pre-process the graph but despite this, A* does offer some improvement over other methods.

2.3.3 Nearest Neighbour

The nearest neighbour (NN) algorithm is a greedy heuristic which constructs solutions to the TSP by always selecting the nearest, unvisited city in relation to the salesman's current city. NN makes decisions based purely off local information (the distance to every city it can travel to from its current position) and is unable to take the future into account, meaning that the NN algorithm is susceptible to make decisions in the present that will lead to worse solutions in the future. Because NN works in a deterministic way, if it always starts at the same depot then it will always produce the same solution.

NN isn't guaranteed to find an optimal solution but can produce reasonable solutions in a short amount of time, however, as the scale of the problem increases, the performance of the algorithm worsens (Ahrens, 2013). NN is not often used to solve TSP problems on its own but has been incorporated into other techniques which has led to improved solutions, for example, the NN algorithm can be used to produce candidate solutions, which other algorithms can then manipulate. With many metaheuristic algorithms candidate solutions are produced at random, so there is a chance the candidate solution will have a very poor fitness, and this may affect the quality of the final solution. By allowing the NN algorithm to produce the candidate solution there is a much higher chance that the initial fitness will be higher, providing a good starting point for the next algorithm. This is how the hybrid algorithm developed

by Dhakal & Chiong (2008) operated, from their experiments we can see that the Hybrid NN algorithm outperforms the standard NN in terms of solution quality, with an average improvement of 13.28%.

2.3.4 2-Opt

2-Opt is a popular local search technique that is often used when solving the TSP. When applied to the TSP, 2-opt takes a complete tour and swaps two cities in order to generate a new, shorter route. 2-opt can also be described as breaking two edges in a graph and reconnecting the graph with two cheaper edges, which can be seen in Figure 2.

The popularity of 2-opt is probably due to the versatility of the algorithm as well as simplicity, there are examples in the literature of 2-opt being used on its own as well as a hybridized algorithm. Sathyan, Boone, & Cohen (2015) compared the performance of 2-opt against a 2-opt and nearest neighbour hybrid algorithm and found that although they performed similarly in terms of solution quality, the hybridized algorithm improved upon the 2-opt's computational time (the time taken to produce the final solution). More complex algorithms were also tested including particle swarm optimisation (PSO) and an evolutionary algorithm (EA), which had slower computational time compared to both 2-opt algorithms. On average, 2-opt also found better quality solutions than the PSO. The TSPs used in this study all had 50 cities, from this we can conclude that the 2-opt method can be beneficial when used for smaller problem instances.

2-opt has also been applied to EAs as a mutation operator, after the EA has generated a new solution there is a chance that solution can be improved on by applying the 2-opt operator. Sedighpour et al. (2011) found that by implementing the 2-op mutation operator the EA was able to produce better results for large-scale problems when

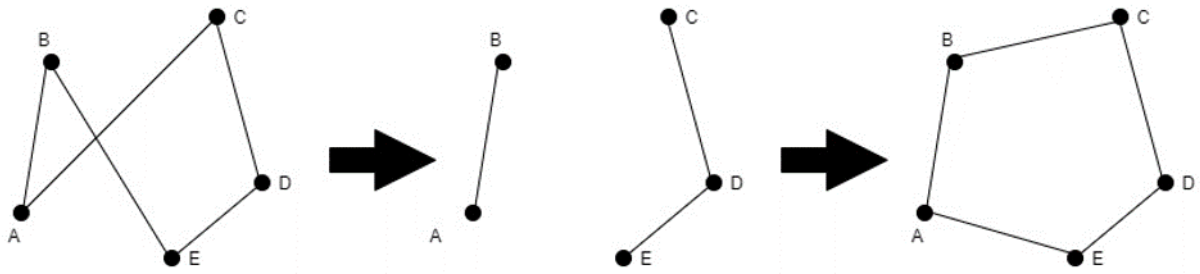


Figure 2 – 2-Opt example

compared to previous algorithms. Singhet al. (2015) compared the use of a 2-opt and nearest neighbour hybrid as a mutation operator against traditional mutation operators such as inversion mutation, insertion mutation and exchange mutation, with the results showing that the hybrid 2-opt performed much better than the other methods.

2.3.5 Hill Climber

A hill climber (HC) algorithm is a simple local search technique which is easy to implement and can produce solutions quickly, although it is unlikely to produce an optimal result. It starts with a randomly initialised solution, from here it uses a move operator to produce new solutions. There are two common move strategies that HCs usually utilise, first-improvement and best-improvement (Basseur & Goëffon, 2015). First-improvement applies the move operator until it produces a solution which improves upon its current solution, it then replaces the old solution with the new one. Best-improvement produces and compares the entire neighbourhood of the current solution and then chooses the best one to replace the current solution. The HC algorithm repeats the move strategy until it converges to a maximum or minimum. Best-improvement is much more computationally expensive option but can converge to an optimum in fewer iterations than a first-improvement strategy.

Because the HC is always trying to improve upon its current solution if it gets to the top of a local maximum or the bottom of a local minimum then the algorithm will get trapped, as there are no better solutions to move to, and it is incapable of accepting worse moves. This makes the HC better when applied to problems with few maxima or minima, as there is less chance of getting stuck. HCs can also get caught in plateaus, flat areas in the search space, to overcome this some additional strategies may be implemented which allow the HC to accept neutral moves (those which neither improve nor worsen the current solution).

Despite the HC algorithm being simple to understand and implement it isn’t often used to solve TSPs, because of this it is hard to evaluate how well this local search technique performs in relation to other techniques. Despite its flaws, the hill climber algorithm can be a useful method to obtain quick results, especially if the overall fitness is not too important.

2.3.6 Simulated Annealing

Simulated annealing (SA) is a metaheuristic which is modelled after annealing, a technique which involves a material starting with a high temperature which is then slowly cooled. The algorithm starts with an initial solution and a temperature value, the algorithm then uses a move operator to choose a new solution from with its neighbourhood. If the new solution is better than the previous one the algorithm keeps it, if the new solution is worse than the previous one then the algorithm uses the temperature value of the system to determine the probability of keeping the new solution. The higher the system temperature is, the greater the chance that the worse solution will be accepted. After each iteration of the algorithm the temperature is decreased, which gradually lowers the chance of picking worse solutions in the future. By allowing the algorithm to pick worse solutions simulated annealing improves upon the hillclimber algorithm by offering a way of escaping from local maxima or minima in order to potentially find a better result. By the time the algorithm is finished it should be left with a set of near optimal solutions.

In a study conducted by Adewole et al. (2012), which compared simulated annealing and an evolutionary algorithm when solving a TSP, simulated annealing was able to find good quality solutions much faster than the evolutionary algorithm, and the difference in solution quality was small enough to consider simulated annealing a viable method for solving the TSP.

Although the SA algorithm can be an effective method of finding good TSP solutions, the algorithm has difficulty solving large TSP instances of over 1000 cities (Geng, Chen, Yang, Shi, & Zhao, 2011) due to the time complexity of the algorithm. Hansen (1992) found that the run time (T_n) of the SA algorithm has time complexity:

$$T_n = O((n^2 + n)\log n))$$

Equation 3 – Simulated annealing big O notation

Simulated annealing is said to be able to find high quality solutions at the expense of time. The way to combat this disadvantage is to implement other local search techniques into the SA algorithm. Geng et al. (2011) implemented a greedy search technique into a SA algorithm. Greedy search techniques are fast and converge quickly but produce less optimal results than other methods, NN is an example of a greedy local search, by hybridizing the two methods they were able to find a reasonable trade-

off between the computation time and solution quality. This improvement allowed the algorithm to solve TSPs with up to 89,900 cities, vastly improving the performance of simulated annealing when applied to large-scale problems.

2.3.7 Tabu Search

Tabu search (TS) is popular metaheuristic which is used for combinatorial optimisation problems as it can produce good quality solutions, this algorithm starts with an initial solution and then moves around the search space by choosing solutions from the search neighbourhood. The main distinction of tabu search from other metaheuristics is the implementation of the tabu list which is fixed in size and continuously updated, this list hold moves which the algorithm deems as 'forbidden', if a move is held in the tabu list then it usually cannot be performed until the list is updated and the move is removed. Forbidden moves are determined by the tabu criteria which is chosen by the designer, common rules include not allowing moves which have the same fitness as the current solution, not allowing moves that were made recently or frequently and not allowing the previous move to be undone. By preventing the algorithm from choosing moves which are forbidden we can guide the algorithm towards unexplored areas of the search space and hopefully towards better solutions.

Tabu search also implements aspiration criteria, if the algorithm wants to make a move and that move is currently held within the tabu list it will not be allowed, the exception is if the move also meets the algorithm's aspiration criteria. Aspiration criteria overrides the tabu criteria, so it is important that the aspiration criteria isn't met too regularly as to not undermine the tabu list function, if the aspiration criteria is always met then the existence of tabu moves is pointless. Aspiration criteria may include allowing a solution provided it is the best solution in the neighbourhood or allowing a solution which is dissimilar to the existing solution in order to promote diversity and exploration of the search space.

According to Marinakis (2010) tabu search is the most widely used technique for solving the vehicle routing problem (VRP). The VRP is a problem which is similar to the TSP, the VRP contains a set of vertices which must be visited only once, the objective is to find an optimal set of routes for a fleet of vehicles to traverse rather than a single vehicle. We can see from VRP heuristic reviews that tabu search methods, particularly the granular tabu search (GTS), have garnered high praise. The GTS eliminates poor solutions by removing edges whose length exceeds the granularity

threshold, a measure that is chosen by the designer of the algorithm. This idea comes from the observation that longer edges are less likely to be included in the optimal solution (Laporte, Gendreau, Potvin, & Semet, 2000). In the guide to VRP heuristics from Cordeau et al. (2002) we can see the performance of tabu search methods compared to other heuristics, tabu search has a much better performance than classical heuristics and is competitive when compared to the other metaheuristics included in the guide. Osaba & Diaz (2012) show that tabu search can also be used to find good solutions for the TSP, in this study the tabu search was compared to an EA that was hybridized with tabu search. From the results we can see that although the tabu search was not as good as the hybrid algorithm when it came to solution quality, the execution time was better for the tabu search.

2.3.8 Evolutionary Algorithm

Evolutionary algorithms (EA) are a type of nature-inspired algorithm which are frequently used when optimising the TSP and similar problems and have been covered extensively in the literature, EAs are modelled after Charles Darwin's theory of evolution and the idea of 'survival of the fittest'. Evolutionary algorithms were first invented by John H. Holland in 1975 (Holland, 1992).

EAs are population based, the population consists of chromosomes which are the solutions to a given problem and each value in the chromosome is known as a gene. How a chromosome is implemented depends on the problem that is being solved, when solving a TSP an EA can use the tour permutation as the chromosome, with each individual vertex being a gene, as each complete tour is a solution to the TSP. The first step of an EA is to initialise the population. Normally an EA will generate the population randomly at runtime, this will help create a diverse population but there is no guarantee that the initialisation will produce good chromosomes. To prevent bad solutions being generated randomly the user can hybridize the EA with another search algorithm which is used to find good starting solutions for the EA (Deng et al., 2015). Developers can also seed high fitness solutions that they know of into the EA rather than randomly initializing the population. MPNS-GRASP is a initialisation algorithm developed by Marinakis (2010) which consists of two phases, first it constructs a random chromosome and then it uses a local search to improve upon the fitness of the chromosome. Marinakis found that by using better initial solutions through the MPNS-

GRASP algorithm led to faster convergence towards better results, thereby improving the runtime of the overall algorithm.

EAs consist of five main methods: initialisation, selection, crossover, mutation and replacement. Each of these methods is used to either explore the search space or improve upon the candidate solutions. Like many other metaheuristics, evolutionary algorithms are stochastic in nature, meaning that when performing multiple runs of an EA it is likely that it will produce a different result each time, because of this, researchers must run their EAs numerous times to calculate an average result.

Selection is the process of choosing parent chromosomes to eventually crossover and produce child chromosomes, this process mimics the act of reproduction in nature. The aim is to select the best chromosomes in relation to the fitness function of the population (Deng et al., 2015) as fitter parents have a higher chance of producing fitter offspring than parents which are less fit. It's also worth noting that unlike real-world reproduction which is limited to two parents, selection and crossover can be implemented with two or more parents if necessary. Razali & Geraghty (2011) studied the performance of three different selection methods: tournament selection, proportional roulette wheel selection and rank-based roulette wheel selection. They concluded that the quality of child chromosomes improved with rank-based roulette wheel selection, which is a method where a chromosome's chance of being picked is relative to its rank in the population.

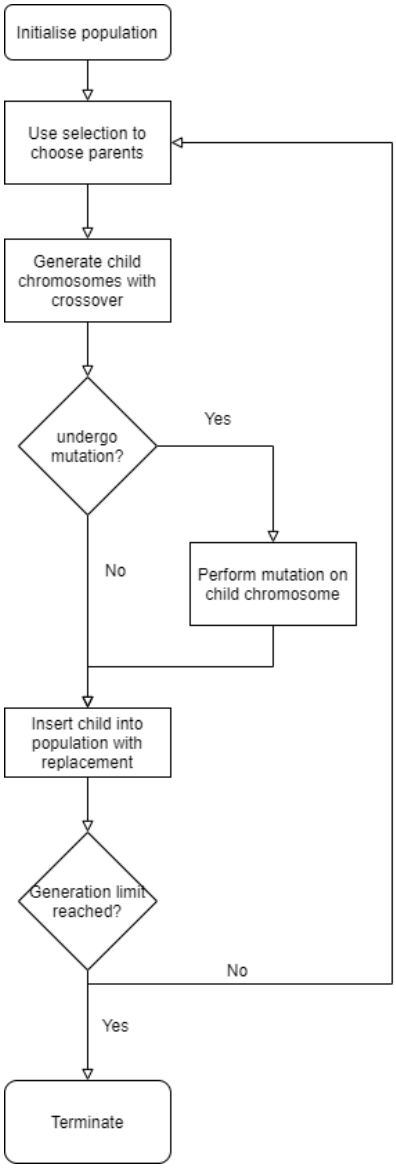


Figure 3 – EA operator flowchart

After the parent chromosomes have been chosen the next step is to cross over the genes from all parents and produce one or more offspring, crossover is an exploitation method as it takes good solutions and attempts to produce better offspring. There are a number of well-known crossover operators such as one-point crossover and uniform crossover, however, crossover is also one of the most well documented operators in the literature with numerous studies exploring new crossover methods. Crossover can either be performed deterministically or randomly (Snyder, Snyder, Daskin, & Daskin, 2006), one-point cross over is an example of deterministic crossover where a cut-off point is established and the genes which appear before the cut-off get inherited from parent 1 and the genes from after the cut-off are inherited from parent 2. Uniform

crossover, where each gene has an equal chance of inherited from either parent, is a random crossover method.

Mutation is an operator with a small chance to slightly modify the genes in a chromosome, it is used to maintain diversity within a population. Mutation frequency is decided by the mutation rate, which should not be set too high or too low. If mutation rate is too high then the children chromosomes may potentially lose any good genes inherited from its parents and the EA will become a random search. If mutation rate is too small then the population becomes less diverse, meaning that the algorithm may prematurely converge and get stuck at a local optima (Xu et al., 2018). This is corroborated by the work of Soon, Anthony, & Teo (2008) who explored the effects of three different mutation rates when applied to an EA. The set of mutation rates consisted of 0.2, 0.02 and 0.002 (or 20%, 2% and 0.2% chance of mutating respectively), the results show that the middle value, 0.02, was the most efficient out of the three.

The final operator found in an EA is replacement, which picks members of the population to be replaced by the offspring that are created during crossover, replacement is necessary as population size is usually fixed within an EA. As with survival of the fittest, solutions that are less fit are less likely to survive until the next generation, however with some replacement operators there is still a chance, albeit a small chance, that good solutions will be replaced. To prevent this elitism strategies can be implemented which preserve good solutions by preventing them from being replaced and can be useful in preventing premature convergence (Nazif & Lee, 2012). Other replacement strategies are implemented to increase population diversity, such as finding and replacing duplicate solutions or by replacing the parents of the newly produced child chromosome.

One of the main challenges of building an EA is deciding which operators to use as the effect of certain combinations of operators may be positive or negative. Any two operators which work better in combination with each other opposed to independently are said to have synergy (Contreras-Bolton & Parada, 2015). Ideally we want to choose a combination of operators that synergise to produce the best possible solutions, but this is extremely difficult due to the amount of existing possible synergies. As a result of this Contreras-Bolton & Parada (2015) successfully designed an evolutionary

algorithm which would explore crossover and mutation combinations in order to find good synergies.

The EA is a very popular choice for solving the EA in the literature and has been seen to be a successful method in many studies, many of which incorporate the use of local search algorithms to improve performance (Nagata & Soler, 2012; Nguyen et al., 2007). Although this may be a useful technique in reducing the tour distance, these studies do not compare their results to a standard EA, so it can be hard to tell how useful the EA is without these local search methods. Another issue which is addressed frequently in the literature is the large number of possible operators an EA can implement. Due to the large variety it is hard to know which operators will produce good solutions, there are a number of studies which focus on testing a specific type of operator such as mutation (Abdoun, Abouchabaka, & Tajani, 2012; Deep & Mebrahtu, 2011) or selection (Razali & Geraghty, 2011). These studies show us important it is to experiment with a number of different operators, as the same algorithm using different operators can produce wildly different results.

2.3.9 Operators

For metaheuristics such as the EA there are a wide range of operators to choose from, as discussed the EA implements initialisation, selection, crossover, mutation and replacement. Other algorithms can also use operators. One of the most common operators is the move operator. This operator decides how an algorithm moves from one solution to another. For example, the swap operator is one which would choose two positions within a solution and swap them, for a TSP this simply means swapping two vertices' positions within the tour permutation. This is a simple way of producing a new solution from a current one.

Another type of operator is the neighbourhood operator. This uses the move operator to create a range of new solutions for the algorithm to choose from, by calculating more than one new solution the algorithm has a higher chance of producing a good solution, however this method is also much more computationally expensive when compared to a move operator.

For the EA operators there are numerous ways to implement them, the selection operator is required to pick solutions to use for crossover, ideally these solutions have a high fitness. These are different methods of ensuring this, for fitness

proportionate selection each solution is given a chance of being picked, but the chance of being picked is directly proportionate to the fitness of the solution, therefore fitter solutions have a better chance of being picked. Tournament selection on the other hand picks *n* solutions from the population to compete in a tournament and the fittest solution is chosen as a parent. Tournament selection produces selection pressure, this is the pressure on the population to select good individuals for reproduction. By decreasing the tournament size, we can decrease the selection pressure, while increasing the tournament size increases the selection pressure. A high pressure means that good individuals are likely to be picked, this leads to faster evolution but can also lead to premature convergence. A low pressure means that weak individuals are likely to be picked, this slows down the rate of evolution and decreases the chances of the algorithm converging toward a single solution.

The number of possible crossover operators for an EA solving a TSP is limited by the fact that the EA chromosomes are likely encoded as permutations. We cannot use operators such as uniform crossover, where the child's genes are randomly selected from one of the parent's corresponding genes, as these are liable to produce solutions which do not visit every city, and hence aren't acceptable solutions. Instead, methods such as order one crossover must be used. This method takes a subset of the first parent, this subset is copied over to the child chromosome in the same position as it appeared in the parent chromosome, seen in Figure 4a. The remaining genes are inherited from the second parent in the order which they appear, if a gene from the second parent has already been inherited by the first parent then it is skipped and the next uninherited gene is chosen, seen in Figure 4b.

