Investigating multiple algorithmic solutions to the multi-objective Capacitated Vehicle Routing Problem with Time Windows

Taylor Aidan Courtney

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Software Engineering.

School of Computing

April 2022

Authorship Declaration

I, Taylor Aidan Courtney, 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:

Date: 03/04/2022

Matriculation no: 40398643

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.

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

This dissertation aims to research the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) and algorithms written to solve it, then implement those algorithms and experiment with them. The primary objective of this project is to analyse the efficiency of algorithms written to solve the problem. The CVRPTW is NP-Hard: a problem with a search space too large to search and likely has no single best solution; instead, there are often multiple high-quality solutions. It is thought to be impossible to solve a CVRPTW (ensure the best solution is found), but it is possible to produce multiple high-quality solutions.

Such solutions can be generated using AI. This dissertation explores a subsection of AI called Genetic Algorithms (GAs), which simulate the process of evolution. A GA takes two parent solutions and joins them together (known as Crossover), or one solution to Mutate, to form a better solution, hopefully. The Fitness (quality) of a solution is determined by how well it satisfies an objective function; an example objective function of the CVRPTW is to reduce the total distance and the number of vehicles required to deliver to every customer. Research has proven that if two already-fit parent solutions are merged, there is a good chance that they will produce a fitter child. The algorithm records which solutions are the best by using natural selection: the fittest solutions survive to be used for future crossovers.

This dissertation experiments with three GAs, two developed by other researchers and a GA written by the researcher of this dissertation. The Feasibility Intensive Genetic Algorithm (FIGA) was written to investigate theories deduced from the CVRPTW's scope and the other two algorithms. A comparison of the three algorithms is made during evaluation. This stage proved FIGA's implementation more effective at producing high-quality solutions than the other two algorithms.

Contents

List of Tables

List of Figures

Acknowledgements

Special thanks are due to Dr Ben Paechter and Dr Stefano Cherubin of Edinburgh Napier University. Ben for his project supervision and Stefano for his interim project advice. Ben’s guidance and support were invaluable to the project's timely research, experimentation, and eventual completion.

Many thanks are also due to Raúl Baños for his personal advice regarding the CVRPTW and questions I had regarding his algorithm, MMOEASA.

Additional thanks to the following researchers, significant contributions were made to the experimentation stage of this dissertation: Raúl Baños (as stated), Julio Ortega, Consolación Gil, Antonio L. Márquez, and Francisco de Toroc for the work of MMOEASA, and Beatrice Ombuki, Brian J. Ross, and Franklin Hanshar for the work of Ombuki’s Algorithm.

1 Introduction

1.1 Background

The original Vehicle Routing Problem (VRP) was first proposed in 1959 and has been researched ever since. In the context of the VRP, the aim is, given multiple customers located at different positions, to find the fastest route for multiple vehicles to service each customer, which minimises the delivery cost. There are multiple editions of the VRP, each with extensive research demonstrating many algorithms that produce solutions. Of the problem flavours that exist, this dissertation will analyse the efficiency of algorithms written to solve the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW).

The CVRPTW was chosen as it is, most recently, widely applicable to real-world delivery situations. Nowadays, time windows are essential as customers on a delivery route like to know when delivery of their goods will arrive and trust that they will be delivered in the specified time window. With the substantial increase in the usage of online markets, the number of customers on delivery routes has also grown exponentially. Finding the most efficient route that services all customers using multiple vehicles, each with capacity limits, is a complex problem that becomes more difficult with more customers and vehicles. With many customers and vehicles comes a search space of possible solutions so large that it will never be feasible to search all of them. Adding cargo and time window constraints to the solutions makes searching for feasible solutions even more difficult.

A possible method of solving the problem is to create an algorithm that can reduce the search space to something much more feasible. Reducing the search space is when, of all possible solutions to the problem, an algorithm avoids working with poor solutions it finds. As a result, the algorithm can make experiments with only high quality, feasible solutions and use them in experiments to try and make them better.

1.2 Aims and Objectives

The aim of a solution to the CVRPTW is dependent on the objective that a user wishes to solve. Therefore, the primary aim of this dissertation is to find effective algorithms and heuristics that can produce high-quality solutions to the objectives in question.

This dissertation will attempt to analyse what makes an algorithm effective at reducing the cost of satisfying the CVRPTW. The cost of a combination of routes will be known as the objective function and is affected by two aspects. The distance of a single vehicle's route impacts the cost via the total distance and number of vehicles taken to complete it. The number of vehicles affects cost as each requires fuel and the driver's labour cost. Solutions cannot focus on reducing drivers to lower the cost as doing so will result in longer routes in which remaining drivers will have to travel. Solutions cannot enforce too many drivers to minimise route lengths as using too many drivers also becomes expensive due to labour costs. Therefore, the most efficient algorithm will develop a method of solving the problem that finds the best compromise.

This dissertation will investigate advancements made recently to the CVRPTW. Researchers have produced many methods devoted to solving CVRPTW. However, the investigator would like to examine the results of more recent approaches to understand how well recent advancements perform and how they compare. A review of literature existing that considers the theory behind the CVRPTW and algorithms that tackle the problem is made. To interpret why the problem must be regarded as an optimisation problem during experimentation, being familiar with the extensive scope is essential to understanding the optimisation process.

This dissertation also aims to distinguish methods of solving the problem by their functionality and solution quality. Algorithms with different operators and heuristics are required to do this. The algorithm that can consistently produce solutions with a lower total cost required to fulfil its best solution will be considered more efficient. To achieve this aim, this dissertation will:

1. Review scientific information that defines the CVRPTW and the requirements for writing an algorithm to find a solution.
2. Identify and select recent, distinct algorithms that explore the CVRPTW based on the effectiveness of any specialised heuristics or operators they use.
3. Develop a practical application containing the chosen algorithms.
4. Gather the results of the algorithms' performance and evaluate findings.

1.3 Scope and Limitations

With the scale of the CVRPTW, the primary scope is to understand and optimise the extensive and complex search space. Algorithms should efficiently reduce the search space to arrive at optimal solutions quickly while respecting the problem's constraints. This challenge is problematic, particularly in the CVRPTW, as any small changes in the order in which a vehicle visits destinations can instantly violate time window constraints and make the solution infeasible. Although, a standard VRP, without time window constraints, is also problematic during evolutionary computation as modifying an optimal solution can quickly make it worse; if the modification is a poor swap in the order of destinations due to their distances apart, for example.

Besides the problem itself, the primary external constraint on the project is time. Six months is allocated to complete the project, but learning the scope of the over-60-year-old CVRPTW, algorithms that can solve it, implement and test them, and then evaluate their performance is demanding.

1.4 Dissertation Structure

Besides the introduction and conclusions, this dissertation covers the following chapters:

- Chapter two reviews previous literature and experimentation surrounding the CVRPTW. The research section highlights the problem's complexity and some compelling solutions.
- Chapter three justifies this dissertation’s practical approach taken to solve the CVRPTW. All aspects of the approach taken are documented to highlight how the algorithms function and justify their implementation.
- Chapter four defines a custom algorithm written to solve the CVRPTW that attempts to solve detailed theories related to the CVRPTW and the other algorithms implemented.
- Chapter five evaluates the implementation’s results justified in chapters three and four. The results will extensively review the solutions produced to the problem by each algorithm present in the implementation.

2 Literature Review

2.1 Capacitated Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem (VRP), initially The Truck Dispatching Problem (Dantzig & Ramser, 1959), entails the problem of finding the optimal set of routes for several delivery vehicles to deliver to several customers. In a VRP, multiple variables can determine a solution's validity. The most popular variables contribute to the total cost required to carry out the series of deliveries in the solution found; the total distance and number of vehicles (Figliozzi, 2010). It is essential to identify an objective that the algorithm aims to satisfy as a VRP can consist of multiple variables, and an algorithm requires a result to aim for (Jozefowiez et al., 2008).

Developing solutions to any VRP requires (an) aim(s) known as the objective function. The objective function considers variables that an ideal solution has minimised/maximised. It can be said that the VRP is inherently a multi-objective problem (Ombuki et al., 2006) as reducing vehicles or another metric is often as desirable as reducing distance. However, there is no constant objective function that algorithms are required to solve, only objectives that users would like to prioritise.

The Capacitated VRP with Time Windows (CVRPTW) aims to ensure that each customer's delivery is completed within their particular time window. At the same time, the physical capacity of each vehicle is not violated (Solomon & Desrosiers, 1988). The problem is complex because travelling between destinations and serving customers their goods impacts the ability to satisfy the time windows. Also, if a vehicle arrives at a customer before their time window opens, the vehicle must kill time waiting for the time window to become available (Desaulniers et al., 2014).

A VRP solution assigns each vehicle at a depot's disposal a sequence of customers to deliver to; this makes the VRP a combinatorial problem (Toth & Vigo, 2001). Combinatorial problems can be infamous for their high complexity, one such high complexity problem being the VRP. Their complexity is due to the number of possible combinations that may be optimal solutions. For example, in a VRP with 100 customers and a depot with 20 vehicles, there are $100^{100} \times 20^{20}$ (1×10^{226}) possible solutions to this problem. To put into perspective why this problem instance is so

complex: if we had a computer powerful enough to examine one million combinations per second, it would take 3.32×10^{212} years to examine each combination via an exhaustive (brute-force) search. This level of complexity calls for more efficient methods of solving combinatorial problems, which can be done via the process known as combinatorial optimisation (Bjorndal et al., 1995).

Combinatorial optimisation intends to refine searches for possible solutions to combinatorial problems. Searching can be optimised by employing heuristics, artificial intelligence (AI), or AI-heuristic hybrids with efficient generations of solutions (Bjorndal et al., 1995). However, some algorithms may not return the most optimal solution. It is widely believed that some combinatorial problems do not have a single, most optimal solution and that they, instead, have multiple. The VRP is classified as an NP problem; the CVRPTW, in particular, is recognised as NP-hard (Solomon & Desrosiers, 1988).

2.2 NP-Hardness

An NP (non-deterministic polynomial-time) problem is a problem where (it is assumed that) it cannot be solved in polynomial time, but an answer can be verified in polynomial time (S. A. Cook, 1971). The difference between a problem being solved and having a verified solution is whether the answer found is definitively the best for the problem. If the answer found is the best, then the problem is solved. If it is not or cannot be confirmed as the best, the answer is a verified solution (Garey et al., 1974). A solution cannot be confirmed as the best when the number of solution permutations to the problem is too large to search.