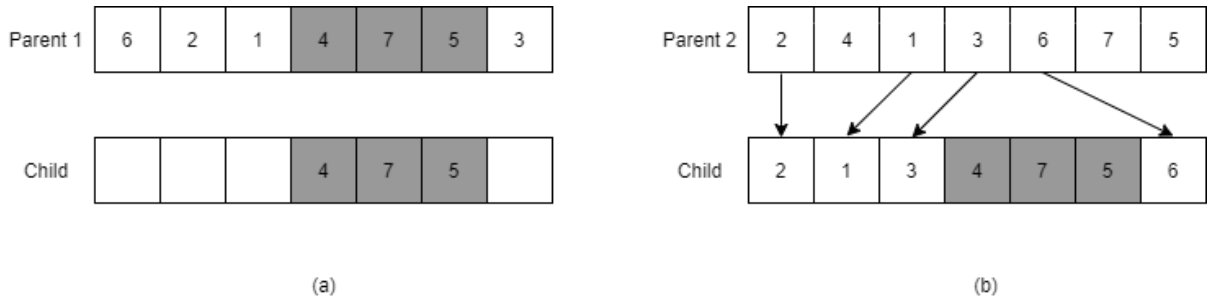


Figure 4 – Order one crossover

An example of a mutation operator which has found success in the literature is reverse sequence mutation (RSM). RSM takes two positions within the tour to define a subset of the tour, the genes within this subset are then reversed, while all of the

genes before and after the subset are kept the same. This operator was used in a study which evaluated the performance of EA mutation operators when used to solve the TSP (Abdoun et al., 2012), out of the four mutation operators evaluated in this study, RSM performed the best overall.

When replacing old members of the population with new child solutions, the ideal solution is to replace members which have a lower fitness, as they are less likely to produce high fitness offspring in the future. Some techniques of choosing appropriate members include always replacing the worst chromosome in the population, a tournament selection where the worst chromosome is replaced, a fitness proportionate selection where weaker chromosomes receive a higher chance of being replaced and replacing chromosomes which are most similar to the child chromosomes in order to maintain diversity.

2.3.10 Parameter Tuning

Some optimisation algorithms can benefit from what is known as ‘parameter tuning’, this is when a developer experiments with the values of an algorithm’s parameters in the hopes of finding the most optimal values. Many of the algorithms discussed above can take advantage of this; EA can experiment with the population size and mutation rate; SA can experiment with the initial and ending temperatures as well as cooling rate and TS can experiment with neighbourhood size and the tabu list size. There may also be other parameters that can be tuned depending on the operators that are implemented, for example, operators such as tournament selection can tune their tournament size.

EA’s population size and TS’s neighbourhood size have very similar functions and act in a similar way. The larger a population is, the greater the chance of initialising a high fitness solution as well as having a greater chance of exploring the search space. With neighbourhood size, a larger neighbourhood means a greater number of possible moves must be evaluated each turn. For both of these parameters, having a larger value will usually increase the average performance of the respective algorithm at the cost of increasing the runtime. This is because the algorithms have a greater number of processes it must perform at each iteration.

Mutation rate determines how often a child chromosome in an EA undergoes mutation, the parameter can be any value between 0 and 1. To decide whether a

child undergoes mutation the EA will randomly generate a number between 0 and 1, if the generated number is less than the mutation rate then the child mutates, otherwise the child will move on to the replacement stage. Typically, an EA has a low chance of performing mutation because if too many chromosomes mutate then there is a greater chance of losing good quality genes inherited through crossover, which may end up decreasing solution fitness. If the mutation rate is too high the algorithm risks becoming a random search. This is confirmed by studies in the literature such as Shim et al. (2011) and Jafarzadeh et al. (2017) who both had mutation rate set to 0.05 for their respective EA's when solving the TSP (5% mutation rate).

When the TS generates a new solution, it is sent to the tabu list, which prevents the TS from moving to this position again until the solution is removed from the list. The tabu list size determines how long a move will be forbidden by the TS, as the size of the list increases, tabu moves take longer to become available again. It also means that more moves can be made tabu, which forces the TS to explore more of the search space. This is good technique for promoting exploration, but once again, having a large number of solutions in a list will slow down the algorithm as the TS must search through the tabu list each iteration to ensure that the solution it intends to move to is not already in the tabu list.

The initial temperature and end temperature of the SA determines the range of temperatures the system can have, while the cooling rate determines how much the temperature cools each iteration. The initial and end temperatures can be any number the developer wishes, while the cooling rate is a value between 0 and 1. The SA starts at the starting temperature and each iteration the current temperature is multiplied by the cooling rate to produce the new temperature. This continues until either the SA terminates or until the end temperature is reached, in which case the cooling function stops, and the SA can no longer accept worse solutions. In the literature the initial temperature is usually very high, and the end temperature is very low. The cooling rate is typically a number which is very close to 1. Ye & Rui (2013) set their SA's initial temperature, ending temperature and cooling rate to 10,000, 0.1 and 0.95 respectively. Having a wide range between initial and ending temperature paired with a cooling rate close to 1 ensures that the cooling function will be active for a sufficient amount of time. If the system temperature drops too quickly then the SA

algorithm will not be able to accept worse solutions and will essentially become a hill climber, which we want to avoid.

2.4 Real World Routing

Although we can apply the Haversine formula defined in Equation 1 to find the great circle distance between two points on Earth, simply finding the great circle distance between all points in a tour would not give us a valid answer to a real-world situation. Many of the studies in the literature which explore real-world TSPs focus on routing vehicles for home delivery purposes or for similar services (Ginting et al., 2019; Jia et al., 2016; Nordin et al., 2012; Sahputra et al., 2016), to accurately solve these real-world TSPs we cannot use great circle distance to connect the vertices. In the real-world we cannot simply travel in a straight line in order to get from point A to B, instead we must use the road network. Only using great circle distance gives us an abstract answer that is infeasible in day to day life. So in order to generate a completely accurate answer we must transform the road network into a graph, by doing this we can create a route which is translatable into a real-world solution. If this step is overlooked and the solution uses great circle distance then the TSP solution is still abstract.

Real-world TSPs are also often very specific to the developer, as previously mentioned real-world TSPs are commonly used for home delivery systems, the specific TSPs being solved will differ for each home delivery service provider. For real problems of this nature it is unlikely that the delivery service provider would wish to share their TSP as it may have sensitive information such as customer's home addresses. Because of this there is a lack of real world TSPs available for benchmark tests. This can prove troublesome when trying to compare the performance of algorithms which are designed to solve these types of problem.

2.5 Conclusion

It is evident that global search methods are obsolete when solving large-scale TSP instances, as the time taken to find the optimum solution will probably be far too long to be of any use in a time constrained environment. Local search is the obvious choice for solving these problems, however, deciding which search techniques to use can vary depending on the problem or the user. Each of the algorithms that were discussed in this literature review have seen success when solving the TSP in some form or another. These algorithms can be useful in different situations, for example, if we

valued getting results quickly then it may be more optimal to choose a simpler algorithm such as nearest neighbour or hill climber. On the other hand, if we had a design problem which required us to find one high fitness solution, regardless of how many runs it takes to find it then the tabu search or evolutionary algorithm may be more useful. These algorithms also vary in complexity, nearest neighbour, 2-opt and hill climber algorithms are simpler and can be implemented relatively quickly with little need for experimentation or parameter tuning, they may also be easier to understand for some people. EA's on the other hand are more complex with greater reliance on operators and the value of its parameters, EA's may be more daunting to implement, but with the right amount of experimentation their performance can be greatly improved. The main objective of this project is to evaluate the performance of different optimisation algorithms when applied to real-world TSPs, given the varying complexity and practical uses of these algorithms they make them a good fit for this project. There is also a mix of heuristics and metaheuristics on display here which will allow for us to evaluate how one set of algorithms perform compared to the other and determine whether algorithms which have problem specific knowledge can improve upon those that don't. Heuristics, although included in the literature, are usually just implemented as a way to improve the performance of metaheuristics. By evaluating how heuristics perform on their own we can explore an area of the subject which is often overlooked in the literature.

The literature tells us that in order to get the best results from a metaheuristic algorithm experimentation needs to take place, depending on which algorithm is being used there will be parameters or operators which can be assessed. This can be a time-consuming aspect of designing an algorithm, as the algorithm should be run multiple times in order to get a sample of solutions that can be used to find averages. Algorithms which do undergo parameter/operator experimentation will also need to be run multiple times in order to confirm whether the change in parameter value had an effect on the algorithm's performance or not.

The literature has also displayed how useful benchmarks are. Many of the studies in this area focus on implementing and optimising one or two algorithms, these studies can do this because they are able to compare their algorithms to others by performing benchmark tests. However, these benchmarks apply to abstract, 2-D TSPs, there is a lack of real-world TSP benchmarks for us to use in order to compare the performance

of the algorithms. Implementing a comparatively large number of algorithms is justifiable as the algorithms can be evaluated against each other. If only one algorithm was to be implemented then we would have no way of determining whether an algorithm had a high fitness or not, as there would be no other fitnesses to compare against. If we have more algorithms then we will have a wider range of solutions to compare against, giving us a clearer idea of how well each algorithm performs.

3 Methodology

3.1 Overview

The first step to achieving the aims of this project is to implement all of the required algorithms as well as the classes that will be used to handle the TSP, for this project I will be using Java. The classes that will need to be implemented include a vertex class, population class, tour class and a tour manager. The vertex class will be used to encode vertex objects, which will have a longitude, latitude and an ID number. The vertex class will also include the Haversine formula implemented into a method that is used to calculate the distance between a vertex and the current vertex. The tour manager class will hold all of the vertices in a list, the manager will be able to add and return vertices from this list. The tour class will encode randomly generated solutions, as well as calculate their total distance and fitness. A method will also need to be included which can save the final tour of an algorithm to a file. The population class will be used by the EA, it will manage all of the chromosomes (which will be encoded as tour permutations) and include methods that will able to initialize a random population, save a new tour into the population, return a chromosome with an index and return the best and worst chromosome according to fitness.

The fitness function implemented into the tour class will use the distance to calculate the fitness:

$$Fitness = \frac{1}{TourDistance}$$

Equation 4 – Fitness function.

This is a simple fitness function that will return a value between 0 and 1. The shorter the tour distance is, the greater the fitness will be. Having the fitness function and distance calculator within the tour class will allow each algorithm to access the same fundamental functions which determine how good a solution is without them being implemented into every algorithm.

After we have implemented all of the algorithms and the necessary classes that is required to solve the TSP we will setup the routing software, which will work independently of the TSP program. The routing software will take the saved tour files, which are generated after solving the TSP, as an input. The software will then calculate the fastest route around all of the vertices when travelling by road. This will

be the final tour, the distance value that is produced from this tour will be used to compare the algorithms against each other.

Now that we can use our program to solve the TSP and use routing software to calculate a real-world route around the vertices, any eligible algorithms will need to undergo parameter tuning in order to maximise their performance. Not all algorithms will include parameters that can be experimented with. Because these algorithms are stochastic, we must run each experiment multiple times, by collecting a sample of data from each experiment we can get the average performance of the algorithms by calculating the mean. By analysing the average performance, we can get a much more reliable and accurate representation of each algorithms performance and how changing the given parameter affects the solution fitness. We will also be calculating the standard deviation of the tour fitness samples. Standard deviation measures how far spread the data is above and below the mean, a larger standard deviation denotes that the data is more spread out, while a smaller deviation shows us that the solutions are very close in value. Standard deviation can be useful when trying to compare two means which are very similar, as the set of data with the smaller standard deviation will produce more consistent solutions which will be more desirable in cases where the algorithm must always produce high fitness solutions.

By calculating the mean and standard deviation we can also perform a t-test, this is a statistical test which allows us to determine whether there is a significant difference between two sets of data. This can be a useful technique when analysing results and determining which parameters to choose, especially when the averages of two sets of results are very similar. The t-test will allow us to see if the difference in averages happened by chance or whether the two sets of data are statistically different from one another. To perform a t-test we start with a null hypothesis, H_0, which states that there is no significant difference between the two samples of data. Then we calculate the t-value, this can be done with the following formula:

$$t - value = \frac{\overline{x_1} - \overline{x_2}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

Equation 5 – T-value formula

In this formula $\overline{x}$ is the mean, σ^2 is the standard deviation and n is the number of values in the sample. Then we must compare the t-vale to a critical value, if our t-value is less than the critical value we accept the hull hypothesis to be true and can infer that there is no significant difference between the samples. If the t-value is greater than the critical value the we reject the null hypothesis, meaning that the two samples are statistically different. The critical value can be obtained from a t-table, to do this we need to decide what probability to use and we need to know the degrees of freedom, typically the probability is around 0.05 (or a 5% chance) and the degrees of freedom can be calculated by: $n_1 + n_2 - 2$. An example t-table can be found in appendix 4. Spreadsheet software such as Microsoft Excel and Google Sheets have t-test methods built in, this is a convenient way of performing the t-tests as there is no need to look up a t-table or calculate the degrees of freedom. The software mentioned will return the critical value, the user only needs to check whether it is greater than or less than the probability. If the critical value produced is greater than our probability then we accept the null hypothesis, otherwise we reject the null hypothesis and the two samples are significantly different. When comparing results from the same algorithm but with different parameters we should use a paired t-test.

Once the algorithms have had their parameters tuned we can collect the data that will be used to compare the algorithms. As with the parameter tuning the algorithms will need to be run multiple times for each dataset in order to calculate the mean and the standard deviation for each algorithm. We will be gathering two values from each algorithm, the distance and the wall clock time. To calculate the distance each algorithm will be allowed to run for 5 minutes, once the algorithm has converged or has run for 5 minutes the tour will be outputted to a JSON file. We can then input this file into the routing software in order to get the real-world distance. To calculate the wall clock time, we will give each algorithm 1,000 iterations, the program will record the times at which the algorithm is started and terminated. By calculating the difference between the two times we can work out the total time the algorithm was running for.

With both the distance and the wall clock time values we will plot the performance of each algorithm on a line graph. These graphs will have the collected data as the y-axis (distance for one graph, wall clock time for the other) and number of vertices as the x-axis. Each algorithm's average distance and wall clock time for each dataset will then be plotted and connected to create a line on their respective graphs. As the number of

vertices increases we will also see the average distance and wall clock time increase. If an algorithm’s corresponding line is higher than another algorithm’s line this shows us that the first algorithm has a worse performance as it either requires more time to run or produces solutions with higher distances. The steepness of the lines will also be able to tell us how well the algorithm scales, steep lines show us that an algorithm scales poorly with the increasing number of vertices, while less steep lines are better at scaling.

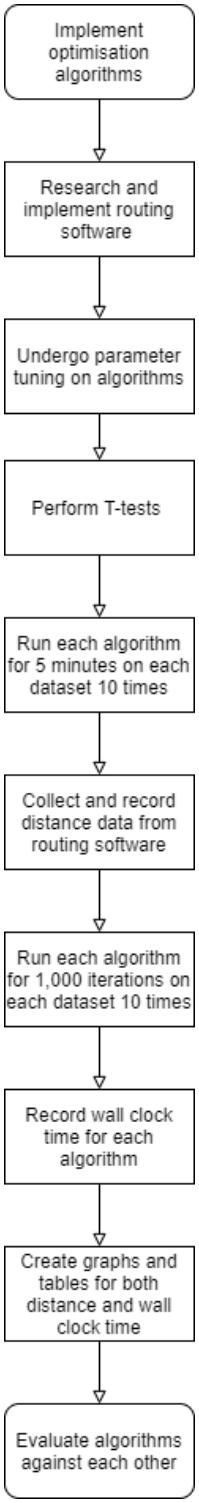


Figure 5 – Methodology overview flowchart.

3.2 Algorithms

The algorithms being evaluated were discussed in Section 2.3; nearest neighbour, 2-opt, hill climber, simulated annealing, tabu search and evolutionary algorithm.

The algorithms will share some common features. Firstly, there will be no fixed starting position, all of the algorithms will be able to start from any arbitrary position. Once the

algorithms have completed their tours, the vertex at the front of the tour (the starting position) will also be added to the end of the tour, creating a complete TSP solution.

None of the algorithms will be hybridized. Although this technique is common and can greatly improve solution fitness, when comparing hybridised algorithms it can be difficult to determine whether any good solutions were found because of one of the search techniques or both. As mentioned in the literature review, all of the chosen algorithms have been successful methods of solving the TSP in the past, by prohibiting hybridization it will be easier to compare the performance of each individual algorithm.

To keep the results fair, the distance values and wall clock time values will be recorded in the same way for every algorithm. The distance will first be produced by using the Haversine formula via the tour class. The tour will then be passed to the routing software, which will produce the updated and final distance. The wall clock time is collected by recording the system times at which the algorithm is started and terminated, by subtracting the starting time from the termination time we can find the total wall clock time for the run of the algorithm.

3.2.1 Operators

The EA will initialize the population randomly, similarly, the other algorithms will also randomly initialize their starting solutions. This does not apply to NN, which starts with a single vertex and builds its solution iteratively.

HC and SA will both use the swap operator at every iteration in order to generate new solutions. The HC's move operator will also be first-improvement operator, meaning that only one new solution will be generated at each iteration. The TS will also utilise the swap operator for its neighbourhood operator, performing the swap operator the same number of times as the size of the neighbourhood to create a range of solutions for the TS to choose from.

The EA will also incorporate the following parameters: tournament selection as the selection operator, order one crossover as the crossover operator, reverse sequence mutation for the mutation operator and finally the EA will replace the worst populations worst solution for the replacement operator.

3.2.2 Parameter tuning

To increase the efficiency of the metaheuristics, parameter tuning will be performed where appropriate. The EA parameters we will experiment with are population size, mutation rate and tournament size, TS will experiment with both tabu list size and neighbourhood size and the SA will experiment with initial temperature and cooling rate. HC, NN and 2-Opt will require no parameter tuning.

While performing the parameter tuning only one parameter will be experimented with at a time, all other parameter will be kept the same until it is their turn to be tuned. The only exception to this is when experimenting with the population size for the EA, the tournament size will scale along with the population size in order to create a constant selection pressure. If the tournament size stayed the same while the population size increased then the selection pressure would start to decrease, and it is likely that the solution fitness would suffer because of it. Therefore, while experimenting with population size, the tournament size will be kept at 10% the size of the population. For all other experiments, other than the tournament size experiment, the tournament size will be kept static.

T-tests will be performed for each experiment, these statistical tests take two samples and determine whether there is any statistical difference between them. The parameter settings which produces the best average solution fitness will undergo t-tests with all of the other parameter settings for that experiment, by doing this we can see if there is any difference between the ‘best’ parameter and the other parameters. The algorithm’s results will be stored in Google docs spreadsheet, this will allow us to utilise Google doc’s built-in t-test function.

For the parameter tuning experiments I will be running each algorithm 10 times, the dataset used for these experiments will contain 10,000 vertices, this dataset will be used both for experimentation and to gather results once the optimum parameters have been chosen. Each algorithm will run for 1,000 iterations.

3.3 Tools

3.3.1 GraphHopper

GraphHopper[3] is a powerful tool that has several useful APIs which can be used for routing and direction services. These APIs have a wide range of abilities, from

[3] https://www.graphhopper.com/

geocoding services to producing many-to-many distance matrices, the service that is used for this project is the routing API. The routing API will take a tour and calculates its total distance without changing the order which the vertices are visited. The routing API will also use roads to connect the vertices rather than great circle distance which will provide more accurate real-world data. GraphHopper can also take other restrictions such as one-way roads into account, which is important for accurately modelling real world problems. As well as calculating the distance of the tour we will also be able to calculate the time it would take a car to travel the total tour, other travel methods such as bicycle and walking can also be taken into account, although they will not be used for this project. If using different modes of transport, the API can take advantage of alternative routes such as footpaths, which are inaccessible to cars. Another benefit of GraphHopper is that it allows the user to avoid certain roads if they wish, the user can choose to avoid ferries, bridges, motorways, tolls and tunnels, although this won't be utilised in this project, it may still be in some users' best interest.

By default, GraphHopper uses OpenStreetMap data in order to produce its graph, in this graph junctions are represented as vertices and the roads are represented as edges. OpenStreetMap provides free map data, which is edited and maintained by OpenStreetMap's community of user and mappers. Because the map is maintained by its users it may not always be accurate or up to date, for example, a new road which was recently constructed may be missing or a one-way street may be classified as a two-way street. Inaccuracies in the map data may result in issues when it comes to routing, however, anyone can edit the OpenStreetMap data, meaning that any issues found by the users can be quickly addressed if needs be.