Polynomial-time is an expression of time calculated via the operational performance of an algorithm and the quantity of the data inputted into the algorithm. Some problems can be solved in polynomial-time; these problems are classified as P (S. A. Cook, 1971). For example, a Linear Search algorithm solves the problem of finding the desired value in a dataset. A Linear Search searches through the dataset of size n one value at a time. Thus, the polynomial-time of the Linear Search algorithm is $O(n)$ (in the worst case, the search requires n operations), which makes the search problem P. Therefore, Linear Search problems are polynomial-time problems, where an algorithm exists (the Linear Search) that can undoubtedly classify the best solution in polynomial-time

On the other hand, deterministic polynomial-time algorithms are problems where an algorithm exists that can determine the optimal solution via calculations. Although, optimal solutions cannot be guaranteed due to the problem's search space complexity. Problems of this class are still NP and are given the class NP-Complete (Garey et al., 1974). Considering the earlier example of a VRP with 100 customers and 20 vehicles, if a deterministic algorithm is found for the VRP, it is still impossible to declare the problem solved and classify it as P because when a solution has been obtained, one solution found from the $100^{100} \times 20^{20}$ different combinations cannot feasibly be compared to every other combination. Because there is no method of comparing a solution to all others, there is no way to ensure a solution is the best one. Therefore, the problem cannot be given a polynomial-time complexity (Young, 1983). However long the algorithm took to get a solution, the solution is merely a calculated combination of participants, based upon each participant's effect on the problem (Cohen, 1979). Alternatively, deterministic algorithms calculate what is likely to be the best solution and can do so in polynomial-time; thus, verifying the solution, in polynomial-time, as one that's possible.

Non-deterministic polynomial-time classifies problems of opposite qualities that are thought not to have an algorithm that can determine the best solution in polynomial-time (S. A. Cook, 1971). Solutions can be derived from non-deterministic algorithms as they, like deterministic algorithms, make calculations based on participants' effect on the problem. However, non-deterministic algorithms are different as they use some degree of probability alongside logic to produce solutions (Floyd, 1967). Such algorithms are applied to the VRP as solutions depend entirely on each destination's (participant's) location and their distances from each other. However, no deterministic algorithm exists for the CVRPTW as it is too complex and constrained (Kumar & Panneerselvam, 2012). Therefore, non-deterministic solutions are employed. Non-deterministic algorithms are stochastic, meaning they will likely give different results after every execution due to probability. Thus, they may determine the optimal solution in polynomial-time but cannot be classed as deterministic due to probability; the algorithm may have had a lucky generation that led it to the optimal solution.

As the name suggests, the hardness of a problem relates to how hard it is. A problem is NP-Hard, not just NP, if it is harder to solve than the hardest NP problems (S. A.

Cook, 1971). One NP problem can be classified as a hard problem if a deterministic algorithm for another NP problem cannot solve the former problem (Knuth, 1974). Consider two NP problems, A and B. Problem A, with a deterministic algorithm, has been recognised that its algorithm can be used to solve problem B. This is known as Reducing; problem B can be reduced to problem A as A's deterministic algorithm can solve B (S. Cook, 2000). However, when trying to use an algorithm that can solve problem B to try and solve problem A, and it has been recognised that B's algorithm cannot solve A (meaning A is not reduceable to B), A becomes a harder problem than B. Therefore, the CVRPTW is an NP-Hard problem as it cannot be reduced to any other problem, but other NP problems can be reduced to it. The Travelling Salesman Problem (TSP) is one problem; the TSP generalises the CVRPTW (Dantzig & Ramser, 1959) as the TSP entails the same problem, but only with one vehicle and no capacity or time window constraints.

Figure 1 visualises the groups of problems contained under the assumptions of P versus NP. P versus NP is the problem that asks whether every problem in NP can exist in P. If one NP problem can be reduced to P, this would mean that every NP problem has a polynomial-time algorithm (S. Cook, 2000). When P ≠ NP:

- The P problem group contains problems that are solvable in polynomial time.
- NP(-Intermediate) problems are not solvable in polynomial time and do not have a deterministic algorithm that can solve it (Ladner, 1975).
- NP-Complete problems were considered NP problems but were reduced to NP-Complete because the optimal solution can be mathematically determined in polynomial time.
- NP-Hard problems are not solvable in polynomial time, and their complexity makes it difficult to verify the most optimal solution in polynomial time. There is likely also many "best" solutions.

When P = NP:

- P, NP, and NP-Complete problems can be solved in polynomial time without failing.
- NP-Hard remains the outlier as problems likely have multiple optimal solutions and can be given multiple objective functions.

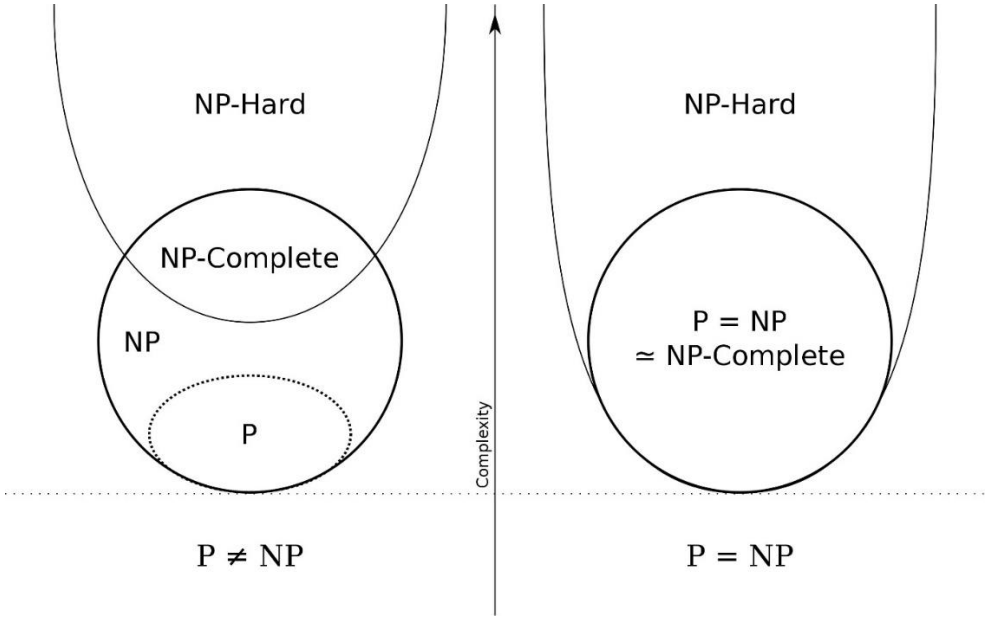


Figure 1: Euler diagram for P, NP, NP-complete, and NP-hard set of problems. The left side is valid under the assumption that P≠NP, while the right side is valid under the assumption that P=NP

To solve the CVRPTW, methods similar to those used to solve other NP problems can be used. Solutions to NP problems typically rely on AI or heuristics to solve them (Dempster, 1982). Modern approaches even combine the two methods, creating a hybrid heuristic algorithm. To arrive at a solution, such methods efficiently solve the CVRPTW by calculations and approximations. AI and heuristics are effective as, in terms of the VRP, they reduce the problem's search space by making calculations to determine if it would be optimal to deliver to one customer instead of another from a vehicle's current location. As solutions can only be derived from customers' effect on the problem instance, by determining one effect on a solution as inefficient, any subsequent combinations containing said effect can be discarded with one calculation as no quality solutions would contain the harmful effect. This is the premise of AI: making logical decisions to enhance the response to a problem (PK, 1984).

2.3 Foundations of Finding Solutions

AI intends to complete tasks that demand human intelligence (PK, 1984). It has also been applied in many problem-solving situations. Many methods of finding solutions based on AI and heuristics have been developed, all with different, highly optimised criteria to find a solution. Russell & Norvig (2002) highlight purposes AI has been used to help us during large, arduous tasks. One task they emphasise is the Mars Rover

and how it utilises AI to maintain itself, such as pathfinding so it does not get stuck and cannot move and trying to diagnose problems and repair itself if there is a fault. Russell & Norvig (2002) describe a practical use of AI, but much theory is applied in accomplishing these essential tasks. For example, a computer needs to be provided with methods that help it understand whether or not a path it could take is too treacherous as a computer has no perception of danger. Computers, like us, use the information available to them to make a decision, which is done by analysing information that their sensors collect. Again, like us, AI's decisions, based on the information they have at hand, are not always perfect. However, decision making can be improved by helping them know what to look for.

2.3.1 Evolutionary Algorithms

There are multiple styles of AI that work out solutions in different approaches. The most promising AI variants for solving the CVRPTW are Evolutionary Algorithms (EA) (Holland, 1992). EAs are algorithms that inhibit traits of systems found in evolution/biology. They are given their name as their procedures are similar to evolution, such as reproduction/recombination, mutation, and natural selection. For an algorithm to aim for the best solution, it requires an objective that determines its suitability to the problem compared to any other solution. A solution will be more suitable if it adheres to an objective function better than another solution.

Genetic Algorithms (GA) (Holland, 1992) are known EAs as they are effective and inherit all of the aforementioned evolutionary properties. GAs begin by generating a set, known as a population or generation, of solutions to the problem and then test the generated solutions' fitness and select the two best. With the selected best solutions of the generated population, which are more than likely bad solutions at the initialisation stage, a GA will perform crossovers (also known as reproductions or recombinations) or mutations of them to create the next population from them. Reproduction/recombination should combine two parents to make one better child solution. Mutation should modify the values (chromosomes – genetic material that make the characteristics of a creature, or in this case, a solution) of a solution in some way to make it better. The algorithm will loop this process until a termination condition is met or when the production of new optimal solutions is plateauing. When performing mutations and crossovers, the processes may destroy the most optimal solution to the

problem before the algorithm knows it has found it. There is a method to prevent this: Elitism (Chang Wook Ahn & Ramakrishna, 2003) means the algorithm will carry over the best solutions from the current population to the next population. As a result, the next population will consist of the current best solutions and crossovers/mutations of them.

There have been alterations to GAs since their proposition that improve their efficiency. One such is performing crossovers on multiple parents (multi-point) instead of with one pair of parents (single-point) (Eiben et al., 1994). The increase in crossovers of the algorithm's current best solutions has increased solutions' suitability, making them more fit than other already-fit solutions to the problems' objective function. Another improvement is known as search space reduction (S. Chen & Smith, 1999), which aims to reduce the range of solutions that will be searched. Search space reduction works by simulating a crossover of two potentially fit, temporary solutions before they are given a fitness. If the simulated crossover child is fit and similar to its parents, it represents that its parents are also likely to be fit.

2.3.2 Heuristics

Heuristics are broader as many heuristics have been developed to solve problems, even to help solve the CVRPTW alone. A heuristic is a mathematical function for making decisions (Pearl, 1984); they rank the options that can be chosen in the current state with the information available to determine which move will lead to the next best state. Heuristics are not a trait available only to computers: they are a quality inspired by the philosophy of how the human mind constructs ideas based on the information available to a person's perceptions at any given moment (Tversky & Kahneman, 1973). Computers exhibit this same trait and can examine many possible combinations of information in a matter of seconds, the difference being that a computer cannot improvise and make better decisions than it knows how to make. A heuristic of a computer is programmable and can be improved to match the given situation.