To find the shortest path between two paths GraphHopper uses Dijkstra's algorithm and A* search algorithm. GraphHopper is also able to use contraction hierarchies in order to provide faster results. Contraction hierarchies are a method of improving computation time of shortest-path algorithms when searching graphs which represent

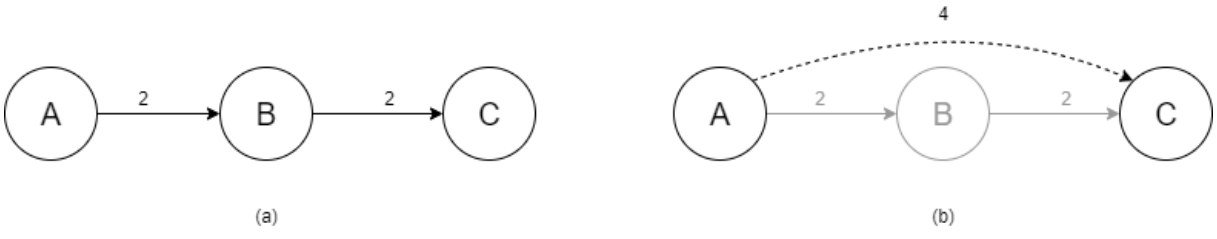


Figure 6 – Contraction hierarchy

road networks, it does this by contracting vertices from the graph in a pre-processing phase. This is a useful technique as road networks contain a large number of roads and junctions to consider. When a vertex, *v*, is contracted it is removed from the graph and shortcut edges may be inserted between the neighbouring vertices of *v*. These shortcuts allow us to skip over 'unimportant' vertices while preserving the shortest path distances (Bauer, Columbus, Rutter, & Wagner, 2016). Figure 6a shows an uncontracted graph while Figure 6b shows vertex B being contracted and the shortcut (represented by the dashed line) between A and C being established.

In order to get the maximum benefit out of GraphHopper the GraphHopper routing engine was cloned from GitHub[4], this allowed me to run an instance of the routing engine locally, which bypasses the restriction on the number of vertices allowed per request. The GraphHopper routing engine can be run from either the command line or from an IDE, if using an IDE GraphHopper recommends using NetBeans or IntelliJ IDEA as they can both run a local web server without further setup. I will be running the application through IntelliJ, as running GraphHopper through an IDE allows the user to debug the application.

Once the GraphHopper application is running the user can send HTTP GET and POST requests to http://localhost:8989/route, if the request is valid then the distance, travel time and directions will be returned to the user. The GET request method is simpler as the only requirement is to specify the parameters (such as the vertices to be visited) in the localhost URL, however, there is a limit to how many vertices can be processed by a single GET request, as URLs can only be so long. Problems which have a large number of locations will have to use a POST request, this feature is currently only available on the master branch of the GraphHopper application.

3.3.2 cURL

cURL is an open-source project which develops two products: a command-line tool and a library, which are used for transferring data via network protocols. For this project the command-line tool will be used, this tool can be used to upload and download data, which is specified with a URL, using internet protocols.

This tool was used in order to send the HTTP POST requests to the local GraphHopper server, the POST request includes a .json file which details which vertices to visit and

[4] https://github.com/graphhopper/graphhopper

in which order to visit them. The cURL command that is required to send the HTTP POST request can be found in appendix 5.

3.3.3 Git & GitHub

Version control systems (VCS) record the changes to a file or a group of files over time. Using a VCS allows users to view changes to files over time, revert files or projects back to a previous state, create development branches that allow users to work on new features without risk of affecting the master branch and more. For this project I used Git as the VCS, Git is a popular distributed VCS which is widely used, many IDEs either include Git support or have extensions which allow users to use Git directly from the IDE. Rather than storing the information as a set of files and the changes made to those files over time, Git stores information as 'snapshots', pictures of what the file looks like at the time of the commit. Git also doesn't store a snapshot of any files which have not been changed, instead it provides a link to the previous, identical snapshot, making Git more memory efficient than other VCSs.

For most of this project I used Git locally, this means that all the information was stored within the local system. This is a simple approach that is best suited for projects which have only one contributor, such as this one, however there are some issues with this approach such as a risk of losing all the data if there is an issue with the local system.

I also used GitHub as a method for transferring the project between systems. When a project is committed to GitHub the information, including the entire project history, is stored on a server. Users can then clone the repository, including its full history, to their local system, once this has been done the user can work on the project locally until they decide to commit the project back to GitHub. This allows multiple people to work on one project simultaneously, as well as having the benefit of having the project saved on a server as well as locally. When I was required to move the project between systems I would commit the project to GitHub, then I could clone the repository on the new system, which would allow me to continue working on the project while also having access to the full project history.

3.3.4 Maven

Maven is a build tool which is used for java-based projects, by providing every Maven project with a POM (project object model) and a shared set of plugins, Maven is able to create a uniform build system. This allows us to automate the build process. The

POM is a configuration file that contains most of the required information to build the project, including the project dependencies. By using a website such as mvnrepository[5] we can search for any dependencies that our project may require in order to run, once the desired dependency has been found it can be copy and pasted into the dependencies section of the POM. The dependency will include three tags, groupid, artifactid and version, these are all used to identify the dependency, once stored inside the POM the project will have access to that dependency and any other dependencies associated with it. Each Maven project also contains its own groupid, artifactid and version number, once the project has been released this will allow others to use the project as a dependency in their own work.

Before cloning and running the GraphHopper application locally, the project used the GraphHopper API, which required the use of GraphHopper dependencies. After removing the GraphHopper API from the project due to it having a restriction on the number of vertices that could be visited there was no dependencies required for the project to work, however, Maven was still used to automate the building of the project. the GraphHopper routing engine is also a Maven project and requires Maven to be installed in order to run. Upon setting the routing engine for the first time the program will automatically download Maven if it is not already available, users can also download the OpenStreetMap data for their desired area at this stage. The GraphHopper routing engine itself consists of several projects, each of which has its own unique POM file and dependencies. By using maven, the developers have made building the application is automatic and easy, allowing users to start using the application as soon as it is set up.

3.3.5 MATLAB

MATLAB is a programming platform that can be used for numerous applications such as numerical computation, data analysis, 3D modelling and more. For this project MATLAB was used in order to visualize the data collected from the algorithm's results by producing graphs, this allows for quick and easy comparisons between different algorithms and parameters.

[5] https://mvnrepository.com/

3.4 Experimental Setup

After the algorithms have had their parameters tuned to maximise their performance we can begin to collect the necessary data that will allow us to compare the different algorithms. As mentioned in the methodology overview there will be two experiments conducted, the first will gather distance data while the other gathers wall clock time. As with the parameter tuning, each algorithm will need to be run multiple times for each dataset in order to calculate an average distance and a standard deviation.

Distance:

When collecting the distance data each algorithm will solve the TSP for each dataset 10 times. For this experiment the algorithms will be allowed to run for a maximum of 5 minutes, once the algorithm has converged or ran for 5 minutes it will return the solution to the TSP, which will be written to a file in JSON format. Each file will need to be inputted to the local GraphHopper routing engine through a HTTP POST request that includes the JSON file. The HTTP POST is sent to the routing engine through the cURL command-line tool. Once the routing engine has received the request it will start to find the shortest distance around the inputted tour. GraphHopper will output the distance to the IDE running GraphHopper as well as the cURL command-line, this distance value will then be recorded. As this is a minimisation problem, the shorter the final route, the better the algorithm is.

Wall Clock Time:

Once again, for this experiment each algorithm will need to be run for each dataset a total of 10 times to collect the mean wall clock time and the standard deviation. We cannot collect the wall clock time alongside the distance values due to each algorithm running for 5 minutes, therefore we need to run a separate test. This experiment requires all algorithms to complete 1,000 iterations, with exception to the NN algorithm which will only need to generate a complete tour, as this algorithm does not utilise iterations like the other algorithms. Instead the NN starts from a single vertex, then it adds the nearest unvisited vertex to the tour, this process is repeated until no more vertices remain, at which point the NN will have produced its complete, final tour. By calculating how long it takes to complete 1,000 iterations we will be able to see a general idea of how fast the algorithms run. The longer it takes the algorithm to run 1,000 iterations then the longer the algorithm needs to perform each iteration. If one algorithm can evaluate iterations faster than another, then the faster

algorithm will be able to perform more iterations in a set amount of time, which would likely be more beneficial to the developer. This is also a minimisation problem so the faster the wall clock time value of an algorithm, the better the performance of the algorithm.

3.4.1 Datasets

There will be four datasets used in this project that will be used for collecting data, the datasets will have 10, 100, 1,000 and 10,000 vertices respectively. By increasing the size of each subsequent dataset by a factor of 10 we will be able to see how well each algorithm scales with the increasing size of the problem, both in terms of solution quality and time taken to produce an answer.

The university of Waterloo in Canada has previously solved a TSP which involved traveling to every pub within the UK[6], the dataset for this problem, which is referred to as UK24727, contained 24,727 pub locations. The datasets used for this project were extracted from the UK24727 dataset, with the first 10 locations being used for the first dataset, then the first 100 locations being used for the second dataset and so on. Because of this, all datasets used for this project contain coordinates for pubs within the UK

[6] http://www.math.uwaterloo.ca/tsp/pubs/data.html

4 Results and Evaluation

4.1 Experimentation Results

This section will present the results gathered from the algorithm experimentation as well as explaining which parameters were chosen for the EA, simulated annealing and tabu search. Results will be presented in tables, the “Best”, “Average” and “Worst” columns represent the best, the mean and the worst distances from each sample. The standard deviation is shown in the “Std. Dev.” Column. Finally, for each experiment t-tests were performed to check whether there is a significant statistical difference between the best performing parameter and the other parameters. T-tests will be represented in their own tables, the parameter value that is constant throughout the t-tests (i.e. the ‘best’ value we are t-testing all other values against) is represented in the first column, labelled as parameter 1. Each subsequent column header corresponds to a parameter value which the best average parameter is being tested against, labelled as parameter 2. If the critical value is less than 0.05 then we reject the null hypothesis and can assume that there is a significant statistical difference between the two samples. Full test data from this section can be found in the appendices (EA data in Appendix 7, SA data in Appendix 8 and TS data in Appendix 9).

Evolutionary Algorithm

As described earlier, the population test was performed first, while selection pressure and mutation rate were kept the same. The results in Table 1 show that the larger populations performed better than lower populations. The population of 400 found better results across the board, as the best, average and worst distances were smaller than all other population sizes. The population size was stopped at 500 to ensure that the EA ran in a reasonable time.

Population	Best	Average	Worst	Std. Dev.
100	2.044	2.062	2.069	0.007
200	2.040	2.055	2.068	0.009
300	2.041	2.050	2.067	0.007
400	2.035	2.045	2.060	0.008
500	2.040	2.052	2.068	0.009

Table 1 – EA solution distance (millions of km) for varying population sizes

Parameter 2 / Parameter 1	100	200	300	500
400	8.32×10^{-5}	0.034	0.252	0.195

Table 2 – P-values from EA population t-tests

When assessing the t-tests we can see that the population size of 400 is significantly different from the smaller populations of size 100 and 200, but there is no significant difference between the larger populations of 300 and 500. From Figure 4 we can see that the population of 300, despite not being able to find solutions as good as the population of 400, is much more consistent, as shown by the smaller boxplot, as well as the standard deviation. Although it is clear that the population sizes of 100-200 are worse than the population sizes of 300-500, there is no clear optimum value. It's possible that the results may be clearer if the algorithm was allowed to fully converge, as the EA would likely still be able to improve upon its solution after 1,000 iterations. It was expected that a larger population would produce greater results as there will be a greater chance of one of the chromosomes producing a good population, given this, the algorithm may have also performed even better with a population greater than 500.

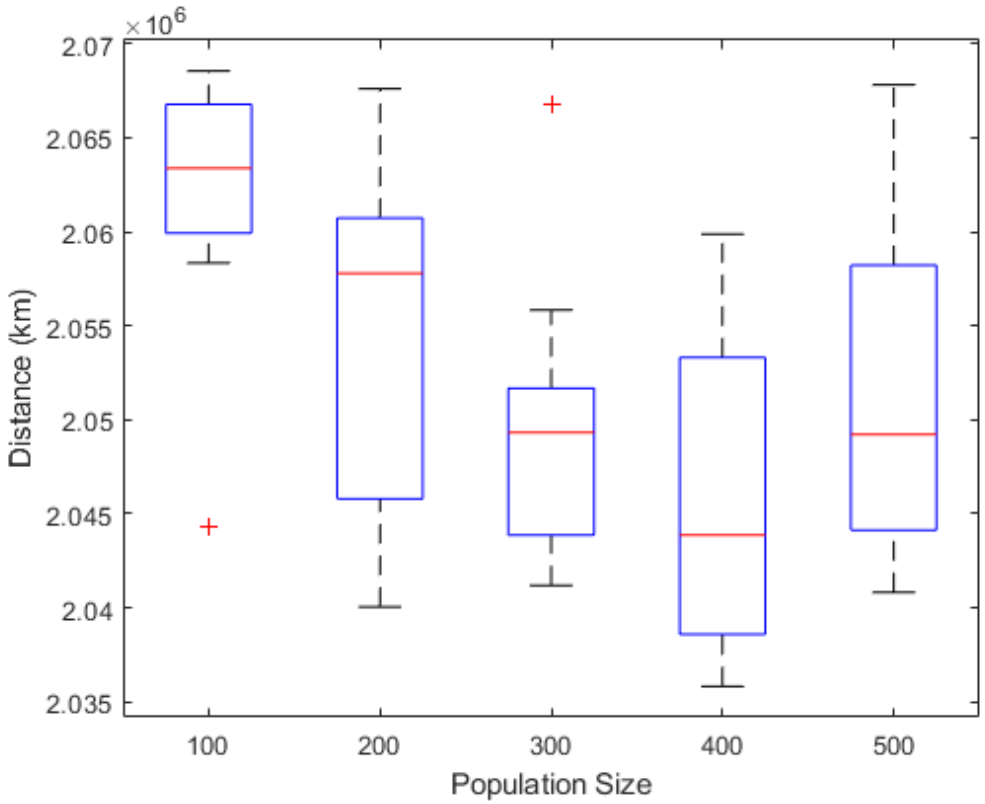


Figure 7 - EA population size experiment results

Changing the tournament size of the EA had the largest effect on the performance of the algorithm, we can see from the mean distances in Table 3 that there is a correlation between tournament size and performance, as the smaller tournament size is, the better the solutions. From this we can determine that the EA performs best with a low selection pressure, with the best performing tournament size, 10, comparing only 2.5% of the total 400 chromosome population each iteration. This means that the fittest population member may rarely be chosen, but the population will remain more diverse. This is because the tournament size is very small in comparison to the population size, and thus, the fittest solutions are less likely to be picked. Large tournament sizes have a much greater chance of choosing fitter parents which can lead to the same parent being chosen over and over, which leads to less diverse offspring. T-test results for the tournament size of 10 shows that the tournament size of 20 is the only tournament which shows no significant statistical difference in the results, however, the associated p-value (0.07) is only just higher than the 0.05 threshold. We can see from Figure 5 that although the size of the box plots for tournament sizes 10 and 20 are roughly similar, the interquartile range of tournament size 10 is much smaller, suggesting that the algorithm is more consistent

in finding good solutions, this is backed up by parameters smaller standard deviation, for these reasons 10 was chosen as the optimum tournament size.

Tournament Size	Best	Average	Worst	Std. Dev.
10	2.018	2.029	2.041	0.007
20	2.020	2.034	2.042	0.008
30	2.034	2.049	2.057	0.008
40	2.034	2.049	2.062	0.009
50	2.045	2.054	2.045	0.006

Table 3 - EA solution distance (millions of km) for varying tournament sizes

Parameter 2 / Parameter 1	20	30	40	50
10	0.070	0.002	2.74×10^{-4}	6.63×10^{-6}

Table 4 - P-values from EA tournament size t-tests

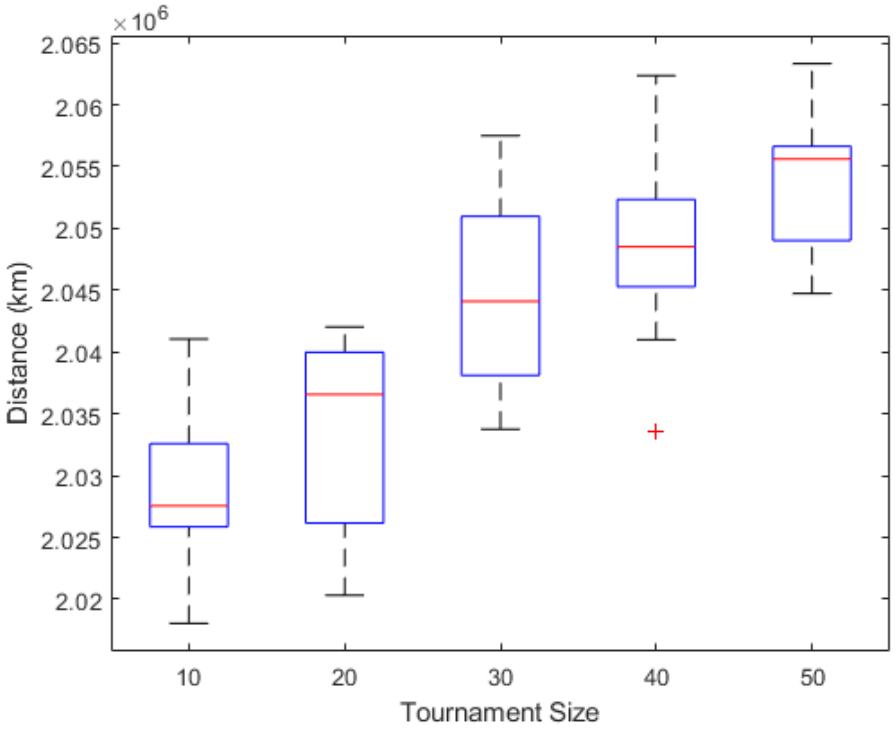


Figure 8 – EA tournament size experiment results

The final experiment for the EA was the mutation rate. The mutation rates tested ranged from 0.1% to 10%. The mutation rate was capped at 10% to stop mutation happening too frequently, as this can lead to the loss of good genes. The results from this experiment are very close, with little indication as to which mutation rate is the optimum (Table 5). For this experiment t-tests compared the mutation rate of 0.05 against all other rates, however, with the exception of 0.001, all of the mutation rates showed no significant difference in the results. We may be able to infer from these results that the mutation has little effect when applied to such a large TSP problem. The biggest difference between the performance of these mutation rates is the consistency, from looking at the standard deviations and size of the interquartile ranges from these parameters we can see that the mutation rate of 0.05 is much more consistent in finding high quality solutions when compared to 0.01 and 0.005, however the most consistent parameter was 0.1. The worst solution found with the mutation rate of 0.05 is better than worst solutions found by all of the other mutation rates and there is only a 65km difference between the best solution found by the 0.05 mutation rate and the overall best solution found by 0.01. Therefore, it is evident that the mutation rate has a much smaller impact on the EA's performance when compared to the population size and the tournament size, especially when applied to such a large problem.

Mutation Rate	Best	Average	Worst	Std. Dev.
0.001	2.027	2.033	2.038	0.004
0.005	2.020	2.029	2.039	0.006
0.01	2.019	2.028	2.040	0.007
0.05	2.019	2.027	2.034	0.005
0.1	2.025	2.032	2.036	0.003

Table 5 – EA solution distance (millions of km) for varying mutation rates

Parameter 2 / Parameter 1	0.001	0.005	0.01	0.1
0.05	0.029	0.517	0.859	0.057

Table 6 - P-values from EA mutation rate t-tests

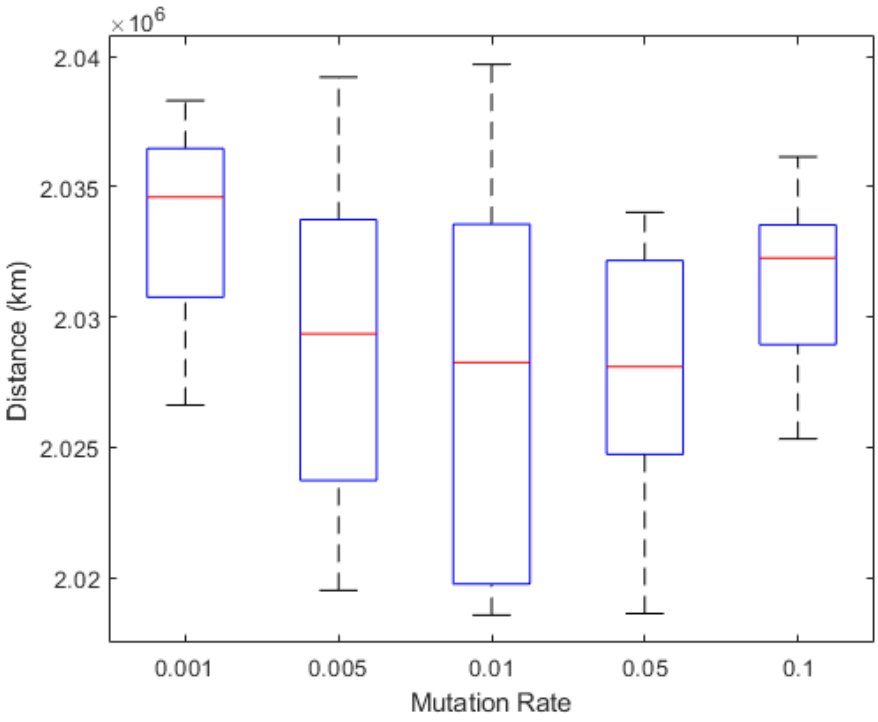


Figure 9– EA mutation rate experiment results

Simulated annealing

The results for this algorithm very clearly show which parameters are the optimal (Table 7). The start temperature parameter was most effective when the temperature was low, as the lowest tested temperature, 500 degrees, outperformed all the other temperatures. It is likely that a lower temperature would have performed even better, however, reducing the starting temperature even more would cause the cooling function to become pointless, as the temperature would decrease too quickly, and the SA would transition into a HC. As we can see from the t-test results, all of the other temperature samples are also significantly different from the 500-degree sample, which reaffirms that 500 degrees is the optimum value for the starting temperature parameter.

Start Temperature	Best	Average	Worst	Std. Dev.
500	2.065	2.076	2.087	0.007
5,000	2.071	2.087	2.103	0.009
50,000	2.083	2.090	2.106	0.006
500,000	2.073	2.090	2.102	0.009

Table 7 – SA solution distance (millions of km) for varying start temperatures

Parameter 2 / Parameter 1	5,000	50,000	500,000
500	0.032	0.002	0.005

Table 8 - P-values from SA start temperature t-tests

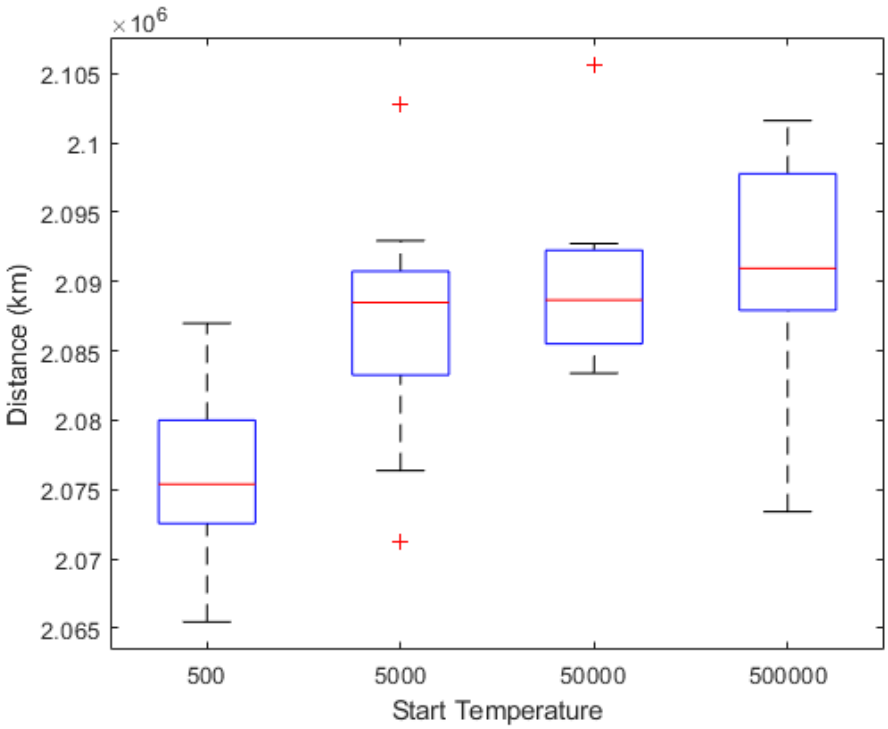


Figure 10 - SA start temperature experiment results

The experiment with cooling rate also produced unexpected results. From previous studies which implemented a SA algorithm to solve the TSP, we expected both the starting temperature and the cooling rate to be high values (Ye & Rui, 2013). For this experiment the most optimum parameter was also the smallest as the cooling rate of 0.8 produced the best average result. Having a small starting temperature means

that the system temperature will drop faster than if there was a high temperature, similarly, a smaller cooling rate also means that the system temperature will decrease more per iteration compared to a higher cooling rate. We can conclude from this that the algorithm works better the faster the temperature drops, this could be due to the algorithm only having 1,000 iterations to run. From the t-test results in table 10 we can see that there is no significant difference between 0.85 and 0.8, so either one of these values would be a suitable choice for the optimum, for this project a cooling rate of 0.8 was chosen. We can also see this from how similar the shape and size of their respective boxplots in Figure 11. As with the starting temperature experiment, the cooling rate was not dropped further as this would undermine the cooling function of the SA.

Cooling Rate	Best	Average	Worst	Std. Dev.
0.99	2.080	2.102	2.117	0.012
0.95	2.051	2.067	2.073	0.007
0.9	2.019	2.040	2.051	0.009
0.85	2.015	2.029	2.039	0.008
0.8	2.010	2.020	2.034	0.008

Table 9 – SA solution distance (millions of km) for varying cooling rates

Parameter 2 / Parameter 1	0.85	0.9	0.95	0.99
0.8	0.149	1.98×10^{-4}	1.13×10^{-7}	7.72×10^{-8}

Table 10 - P-values from SA cooling rate t-tests

As the cooling rate was also smaller than expected this confirms that in this scenario the SA algorithm performs better the quicker the system temperature drops. The quicker the temperature decreases then the quicker the SA algorithm will transition to a HC, as it will no longer be able to accept worse solutions. The reason for accepting worse solutions is to help the algorithm escape from local optima, however, with a search space as large as a 10,000 vertex TSP, the simulated annealing algorithm may never find a local optimum before it runs out of allocated iterations. This may explain why simulated annealing produces better results with lower temperatures and

cooling rates in this situation, as the quicker the temperature drops, the sooner the algorithm can start focussing on better solutions. Given a situation like this where the algorithms have a limited number of iterations a HC may be more appropriate than the SA, as the HC will only choose better solutions, while the SA can choose worse solutions until the system temperature drops. Therefore, it is more likely that the HC will perform better as the SA's temperature might not drop fast enough, causing the SA to choose solutions which move it further away from the optimum.

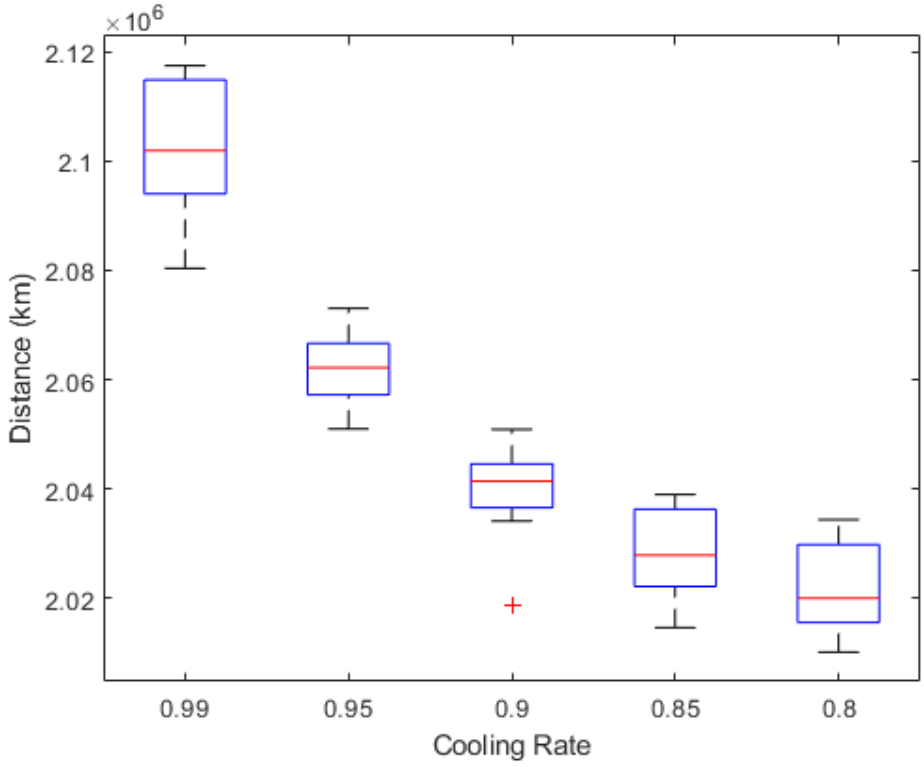


Figure 11 - SA cooling rate experiment results

Tabu search

The first experiment conducted was related to how the neighbourhood size effects the solution quality, we hypothesise that solution quality to increase alongside the number of solutions in the neighbourhood as there are more chances for the neighbourhood operator to produce good solutions. The results from Table 6 confirm this hypothesis to be true, as each increase in neighbourhood size reduces the distance of the tours produced. Had the neighbourhood size been increased further it likely would have been able to produce even better results, however, doing so would greatly increase the time taken to run the algorithm. The neighbourhood size of 250 produced the best results in terms of best, average, and worst solutions found, as well as this we know from the t-tests that this sample was significantly different from

all of the other samples, therefore, this value is the obvious optimum. When examining figure 12 it is obvious that the neighbourhood size of 250 is clearly the optimum as it's corresponding boxplot is much lower than all of the other boxplots.

Neighbourhood size	Best	Average	Worst	Std. Dev.
50	1.574	1.585	1.604	0.008
100	1.514	1.528	1.544	0.009
150	1.483	1.491	1.501	0.005
200	1.469	1.474	1.478	0.004
250	1.448	1.456	1.466	0.005

Table 11 – TS solution distance (millions of km) for varying neighbourhood sizes

Parameter 2 / Parameter 1	50	100	150	200
250	0	2.04×10^{-9}	6.81×10^{-8}	2.73×10^{-5}

Table 12 - P-values from TS neighbourhood size t-tests

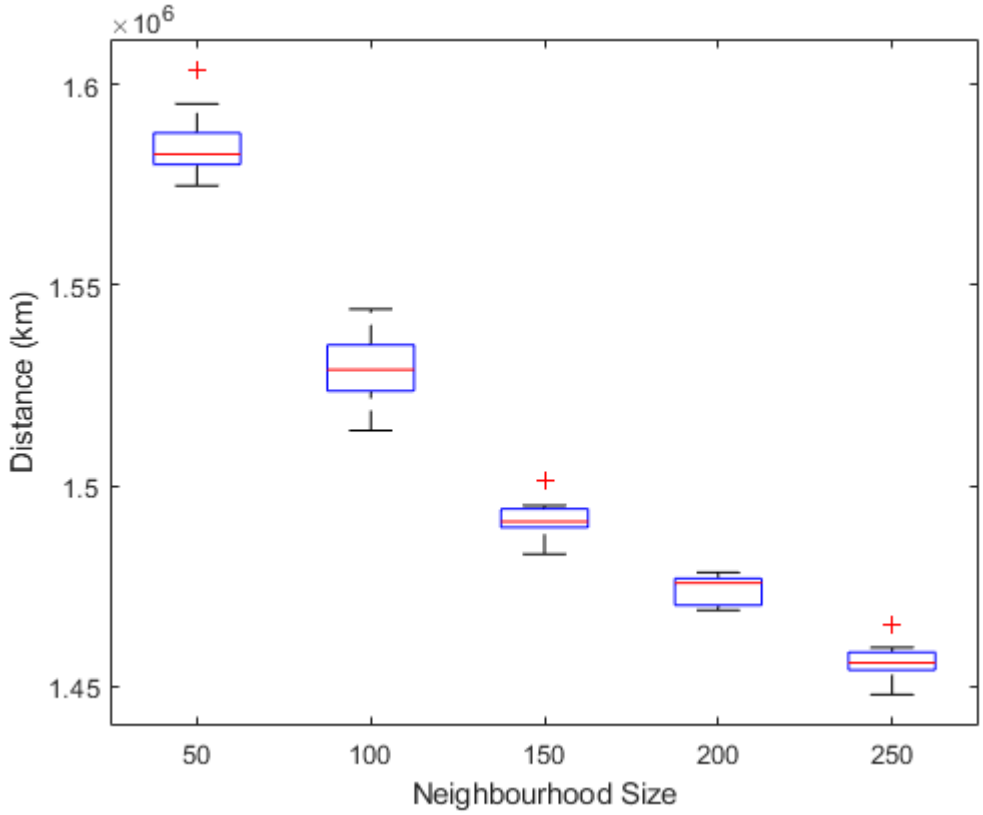


Figure 12 - TS neighbourhood size experiment results

The last experiment's purpose was to explore how the size of the tabu list effected the solution quality of the tabu search. Having a larger tabu list means more solutions can be restricted, which will encourage the algorithm to visit unexplored areas of the search space, it also means that solutions which are restricted will be restricted for longer. The results in Table 13 show little difference between the various tabu sizes that were tested. For this experiment the t-tests compared the results of tabu size of 100 against all other results, however, since all of the p-values from these tests are greater than 0.05 we know there is no difference between any of these tabu list sizes. We can see from Figure 13 that each sample of results was quite similar and there also seems to be no general trend to the data, all of these factors suggest that out of the values tested there is no optimum. This is probably due to the size of the search space, the tabu search makes a small change to its solution each iteration, simply swapping two arbitrary vertices in the tour permutation. When the solution permutation contains 10,000 vertices, as it does in this case, then the number of possible solutions we can make from swapping two vertices is an extremely large number. The chances that the tabu search would swap the same two vertices twice is therefore extremely low, even with a neighbourhood size of 250, therefore the size of the tabu list may have less of an overall effect as the algorithm is already unlikely to visit the same location twice anyway.

Tabu list size	Best	Average	Worst	Std. Dev.
50	1.575	1.585	1.603	0.008
100	1.571	1.584	1.593	0.008
150	1.579	1.586	1.597	0.006
200	1.573	1.590	1.601	0.009
250	1.579	1.588	1.596	0.005

Table 13 – TS solution distance (millions of km) for varying tabu list sizes

Parameter 2 / Parameter 1	50	150	200	250
100	0.651	0.507	0.214	0.137

Table 14 - P-values from TS tabu list size t-tests

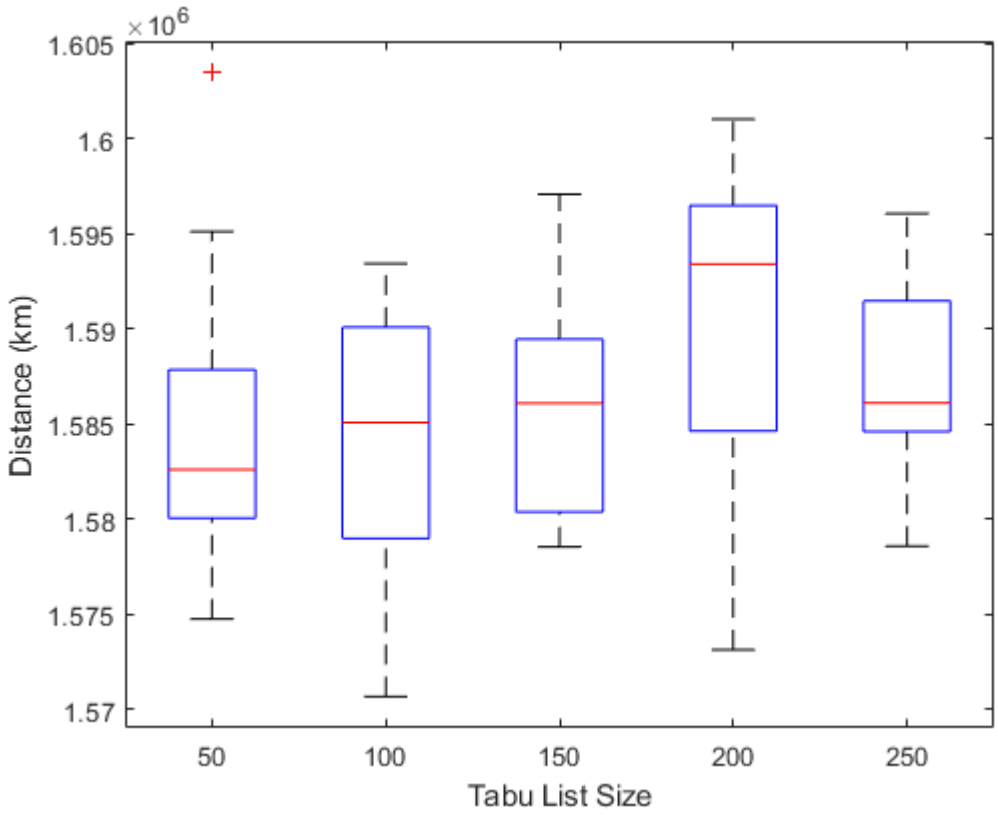


Figure 13 - TS tabu list size experiment results

4.2 Results

In this section I will be reporting on the results that were gathered from each algorithm solving the set of four datasets outlined in Section 3.3.1. Full results tables for this section can be found in the appendices (Distance data in Appendix 10, wall clock time data in Appendix 11).

4.2.1 Distance results

First we will analyse the results from allowing each algorithm to run for 5 minutes, here each algorithm is allotted the same amount of time to run and therefore, the algorithms which produce the shortest tours will be more successful than those which produce longer tours. Table 15 displays the results for this data, in this table the first column represents the algorithm, after this there are four more columns, 10-vertices, 100-vertices, 1,000-vertices and 10,000 vertices, each of these columns represent one of the four datasets. Figure 14 shows a visual representation of the data collected in Table 15 and gives us a better idea of how well each algorithm performs as the scale of the problem increases.