An example of a computational heuristic, and probably the most simple heuristic, is the Greedy Heuristic. Namely greedy because it is a heuristic programmed to always go with the most optimal-appearing option upon first inspection of the options available (Bang-Jensen et al., 2004). It is a heuristic as its ability to make decisions based on

the information given is used to approximate a solution. However, it is very likely to be inefficient as simply going with the optimal appearing decision at every stage, without doing any calculations to ensure it is optimal, yields pretty poor results. In the CVRPTW, a greedy heuristic may generate routes based on customers' shortest distance from one another.

Heuristics perform either a Local or Global Search; methods of searching for some form of a solution from a given state (Price et al., 2006). A Local Search strategy would take a single point in a current solution and determine a small change that can be made to improve the solution. On the other hand, a Global Search searches an entire problem instance's dataset and calculates the most efficient solution (Schonlau et al., 1998). A Local Search could make an incomplete solution complete, but not necessarily optimal, by analysing information at one point to choose which point to move to next (Hoos & Stützle, 2005). Consider the Greedy Heuristic as an example of a Local Search, making calculated decisions at each stage. Both search methods have their drawbacks: a Local Search can never guarantee that it has found the best solution, but it will be able to search for a solution when it is not feasible to evaluate every solution, as for the VRP. A Global Search is the opposite; it can guarantee that it has found the best solution, but only by searching every solution in the global problem scope, which is not feasible for the VRP.

2.3.3 Other Methods

Some algorithms do not employ the properties of evolution. Instead, they utilise strategies found in mathematics or other areas of biology.

Ant Colony Optimisation (ACO) (Dorigo et al., 1991) is another biologically-inspired method of finding solutions to combinatorial problems. More specifically, they are used to solve problems that can be reduced to finding a path; this is due to the functionality of ACO. The biological inspiration is that real ants work together to find the optimal path when finding goods, transporting them, and returning unharmed. Ants place pheromones that guide other ants down the more efficient path. Multi-agent systems can simulate the behaviour of ants; in AI, an agent is anything that acts with intelligence, which is generally taking information from the environment it is in and putting it to use. A multi-agent system can simulate pheromones by recording optimal paths, and artificial ants (agents) can follow those recordings and find better paths.

Another method is Branch-and-Cut algorithms (Crowder et al., 1983), which are mathematics-based and use a combination of Branch-and-Bound algorithms and Cutting Planes. Branch-and-Bound algorithms follow the structure of a tree by placing the collection of all data in the problem's dataset at the root, then branching out into smaller, more specific datasets that reach closer to a solution. Like trees, branches may split into branches, leading to other solutions derived from their parent branch. Branches of solutions that are not better than the best solution discovered so far by the algorithm are discarded. Cutting Planes are a method of finding solutions containing only integers to mixed-integer linear problems. A problem is mixed-integer if it is unknown whether all of the problem variables are integers or not. Suppose variables in the most recently found optimal solution are not all integers. In that case, Linear Programming theory (Dantzig, 1951) dictates that a cut can be applied to the mixed-integer values in the current set of variables to guarantee the algorithm finds the set of integer-only values of the most optimal solution. A cut acquires two numbers' inequality (difference) and is either added to one or taken away from the other.

2.4 Evaluation Strategies

As previously discussed, the fitness functions of GAs determine the suitability of solutions, and the objective function determines solutions' fitness that the user is aiming to solve. As a VRP example, if the objective is to reduce the cost, the function is the mathematical equation that may use the total distance and the number of vehicles in a solution to determine how well the objective has been fulfilled by a solution (J. Chen & Chen, 2008). Non-deterministic algorithms use ranking systems that compare solutions based on the desired objectives to find the solution with the most optimal objective function. However, in a multi-objective problem, ranking systems cannot compare solutions' multiple objectives as each objective represents a different characteristic of a solution and can have largely differing values. A multi-objective method of solving a problem aims to compare solutions using more than one objective in the objective function (Jozefowiez et al., 2008).

Pareto Efficiency (Goldberg, 1989) is commonly used when solving the VRP and many other combinatorial problems. Pareto is a ranking system utilised in optimisation that uses the multiple variables in the objective function to compare solutions to each other. In other words, instead of comparing solutions by the cost of their routes, they are

compared using the variables that contribute to their cost. Pareto systems decide which solution is best by applying a technique named Pareto-dominance. Pareto-dominance determines whether or not one solution dominates another. For one solution to dominate a second solution, at least one of its objective variables must be better than the same variable from the second solution. However, none of the first solution's other objective variables can be worse than the second solution's for the first solution to dominate it. If one is better, but the other is worse, neither solution dominates one another (Varian, 1976). For this reason, algorithms for NP-Hard problems list every non-dominated solution after termination, known as a non-dominated set (Zelany, 1974). Considering the VRP objective function of reducing the cost by minimising distance and vehicles, for one set of routes to dominate another, one solution's total distance, for example, can be shorter than another solution's total distance. Although, the number of vehicles used must be at least the same or less than the other solution's number. The most optimal solution found by this evaluation strategy is known as the Pareto-optimal solution. The Pareto Efficiency ranking system is one of the most well-suited when using a method of solving the VRP that considers the problem multi-objective.

Another multi-objective ranking system is the Weighted Sum (von Neumann & Morgenstern, 2007). The Weighted Sum calculates the final sum of the variables contributing to an objective, and the sum for each solution helps determine the best. However, a GA cannot calculate a multi-objective function because the different objectives cannot be added together as they represent different objectives (Marler & Arora, 2010). Selection needs to take each of the variables that determine solutions' suitability to the objective function and add them to the "weight" of a solution. So, for example, the total distance and number of vehicles in a VRP solution cannot be added together as the total distance can be a number in the thousands, and the number of vehicles will likely only ever be as high as double digits. The weight is a single value that determines suitability. The Weighted Sum method can only be employed in single-objective problems. Coming back to the VRP, the weight of solutions in a single-objective problem could be each vehicle's total (summed) distance.

2.5 Solution Methodologies

GAs have made significant contributions to the VRP following Thangiah et al. (1991). Since then, GAs have been modified in different ways, with each variation using (multiple) heuristics to improve the solutions within each population. These algorithms are known as Hybrids; they combine the features of heuristics and EAs. More specifically, GAs use a mix of crossovers, mutations, selections, and heuristic decisions simultaneously to improve the quality of solutions. Recent Hybrid algorithms developed specifically to solve the CVRPTW aim to utilise multiple special operators (such as Pareto Efficiency) and heuristics in the same algorithm. The aim is to compensate for the disadvantages of one operator by using another alongside it (Baños et al., 2013). This dissertation investigates two algorithms and their specialisations to generate solutions to the CVRPTW.

As both algorithms are GAs, they both use crossover and mutation operators. Another feature they both have is a Selection Tournament, a common GA optimisation technique. Selection Tournaments search the population, in its active state, for the best-known solution so far. Once the best solution has been found, the selection tournament returns it to the GA. The GA uses this solution as the second parent when performing crossover operations (Ombuki et al., 2006). A selection tournament aims to improve the quality of GAs' crossover: to merge the fittest solutions with other solutions, potentially improving crossover quality. Merging solutions with the most optimal solution will likely achieve the highest results during the crossover operation.

2.5.1 MMOEASA: A Multi-Start, Multi-Objective Evolutionary Algorithm with Simulated Annealing

MMOEASA (Baños et al., 2013) is a hybrid GA that utilises multiple functions to improve its solutions. MMOEASA implements a Pareto-based multi-objective approach to finding solutions. It employs the Simulated Annealing heuristic to improve searches and solutions' acceptance, a multi-objective Metropolis function – a Pareto-based acceptance criterion (Metropolis et al., 1953), and a multi-start local search method. The heuristics used by MMOEASA have been widely researched and shown to be very efficient at optimising combinatorial solutions, but the crossover implemented is not specific to optimising CVRPTW solutions.

2.5.1.1 Simulated Annealing

Simulated Annealing (SA) (Kirkpatrick et al., 1983) is derived from the Physics process known as Annealing, which means exposing a metal material to a high temperature and then cooling it over time. Annealing aims to make metals weaker and easier to reshape. Simulated Annealing in the MMOEASA algorithm aids the Metropolis Function in deciding whether to record generated solutions based on their parent's current temperature (Baños et al., 2013). The Metropolis Function uses probability to decide whether to record the new child solution if it does not dominate its parent. Probability is based on whether the parent's temperature is high (in other words, making it easier to change; reshape), and the ratio of the difference between the parent's and child's fitness is not high; based on their variables in the objective function. The lower the difference in fitness and the higher a parent's temperature, the higher the probability is of a child being recorded.

2.5.1.2 Multi-Objective Metropolis Function

The original Metropolis Function (Metropolis et al., 1953) contains an acceptance criterion that only considers one objective function and the Simulated Annealing temperature of a solution. Again, child solutions are accepted if they dominate their parent or are accepted based on probability if their parent's temperature is high. However, the standard Metropolis function cannot consider a multi-objective function as parents and child solutions need to be compared using every objective in the objective function.

A multi-objective comparison cannot be made by finding the difference between the multiple objectives of a parent and its child, as each value represents a different characteristic of a solution. For the same reason that a Weighted Sum cannot be used, there is no way to convert two or more objectives into a single value to allow a more straightforward comparison of solutions. For example, if there are two objectives and two solutions: solution one's objective one is high and objective two is low, and solution two's objective one is low, and two is high, there is no way to determine which is better. Therefore, the Multi-Objective Metropolis Function (Baños et al., 2013) calculates the dispersion between the objectives by finding the difference between the parent's and child's fitness and dividing the differences by the problem instance's Pareto-optimal (most optimal multi-objective) Hypervolume. Hypervolume is the name given to the

group of values from a multi-objective objective function (Zitzler & Thiele, 1999). MMOEASA's acceptance criterion (MO Metropolis) utilises an approximated Hypervolume of the Pareto-optimal solution to the problem instance. Hypervolumes in multi-objective acceptance criteria have proven to be sensitive to minuscule differences in two multi-objective solutions' fitness (Zitzler et al., 2007).

2.5.1.3 Multi-Start

Multi-Start (Bräysy et al., 2004) is a local search method used in MMOEASA that allows the algorithm to restart the Simulated Annealing process once the temperature of every solution in the current population has fallen to the minimum temperature. As long as the algorithm's termination condition (some iterations, time limit, or some genetic generations) has not been reached, Multi-Start will cause the temperatures of every solution in the population to reset to the maximum temperature. Resetting the temperature of each solution to their initial max temperature will once again allow more children of said solutions to be recorded; how it was prior to the solutions in the population being cooled to the minimum temperature.

2.5.2 Ombuki's Algorithm

Ombuki's Algorithm (Ombuki et al., 2006), as per the author's name, is another multi-objective GA that attempts to solve the CVRPTW based on the total distance and number of vehicles in a solution. Therefore, the algorithm also utilises a Pareto-based acceptance criterion. Ombuki's Algorithm is not a hybrid algorithm and, instead, focuses on crossover and mutation operators to generate its solutions. The crossover used is contrary to the research behind MMOEASA: the crossover is explicitly designed for the CVRPTW, but as it is written by Ombuki et al. (2006), there is not much research exploring their crossover.

2.5.2.1 Routing Scheme

Developed by (Ombuki et al., 2006), the routing scheme is simply a method of attempting to convert infeasible solutions in the population to feasible. The scheme takes a solution as it currently is, takes its first vehicle and iteratively inserts that vehicle's destinations into a new solution's route. A new vehicle is created when a destination (to be inserted) is reached that breaches the feasibility constraints. The infeasible destination is inserted to the start of its route, and the remaining insertions

continue from the new vehicle. Once this process is finished, the routing scheme then attempts to shift the last customer from the end of each route to the start of the following route. If these swaps destroy the feasibility of a solution, then they are discarded.

2.5.2.2 Best Cost Route Crossover

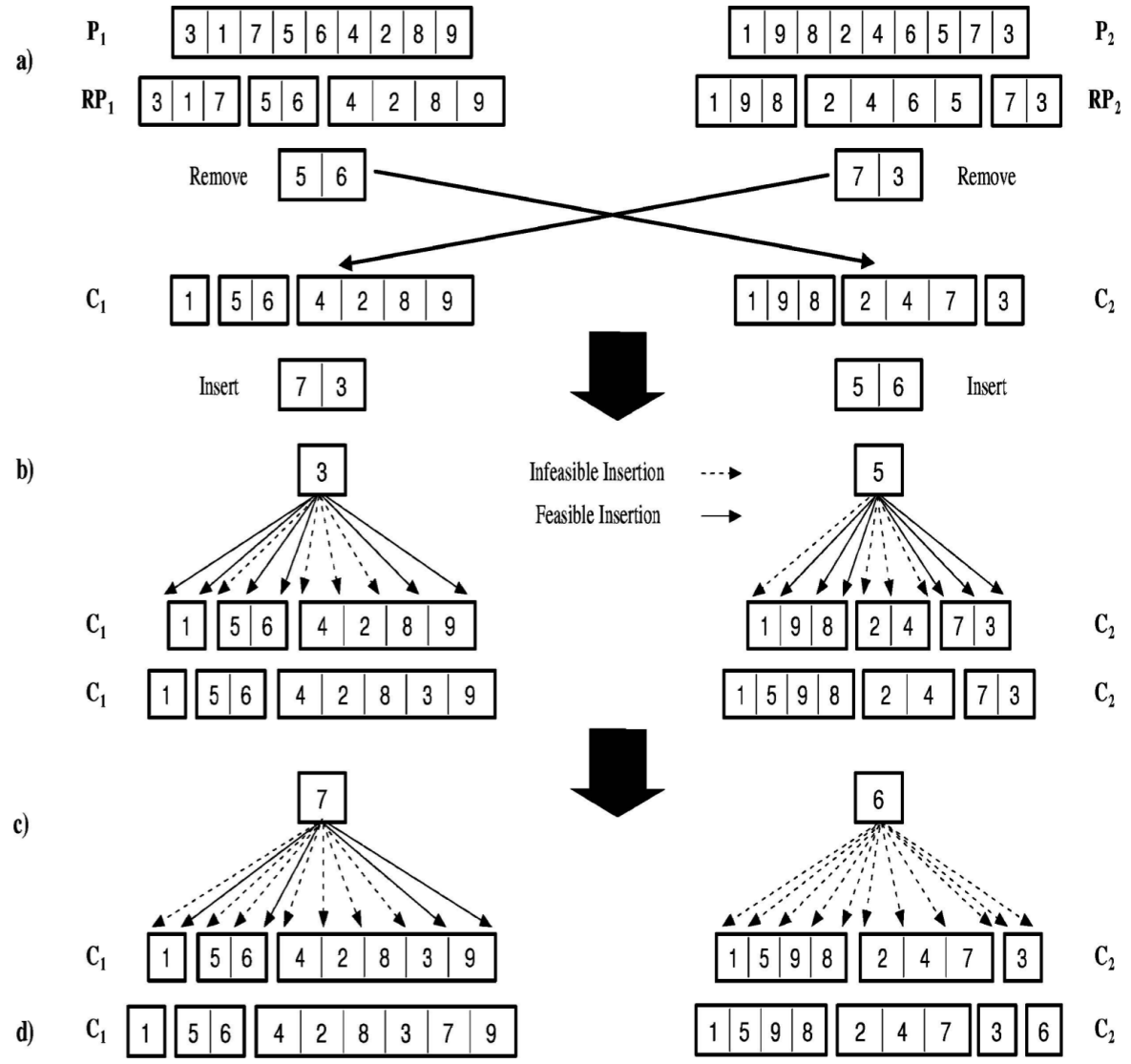


Figure 2: Best Cost Route Crossover (BCRC) operator (Ombuki et al., 2006).

The Best Cost Route Crossover (BCRC) aims to perform the same function as a traditional crossover operator, but with additional conditions relating specifically to the NP-hard CVRPTW (Ombuki et al., 2006). A standard crossover operator would be given one solution, remove (an) entire route(s) from it, select another solution from the population at random, and add some of the randomly selected solution's routes to the

given solution. However, a route from the random solution can only be added to the given solution if any of the routes remaining after removal do not already visit any of the random solution's destinations from the route in question. Therefore, with this logic (which is required to prevent duplicate visits to a node), some destinations may not be added after being rejected if they are on a route with a destination that the given solution still visits after removal. When such a situation does occur, missing destinations are either appended to the end of an existing route as long as the insertion does not make the route invalid. If it does, an entirely new route with a new vehicle needs to be created to accommodate the missing destination. BCRC instead inserts destinations that have been removed from the given solution back into it but within the best fitting point in its remaining routes (see Figure 2).

Ombuki et al. (2006) stated that the operator in Figure 2 represents every route together in one chromosome. A chromosome is a solution made up of genes (in this case, the genes are the customers). P_1 and P_2 in Figure 2 are the original chromosomes, and RP_1 and RP_2 show the solution's customers divided into their routes. C_1 and C_2 are the children produced by the crossover operator.

2.5.2.3 Constrained Route Reversal Mutation

While MMOEASA utilises multiple mutation operators (Baños et al., 2013), Ombuki et al. (2006) intend to reduce disruption to existing routes in a solution due to the CVRPTW's strict time window constraints; the constraints can cause even a slightly mutated solution to become infeasible. The mutation proposed selects two random points in a solution's entire chromosome and reverses the visiting order of the destinations between said points. To further minimise the disruption, the points are placed at close indexes in the chromosome so that the number of destinations between the points will be no greater than three. Because the chromosome is considered as a continuous list of destinations in their visiting order and not separated by the different routes, the mutation operator can also reverse destinations across different routes. For example, in Figure 2, say the reversal's start point is placed at destination 7 in P_1 and the end point is at destination 6 (where RP_1 shows how they are placed in different routes). After the reversal, route one would become "3, 1, 6" and route two "5, 7".

2.6 Conclusion

Since its initial proposal, algorithms written to solve the CVRPTW have been optimised via many mathematical developments. While still considered NP-hard, the problem has seen heuristic and algorithmic solutions that can generate highly optimal solutions quickly, overcoming the infeasibility of an exhaustive evaluation of all solutions to find the best. Although inspecting the results gathered by Baños et al. (2013) and Ombuki et al. (2006) reveals that the process of solving the CVRPTW is still incomplete. The results show that the values of the objective functions for every non-dominated solution after multiple terminations of the algorithm is non-deterministic of the optimal solution's objectives.

When an NP problem is solved, the algorithm that solves the problem will declare it as NP-complete. For the CVRPTW to become NP-complete and no longer NP-hard, an algorithm will need to solve the problem with one hundred per cent accuracy. Acquiring one hundred per cent accuracy would become apparent when the algorithm arrives at the same solution across multiple iterations; there are no better solutions to be discovered after the optimal solution is found.

3 Technical Justification

3.1 Languages and Libraries

Implementation of MMOEASA and Ombuki's Algorithm is a tricky process as, together, they consist of many operators and heuristics. Therefore, to make the development and debugging process of the algorithms smooth, it was decided to avoid low-level languages that are easy to make mistakes in and can be complex to debug. The purpose of this decision was to save time during the implementation process. The downside, however, that was foreseen before this decision was final is that the algorithms will not have the immense performance boost from being developed at a low level.

Low-level development of the algorithms would allow for a higher termination condition. For example, the algorithm could reach a termination condition of 1000 iterations in C++ faster than Python (the chosen language). A C++ implementation could reach said higher termination condition in the same amount of time it would take Python to reach a lower termination condition. Therefore, in C++, the termination condition could be set to a higher amount, allowing for more crossovers and mutations to be made to generate better solutions for longer.

3.1.1 SDKs

3.1.1.1 Python 3.9

The experimental application that consists of the chosen algorithms has been developed with the Python 3.9 SDK. Therefore, a version of the SDK ≥ 3.9 is recommended; SDKs < 3.9 may work, but this cannot be guaranteed.

The Python application contains functionality from the library NumPy 1.21.4 and, therefore, requires version $\geq$ 1.21.4. The NumPy package has been used to perform some mathematical functions in Python scripts.

3.1.1.2 C17

The application contains a non-essential C program. The application was developed on Windows using C17 (the latest stable release of the C standard library) and Microsoft's C/C++ compiler version 19.30.30706. Due to the application being

developed on Windows, it may not abide by standards of the GCC compiler or any others for that matter. If this is the case, some modifications may need to be made to compile the application.

The C application has also been given a Makefile and a CMake configuration to easily compile the program. The CMake file can also be used to build a Visual Studio Solution application for debugging; this has been tested using Visual Studio 2022.

3.1.2 Included Applications

The Python application consists of multiple files containing the functionality of the experimental implementations of the Genetic Algorithms. The instructions describing how to use the application can be found in the README markdown file included with the application.

The C application, included with the MMOEASA algorithm, also comes with instructions included in the README. The purpose of the C application is to allow for an external calculation of the MMOEASA objectives. Using an external validator allows for testing of MMOEASA's implementation. If the validator gives the same results as the Python implementation, it proves that the Python application generates valid solutions and evaluates them correctly. The solution validator is written in C as it contains the objective function from the original MMOEASA code, also written in C. An external check is required to ensure that the algorithm's objective function is bug-free; the results of an external validator address the issue that there could be a bug in the Python application, causing it to give false positives.