Algorithm	10-vertices		100-vertices		1,000-vertices		10,000-vertices	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
EA	0.536	0.114	6.035	0.571	114.7	3.046	1987	11.82
TS	0.616	0.017	2.202	0.240	26.56	1.074	1914	8.432
SA	0.509	0.128	2.690	0.328	25.19	0.909	1011	6.313
HC	0.451	0.145	3.129	0.393	25.34	1.287	1643	265.4
2O	0.610	0.021	1.052	0.020	3.993	0.121	724.4	13.15
NN	0.363	0.025	0.973	0.023	3.887	0.148	29.39	0.253

Table 15 – Tour distance (thousands of km) results of algorithms

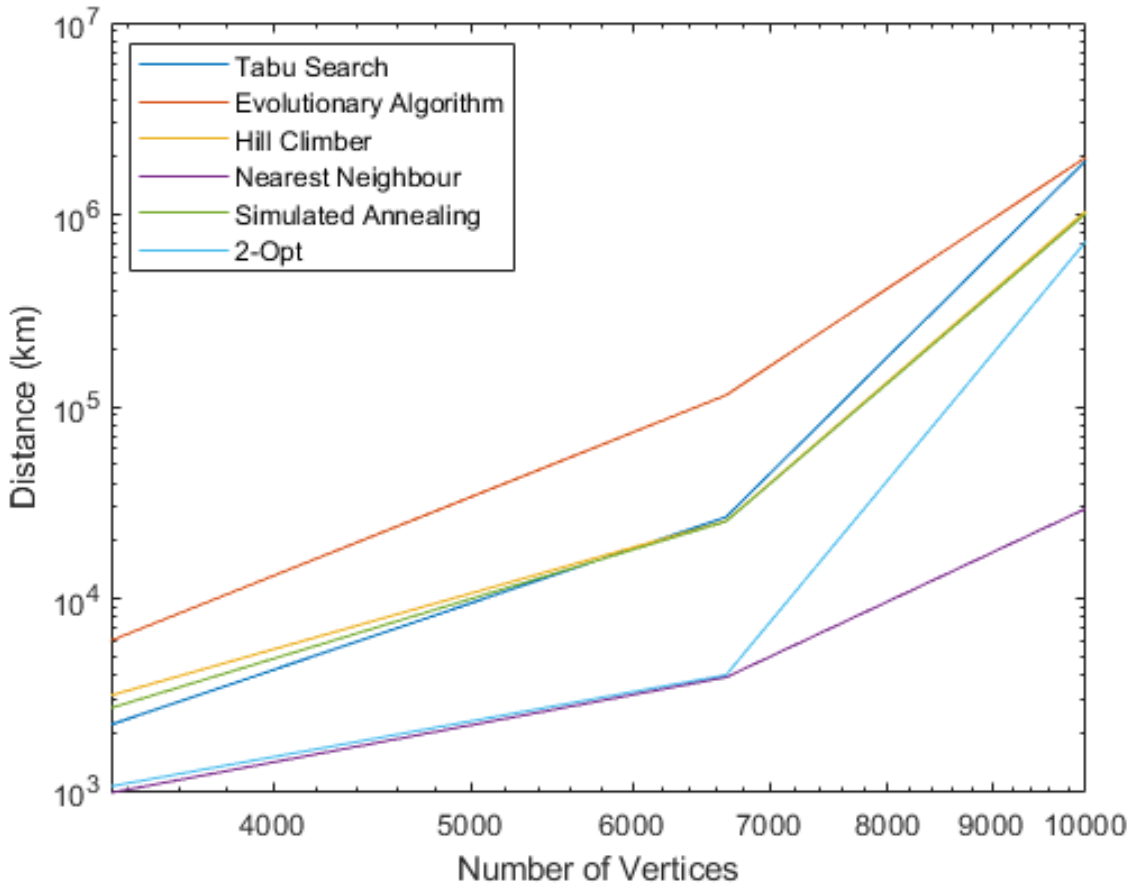


Figure 14 – Average tour distance (km) of algorithms

From this first set of results which compares the solution quality of each algorithm against TSP size we can see that overall, the nearest neighbour was the best algorithm, as it consistently found the highest fitness solutions for each of the datasets. Not only this, but the nearest neighbour also had small standard deviations for each dataset sample, which tells us that this method is extremely consistent in finding high quality solutions. Compared to the other algorithms the nearest

neighbour was much more effective when solving the largest dataset of 10,000 points, the nearest neighbour's average distance of 29,388.41km is a 95% decrease in distance when compared to the next best average distance which was found by the 2-opt method (724,396.75km). This demonstrates how efficient this heuristic can be for solving large problems.

Although the 2-Opt algorithm did not perform as well as the nearest neighbour for the smallest and largest datasets, there was very little difference between the 2-opt's average solution fitness and the NN's average solution fitness for the datasets containing 100 and 1,000 vertices, with the nearest neighbour being slightly better for both. 2-Opt also showed that it was capable of being extremely consistent in terms of solution quality, as the standard deviations for the first three datasets were the smallest of all 6 algorithms, although the consistency did drop for the final and largest dataset.

Both nearest neighbour and 2-opt have shown themselves to be good options for the TSP, as they can find better quality solutions when compared to the other algorithms that were used. Nearest neighbour and 2-opt are also quite simple methods for solving the TSP, especially when compared to methods such as the evolutionary algorithm which contains many parameters and operators that should be optimised and tuned in order to get the best results. Interestingly, the EA performed the worst overall. For the smallest dataset the EA's performance was middling, not the best performance but also not the worst. As the size of the TSPs increased the performance gradually got worse for the EA, for the second dataset of 100 vertices the EA's average distance was almost double that of the next highest average which was produced by the hill climber, and the gap between the EA and the other algorithms only increased for the 1,000 vertex TSP.

The performance of the hill climber and simulated annealing algorithms was fairly similar throughout these results, however, with exception to the first dataset, the simulated annealing algorithm generally produced better results than the hill climber. This is what we would expect to see based on the fact that the hill climber has no way of escaping a local optimum once it has been caught in one, while the simulated annealing's cooling function allows it to escape local optima in order to explore more of the search space and potentially find better solutions. The results from these two algorithms are nonetheless extremely close, as can be seen by how similar the two

corresponding algorithm's lines are on figure 14. Simulated annealing seems to have had the biggest advantage over the hill climber when applied to the largest problem, as there is a much greater difference between the two algorithm's average solutions for this dataset.

Finally, there is the tabu search, this method was on par with the hill climber and simulated annealing algorithms for most of the datasets, showing fairly similar results for the 10, 100 and 1,000 vertex datasets. We can also see this trend in figure 14 as the line for tabu search is very close to the lines for hill climber and simulated annealing, however, as the number of vertices increases the solution quality of the tabu search gradually starts to increase and the performance becomes more comparable to the evolutionary algorithm. From this we can assume that the tabu search will work more efficiently for smaller datasets.

4.2.2 Wall Clock Time results

The next set of results will show us how fast the algorithms are at completing 1,000 iterations for each dataset. The only algorithm which will not complete 1,000 iterations is the NN, because the NN starts from one vertex and iteratively adds vertices to the solution until there are no more remaining vertices to be visited. After the NN has found its final solution it does not make any changes to it. NN also does not take iterations into account unlike the other algorithms, therefore for each run of the NN we will record the time take to generate a complete tour. Table 16 will be set out in a similar way to Table 15 for the previous results, with the exception that the average and standard deviation sub-columns for each dataset will be referring to the wall clock time in seconds at the end of the 1,000 iterations rather than the distance of the tour.

Algorithm	10-vertices		100-vertices		1,000-vertices		10,000-vertices	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
EA	0.020	0.019	0.057	0.033	0.900	0.076	63.44	1.280
TS	0.617	0.066	5.689	0.084	64.02	0.472	1559	3.189
SA	0.004	0.003	0.005	0.004	0.027	0.017	0.704	0.077
HC	0.007	0.003	0.039	0.008	0.356	0.026	3.628	0.164
2O	0.006	0.003	0.038	0.010	0.344	0.054	3.627	0.104
NN	0.002	0.001	0.006	0.004	0.123	0.022	10.19	0.197

Table 16 – Wall clock time (s) of each algorithm when solving 1,000 iterations

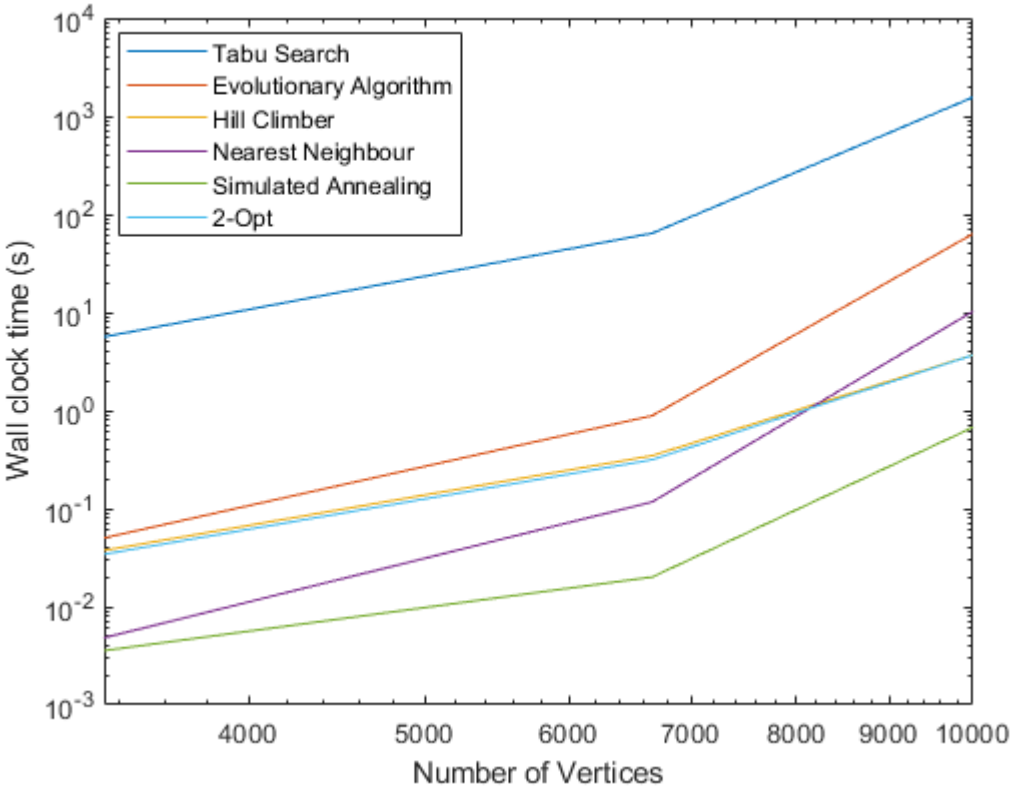


Figure 15 – Average runtime (s) of algorithms

Being able to perform iterations more quickly is a useful technique as it means that given a time limit, a faster algorithm will be able to perform more iterations and will therefore have a greater chance of exploring the search space and finding good solutions. From table 16 we can see that the performance of all the algorithms is similar for the smallest dataset. Despite this, simulated annealing is clearly the fastest algorithm. Although the nearest neighbour was slightly faster for the 10-vertex dataset, simulated annealing had the advantage in all other datasets, most impressively the algorithm was able to perform 1,000 iterations of the largest dataset in under 1s.

The nearest neighbour is arguably the next best algorithm in terms of wall clock time, we can see from Table 16 that this algorithm performed well up until the final dataset, where it was overtaken by both 2-Opt and hill climber. However, we can also conclude from Figure 15 that the nearest neighbour was the worst algorithm when it came to scaling with the size of the problem, as it has the steepest line. This may also be due to the fact that nearest neighbour had to generate a 10,000-vertex tour while all of the other algorithms just had to complete 1,000 iterations. Either way, the

nearest neighbours' impressive performance in terms of both distance and wall clock time make show us that it is clearly the best algorithm.

The performances of the 2-opt and Hill climber algorithms exhibited extremely similar runtimes throughout, at each dataset there was less than a difference of 1s between each algorithm, with 2-opt having the slight edge. Because the difference between these two algorithms is so minute, it can be assumed that they are the same in terms of wall clock time. However, as 2-opt had a faster runtime and produced higher fitness solutions, 2-opt is the overall better algorithm.

Once again, it was a poor performance for the EA, which ended up having the second worst wall clock time. The reason that the EA is slower than the previous four mentioned algorithms is due to the complexity and number of computations that must be made for each iteration of the EA. As well as having to handle a population of 400 chromosomes the EA must also undergo selection, crossover, replacement and possibly mutation at each iteration. This is a much greater number of computations per iteration when compared to that of the 2-opt, hill climber, simulated annealing or nearest neighbour algorithms. Unfortunately, the complexity of the algorithm doesn't seem to help find high quality solutions, like with tabu search.

Finally, tabu search was the worst algorithm when trying to solve 1,000 iterations as quickly as possible. Even for the smallest dataset, despite taking on average only 0.620 seconds to complete all the iterations, this is still much longer than the next slowest average time, which was 0.019 seconds by the EA. As the number of vertices reaches 10,000 the average time to complete 1,000 iterations has increased to over 25 minutes. Like the EA, this big gap in wall clock time between the tabu search and the other algorithms is due to the complexity of the algorithm. The tabu search generates, calculates and compares the distance for 250 possible solutions per iterations, when compared to the simpler algorithms such as hill climber which only needs to generate one solution per tour, it is clear why the tabu search takes much longer, especially as the size of the problem increases. The size of the neighbourhood also affects the time of the tabu search, as the smaller the number of neighbours the faster the algorithm will run but doing so will likely cause a decrease in the distances found. The tabu search can be a balancing act between quality of solutions and time taken, which allows the algorithm to be tweaked to match the designers desired outcome.

4.3 Discussion

There are a couple of factors that contribute to the nearest neighbour's dominance over the other algorithms, first, the nearest neighbour doesn't start with a completed tour, instead it starts from one vertex and builds the rest of the tour with the help of problem specific knowledge. Because the nearest neighbour is able to identify the closest unvisited vertex, it gradually builds its final solution iteration by iteration, ultimately leading to high quality solutions, thanks to the problem specific knowledge it is able to utilise. The other algorithms all start with one or more randomly generated permutation and make random changes to that solution in order to create new and hopefully better solutions. However, as the size of the problem and the size of the search space increase the chances that the other five algorithms will randomly generate a high-quality solution as the starting solution is increasingly unlikely. A technique employed by some when designing an optimisation algorithm is feeding good quality solutions into the algorithm to give it a better starting position within the search space. In the literature, the nearest neighbour algorithm has often been utilised in this way. These projects usually do not record the results of the standard nearest neighbour algorithm (Singh et al, 2015) or will only show the results of the nearest neighbour when compared to a hybrid algorithm that has also employed the nearest neighbour approach (Dhakal & Chiong, 2008). This makes it difficult to compare the results of the nearest neighbour implemented in this project to those implemented in previous works. These previous works agree that the nearest neighbour is an algorithm which can produce high quality results in a short amount of time and that implementing the results of the nearest neighbour into another algorithm can improve the solution quality/computation time of that algorithm. However, they all fail to mention how well the nearest neighbour and the algorithm it is going to be hybridised with perform on their own. For instance, the nearest neighbour was hybridised with the 2-opt algorithm by Sathyan et al. (2015), the results found that 2-opt with nearest neighbour outperformed the standard 2-opt algorithm in terms of computation time, however, there are no results comparing the performance of the standard nearest neighbour algorithm. When evaluating the results found in this project we can derive that the nearest neighbour does indeed outperform the 2-opt algorithm, as well as all of the other included algorithms, especially when it comes to large scale problems.

Like the nearest neighbour algorithm, 2-opt is a simple method as it requires no experimentation with parameters or operators. Despite its simplicity, 2-opt garnered high-quality solutions throughout this study. Given how easy it is to implement the 2-opt algorithm and the quality of the solutions it produces, it is apparent why 2-opt has been typically hybridized with other algorithms in order to improve upon their solutions. As mentioned earlier, Sathyan et al. (2015) hybridized the 2-opt and nearest neighbour algorithms. When we compare the results of the standard 2-opt against the hybrid 2-opt we can see that there is very little difference between the two in terms of solutions quality (the standard algorithm was actually able to find a slightly shorter minimum distance). Many other algorithms which are hybridized with the nearest neighbour technique usually improve upon on the results they found without the heuristic's help, such is the case with Dhakal & Chiong (2008). However, because the 2-opt algorithm made no improvement, we can conclude that the 2-opt algorithm is generally on par with the nearest neighbour heuristic. The results of this project confirm this as there is only a minor difference between the two algorithms when applied to the 10, 100 and 1,000 vertex datasets, but the difference between the two algorithm's quickly increased as the number of vertices increased from 1,000 to 10,000, suggesting that the 2-opt algorithm either cannot handle large scale problems, or, needed more than 5-minutes in order to further increase its solution quality.

The hill climber algorithm is the last 'simple' algorithm included in this project but has been explored much less than the previous two algorithms in the literature. As with the nearest neighbour and 2-opt algorithms, most studies which do mention the hill climber usually are trying to combine the hill climber strategy with other algorithms in order to increase solution quality. From this study we can see that the performance of the hill climber was rather average throughout, in terms of both solution quality and speed the hill climber was never produced the best results but also never produced the worst results. The average performance of the algorithm, along with the fact that the hill climber is a greedy technique which will not accept a worse answer under any circumstances may explain the lack of interest in this algorithm in the literature, as the hill climber is likely to converge towards local optima. These drawbacks are both improved by simulated annealing, which is able to escape local optima by accepting worse solutions and overall was faster and found better solutions. Even when trying to implement greedy techniques to improve starting solutions of previously existing

algorithms, nearest neighbour is clearly the superior choice, as it produces solutions which are far better than those produced by the hill climber. 2-opt would also be a good choice for producing good starting solutions, as 2-opt also produces higher quality solutions than the HC. As well as this, when compared to the hill climber, both the nearest neighbour and 2-opt had faster runtimes for 3 of the 4 datasets. Given that nearest neighbour and 2-opt have a superior performance when compared to hill climber we can see why the hill climber may been used less frequently in previous studies.

Although simulated annealing was able to improve upon the hill climber algorithm, there isn't a huge increase in solution quality. It may be the case that the parameters chosen for the simulated annealing algorithm weren't optimal, which prevented the algorithm from finding even better results. The simulated annealing algorithm in this study has a cooling rate of 0.8 and a staring temperature of 500 degrees, as explained in Section 4.1, these parameters are much lower than expected, probably due to the parameter experiments not allowing the algorithm enough time to converge towards a local optimum, which made cooling at a faster rate more desirable. Previous works which have studied the performance of simulated annealing when applied to the TSP have found that higher temperatures produce better results. For example, Ye & Rui (2013) and Adewole et al. (2012) both used an initial temperature of 10,000, while using cooling rates of 0.95 and 0.9999 respectively. Alternatively, another improvement could be to make the algorithm adapt to the size of problem it is given. This type of simulated annealing algorithm was used by Geng et al. (2011), here the initial temperature is set to 1,000, but the cooling rate is dependent on the number of cities the algorithm has to travel to. This technique is useful as we would expect larger problems to need a higher cooling rate as the algorithm will likely have a much larger search space to explore, and in order to explore that search space the temperature of the system needs to decrease more slowly. This method could potentially increase the solution quality of simulated annealing for all problem sizes.

Tabu search, while being able to find high quality solutions thanks to its neighbourhood function, is weighed down by its runtime, which is far greater than the other algorithms included in this study. As the size of the problem increases, the time required to run the algorithm increases exponentially. When the algorithms were

given a set amount of time to run, the tabu search produced solutions which were similar in quality to those produced by the hill climber and simulated annealing algorithms, despite the fact that the tabu search would have fewer iterations to work with. This shows us that despite having a long runtime, the tabu search has a much greater capacity for improvement per iteration, as the tabu search produces 250 times more solutions per iteration when compared to the hill climber and simulated annealing techniques. So, despite the run time of tabu search, it can still be a competitive method when solving the TSP. Given the algorithms long run time, tabu search would be most appropriately used when applied to design problems, which do not require the algorithm to produce a quality solution in a short amount of time.

Due to the poor solution quality and relatively slow run time of the EA used in this project, especially when compared to the other algorithms included, it would be hard to suggest it as viable method for solving real-world TSPs. However, this is not reflected by the results of many similar projects, where the EA has been praised for being able to find high quality solutions. The difference between many of those projects and this project is the use of experimentation with a range of different operators. As previously stated, there was no experimentation with different EA operators in this project in order to keep the experimentation time down. Other projects, especially those which focus solely on an EA, experiment with different operators in order to find the best combination of operators possible. Some projects focus on one specific type of operator, such as mutation or selection (Abdoun et al., 2012; Razali & Geraghty, 2011), while others focus on all operator types and finding good synergies between different operators (Contreras-Bolton & Parada, 2015). Regardless, by experimenting with different operators these researchers were able to improve upon their EAs and choose operators that suit their specific problem, which would cause their EAs to produce better results. Adewole et al. (2012) performed a comparison between simulated annealing and an evolutionary algorithm for solving the TSP, they found that the EA's runtime was much greater than the simulated annealing's as the size of the problem increased. This matches what was found in this study, however, Adewole also found that the distance found by the algorithms was quite similar for datasets that ranged from 10-100 vertices. Although this is true for the 10-vertex dataset used in this project, there is a huge difference between the two algorithms as the dataset increases to include 100 vertices. Therefore, we can assume that our EA is underperforming when compared to other studies' EAs, and

that experimenting with different operators is just as important as experimenting with parameters, if not more important. Another important factor which sets apart the EA used in this project and EAs from the literature is the inclusion of alternative search techniques in order to improve upon the answers that the EA produces. The EA in this project does not rely upon any other algorithm in order to improve its answers, although, this has proven to be an extremely useful technique in the literature. For example, take the study conducted by Singh et al (2015), where both a nearest neighbour and a 2-opt algorithm were implemented into an evolutionary algorithm. The result of hybridising the EA was that it “remarkably increases the performance of EA” when applied to TSPs. So, in order to take full advantage of an EA it is advised to explore different parameters, operators and search techniques in order to find the best combination of settings for the given problem. Through proper experimentation and even the inclusion of some addition search techniques these complex techniques can greatly improve their performance. For instance, by feeding the output of the nearest neighbour into another algorithm we can guarantee that the new algorithm will perform at least as well as the nearest neighbour, however, from here the algorithm has the ability to improve upon the solution that the nearest neighbour produced. And with the large number of possible operators that have been developed over the years, it is unlikely that a developer will find the best combination of operators for the given problem on the first try. Operators that found high quality solutions for one problem will not guarantee good solutions from another problem, thus, to get the most out of metaheuristic optimisation algorithms it is recommended to explore and experiment with a wide range of parameters and operators where possible.

Of the algorithms which were included in this project, the nearest neighbour was the most efficient, closely followed by the 2-opt algorithm. Even though these algorithms are simplistic in nature they have shown that greater complexity does not lead to greater solution quality.

5 Conclusion

In this study six algorithms were implemented in order to solve real-world TSPs of various different sizes. The algorithms included all demonstrated different methods for solving the same problem, each with different success rates. All of the algorithms implemented work as intended although some of the algorithms may not have been implemented as effectively as possible as the more complex algorithms such as the EA missed out on operator experimentation. Had there been sufficient testing these algorithms could have likely performed even better.

GraphHopper also was implemented successfully and was able to calculate the real-world distance around the tours that were produced by the algorithms. While this aim was met, it could have been improved. The tours could only be posted to the GraphHopper application through cURL, this is because GraphHopper was implemented independently of the main project which had all of the algorithms and TSP classes. This also led to having to implement features which were not initially intended such as saving the tour to a JSON file. Implementing the GraphHopper application into the TSP project could have allowed for a more intuitive way to use the routing software, as getting the results from GraphHopper was a time-consuming task. This is because only one file could be posted at a time and each algorithm had to be run 10 times, so when collecting the data for the results section 60 files had to be posted to the GraphHopper server. If there was a quicker way of using the routing software there could have been extra time for operation experimentation. Visualisation of the route was originally one of the aims for this project (Appendix 1), this was implemented but later dropped as the number of vertices increased and GraphHopper could not handle visualising 1,000 or 10,000 vertices at a time.

For each algorithm we were able to collect the average distance and runtime for each of the four datasets, however, limitations on the algorithms such as iteration/time constraints may have also produced results that are not representative of how the algorithms performed when given unlimited iterations and time, which could make the results less reliable. Initially, it was expected that the more complex metaheuristics such as EA and TS would outperform the simple heuristic like the NN and 2-Opt, however the results show the opposite of this. The simplistic heuristics performed better in terms of both tour distance and runtime, while the EA was the worst algorithm out of the bunch. The NN was the best algorithm overall, likely due to its

ability to utilise problem specific knowledge, something that none of the other algorithms could do. Being able to see which vertices are closest to the current vertex allows the NN to generate good, but not optimal answers. However, without a good starting position it is hard for the metaheuristics to reach the same highs as the NN, first, the likelihood of generating a good permutation to use as the starting solution get increasingly unlikely as the number of vertices increases. Second, the metaheuristics can only make random changes to the solution and can only see how good their current solution is based on the fitness function, meaning there is no guarantee that the algorithm will make the right choices. Metaheuristics are not without their merit however, it is understandable why many metaheuristics are paired with heuristics, as the heuristics are able to produce good starting solutions and the metaheuristics are then able to improve upon them. We are also able to see the scalability of the algorithms, for example 2-opt performed well for the first three datasets, but once the TSP increased to 10,000 vertices the distance produced by the algorithm quickly rose. Other algorithms such as the TS also had trouble scaling, as the size of the problem increased the time required for each iteration grew exponentially. The EA also had a large increase in runtime as the size of the problem increased, although not as much as the TS. Overall SA was the best algorithm when it came to scaling, as the runtime of the algorithm was still extremely competitive, even when the problem had 10,000 vertices to visit.

5.1 Future work

The biggest limitation faced in this project was the parameter tuning and operation experimentation. Due to having to run each algorithm 10 times when performing each parameter tuning experiment, there was a large number of runs that had to be performed, 340 runs were required for the parameter tuning alone. Because of the large number of runs that had to take place some compromises had to be made, firstly, as the algorithms were being tested on a 10,000 vertex dataset, the algorithms would not be allowed to run until convergence as this would take far too long, especially for algorithms such as the TS which took over 25 minutes to perform 1,000 iterations. This is why the parameter tuning runs were capped at 1,000 iterations; however, this may have affected the results of the parameter tuning, making the results less accurate. As explained there were some unexpected results collected during this stage, especially for the SA parameters, the iteration limit could have been the reason behind these unexpected results. The decision to forego the

operator experimentation was meant to alleviate the amount of time spend testing but doing this probably decreased the chances of finding higher quality for some algorithms. Not allowing the algorithms to run until they converged was a decision made after realising how long it took the tabu search algorithm to run for the larger datasets, If the tabu search didn't have a time or iteration limit it could have potentially run for days before converging, which isn't a possibility when each algorithm needs to run each dataset multiple times in order to collect averages. However, in an ideal situation the algorithms would have been allowed to run until they converged, this way the distance and wall clock time could have been gathered from one set of experiments rather than two and the results would be more trustworthy.

Future work could address these limitations. From the research conducted for this project, there were no studies in the literature which compared the performance of a standard EA with no operator experimentation (such as the one in this project) with an EA which had been extensively experimented with and tuned to its specific problem. There could also be work conducted into which local search algorithms complement the EA best when hybridised, as most studies which include hybrid EAs never experiment with more than one or two local search algorithms.

Furthermore, if all of the algorithms that were included in this project were allowed to run up until the point of convergence, the results found may be wildly different from those reported in section 4.2. Design problems, which require only a single good solution with no regard to the time taken, were not considered in this project, as all of the algorithms were run with restrictions, either a time or iteration limit. Therefore, the results of this project only show us which algorithms would be best for repetitive problems – problems which require reliable algorithms that consistently produce good solutions.

5.2 Self-appraisal

This project was completed over the course of two semesters, in that time I had to conduct research into numerous different areas of the subject including: optimisation algorithms and how to implement them, routing software, sending HTTP requests as well as how to use software such as cURL, Maven and MATLAB. I found that learning how to use GraphHopper was the most difficult aspect of the project, especially since the version of GraphHopper I was using was cloned from GraphHopper's master

branch on GitHub. Because of this there were a few occasions where I ran into bugs that I did not know how to fix as there was no information about the bug in the developer documentation. In order to address these bugs, I had to make full use of GraphHopper's online forum, as well as issues tab on GraphHopper's GitHub page, in order to get the necessary help from the developers (Appendix 6).

From a project management standpoint, I believe I've worked quite well, throughout the duration of the project I followed my Gantt chart (Appendix 3) quite closely and made amendments to it when necessary. Because of this I never felt like I was behind schedule and was able to balance the workload of this project with my other studies. Although there were some aspects of the project I wish I could have dedicated more time to such as testing the algorithms for each individual dataset for more precise results and allowing the algorithms to run until convergence, I made the decision not to go ahead with either of these. This was because I was aware that if I conducted all of the testing I wanted to do I would soon run out of time, being realistic with the scope of the project allowed me to stay on schedule and complete the project on time. I also attended weekly meetings with my supervisor, taking notes at each meeting to track my progress for each week (Appendix 3). Attending these supervisor meeting was imperative to the success of the project as it gave me crucial support and guidance throughout the project and allowed me to discuss and develop ideas for the project.

References

Abdoun, O., Abouchabaka, J., & Tajani, C. (2012). *Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem*. Retrieved from http://arxiv.org/abs/1203.3099

Adewole, A. P., Otubamowo, K., & Egunjobi, T. O. (2012). A Comparative Study of Simulated Annealing and Genetic Algorithm for Solving the Travelling Salesman Problem. *International Journal of Applied Information Systems*, *4*(4), 6–12. https://doi.org/10.5120/ijais12-450678

Ahrens, B. M. (2013). A tour construction framework for the travelling salesman problem. *Conference Proceedings - IEEE SOUTHEASTCON*, 1–8. https://doi.org/10.1109/SECON.2013.6567459

Ancău, M. (2008). The optimization of printed circuit board manufacturing by improving the drilling process productivity. *Computers & Industrial Engineering*, *55*(2), 279–294. https://doi.org/10.1016/j.cie.2007.12.008

Basseur, M., & Goëffon, A. (2015). Climbing combinatorial fitness landscapes. *Applied Soft Computing Journal*, *30*, 688–704. https://doi.org/10.1016/j.asoc.2015.01.047

Bauer, R., Columbus, T., Rutter, I., & Wagner, D. (2016). Search-space size in contraction hierarchies. *Theoretical Computer Science*, *645*, 112–127. https://doi.org/10.1016/j.tcs.2016.07.003

Bland, R. G., & Shallcross, D. F. (1989). Large travelling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation. *Operations Research Letters*, *8*(3), 125–128. https://doi.org/10.1016/0167-6377(89)90037-0

Coello, C. A. C. (2005). Twenty Years of Evolutionary Multi-Objective Optimization: A Historical View of the Field. *Evolutionary Computation Group*, 1–20. Retrieved from http://delta.cs.cinvestav.mx/~ccoello/EMOO/coello06a.pdf.gz

Contreras-Bolton, C., & Parada, V. (2015). Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PLoS ONE*, *10*(9), 1–26. https://doi.org/10.1371/journal.pone.0137724

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., & Semet, F. (2002). A Guide to Vehicle Routing Heuristics. *Source: The Journal of the Operational Research Society*, *53*(5), 512–522. https://doi.org/10.1057/palgrave/jors/2601319

Deep, K., & Mebrahtu, H. (2011). Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman Problem. *International Journal of Combinatorial Optimization Problems and Informatics*, *2*(3), 2–24.

Delgado-Antequera, L., Laguna, M., Pacheco, J., & Caballero, R. (2019). A bi-objective solution approach to a real-world waste collection problem. *Journal of the Operational Research Society*, *0*(0), 1–12. https://doi.org/10.1080/01605682.2018.1545520

Deng, Y., Liu, Y., & Zhou, D. (2015). An Improved Genetic Algorithm with Initial Population Strategy for Symmetric TSP. *Mathematical Problems in Engineering*, *2015*. https://doi.org/10.1155/2015/212794

Dhakal, S., & Chiong, R. (2008). A hybrid nearest neighbour and progressive improvement approach for travelling salesman problem. *Proceedings - International Symposium on Information Technology 2008, ITSim*, *1*, 6–9. https://doi.org/10.1109/ITSIM.2008.4631549

Duman, E., & Or, I. (2004). Precedence constrained TSP arising in printed circuit board assembly. *International Journal of Production Research*, *42*(1), 67–78. https://doi.org/10.1080/00207540310001601073

Flood, M. M. (1956). The Traveling-saleman problem. *Organization Science*, *4*(1), 61–75.
Geng, X., Chen, Z., Yang, W., Shi, D., & Zhao, K. (2011). Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing Journal*, *11*(4), 3680–3689. https://doi.org/10.1016/j.asoc.2011.01.039
Ginting, H. N., Osmond, A. B., & Aditsania, A. (2019). Item Delivery Simulation Using Dijkstra Algorithm for Solving Traveling Salesman Problem. *Journal of Physics: Conference Series*, *1201*(1). https://doi.org/10.1088/1742-6596/1201/1/012068
Hansen, P. B. (1992). Simulated annealing. *Natural Computing Series*, 59–79. https://doi.org/10.1007/978-3-319-93073-2_4
Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press.
Jadon, R. S., & Datta, U. (2013). Modified ant colony optimization algorithm with uniform mutation using self-adaptive approach for travelling salesman problem. *2013 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013*, (c), 1–4. https://doi.org/10.1109/ICCCNT.2013.6726752
Jafarzadeh, H., Moradinasab, N., & Elyasi, M. (2017). An Enhanced Genetic Algorithm for the Generalized Traveling Salesman Problem. *Engineering Technology & Applied Science Research*, *7*(6), 2260–2265.
Jalaluddin, M. F., Kamaru-zaman, E. A., Abdul-Rahman, S., & Mutalib, S. (2017). Courier Delivery Services Visualisor (CDSV) with An Integration of Genetic Algorithm and A* Engine. *Proceedings of the 6th International Conference on Computing and Informatics, ICOCI 2017*, (155), 126–131.
Jia, J., Pan, J. S., Xu, H. R., Wang, C., & Meng, Z. Y. (2016). An Improved JPS Algorithm in Symmetric Graph. *Proceedings - 2015 3rd International Conference on Robot, Vision and Signal Processing, RVSP 2015*, 208–211. https://doi.org/10.1109/RVSP.2015.56
Juneja, S. S., Saraswat, P., Singh, K., Sharma, J., Majumdar, R., & Chowdhary, S. (2019). Travelling Salesman Problem Optimization Using Genetic Algorithm. *Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019*, 264–268. https://doi.org/10.1109/AICAI.2019.8701246
Karabulut, K., & Fatih Tasgetiren, M. (2014). A variable iterated greedy algorithm for the traveling salesman problem with time windows. *Information Sciences*, *279*, 383–395. https://doi.org/10.1016/j.ins.2014.03.127
Laporte, G., Gendreau, M., Potvin, J. Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, *7*(4–5), 285–300. https://doi.org/10.1111/j.1475-3995.2000.tb00200.x
Marinakis, Y. (2010). A hybrid genetic Particle Swarm Optimization Algorithm for the vehicle routing problem. *Expert Systems with Applications*, *37*(2), 1446–1455. https://doi.org/10.1016/j.eswa.2009.06.085
Nagata, Y., & Soler, D. (2012). A new genetic algorithm for the asymmetric traveling salesman problem. *Expert Systems with Applications*, *39*(10), 8947–8953. https://doi.org/10.1016/j.eswa.2012.02.029
Nazif, H., & Lee, L. S. (2012). Optimised crossover genetic algorithm for capacitated vehicle routing problem. *Applied Mathematical Modelling*, *36*(5), 2110–2117. https://doi.org/10.1016/j.apm.2011.08.010
Nguyen, H. D., Yoshihara, I., Yamamori, K., & Yasunaga, M. (2007). Implementation

of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *31*(1), 92–99.

Nordin, N. A. M., Zaharudin, Z. A., Maasar, M. A., & Nordin, N. A. (2012). Finding shortest path of the ambulance routing: Interface of A algorithm using C# programming. *SHUSER 2012 - 2012 IEEE Symposium on Humanities, Science and Engineering Research*, 1569–1573. https://doi.org/10.1109/SHUSER.2012.6268841

Osaba, E., & Diaz, F. (2012). Comparison of a memetic algorithm and a tabu search algorithm for the traveling salesman problem. *2012 Federated Conference on Computer Science and Information Systems, FedCSIS 2012*, 131–136.

Piarsa, I. N., Hadi, E. S., & Wirdiani, N. K. A. (2015). *Rural Road Mapping Geographic Information System Using Mobile Android*. *12*(3), 95–100.

Razali, N. M., & Geraghty, J. (2011). Genetic algorithm performance with different selection strategiesin solving TSP. *Proceedings of the World Congress on Engineering 2011, WCE 2011*, *2*, 1134–1139.

Sahputra, M. F. A., Devita, R. N., Siregar, S. A., & Kirana, K. C. (2016). Implementation of Traveling Salesman Problem (TSP) based on Dijkstra ' s Algorithm in Logistics System. *International Journal of Electrical and Electronics Engineering*, *14*(1), 39–44.

Sathyan, A., Boone, N., & Cohen, K. (2015). Comparison of Approximate Approaches to Solving the Travelling Salesman Problem and its Application to UAV Swarming. *International Journal of Unmanned Systems Engineering*, *3*(1), 1–16. https://doi.org/10.14323/ijuseng.2015.1

Sedighpour, M., Yousefikhoshbakht, M., & Mahmoodi, N. (2011). An Effective Genetic Algorithm for Solving the Multiple Traveling Salesman Problem. *Journal of Optimization in Industrial Engineering*, *8*, 73–79.

Shim, V. A., Tan, K. C., Chia, J. Y., & Chong, J. K. (2011). Evolutionary algorithms for solving multi-objective travelling salesman problem. *Flexible Services and Manufacturing Journal*, *23*(2), 207–241. https://doi.org/10.1007/s10696-011-9099-y

Singh, D. R., Singh, M. K., & Singh, T. (2015). A hybrid heuristic algorithm for the Euclidean traveling salesman problem. *International Conference on Computing, Communication and Automation, ICCCA 2015*, (1), 773–778. https://doi.org/10.1109/CCAA.2015.7148514

Singh, G., & Chopra, V. (2012). An Analysis of various techniques to solve Travelling Salesman Problem: A Review. *Numerische Mathematik*, *1*(1), 269–271.

Snyder, L. V, Snyder, L. V, Daskin, M. S., & Daskin, M. S. (2006). A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, *174*, 38–503. https://doi.org/10.1016/j.ejor.2004.09.057

Soon, G. K., Anthony, P., & Teo, J. (2008). The effect of varying the crossover rate in the evolution of bidding strategies. *Proceedings of the 4th IASTED International Conference on Advances in Computer Science and Technology, ACST 2008*, 162–167.

Urquhart, N., Scott, C., & Hart, E. (2010). Using an evolutionary algorithm to discover low CO 2 tours within a travelling salesman problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6025 LNCS*(PART 2), 421–430. https://doi.org/10.1007/978-3-642-12242-2-43

Xu, J., Pei, L., & Zhu, R. Z. (2018). Application of a genetic algorithm with random

crossover and dynamic mutation on the travelling salesman problem. *Procedia Computer Science*, *131*, 937–945. https://doi.org/10.1016/j.procs.2018.04.230

Ye, G., & Rui, X. (2013). An improved simulated annealing andgenetic algorithm for TSP. *Proceedings of 2013 5th IEEE International Conference on Broadband Network and Multimedia Technology, IEEE IC-BNMT 2013*, (2), 6–9. https://doi.org/10.1109/ICBNMT.2013.6823904

Appendix 1 Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Optimizing a real-world travelling salesman problem with evolutionary computing

Overview of Project Content and Milestones

The Main Deliverable(s):

- An application that includes: a search space for the TSP that reflects real world map data, algorithms (including at least one evolutionary algorithms) to solve the TSP as optimally as possible.
- A set of results for each algorithm that will be used to compare the different approaches (distance travelled, travel time, execution time)

The Target Audience for the Deliverable(s):

- Optimization/TSP researchers
- Logistics companies who plan delivery routes for businesses.
- Manufacturers of satnav systems.
-

The Work to be Undertaken:

- Investigate methods that are typically used to solve the TSP and conduct a literary review of the TSP,
- Finding and applying a real-world situation to the TSP,
- Implementing a real-world map into the TSP search space,
- Test the implemented algorithms to ensure they work as intended,
- Analysing and comparing the results in order to evaluate which methods are the best when optimising the TSP.

Additional Information / Knowledge Required:

- I will need to explore the evolutionary computing methods that I decide to use, how they work and how to implement them.

- I will also need to learn how to implement a real map as a TSP search space.

Information Sources that Provide a Context for the Project:

- Michalewicz, Z. and Fogel, D. (2011). *How to solve it*. Berlin: Springer.
- An Analysis of various techniques to solve Travelling Salesman Problem: A Review (Singh & Chopra, 2012)
- Brown, L. (2019). *How green is your parcel?* [online] BBC News. Available at: https://www.bbc.co.uk/news/science-environment-47654950 [Accessed 3 Oct. 2019].