3.2 Datasets

3.2.1 Problem Instances

For the algorithms to carry out tests on a CVRPTW, the application needs a dataset consisting of a problem instance in numerical values. It was decided that the problem instances would be those developed by Solomon (1987) as they are famous instances and are thoroughly detailed. Therefore, in this dissertation, the algorithms that were chosen for experimentation also investigate the instances developed by Solomon. Solomon's problem instances are a popular benchmarking set of problem instances with multiple variations of the CVRPTW:

- Number of customers:
    - 25,
    - 50,
    - 100.
- Customers' location:
    - C - clustered customers,
    - R - uniformly distributed customers,
    - RC - a mix of R and C.
- Width of deliveries' time windows:
    - Set 1 - destinations with narrow time windows.
    - Set 2 - destinations with wide time windows.

3.2.2 Hypervolumes

Because MMOEASA utilises Hypervolumes in its multi-objective acceptance criterion (Multi-Objective Metropolis function), the Hypervolumes' values need to be provided before executing the algorithm. The original MMOEASA algorithm includes approximations of the objectives, but only for the 100 customer instances of Solomon's problems. Using these values would restrict the experimentation process to these problem instances and no others, excluding Solomon's 25- and 50-customer problems.

However, as the Hypervolumes are essential to the acceptance criterion's ability to distinguish slightly better solutions, they need to be provided before the genetic algorithm begins multi-objective acceptance. They cannot be calculated at run time as the algorithm will constantly be finding more optimal solutions and, thus, the Hypervolumes would change. Therefore, as long as the algorithm has not found the most optimal solution, the Hypervolumes will not reach values accurate to the optima. The effect becomes evident when the acceptance criterion uses inaccurate Hypervolumes; it is more lenient with accepting poor solutions. Even if the algorithm discovers accurate optimum values, previous generations made before the discovery would still have accepted poor solutions with more lenience.

As a result, the decision was made to use the original MMOEASA's Hypervolumes. The only other option is to calculate the Hypervolumes for every other problem

instance to prepare them prior to execution. Doing so would require three to four algorithm executions on each problem instance, recording the objective functions of the non-dominated solutions found to each instance (and each from the three to four executions), then using them in the Hypervolumes calculation. Because the algorithm can take 10 minutes to complete, three to four executions for every problem instance would take some time. Because Ombuki's algorithm also experiments on Solomon's 100 customer problem instances, it is safe to be restricted to experimenting on these instances by using these Hypervolumes.

To be able to do so, MMOEASA's acceptance criterion has to be implemented in Ombuki's Algorithm and vice versa. This is because MMOEASA's acceptance criterion does not have the Hypervolumes provided for the number of vehicles as an objective.

3.3 Areas of Research

The Genetic Algorithms chosen for research are reasonably recent and have been shown to produce good results. The decision to research recent algorithms is to discover newer advancements in the problem area. Recently developed algorithms will likely have improved functionality and heuristic methods of solving the problem that other researchers have previously discovered, so the chosen algorithms should have a more optimised approach to solving the problem than their predecessors.

As shown in Figure 2, BCRC creates two children, but Ombuki et al. (2006) do not specify how both children are used. Some GAs use optimal child solutions to overwrite the most unfit solutions in the population; in other words, all of the solutions that the child dominates. However, doing so can create a population with many similar solutions if a child solution is better than most in the current population. As a result, the crossover operator would be less effective as it would likely remove a route and replace it with the same route, rendering the crossover operator ineffective from that point forward. Even though Ombuki's BCR crossover operator inserts destinations into the fittest position of existing routes instead of moving an entire route from another solution, the population will reach a point where every route is (almost) as fit as they will get. Therefore, by having identical solutions, insertions to different positions in highly optimal routes will likely be less- or in-feasible, restricting the crossover to replicating the solution from which it just removed a vehicle. The edition of Ombuki's

algorithm implemented in this dissertation will only replace the parent solution from the iterator (and not any other index of the population) with the most dominant child.

Alternatively, the two crossover children could be used to replace both the parents from the iterator and the selection tournament if they dominate. However, if either of the children generated is the most dominant in the population during the next iteration, the selection tournament would select one of them. If the tournament-replacing-child solution is the most dominant, any subsequent solutions in the population (after the solution where the crossover occurred) would be crossed-over with the same solution again.

MMOEASA and Ombuki's Algorithm research slightly different objective functions but benchmarked using Solomon's problem instances. MMOEASA aims to control the number of vehicles by evenly distributing either the distance travelled or the cargo carried, while Ombuki's Algorithm attempts to minimise the literal number of vehicles. The differing objective functions can be corrected by modifying both algorithms to use both objective functions within each, using MMOEASA's objective function in Ombuki's algorithm and vice versa. However, the acceptance criteria also had to be used in both algorithms because MMOEASA's acceptance criterion requires Hypervolumes of the objective function it is solving. Thus, to use Ombuki's objective function in MMOEASA, Ombuki's acceptance criteria must also be moved as there are no Hypervolumes present in MMOEASA's original Hypervolumes for the amount of vehicles objective. Note that Simulated Annealing has been nullified in Ombuki's Algorithm so that it performs the Metropolis function without the effects of Simulated Annealing. The temperature is constantly the max temperature from MMOEASA for this.

The dataset used in MMOEASA's and Ombuki's Algorithm's research is Solomon's, the same as this dissertation intends to research. Using the same objective function and dataset will allow fair comparisons between results to be made as they will contain identical benchmarks and criteria to satisfy.

The algorithms chosen for experimentation also investigate the CVRPTW. The CVRPTW was chosen for investigation as it is a widely applicable and complex problem. Modern delivery schedules experience a high demand from customers to arrive as soon as possible. Therefore, because the CVRPTW is in high demand and

is a complex problem, it must be solved efficiently to save delivery time and not keep customers waiting.

3.4 Evaluation Strategy

The original research conducted upon MMOEASA and Ombuki's Algorithm is evaluated using different methods. The results of Ombuki's Algorithm are evaluated with a straightforward approach: the best-known solution (with the lowest objective function) to each of Solomon's 100-customer problem instances is listed, and next to those values, Ombuki's Algorithm's result. Therefore, if the algorithm's results are better than the best known, the algorithm can produce a better solution. However, evaluating the results using this method is somewhat trivial as the best solution the GA finds is used to evaluate it, regardless of the execution time. Even if the execution time to find the best solution is low, it may mean that the algorithm got lucky as crossover and mutation operators generate new solutions based on probability. Algorithms of this kind are stochastic; every execution likely generates a different result. This is important because, theoretically, giving any Genetic Algorithm long enough to execute would allow it to generate the best solution eventually.

To effectively evaluate the algorithm's performance compared to others, it is better to take an average or median value of results produced over multiple iterations, with each execution terminating at a certain point. This calculation would reveal whether or not one algorithm can produce higher quality solutions within a shorter time frame. It is better to take a median result rather than an average, as taking an average would be affected by outlying results; lucky or unlucky runs.

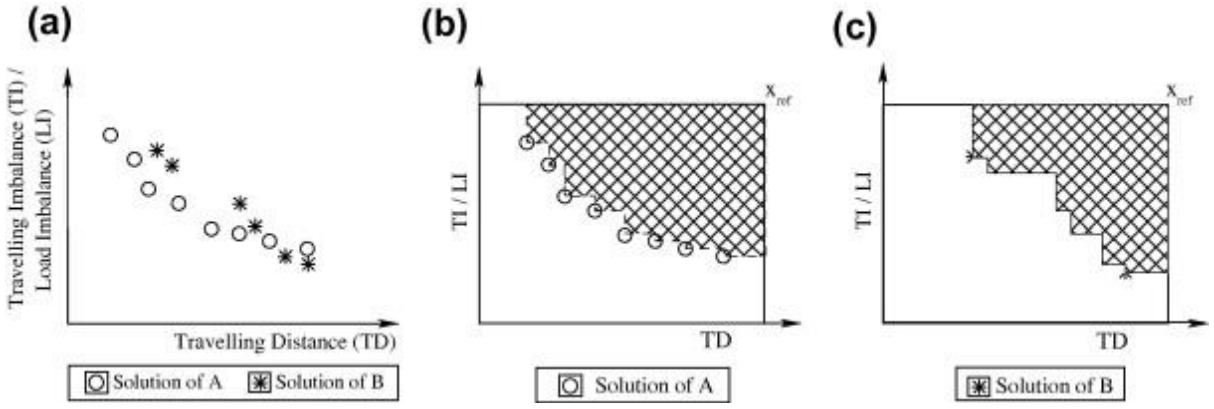


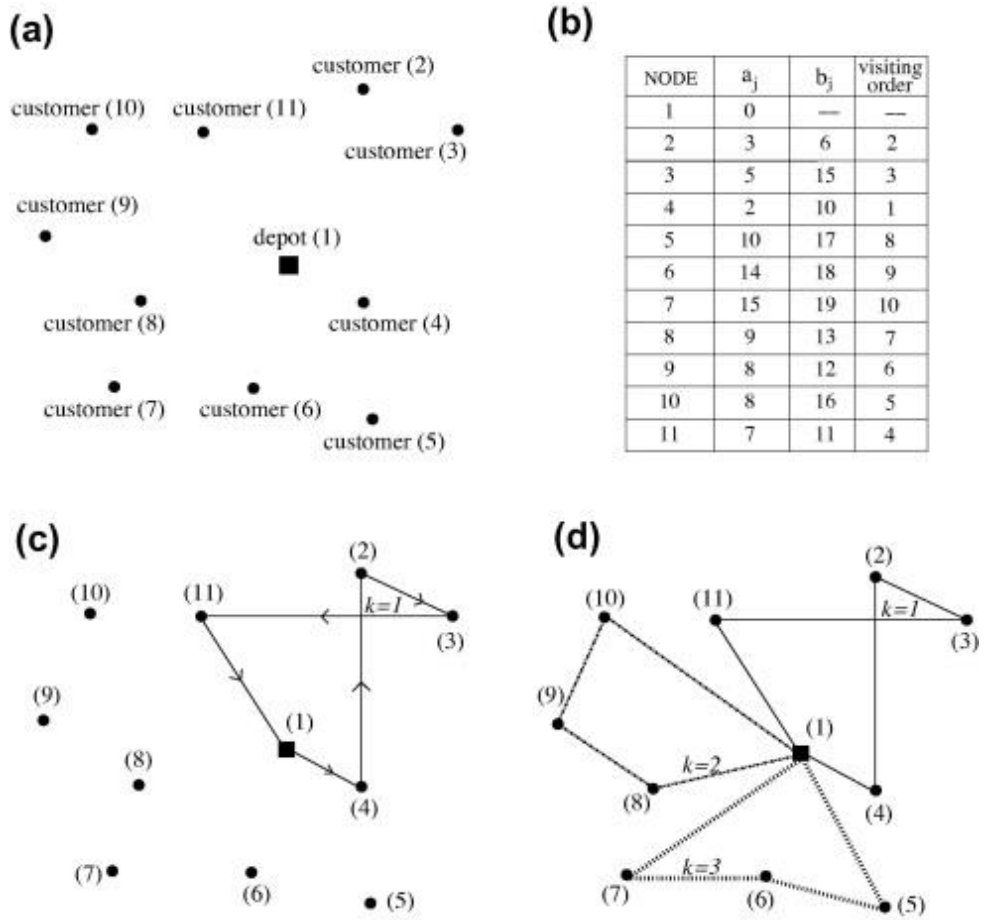
Figure 3: Graphical explanation of the metrics used over two non-dominated sets A and B.