The Importance of the Project:

- Although the TSP is an abstract problem, it can be applied to real world problems, such as trying to find the optimal route for delivery drivers.
- By applying evolutionary computing methods to a real-world TSP problem and comparing the results we can find which methods are most valuable when optimizing routing problems such as the TSP.
- Optimising routing problems has numerous benefits including reducing cost of fuel, reducing carbon dioxide emissions and reducing time taken to complete the tour of every destination. This can benefit businesses, consumers and the environment.

The Key Challenge(s) to be Overcome:

- Finding a real-world situation (such as a delivery route) to implement into the TSP may be difficult as companies may not be able to provide their customers' home addresses for security reasons.
- Understanding, implementing and testing the evolutionary algorithm formulas may take some time.
- Managing the project on top of other modules and part-time work may prove to be a challenge.

Appendix 2 Second Formal Review Output

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Robbie McNeil

Supervisor: Brian Davison

Second Marker: Neil Urquhart

Date of Meeting: 26 Nov 2019

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes**

If not, please comment on any reasons presented

Please comment on the progress made so far

Because of the confusion around the aim, the lit review is incomplete. Either the aim needs to change or the lit review also needs to cover vehicle routing problems.

Is the progress satisfactory? **yes**

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

Clarify aim with regard to the specific problem – TSP or VRP

Be careful of the distinctions. VRP problems are constrained – the constraint provides the rationale for having multiple vehicles. Given the current timescale, probably best to concentrate on TSP.

Does the student have a plan of work? **yes**

If yes then please comment on that plan otherwise write down your suggestions.

Needs further development. Be sure to update Gantt chart on a regular basis, but the actual work is the main priority.

Does the student know how they are going to evaluate their work? **yes**

If yes then please comment otherwise write down your suggestions.

Use known algorithms such as nearest-neighbour, 2-opt, etc. as benchmarks for evaluation

Any other recommendations as to the future direction of the project

Be clear about definitions and taxonomy of type of algorithm.

Code metaheuristics directly without using tools; then use GraphHopper to find shortest routes.

Invent specific problems for TSP of adequate size (100+) then compare performance of different algorithms.

Signatures: Supervisor Second Marker

Student

The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

Appendix 3 Diary Sheets (or other project management evidence)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 20.09.2019 **Last diary date: -**

Objectives:

- Fill in initial project overview
- Make a Gantt chart for the project
- Find/Review literature and sources related to the subject for use in the literature review
- Make a list of potential titles for the project

Progress:

- Completed all IPO sections
- Made Gantt chart draft, checked by Brian, to be updated over the next week
- Found 7 relevant literature sources

Supervisor's Comments:

- Good reflective questions about the process
- Draft of IPO ahead of schedule
- Found good reference

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 30.09.2019 **Last diary date: 20.09.2019**

Objectives:

- Find more literary sources to use for literature review
- Begin to read the literature I have found and take notes on the articles/journals read
- Update Gantt chart to include non-working hours (Exam, coursework deadlines, holidays)
- Download Mendeley & update IPO with correctly formatted references

Progress:

- Found 10 more journals, started to read some literature and make notes in a word document
- Started planning the structure of the literature review
- Gantt chart updated with exam dates, coursework deadline and out-of-work hours
- Downloaded mendeley and uploaded literature sources to it for easier citing
- Did not get as much reading done as I would have liked due to trying to keep up with background reading and practical work for other modules (However I'm still ahead of time according to my Gantt chart)

Supervisor's Comments:

- Use iterative approach for literature – read introduction and conclusion
- Approach to reading literature will improve over time

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 07.10.19 **Last diary date: 30.09.19**

Objectives:

- Read through introductions and conclusions of found literature and decide which sources will and won't be useful
- Start reading through lit sources that are useful
- Continue to take notes

Progress:

- Read through all introductions and conclusions and got rid of the articles that won't be useful to the project,
- Started to read through the rest of the remaining literature, highlighted important or useful parts via Mendeley & took notes on word,
- Read Neil's article on using GAs to reduce CO2 emissions and came up with some questions for him regarding it,
- Thought about possible real-life TSPs I could implement.

Supervisor's Comments:

- Hold off on writing the literature review draft until I know what the objective of the project is.
- Don't worry about implementation methods just yet (i.e. google maps, open street view, databases…)
- Make decision as to what I'm optimising and for whom.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 14.10.19 **Last diary date: 07.10.19**

Objectives:

- Research using time as an evaluation metric for TSP
- Research MOTSP
- Continue reading found literature
- Email Neil and set up a meeting to discuss fleshing out the project
- Come to a decision as to what the project is and who it is for

Progress:

- Had a meeting with Neil and he gave me some new ideas for the project revolving around urban deliveries
- Read some articles on TSP with time windows and MOTSPs

Supervisor's Comments:

- Good progress so far
- Main priority now is to get a clear statement of the aim

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 21.10.19 **Last diary date: 14.10.19**

Objectives:

- Research constraints
- Get a clear view of the objectives for the project

Progress:

- Read a few articles which deal with either TSP or VRP with constraints and make note of what those constraints were.
- This week I spent a lot of time focussing on my practical work for my other modules as they will be great help when it comes to starting my courseworks in the upcoming week

Supervisor's Comments:

- Good set of notes, coming together well
- Time to start turning notes into text
- Start thinking about how to visualise results and that will help me get there

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 28.10.19 **Last diary date: 21.10.19**

Objectives:

- Start writing literature review draft.

Progress:

- Wrote ~1300 words for literature review
- Read some more literature

Supervisor's Comments:

- For next week start thinking about concrete decisions – graph hopper, data types/sets etc.
- Start to think about how to evaluate and compare results.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 04.11.19 **Last diary date: 28.10.19**

Objectives:

- Continue writing lit review draft
- Research data needed for the application

Progress:

- Lit review is around 2000 words
- Looked at GraphHopper tutorials and downloaded a REST client so I could experiment with the API

Supervisor's Comments:

- Good progress on literature review.
- I've got to the critical point in expressing the aim of the project.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 11.11.19 **Last diary date: 4.11.19**

Objectives:

- Continue writing literature review
- Research the jspirit optimisation engine

Progress:

- Finished the first draft of my lit review (~4000 words)
- Emailed Brian and Neil about setting up an interim meeting

Supervisor's Comments:

- Well done for finishing the draft of the lit review on schedule.
- Bear in mind it is a draft and a second draft will likely be in the works.
- Try to get a good idea for the aim .
- Think about what it means to combine two search method.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 18.11.19 **Last diary date: 11.11.2019**

Objectives:

- Prepare for interim meeting
- Continue to research the methodology

Progress:

- Practiced with maven and graphhopper
- Continues to investigate the methodology, made a simple plan

Supervisor's Comments:

- Nailed the aim & the question on how to combine approaches

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 26.11.19 **Last diary date: 18.11.2019**

Objectives:

- Make amendments to literature review
- Continue to research methodology

Progress:

- Went to interim review meeting.
- Further research into the tools – got an instance of GraphHopper running locally on my PC.
- Further narrowed down the aim and methodology – no VRP, no jsprit

Supervisor's Comments:

- Need to ideally finish lit review and methodology before Christmas
- Look into clustering algorithms

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 20.12.2019 **Last diary date: 05.12.2019**

Objectives:

Over the holidays:

- Finish lit review – amend sections and add conclusion
- Start building the application, start by implementing the algorithms, then adding gpraphhopper API

Progress:

- Made changes to lit review
- Read some new papers about nearest neighbour & clustering
- Got the GraphHopper application running through IntelliJ
- Most time and effort went into revising for exams the past 2 weeks

Supervisor's Comments:

- Distinguish between what graphhopper does and what it doesn't do
- Find a way to show an abstraction of the network
- Number of nodes, scaling, how does the algorithm deal with 10, 100, 1000 nodes?
- Figure out an easy testing instance
- Top priority – get project platform set up, how to test, how to run, what data?

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 24.1.2020 **Last diary date: 20.12.2019**

Objectives:

- Set up the platform
- Research what GraphHopper does and doesn't do
- Get methodology section started
- Implement nearest neighbour

Progress:

- Set up the platform, implemented evolutionary algorithm, tabu search, hill climber, simulated annealing and 2-opt algorithm
- Compiled a data set consisting of all major town/cities in Scotland (164 vertices)
- Implemented a method which produces a url that can be used to create a graph in a local instance of Graphhopper
- Updated literature review

Supervisor's Comments:

- Start thinking about evaluation in systematic way
- Gantt chart -. Start at submission date and work backwards
- Start implementation of final algorithm

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 05.12.2019 **Last diary date: 24.11.2019**

Objectives:

- Based on the interim meeting, make suggested changes to literature review.
- Find more literature.
- Continue to experiment with GraphHopper.

Progress:

- Little time was spent on the honours project as I had two coursework hand-in due for this week.

Supervisor's Comments:

- Not much work – but that was in the plan.
- Good that I've made some tweaks to the lit review.
- Keep making small progress over the next couple weeks while exams are coming up.
- Read clustering paper.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Robbie McNeil **Supervisor: Brian Davison**

Date: 31.01.2020 **Last diary date: 24.01.2020**

Objectives:

- Implement nearest neighbour algorithm
- Start methodology
- Update Gantt chart

Progress:

- Implemented nearest neighbour
- Finished adding final segments of lit review
- Updated Gantt chart

Supervisor's Comments:

- Methodology: need to know decisions (why algorithms were chosen), need to know the process
- Keep ideas of combining algorithm on the backburner while I focus on writing the methodology

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 7.02.2020 **Last diary date: 31.01.2020**

Objectives:

- Test the datasets on the algorithms
- Experiment with parameters
- Write algorithm section for methodology

Progress:

- Program won't accept large datasets
- Started experimenting with parameters and recording results
- Started methodology

Supervisor's Comments:

- Hit a technical limitation, but making good timing
- Fix post issue for next week
- Keep adding to methodology

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 18.02.2020 **Last diary date: 7.02.2020**

Objectives:

- Fix issue with sending POST requests to local GH server
- Continue writing methodology

Progress:

- Can now send POST requests to GraphHopper, but still does not return a visualisation of the tour
- Continued work on the methodology adding evaluation section and updating previous sections

Supervisor's Comments:

- Set up a meeting with Neil to query about getting graph working
- If graphhopper doesn't work start experimenting with other services – not necessarily starting over
- Still making good time nonetheless
- Continue working on the methodology

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 28.02.2020 **Last diary date: 18.02.2020**

Objectives:

- Organise a meeting with Neil
- Fix visualisation of tours

Progress:

- Had meeting with Neil, he informed me to use .KML files for visualisation, however he also said that for this type of project visualisation would not be necessary – the distance of tours and CPU time are what's important

Supervisor's Comments:

- Just comparison is enough for the project to hit the top rank
- If I wanted to nail it, I could add in other methods such as clustering or combining methods
- Still in a good position
- Find a good tool for visualising data (i.e. tabaeu.com, lucidchart)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 03.03.2020 **Last diary date: 28.02.2020**

Objectives:

- Collect results data
- Produce visualisations for data
- Continue results section

Progress:

- Finished testing and collected results for all 4 datasets
- Started using MATLAB to produce graphs

Supervisor's Comments:

- Brilliant set of intermediate results
- Concentrate on hybridization
- If the project was handed in now it would pass – making good progress

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Rob McNeil **Supervisor: Brian Davison**

Date: 10.03.2020 **Last diary date: 03.03.2020**

Objectives:

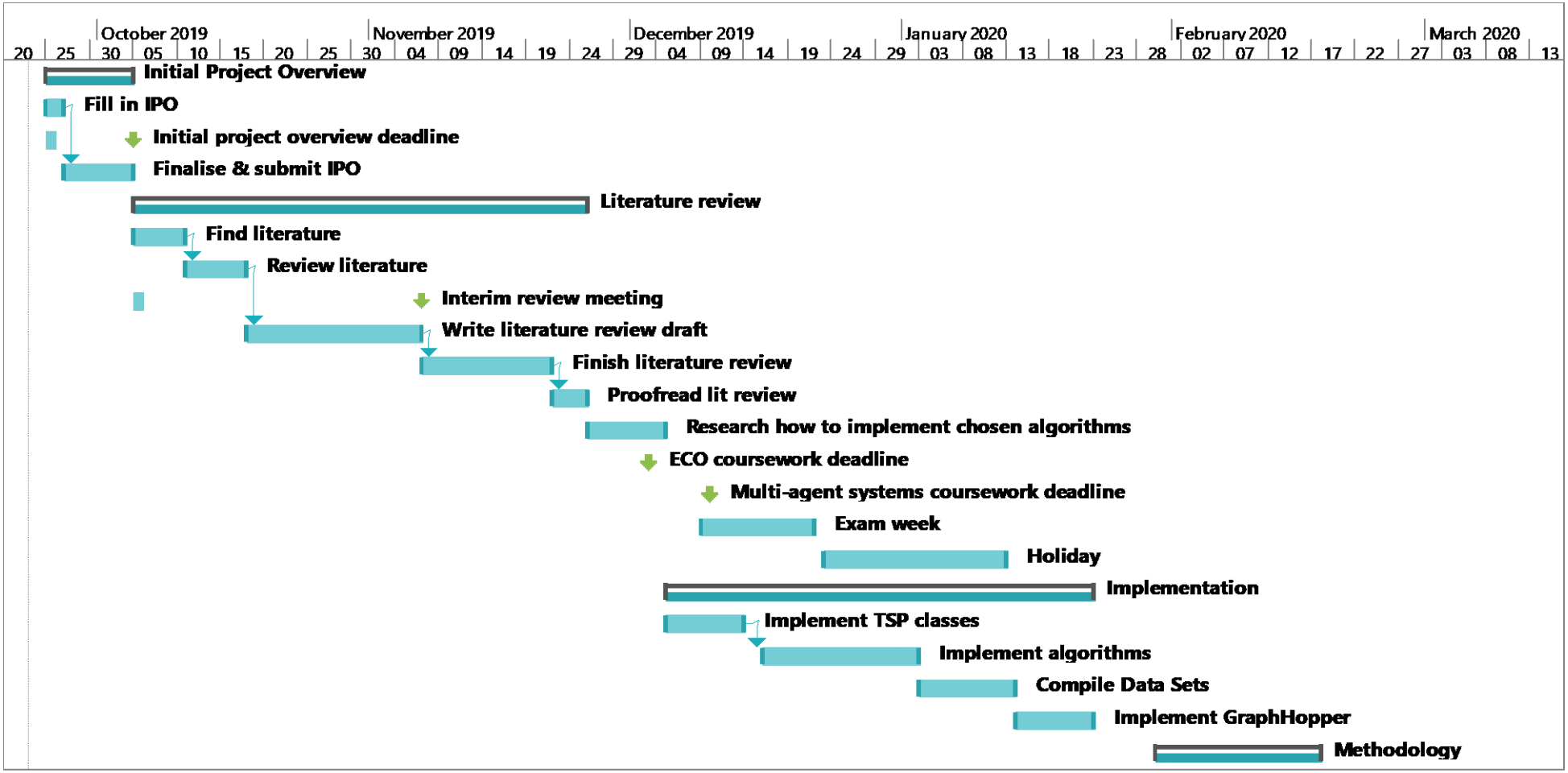
- Add all necessary test data to the dissertation (tables/graphs)
- Write up results section
- If there's time, finish up the evaluation, conclusion and introduction

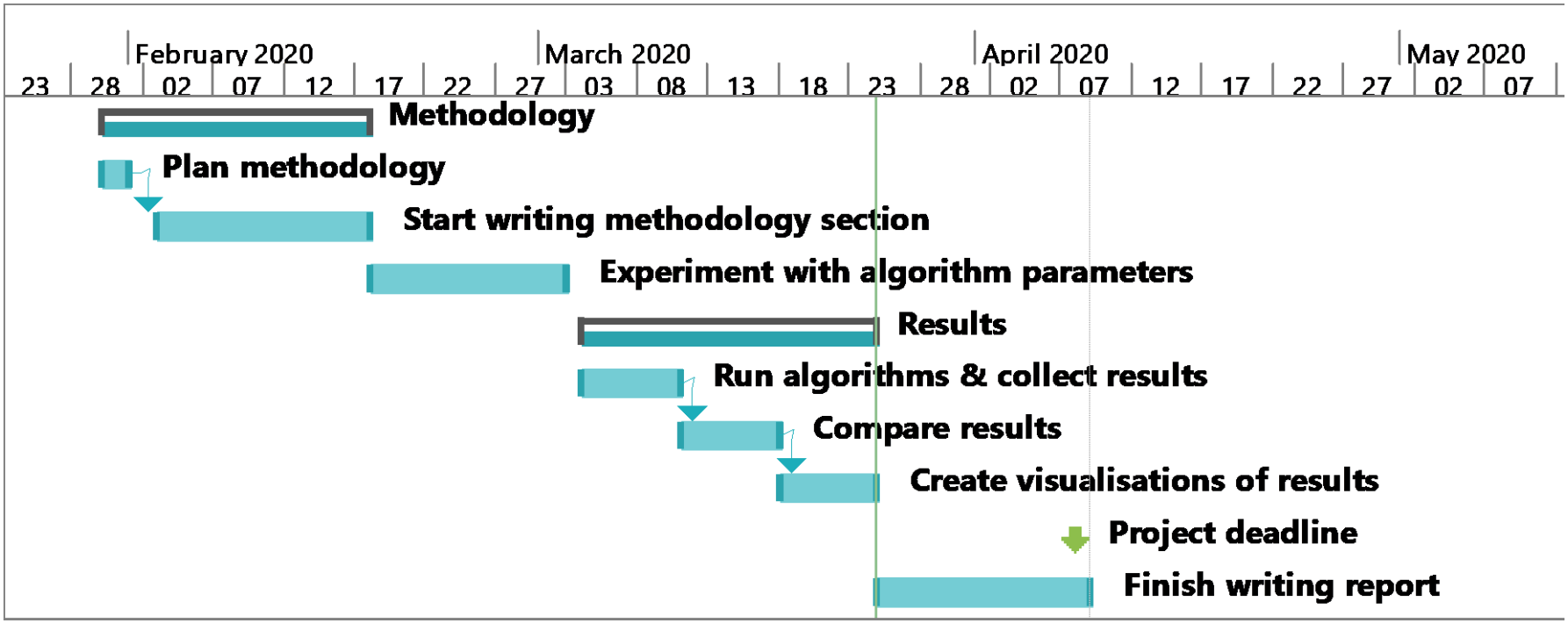
Progress:

- Results section finished, including all tables and graphs
- Started writing evaluation, need to reread through some literature in order to make some comparisons between my work and others

Supervisor's Comments:

- Concentrate on one thing at a time
- Look at the conventions of other works to see how to represent the work
- Next week we will talk about the draft


October 2019
November 2019
December 2019
January 2020
February 2020
March 2020
20 25 30 05 10 15 20 25 30 04 09 14 19 24 29 04 09 14 19 24 29 03 08 13 18 23 28 02 07 12 17 22 27 03 08 13
Initial Project Overview
Fill in IPO
Initial project overview deadline
Finalise & submit IPO
Literature review
Find literature
Review literature
Interim review meeting
Write literature review draft
Finish literature review
Proofread lit review
Research how to implement chosen algorithms
ECO coursework deadline
Multi-agent systems coursework deadline
Exam week
Holiday
Implementation
Implement TSP classes
Implement algorithms
Compile Data Sets
Implement GraphHopper
Methodology


February 2020
March 2020
April 2020
May 2020
23 28 02 07 12 17 22 27 03 08 13 18 23 28 02 07 12 17 22 27 02 07
Methodology
Plan methodology
Start writing methodology section
Experiment with algorithm parameters
Results
Run algorithms & collect results
Compare results
Create visualisations of results
Project deadline
Finish writing report

Appendix 4

Degrees of freedom	Significance level					
	20% (0.20)	10% (0.10)	5% (0.05)	2% (0.02)	1% (0.01)	0.1% (0.001)
1	3.078	6.314	12.706	31.821	63.657	636.619
2	1.886	2.920	4.303	6.965	9.925	31.598
3	1.638	2.353	3.182	4.541	5.841	12.941
4	1.533	2.132	2.776	3.747	4.604	8.610
5	1.476	2.015	2.571	3.365	4.032	6.859
6	1.440	1.943	2.447	3.143	3.707	5.959
7	1.415	1.895	2.365	2.998	3.499	5.405
8	1.397	1.860	2.306	2.896	3.355	5.041
9	1.383	1.833	2.262	2.821	3.250	4.781
10	1.372	1.812	2.228	2.764	3.169	4.587
11	1.363	1.796	2.201	2.718	3.106	4.437
12	1.356	1.782	2.179	2.681	3.055	4.318
13	1.350	1.771	2.160	2.650	3.012	4.221
14	1.345	1.761	2.145	2.624	2.977	4.140
15	1.341	1.753	2.131	2.602	2.947	4.073
16	1.337	1.746	2.120	2.583	2.921	4.015
17	1.333	1.740	2.110	2.567	2.898	3.965
18	1.330	1.734	2.101	2.552	2.878	3.922
19	1.328	1.729	2.093	2.539	2.861	3.883
20	1.325	1.725	2.086	2.528	2.845	3.850
21	1.323	1.721	2.080	2.518	2.831	3.819
22	1.321	1.717	2.074	2.508	2.819	3.792
23	1.319	1.714	2.069	2.500	2.807	3.767
24	1.318	1.711	2.064	2.492	2.797	3.745
25	1.316	1.708	2.060	2.485	2.787	3.725
26	1.315	1.706	2.056	2.479	2.779	3.707
27	1.314	1.703	2.052	2.473	2.771	3.690
28	1.313	1.701	2.048	2.467	2.763	3.674
29	1.311	1.699	2.043	2.462	2.756	3.659
30	1.310	1.697	2.042	2.457	2.750	3.646
40	1.303	1.684	2.021	2.423	2.704	3.551
60	1.296	1.671	2.000	2.390	2.660	3.460
120	1.289	1.658	1.980	2.158	2.617	3.373
∝	1.282	1.645	1.960	2.326	2.576	3.291

Appendix 5 cURL command

curl -X POST -H "Content-Type: application/json" "http://localhost:8989/route" -d @<file_name>

Appendix 6 GraphHopper forum posts

POST request to locally hosted GraphHopper instance

Open Source Routing Engine

Robbie_McNeil #1 February 12, 2020, 5:05pm

Hi there,

I'm currently trying to solve TSPs using the open source routing engine, I am able to send a GET request to **http://localhost:8989** for smaller problems, however, I want to eventually increase the size of my problems to include ~5000-10,000 vertices, which isn't possible using this method as the URL is much too long.