The MMOEASA researchers did precisely this; results are evaluated using a median of the area of the final non-dominated set's Hypervolumes, plotted on a graph. Figure 3 shows the areas covered by two non-dominated solution sets, A and B; the x-axis is one objective (in this case, the total distance of a solution), and the y-axis is the second (the number of vehicles). Figure 3 (b) and (c) shows the Hypervolumes of solutions in set A and B plotted on a graph. Figure 3 (c) shows the area that both sets occupy. The resulting area of each set on the graph is between the Hypervolumes and a reference Hypervolume point with objective values equal to twice that of the initial solution: the result of the Time-Window-Based Insertion Heuristic (TWIH) (Baños et al., 2013). The area calculated is used in calculating the median area occupied by every non-dominated set generated by the algorithm.

A median value is used to approximate the actual performance using the results of the stochastic algorithm. The results' median is used instead of averages, as averages are less significant figures. The algorithms are stochastic; some results may have been obtained based on luck. If this is the case, the lucky results will increase the average graph coverage of each non-dominated set. This is also true for unlucky results; low graph coverages of some non-dominated sets would result in reduced average coverage. If no non-dominated set is produced, the algorithm in question could not solve that problem instance, and thus, the coverage of the non-dominated set is given as 0.0%.

The TWIH reference point (X_{ref} in Figure 3) has the values $x = 2 \times F1(TWIH)$ and $y = 2 \times F2(TWIH)$) (where $F1$ is one objective, and $F2$ is another, from the objective function) as the solution generated by the TWIH is present in every execution during the initialisation of the MMOEASA population. Figure 4 shows how the TWIH is calculated.

X_{ref} is always the top right of the graph as it is supposed to be a substantial value so that every Hypervolume in the non-dominated set will be less. Therefore, it is not essential how the reference point is calculated, as long as it is positioned at large coordinates. This dissertation will use a reference point relative to the problem instance and will be calculated as so:



NODE	a_j	b_j	visiting order
1	0	—	—
2	3	6	2
3	5	15	3
4	2	10	1
5	10	17	8
6	14	18	9
7	15	19	10
8	9	13	7
9	8	12	6
10	8	16	5
11	7	11	4

Figure 4: Time-Window-based Insertion Heuristic: (a) Spatial location of the depot and customers (N = 11). (b) Customers' time windows and visiting orders. (c) Customers visited by the first vehicle (k = 1). (d) The initial solution obtained by the TWIH( ) (Baños et al., 2013).

- The total distance objective will be fixed at an empirically calculated number: 10,000. The number is obtained from a few test runs as no non-dominated set contained any distance greater than 5,000. Thus, 5,000 is doubled to ensure safety.
- The other values, cargo disbalance (from MMOEASA's objective function) and the number of vehicles (from Ombuki's objective function), are taken from the problem instance. The values used are the max number of vehicles and cargo per vehicle permitted by the problem instance.

The reference point will be calculated this way to attempt a lower reference point. Having a lower reference point would reveal more minuscule differences in areas.

A set of non-dominated solutions is better if the area mentioned earlier is higher, covering most of the graph, indicating low Hypervolumes (close to the axes).

Therefore, a higher median area covered by the graphs indicated that an algorithm generates better non-dominated solutions in the same amount of time as another algorithm. Note that (as seen in Figure 3 (a)) the number of Hypervolumes in the final non-dominated set is permitted to vary as long as it does not exceed a certain number (Baños et al., 2013). The number does not have any impact on representing solutions' quality. If the number is low, this may demonstrate that the run found optimal, non-dominated solutions quickly or struggled to find any non-dominated solutions.

Due to the justification of MMOEASA's evaluation strategy, it will be employed as the evaluation strategy of this dissertation. It is considerate of stochastic algorithms and the theory that, with enough time, any Genetic Algorithm can produce the optimal solution.

3.4.1 Undefined Scenario in Ombuki's Specification

Figure 2 (d) shows that the BCR crossover creates a new vehicle for destinations with no feasible insertion point. However, it is not specified where the infeasible destination is inserted when the number of vehicles currently in the solution is equal to the problem instance's limit. For example, suppose the problem instance only permits 25 vehicles, and the crossover child being given new destinations already utilises 25 vehicles. In that case, the operator should not allocate a new vehicle for any infeasible destinations. Ombuki's Routing Scheme also has the same anomaly, and both the crossover operator and the routing scheme are important to Ombuki's Algorithm.

Because the original research does not cover what should happen in this scenario, the evaluation stage of this dissertation will contain experiments with two solutions to this problem. One is true to the original research: a new vehicle is created, regardless of whether it exceeds the problem instance's limit or not. This operator may be what the researcher intends as, over time, the crossover and routing scheme may reduce the number of vehicles. The second will use MMOEASA's alternative to inserting infeasible crossover destinations. The last destination of each vehicle already in the child solution will be compared based on their distance to the infeasible destination. The infeasible destination will be appended to the vehicle's route, where the distance, as mentioned earlier, is the shortest.

Two editions of the crossover and the routing scheme are implemented because the first implementation is likely inefficient. Inserting more routes than is permitted will create more vehicles for the crossover and routing scheme to merge into other solutions. Therefore, reducing the number of vehicles will take longer, mainly because the operators that can do so are also the cause of the additional vehicles. Attempting to remedy this with MMOEASA's strategy for inserting destinations with no feasible insertion will also make the performance comparison between MMOEASA and Ombuki's Algorithm fairer than using a new alternative method.

4 Feasibility Intensive Genetic Algorithm (FIGA)

As part of a custom Genetic Algorithm, this dissertation proposes three new GA functions, consisting of both operators and heuristics. Proposals aim to increase/maintain the feasibility of solutions during the GA's operations. The crossover and mutation operations also seek to perform optimal modifications in terms of customers' distances from one another.

The CVRPTW is NP-hard due to its heavy constraints. It could be said that it is necessary to use crossover and mutation operators that satisfy the constraints of the problem as they are one of the primary reasons the problem is as complex as it is. Theoretically, doing so would give those operators that maintain feasibility a better chance at exploring more exclusively the range of solutions that is feasible.

4.1 Initialisation: Distributed Time-Window-based Insertion Heuristic (DTWIH)

The DTWIH (see Figure 5) attempts to provide at least an almost-feasible solution from the initialisation stage. The idea is to give the algorithm a head start at meaningful genetic computations by attempting to give the crossover and mutation operators feasible parent solutions to use from the initialisation stage. Suppose the heuristic can produce a feasible solution to every problem (disregarding that solution's fitness as it is not important to the heuristic's function). In that case, this takes away the responsibility from the GA to first generate a feasible solution, solving the problem of a GA not being able to produce a feasible solution to a problem.

First of all, the nodes are sorted by their ready time. Then, the number of routes (n) that the nodes will be inserted into is calculated as roughly half of the maximum number of vehicles as allowed by the problem instance; see how n is calculated in Figure 5. Next, the first n sorted customers (nodes) are shuffled into a random order and appended to the end of the available n routes in the shuffled order. When the first n nodes of the sorted list have been inserted into their routes, they are removed from the sorted list to allow for the allocation of the next n nodes.

(a) Available nodes

Node	ready_time	cargo
1	12	5
2	14	10
3	5	5
4	10	10
5	3	10
6	9	5
7	11	10
8	15	5
9	7	10
10	8	10

(b) Sorted by ready_time

Node	ready_time	cargo
5	3	10
3	5	5
9	7	10
10	8	10
6	9	5
4	10	10
7	11	10
1	12	5
2	14	10
8	15	5

Example max capacity of vehicles = 30
***n* = ceil(m/2)** - where 'm' is the max numer of vehicles in the problem (in this example, say **m = 5**, so ***n* = 3**)

(c.i) shuffle the first *n* nodes of the sorted nodes; example result:

List index	shuffled nodes
0	3
1	9
2	5

(c.ii) insertion of sorted nodes: **iteration 1**

Route	nodes (in delivery order)
1	**3**
2	**9**
3	**5**

(d.i) shuffle the first *n* nodes of the remaining sorted nodes; example result:

List index	shuffled nodes
0	6
1	10
2	4

(d.ii) insertion of sorted nodes: **iteration 2**

Route	nodes (in delivery order)
1	3, **6**
2	9, **10**
3	5, **4**

(e.i) shuffle the first *n* nodes of the remaining sorted nodes; example result:

List index	shuffled nodes
0	2
1	7
2	1

(e.ii) insertion of sorted nodes: **iteration 3**

Route	nodes (in delivery order)
1	3, 6, **2**
2	9, 10, **7**
3	5, 4, **1**

8 can also be appended to route 1 as insertion there still does not violate capacity constraints

(f.i) shuffle the first *n* nodes of the remaining sorted nodes; example result:

List index	shuffled nodes
0	8

(f.ii) insertion of sorted nodes: **iteration 4**

Route	nodes (in delivery order)
1	3, 6, 2, **8**
2	9, 10, 7
3	5, 4, 1

Figure 5: Custom insertion heuristic; Distributed Time-Window-based Insertion Heuristic (DTWIH)

The heuristic is performed p times, where p is the population size; in other words, p solutions are created and inserted into each population index. The n customers that are iteratively inserted into the routes are always shuffled to create diversity in the initialised population's solutions. Capacity constraints are always accounted for: if a

vehicle's capacity is at its limit and there is a node from the sorted list to be inserted into it, one of the unused vehicles will be assigned the infeasible node. Assigning infeasible nodes to a new vehicle is not optimal due to the objective function, but there is no other choice when an insertion is infeasible.

4.2 Crossover Operator

The crossover operator utilised in Ombuki's Algorithm specialises in producing feasible CVRPTW child solutions. As previously discussed, Ombuki's crossover operator aims to merge two solutions in a manner specific to the problem. In contrast, MMOEASA's operator is a standard crossover method that often results in missing customers, meaning they will have to be reinserted in some manner. MMOEASA reinserts missing customers to the end of a route as long as insertion does not violate the vehicle's capacity. Otherwise, in both algorithms, a new route is created and is allocated the infeasible customer. However, Ombuki's crossover operator also has one issue. As shown in Figure 2, the position chosen for customers during crossover is based on the fittest location in the entire chromosome of customers. The problem with this method is that when one customer finds a feasible solution and is inserted, the next customer may have been a better fit in that position. The operator cannot interpret this as the previously inserted customer has taken the position and likely worsened that position for the next customer.

The proposed solution uses a decision tree, an iterative and recursive search for the fittest insertion points for each customer during Ombuki's crossover. A decision tree aims to find the fittest order of insertions and the fittest insertion points. Figure 6 shows the decision tree algorithm written to solve the problem. The issue with using decision trees is that they are an expensive procedure that can become exceptionally deep searches. The variable $customers\_to\_insert$ in Figure 6 is the list of customers in a parent vehicle's route from a solution given to the crossover operator. This decision tree would have a depth relative to the number of customers that vehicle visits. For example, if the vehicle visits ten customers, the tree depth will equal ten. There are a couple of conditions in place to reduce the search space:

1. With every recursive invocation, one customer is removed from the set of customers to insert. Therefore, the number of evaluations made at the next

recursion in every depth is equal to $customers\_to\_insert.length - 1$ (see line 19 in Figure 6).

2. Evaluations of remaining nodes are only performed if the insertion of a previously un-inserted node is feasible; notice that the recursive invocation only happens if line 14 in Figure 6 is true.

The decision tree operator is unsuccessful because it takes too long to execute one tree. The point of the GA is to create a child by inserting (a) route(s) from one parent into another, then check if the child dominates the parent(s), and iterate to the next parent solution. Not many genetic operations will occur if the decision tree requires much time to execute. If the algorithm's termination condition is measured in execution time, much time will be wasted on performing crossovers with the decision tree. Therefore, the decision-tree-based operator will not be used in this experiment; instead, Ombuki's crossover operator shall be implemented.

```
1  fittest_insertion_points_decision_tree(child_solution, customers_to_insert, best_solution_so_far, depth):
2      if depth == 0 and child_dominates(child_solution, best_solution_so_far):
3          return child_solution
4
8      for each customer in customers_to_insert:
5          shortest_from_previous = INT_MAX
6          shortest_to_next = INT_MAX
7
9          for each vehicle in child_solution.vehicles:
10             for each destination in range 1 .. vehicle.destinations.length:
11                 distance_from_previous = euclidean_distance(vehicle.destinations[destination - 1], customer)
12                 distance_to_next = euclidean_distance(customer, vehicle.destinations[destination])
13
14                 if route is feasible with customer inserted into vehicle.destinations at point destination
                       and ((distance_from_previous < shortest_from_previous
                             and distance_to_next ≤ shortest_to_next)
                         or (distance_from_previous ≤ shortest_from_previous
                             and distance_to_next < shortest_to_next)):
15
16                     child_solution_copy = copy of child_solution
17                     calculate objectives and time windows of child_solution_copy
18
19                     evaluation_result = fittest_insertion_points_decision_tree(child_solution_copy,
                                                 customers_to_insert - customer, best_solution_so_far, depth - 1)
20                     if evaluation_result ≠ null:
21                         best_solution_so_far = evaluation_result
22                         shortest_from_previous = distance_from_previous
23                         shortest_to_next = distance_to_next
24     return best_solution_so_far
```

Figure 6: Pseudocode of the decision tree crossover operator.

As an alternative to the decision tree crossover, a modified edition of Ombuki's BCR crossover is implemented. As shown in Figure 2, Ombuki's operator adds customers

with no feasible insertion to a new route, regardless of whether the new route creation exceeds the maximum number of vehicles and, thus, creates an infeasible child. The modification proposed attempts to remedy that by placing the new, infeasible destination before the destination with the highest wait time in the entire solution. The primary reason for this is to insert an unvisited customer. However, inserting a customer at this position also aims to reduce wait times. Wait times can waste much time waiting for a destination's ready time to be available, significantly decreasing the likelihood of satisfying the time windows of destinations following the customer that a vehicle spends much time waiting to serve. Therefore, with high wait times removed and put to use, positions in the route that follow the destination with the previously high wait time should have a higher chance of having their time windows satisfied.

Although, the crossover of the destination that replaced the wait time resulted from no feasible insertion points. That means that this insertion point is also infeasible. It is left to the remaining crossover insertions and the mutation operators to attempt to find a destination before/after the new, infeasible destination to make it feasible again. The mutations written aim to increase and maintain feasibility; they may also restore infeasibility caused by the crossover operator. This signifies that it is likely that this insertion heuristic cannot be converted to a deterministic heuristic for the problem as it is dependent on the operator inserting the destination into a good position. The best insertion point cannot be calculated (for example, based on the shortest distance and longest wait time) on the first iteration for the same reason that the best crossover insertion point cannot be calculated without using a decision tree. Also, high and low wait times do not indicate the best insertion point, only whether the order is good.

4.3 Mutation Operators

The mutation operator originally proposed with Ombuki's Algorithm is has a minimalistic effect on solutions. As previously discussed, the purpose is not to change the order of a route too much, as doing so will likely destroy its feasibility. MMOEASA, on the other hand, has ten mutation operators (Baños et al., 2013). It could be argued that more mutations lead to a more comprehensive variety of solutions. Therefore, this dissertation will explore multiple mutation operators that have been designed to attempt to maintain the feasibility of solutions. Although, some do not prioritise

feasibility to create diversity in solutions produced by not relying on time-window-based mutations. Which operator to be executed is selected randomly.

4.3.1 Time-Window-based Sort Mutator

There is a $^2/_3$ chance that the route selected for mutation is selected based on the total wait time of each customer on that route. The other $^1/_3$ is a random selection. The element of randomness is to account for solutions being unchanged before they arrive at mutation again (for example, if they are highly optimal). All customers in that route are sorted in ascending ready time order, identical to Figure 5 (b).

4.3.2 Time-Window-based Swap Mutator

It has the same functionality as the Time-Window-based Sorting Mutator but only swaps the order of one destination whose ready time is later than the following destination's.

4.3.3 Wait-Time-based Swap Mutator

Again there is a $^2/_3$ chance that the route with the longest total wait time is selected. In this mutator, only customers with a wait time are reordered by swapping them with the destination positioned after them. Swaps are only made if the first destination's wait time is greater than the second's.

4.3.4 Single Wait-Time-based Swap Mutator

The same function as the Wait-Time-based Swap Mutator, but only performs one swap on destinations in the entire route.

4.3.5 Distance-based Swap Mutator

This mutator is less considerate of time window constraints but does not make any drastic changes; to try and maintain feasibility. Take the longest route and iterate its destinations. Take the iterator destination and compare the distance between the next destination and the destination after the next. If the next again destination is closer than the next, swap them. In an ideal scenario, this would only increase wait times.

4.3.6 Single Distance-based Swap Mutator

Again occurs in a single instance; identical function of the Distance-based Swap Mutator, but only swaps one pair of destinations in the entire route.

4.3.7 Time-Window-based Move Forward Mutator

Pick a random route. Only the destination with the lowest ready time is reordered in this mutator and moved to the route's beginning. If the lowest is already at the beginning, try to move the second-lowest to the second position in the ordered route, or the third, and so on.

4.3.8 Time-Window-based Push-back Mutator

The opposite of the Time-Window-based Move Forward Mutator. The destination with the highest ready time is moved to the last position, or the second-highest to the second-last, and so on.

4.4 Evaluation of the Feasibility Intensive Genetic Algorithm

4.4.1 Strategy

Again, the area coverage of the non-dominated solutions on the graph produced after the algorithm’s termination will be used to determine FIGA’s effectiveness at producing quality solutions. Note that FIGA will store non-dominated solutions in a non-dominated set separate from the population. This is the strategy employed by MMOEASA in which non-dominated solutions are added to a non-dominated set, and if another solution ever dominates them, they are removed.

The algorithm as a whole will need to be analysed to evaluate the effectiveness of the operators implemented in FIGA. In a GA, the operators work together to produce a result. The operators could be tested individually, but these results would only be relevant when the research aims to analyse precisely the effectiveness of GA’s operators at solving a problem. As this dissertation aims to find what algorithms effectively solve the CVRPTW, the results should aim to show whether one algorithm can produce a better result in the same amount of time as another. Therefore, if the results show that FIGA outperforms the others, this information is more relevant than which operators succeeded more often.

4.4.2 Proving Results' Significance

From the resulting areas gathered for each problem from each algorithm, the Mann-Whitney Test (Mann & Whitney, 1947) calculates the likelihood that the results obtained were produced by chance. For example, suppose one algorithm's (algorithm A) resulting areas are higher than another (algorithm B). The Mann-Whitney test can calculate the likelihood that algorithm A's areas are higher because the algorithm is genuinely better and are not higher because the algorithm had (a) lucky run(s).

To prove an algorithm is better, the Mann-Whitney test of this dissertation investigates the probability of two hypotheses being true after any number of runs of the two algorithms. Those two hypotheses are as follows:

1. Algorithm B will produce a result that is at least as good as, if not better than, algorithm A's result, and
2. Algorithm A will produce a better result than algorithm B's result.

Where algorithm A is the algorithm with higher areas and algorithm B is the other.

Two primary values are considered in the test: the Significance Level (α) and the p-value. α is used to determine the minimum probability of a hypothesis occurring for it even to be considered a plausible hypothesis, and the p-value is the actual probability of a hypothesis being true. Where the p-value is less than the Significance Level, the idea that the areas may be higher by luck can be discarded as this probability is insignificant. When this is the case, it shows that comparing the results given to the test will be significant; the results gathered can be trusted enough (trusted to be authentic and not lucky) to be compared.

The reason for doing this check is because, within the time scope of this dissertation, the experiments will not be performed enough times to ensure that lucky results do not have a significant impact on the actual median areas of the algorithms.

4.5 Rejected Ideas

As FIGA focuses on performing operations that are considerate of the CVRPTW's constraints, the ideology behind focusing solely on operators is twofold. Firstly, operators are written to focus on maintaining feasibility during genetic computations.

Secondly, because the CVRPTW is an NP-hard problem, the heuristics implemented by MMOEASA (Simulated Annealing, Multi-Start, and the Metropolis Function) are not used. While MMOEASA's heuristics most definitely perform well in multi-objective problems, the function of Simulated Annealing is theoretically less prominent in NP-hard scenarios.