I've tried looking up ways of sending a POST request, with the required vertices sent in JSON format, but I've found some conflicting results, as the documentation says it is possible to send POST requests while some forum posts I have found state otherwise.

Is it possible to send a HTTP POST request to the a local instance of GraphHopper? I've made some attempts to do so but I keep getting 400 and 405 response codes. If it is possible, any additional information you have about how to do so would be greatly appreciated!

Rob.

karussell #2 February 12, 2020, 6:46pm

In the master you can use it already: **https://github.com/graphhopper/graphhopper/pull/1762**

Robbie_McNeil #3 February 13, 2020, 4:24pm

Thank you for the quick reply! I was using the 0.13 branch, I've tried switching to the master branch but when I try and run the program from intelliJ I get this error:

java.lang.IllegalArgumentException: If no graph.location is provided you need to specify an OSM file.

Any ideas where I'm going wrong?

ratrun #4 February 13, 2020, 5:55pm

You might have the issue discussed in
https://github.com/graphhopper/graphhopper/issues/1897 .

1 Like

Robbie_McNeil #5 February 14, 2020, 4:44pm

I have updated to the master branch, I am still struggling with sending the POST request though, the only example of a POST request I can find is in the RouteResourceTest file and I am unsure of whether I can replicate this method outside of the test. Are there any other examples of how to send a POST request?

karussell #6 February 14, 2020, 5:29pm

This should work:

```
curl -X POST -H "Content-Type: application/json" "http://localhost:8989/route" -d
```

1 Like

Powered by **Discourse**

https://discuss.graphhopper.com/t/post-request-to-locally-hosted-graphhopper-instance/5187?u=robbie_mcneil

Appendix 7 EA parameter tuning data

	Population size				
Run	100	200	300	400	500
1	2061314	2045281	2066742	2035826	2057549
2	2067481	2040048	2045482	2059875	2050221
3	2059926	2063352	2051678	2053451	2040952
4	2068545	2059565	2049293	2053312	2040829
5	2044342	2058606	2055834	2039056	2044731
6	2064254	2067596	2051199	2043425	2067810
7	2065271	2056958	2043421	2038594	2065206
8	2058334	2045794	2041188	2036243	2048227
9	2066770	2060722	2043869	2046207	2044126
10	2062485	2047825	2049364	2044319	2058216
Mean	**2061872.2**	**2054574.7**	**2049807**	**2045030.8**	**2051786.7**
Std. Dev	**7003.965001**	**9136.472417**	**7439.021217**	**8187.941103**	**9845.573038**

Table 17– Population size parameter tuning full results (Distance in km)

	Tournament size				
Run	10	20	30	40	50
1	2025859	2024101	2050172	2062367	2061409
2	2030365	2042037	2038111	2045294	2044748
3	2032600	2041989	2039558	2048719	2054620
4	2026918	2039982	2057499	2047432	2056148
5	2018049	2026176	2044538	2041005	2044748
6	2028230	2020332	2033759	2062292	2049029
7	2041060	2038952	2043661	2052344	2055494
8	2022780	2037633	2050985	2050184	2063336
9	2035432	2032547	2037150	2033547	2055750
10	2025884	2035518	2053097	2048302	2056649
Mean	**2028717.7**	**2033926.7**	**2044853**	**2049148.6**	**2054193.1**
Std. Dev	**6531.600775**	**7833.716389**	**7825.902575**	**8738.114368**	**6279.607126**

Table 18 - Tournament size parameter tuning full results (Distance in km)

	Mutation rate				
Run	0.001	0.005	0.01	0.05	0.1
1	2034477	2033739	2019590	2027971	2025315
2	2038311	2027510	2019750	2019407	2028879
3	2034730	2031170	2039701	2018616	2036145
4	2030758	2036274	2033559	2025415	2028938
5	2031012	2039209	2018551	2034020	2033535
6	2029825	2029323	2027442	2024727	2032574
7	2026617	2023722	2026834	2031342	2032665
8	2036464	2022379	2029061	2028204	2029742
9	2037099	2029381	2035301	2032238	2035687
10	2034739	2019506	2029937	2032172	2031945
Mean	**2033403.2**	**2029221.3**	**2027972.6**	**2027411.2**	**2031542.5**
Std. Dev	**3702.591999**	**6219.258549**	**7115.989713**	**5360.81517**	**3345.264588**

Table 19 – Mutation rate parameter tuning full results (Distance in km)

Appendix 8 SA parameter tuning data

	Starting temperature			
Run	500	5000	50000	500000
1	2073447	2088442	2105597	2087920
2	2076805	2090746	2092254	2099745
3	2072557	2083260	2084154	2095226
4	2079444	2102781	2092739	2090173
5	2080011	2088517	2083388	2090659
6	2065451	2092947	2091639	2091248
7	2073957	2086707	2088033	2101608
8	2086985	2071203	2086704	2073409
9	2070963	2090122	2089283	2076889
10	2084693	2076361	2085500	2097777
Mean	**2076431.3**	**2087108.6**	**2089929.1**	**2090465.4**
Std. Dev	**6516.425708**	**8751.227612**	**6428.030223**	**9242.686228**

Table 20 - Starting temperature parameter tuning full results (Distance in km)

	Cooling rate				
Run	0.99	0.95	0.9	0.85	0.8
1	2117457	2057231	2043464	2029258	2011470
2	2115147	2070124	2040036	2030715	2020896
3	2102348	2051002	2034143	2039005	2018855
4	2106124	2066646	2042824	2022199	2031581
5	2114858	2064709	2050097	2038386	2022530
6	2086545	2059751	2036601	2026575	2010169
7	2094012	2057211	2044578	2036339	2019147
8	2101526	2066281	2018687	2014670	2015599
9	2080360	2073045	2050917	2022892	2029851
10	2096236	2058480	2039500	2019936	2034412
Mean	**2102041.889**	**2062888.889**	**2040149.667**	**2028893.222**	**2020010.889**
Std. Dev	**12453.11325**	**6809.450998**	**9203.524857**	**8245.958973**	**8272.001692**

Table 21 - Starting temperature parameter tuning full results (Distance in km)

Appendix 9 TS parameter tuning data

Run	50	100	150	200	250
1	1582693	1570665.561	1586334	1595772	1578576
2	1595132	1583466.023	1584791	1593839	1585105
3	1574748	1580319.365	1588600	1575201	1584605
4	1580026	1588847.284	1597088	1584636	1586739
5	1587864	1586711.803	1580379	1601039	1584279
6	1581127	1590097.438	1578546	1592997	1594273
7	1603501	1593446.172	1589459	1573130	1585488
8	1582520	1574747.276	1585869	1596817	1590709
9	1587301	1578984.492	1594705	1596495	1591473
10	1580055	1593265.495	1578825	1591096	1596077
Mean	**1585496.7**	**1584055.091**	**1586459.6**	**1590102.2**	**1587732.4**
Std. Dev	**8411.264908**	**7785.211705**	**6285.449762**	**9434.149044**	**5309.715439**

Table 22 - Starting temperature parameter tuning full results (Distance in km)

	Neighbourhood size				
Run	50	100	150	200	250
1	1582693	1518007	1495026	1475150	1456061.383
2	1595132	1527919	1493975	1470207	1454047.651
3	1574748	1524601	1494187	1473720	1455682.972
4	1580026	1535062	1484928	1478336	1455482.95
5	1587864	1529908	1489614	1476842	1449826.038
6	1581127	1536164	1482893	1476340	1458453.178
7	1603501	1543960	1501061	1468927	1457864.111
8	1582520	1531218	1491194	1469618	1465537.426
9	1587301	1513749	1490795	1476560	1459648.491
10	1580055	1523612	1490686	1477987	1447905.253
Mean	**1585496.7**	**1528420**	**1491435.9**	**1474368.7**	**1456050.945**
Std. Dev	**8411.264908**	**8927.122742**	**5155.706535**	**3561.660569**	**4961.709348**

Table 23 – Neighbourhood size parameter tuning full results (Distance in km)

Appendix 10 Algorithm distance data

	Algorithm				
Run	2O	EA	HC	SA	TS
1	625.698811	620.8093669	649.9838225	575.9019947	575.9019947
2	620.541817	581.4541568	649.9838225	620.5402749	620.8467559
3	574.7788927	382.5516492	649.9838225	604.5437185	625.119363
4	625.4119167	625.119363	329.9683095	624.6433395	625.5213941
5	625.5213941	380.9689792	329.9683095	385.2762371	624.5338244
6	620.8467559	574.8645509	329.771709	379.8148962	604.5437185
7	625.4119167	626.3020024	379.8148962	575.7800918	575.7800918
8	576.1441188	631.6804672	379.8148962	406.9858422	620.0642471
9	624.3742475	396.2400132	379.8148962	650.95771	625.119363
10	605.1172855	627.2999958	625.698811	329.9683095	624.545796
Mean	**610.9053716**	**536.2756864**	**450.535497**	**508.72338**	**616.230506**
Std. Dev	**21.07895795**	**114.1692115**	**145.3024654**	**129.4264731**	**16.55014217**

Table 24 – Algorithm distance data for 10 vertex TSP (Distance in km)

	Algorithm				
Run	2O	EA	HC	SA	TS
1	1053.892304	6711.854893	2336.482977	2751.539214	2117.618923
2	1058.337442	6665.665657	3054.998429	2650.698013	2260.074787
3	1056.834693	6502.219004	3195.510715	2891.550762	2227.795065
4	1054.107361	5461.308284	3106.334623	2398.999181	2510.716078
5	997.6751883	5461.162623	3964.531693	3081.522953	2355.281899
6	1044.21716	5957.65524	3038.427028	3114.059956	2331.379917
7	1056.218153	6112.22624	3038.427028	2504.673546	2237.934027
8	1065.345034	5599.12507	3038.427028	2463.537672	1755.554848
9	1070.983418	5239.721242	3257.657046	2096.90756	1833.318473
10	1059.248033	6642.49141	3257.657046	2949.49147	2392.227521
Mean	**1051.685879**	**6035.342966**	**3128.845361**	**2690.298033**	**2202.190154**
Std. Dev	**20.25152165**	**571.1608873**	**393.6052016**	**328.0375376**	**240.2348613**

Table 25 – Algorithm distance data for 100 vertex TSP (Distance in km)

	Algorithm				
Run	2O	EA	HC	SA	TS
1	4000.267796	114857.4747	25988.1349	27378.49936	26876.25699
2	4067.753442	112217.0418	26562.66279	24200.83788	26410.51311
3	3880.396904	111937.516	24654.82275	24876.31016	24623.88905
4	4085.889582	120690.509	26580.89625	25741.32289	28392.28069
5	3857.354726	117927.0111	23886.83679	25303.30553	25418.40739
6	3960.656701	113862.4282	25367.44157	24729.91972	26325.80536
7	4129.457344	116603.6043	24743.08848	24831.41937	26973.25248
8	3781.756489	110567.0344	27264.00088	25616.98577	26755.80473
9	4029.549381	113296.7325	25252.65883	24415.47497	26119.62723
10	4137.465479	114813.1299	23148.48803	24827.72197	27743.83718
Mean	**3993.054784**	**114677.2482**	**25344.90313**	**25192.17976**	**26563.96742**
Std. Dev	**121.0349302**	**3046.025054**	**1286.895766**	**909.0181026**	**1074.480861**

Table 26 – Algorithm distance data for 1,000 vertex TSP (Distance in km)

	Algorithm				
Run	2O	EA	HC	SA	TS
1	720796.3742	1979293.839	1762509.112	1025174.913	1900533.068
2	743709.8001	1995484.066	1018054.064	1016453.602	1911890.256
3	732362.62	1975827.835	1014313.995	1008099.835	1914255.012
4	746789.936	1992644.018	1003685.4	1011551.815	1907230.229
5	724492.3335	1974841.763	997812.4378	1008481.752	1924039.348
6	706068.0994	1994103.826	996556.8691	1002817.071	1927865.571
7	712895.0813	1986422.86	1000217.882	1005012.009	1921908.587
8	713921.7269	2005087.421	1008928.342	1012158.275	1917955.994
9	723802.7942	1968203.987	856188.5707	1012052.462	1907493.48
10	719128.7044	1996553.631	776250.9112	1008221.911	1914255.012
mean	**724396.747**	**1986846.325**	**1043451.758**	**1011002.365**	**1914742.656**
std dev	**13152.67029**	**11823.41014**	**265398.1914**	**6313.216464**	**8431.599066**

Table 27 – Algorithm distance data for 10,000 vertex TSP (Distance in km)

Appendix 11 Algorithm wall clock time data

	Algorithm					
Run	2O	EA	HC	SA	TS	NN
1	0.0146792	0.0728648	0.0159645	0.0103697	0.7546511	0.0031804
2	0.0066903	0.0201172	0.0058486	0.0083349	0.7239955	0.0010496
3	0.0075458	0.0177118	0.0055438	0.0036264	0.6119204	0.0009736
4	0.0078311	0.0133979	0.0075619	0.0036931	0.5807057	0.0013493
5	0.0042332	0.0115712	0.005127	0.0031044	0.5766291	0.0013129
6	0.0047001	0.0107192	0.0053624	0.0022039	0.5879921	0.0035327
7	0.0038934	0.0140172	0.0059246	0.0020833	0.5730776	0.0011977
8	0.0038091	0.0165004	0.0045366	0.0017053	0.578288	0.0007367
9	0.0037488	0.010925	0.0048654	0.0022067	0.5831893	0.0007437
10	0.0043843	0.0120396	0.0046609	0.0017623	0.5967702	0.0009967
Mean	**0.00615153**	**0.01998643**	**0.00653957**	**0.003909**	**0.6167219**	**0.00150733**
Std. Dev	**0.00338371**	**0.01883987**	**0.00342291**	**0.00299455**	**0.06598791**	**0.00099956**

Table 28 – Algorithm wall clock time data for 10 vertex TSP (Time in s)

	Algorithm					
Run	2O	EA	HC	SA	TS	NN
1	0.0669257	0.1498796	0.0622487	0.015519	5.8009745	0.0180612
2	0.0355053	0.0509097	0.0413212	0.0085259	5.6431992	0.0092953
3	0.0381038	0.0475274	0.0369473	0.0039059	5.770711	0.0074784
4	0.0354466	0.0454253	0.0365008	0.0036426	5.6899919	0.004623
5	0.0358456	0.0445562	0.0362142	0.0033243	5.8040659	0.0053115
6	0.0315012	0.0446668	0.0356374	0.0028728	5.5874115	0.0038683
7	0.0365439	0.0540241	0.0365599	0.0033534	5.6561737	0.0034212
8	0.0306341	0.0447599	0.0353687	0.0036135	5.6253284	0.0051984
9	0.035051	0.0421416	0.0355925	0.0032334	5.7294834	0.0032654
10	0.034624	0.0450375	0.0362125	0.0030058	5.5825515	0.0040647
Mean	**0.03801812**	**0.05689281**	**0.03926032**	**0.00509966**	**5.6889891**	**0.00645874**
Std. Dev	**0.01039655**	**0.03285795**	**0.0082547**	**0.00401505**	**0.08364999**	**0.00449498**

Table 29 - Algorithm wall clock time data for 100 vertex TSP (Time in s)

	Algorithm					
Run	2O	EA	HC	SA	TS	NN
1	0.4154028	1.0742794	0.4150251	0.062843	64.2901286	0.1804758
2	0.3136337	0.8481303	0.3420969	0.044046	63.3001111	0.1275174
3	0.406088	0.8926231	0.3437255	0.0370602	64.5945792	0.134318
4	0.3442445	0.872075	0.3624835	0.0325353	64.7343663	0.1200252
5	0.4288159	0.8552729	0.3668552	0.0201919	63.7913403	0.1194834
6	0.3132216	0.8440676	0.3653894	0.0140023	64.2153922	0.1122193
7	0.3313224	0.8499132	0.3665321	0.0161942	63.5954036	0.106558
8	0.3071878	0.8445289	0.3330223	0.012857	63.6872572	0.1053523
9	0.2962017	0.9594281	0.3327126	0.0149937	63.7009755	0.1086614
10	0.2804944	0.9575188	0.3301216	0.013313	64.3215877	0.1188177
Mean	**0.34366128**	**0.89978373**	**0.35579642**	**0.02680366**	**64.02311417**	**0.12334285**
Std. Dev	**0.05359192**	**0.07561035**	**0.02566111**	**0.01689942**	**0.4720838**	**0.0220935**

Table 30 - Algorithm wall clock time data for 1,000 vertex TSP (Time in s)

	Algorithm					
Run	2O	EA	HC	SA	TS	NN
1	3.7912408	66.645884	3.8351905	0.9087401	1566.270035	10.2872209
2	3.5742609	63.6178605	3.4785763	0.739031	1559.747996	9.9079236
3	3.5384004	63.0172472	3.5047422	0.6690689	1557.466373	10.0002962
4	3.5751722	61.8500727	3.4872872	0.6793715	1563.424847	10.0982671
5	3.5549643	62.7279227	3.4163946	0.6639245	1558.401591	10.1535971
6	3.4760177	63.4078441	3.6820776	0.6837166	1557.730308	9.9451328
7	3.6502169	63.9999559	3.5297495	0.66423	1556.407696	10.4068071
8	3.6464295	62.8125683	3.6937884	0.6902044	1556.733383	10.3340269
9	3.7839115	62.7498538	3.8376329	0.6393836	1557.345057	10.3092787
10	3.6782335	63.5884184	3.8181094	0.7093876	1558.65845	10.4572649
Mean	**3.62688477**	**63.44176276**	**3.62835486**	**0.70470582**	**1559.218574**	**10.18998153**
Std. Dev	**0.10351583**	**1.27930204**	**0.16398398**	**0.07671099**	**3.18938764**	**0.1966705**

Table 31 - Algorithm wall clock time data for 10,000 vertex TSP (Time in s)