Simulated Annealing reduces the probability that the child will overwrite the parent when neither the parent nor child dominates the other. The odds of overwriting a non-dominated parent in an NP or NP-complete problem is effective in these problems as they likely have a determinable optimal solution that the algorithm should be guided towards achieving. The genetic computations are guided using Simulated Annealing because as the algorithm gets closer to generating the optimal solution, Simulated Annealing gradually reduces the odds that currently known optimal solutions will be overwritten with child solutions that do not dominate their parents.

NP-hard problems, however, often do not have a single, most optimal solution and instead provide a set of multiple non-dominated solutions. If Simulated Annealing allows a generated child solution to overwrite the parent, the child can only be as good as the solutions in the non-dominated set. The parent will have already been evaluated to check if it should become a member of the non-dominated set. Therefore, since the child does not dominate the parent, if the child overwrites the parent, the outcome of this event is that the non-dominated set can be given an additional solution to list as another one of the multiple solutions to the NP-hard problem.

Instead of using Simulated Annealing and the Metropolis Function as acceptance criteria, FIGA will record solutions only if they dominate their parent. Multiple non-dominated solutions do not have to be provided, and it is more desirable that a GA prioritises dominating solutions.

4.6 Additional Information and Pseudocode

Both MMOEASA and Ombuki's Algorithm implement a selection tournament. Selection tournaments are efficient when performing crossovers, so FIGA employs a selection tournament. Ombuki's Algorithm has some probability of selecting either the best solution for a crossover or a random solution from the population (Ombuki et al., 2006). MMOEASA has a $^1/_2$ chance of using the non-dominated set to select a

random parent; otherwise, the population is used. FIGA will instead give the tournament an 80% chance of selecting a solution from the non-dominated set. If the 80% probability fails or the nondominated set is empty, a random solution from the population is selected.

```
1  FIGA(problem_instance, population_size, termination_condition, crossover_probability, mutation_probability):
2      population = list
3      nondominated_set = list
4
8      for i in range 0 .. population_size - 1:
5          population[ i ] = DTWIH()
6
7      while not termination_condition has been met:
9          crossover_parent_two = selection_tournament(nondominated_set, population)
10
11         for each solution in population:
12             //if crossover/mutation probability fails, they return the solution (first) parameter given to them
13             child = crossover(solution, crossover_parent_two, crossover_probability)
14             child = mutation(child, mutation_probability)
15
16             if child is not solution:
17                 if not solution.feasible or child dominates solution:
18                     overwrite solution in population with child
19                     check_nondominated_set_acceptance(nondominated_set, child)
20     return nondominated_set
```

Figure 7: FIGA main algorithm pseudocode.

The main algorithm of FIGA is represented in Figure 7. The main algorithm contains no particular functionality explicitly developed to aid the previously described operators. However, the functionality of the acceptance criterion is as follows. If a child solution was generated and is feasible, but the parent is not feasible, or the parent is feasible, and the child dominates it, then overwrite the parent. Then whether or not the child should be accepted into the non-dominated set is checked. Figure 8 details the procedure to work out the non-dominated set.

The non-dominated set acceptance procedure first adds the new child solution to the non-dominated set. Then, an evaluation of the entire non-dominated set is performed to determine which solutions should remain after the new solution that dominated its parent has been appended. Solutions that should remain are those in the non-dominated set that are still non-dominated with the new solution. The new solution can also be removed during this procedure if any solution in the non-dominated set dominates it. Line 13 in Figure 8 also checks for duplicate solutions based on whether two solutions' objectives are equal. Lines 9 and 10 are loops that reduce this check from $O(n^2)$ complexity to $O(n \log n)$. The non-dominated set will likely never be large

enough for this reduction to be noticeable, but it is implemented to spend less time focusing on correcting the non-dominated set to return to genetic operations quicker.

```
1  check_nondominated_set_acceptance(nondominated_set, child_solution):
2      if not child.feasible:
3          return
4
8      nondominated_set.append(child)
5      solutions_to_remove = set //a set is used for optimisation; it's faster to check if a value is in a set than a list
6
7      if nondominated_set.length > 1:
9          for s in range 0 .. nondominated_set.length - 2:
10             for s_aux in range s + 1 .. nondominated_set.length:
11                 if nondominated_set[s] is dominated by nondominated_set[s_aux]:
12                     solutions_to_remove.add(s)
13                 else if nondominated_set[s_aux] is dominated by nondominated_set[s] \
                       or both solutions at indexes s and s_aux are equal:
14                     solutions_to_remove.add(s_aux)
15
16         if not solutions_to_remove is empty:
17             i = 0
18             for s in range 0 .. nondominated_set.length - 1:
19                 if s not in solutions_to_remove:
20                     nondominated_set[i] = nondominated_set[s]
21                     i++
22             remove solutions from i to nondominated_set.length from nondominated_set
```

Figure 8: FIGA's "check_nondominated_set_acceptance" procedure pseudocode.

5 Evaluation

5.1 Test Data, Parameters, and Termination Condition

As MMOEASA and Ombuki's Algorithm explore different objective functions, results have been gathered that represent both algorithms' efficiency at solving both objective functions. The measurements taken are results from the first versions of each type of Solomon's benchmarking problem instances. The instance names are:

- C101,
- R101,
- RC101,
- C201,
- R201,
- RC201.

One of each type of Solomon's problem instances has been experimented with to examine the flexibility of the algorithms. Each algorithm is executed on the problems listed five times, and the final resulting areas for each problem will be the median of those five areas. It would be a more rigorous testing process to use more of Solomon's instances more than five times. This would represent how well each algorithm performs when solving each obstacle that the different instances contain.

Some of the problems chosen for experimentation will be more challenging to solve than the others; primarily, the narrow-time-window instances are more challenging than the wide time windows. However, using the same benchmarking instances across each algorithm is still a fair comparison of their abilities. The evaluation process would be more effective by carrying out experiments more than five times to reduce the error of having lucky and unlucky runs due to the stochastic algorithms. Doing so was not feasible within the scope of this dissertation's time duration.

Each algorithm employs different operators and heuristics from each other. Therefore, the parameters used for MMOEASA and Ombuki's algorithms are specified by the recommendations of their original research conducted, except the termination condition. The parameters have remained unchanged from the original research as

their values were likely set empirically by the original researchers or were calculated. For MMOEASA:

- Population size = 20 solutions,
- Multi-starts = 10,
- Crossover probability = 25%,
- Mutation probability = 25%,
- Temperature max = 100,
- Temperature min = 50,
- Temperature stop = 30.

For Ombuki's Algorithm:

- Population size = 300 solutions,
- Crossover probability = 80%,
- Mutation probability = 10%.

Finally, for FIGA:

- Population size = 30 solutions,
- Crossover probability = 80%,
- Mutation probability = 50%.

For FIGA, the population size is set to a moderate value to allow for more operators to be applied to the population before termination. A larger population size would lead to a more extensive set of solutions with only one operator applied to them during each iteration. The smaller population size allows more iterations and, therefore, more crossovers and mutations. The idea is inspired by the multi-point crossover modification highlighted in the literature review (Eiben et al., 1994); more crossovers have been shown to lead to better solutions. The crossover's probability of occurring remains the same as Ombuki's as the operator is similar to Ombuki's crossover. The mutation operator's invocation probability is 50% to help avoid infeasible children.

The termination condition of each algorithm during testing is measured in seconds. Each algorithm is given 600 seconds (10 minutes) to perform one execution. Experiments were performed independently from each other (one at a time) on PCs with the following OS and hardware capabilities:

- Windows 10 Enterprise version 20H2 (64-bit)
- Intel Core i7-10700 vPRO x64 CPU (8 cores with 16 threads at 2.90 GHz)
- 32 GB of RAM

The termination condition is constant to conduct a fair comparison and recognise which algorithms can produce a better result than the others within the same amount of time. Time is the only consistent unit as a termination condition throughout each algorithm. If, for example, 2000 iterations is used as a termination condition, then the measurements taken will not be fair as different algorithms can have different population sizes and crossover and mutation probabilities. In this case, MMOEASA has a 25% chance of performing crossovers or mutations on 20 solutions. Ombuki's Algorithm is given an 80% chance of crossover and a 10% chance of mutation with 300 solutions. Therefore, theoretically, one iteration of Ombuki's algorithm would perform 270 operations ($300 \times (0.8 + 0.1)$), and MMOEASA only 10 ($20 \times (0.25 + 0.25)$). Time has been used to terminate the algorithms while maintaining the original probability parameters to honour the original research.

5.2 Experimentation Results

During experimentation, algorithms that failed in one execution of a problem instance also often failed at every subsequent execution of the same problem instance. In other words, algorithms were often consistent in not producing results to the same problem after every execution. The cause is that when an algorithm fails at one problem, the operators or heuristics, or both, are not efficient at solving that problem. The best example of this is the original edition of Ombuki's Algorithm compared with the no feasible insertion fix (see Table 1).

5.2.1 Comparing Ombuki's Original Operators and MMOEASA's Alternative Infeasible Destination Insertion

Table 1 contains the median areas for Ombuki's Algorithm with the original vehicle-limit-exceeding implementation and MMOEASA's infeasible insertion alternative. The results show that the effect of the operators can drastically impact an algorithm's ability. The original operators that create excess vehicles that exceed the instance's limit cannot produce any feasible solutions to the problems listed. In contrast, the operators with MMOEASA's alternative (that appends infeasible destinations to the

route with the nearest last destination) generated feasible results. The exceptions are problems R101 and RC101 with no results. These problems are more complicated because customers are less clustered, further apart, and have narrow time windows. This makes them more challenging because the strict time windows plus the long distances make many combinations infeasible.

Algorithm	Solomon's Problem Instance – Median Area Covered					
	C101	R101	RC101	C201	R201	RC201
Ombuki (original)	0%	0%	0%	0%	0%	0%
Ombuki (with MMOEASA's alternative)	73.3%	0%	0%	72.89%	75.81%	67.32%

Table 1: Results of Ombuki's Algorithm, comparing the original crossover and MMOEASA's no feasible insertion point alternative.

Ombuki's original implementation will be discarded as the operators yield infeasible results. Any further evaluations that use results from Ombuki's Algorithm will do so with the results gathered from MMOEASA's infeasible insertion fix.

5.2.2 Comparing MMOEASA and Ombuki's Algorithm: MMOEASA's Objective Function

As previously mentioned, MMOEASA's objective function is to reduce the total distance travelled by each vehicle in a solution and the imbalance of the amount of cargo carried by each vehicle.

Algorithm	Solomon's Problem Instance – Median Area Covered					
	C101	R101	RC101	C201	R201	RC201
MMOEASA	49.26%	0%	47.07%	64.45%	68.43%	64.55%
Ombuki	30.12%	0%	0%	61.84%	68.93%	61.42%

Table 2: MMOEASA and Ombuki's Algorithm at solving the CVRPTW with total distance and cargo imbalance as the objective function.

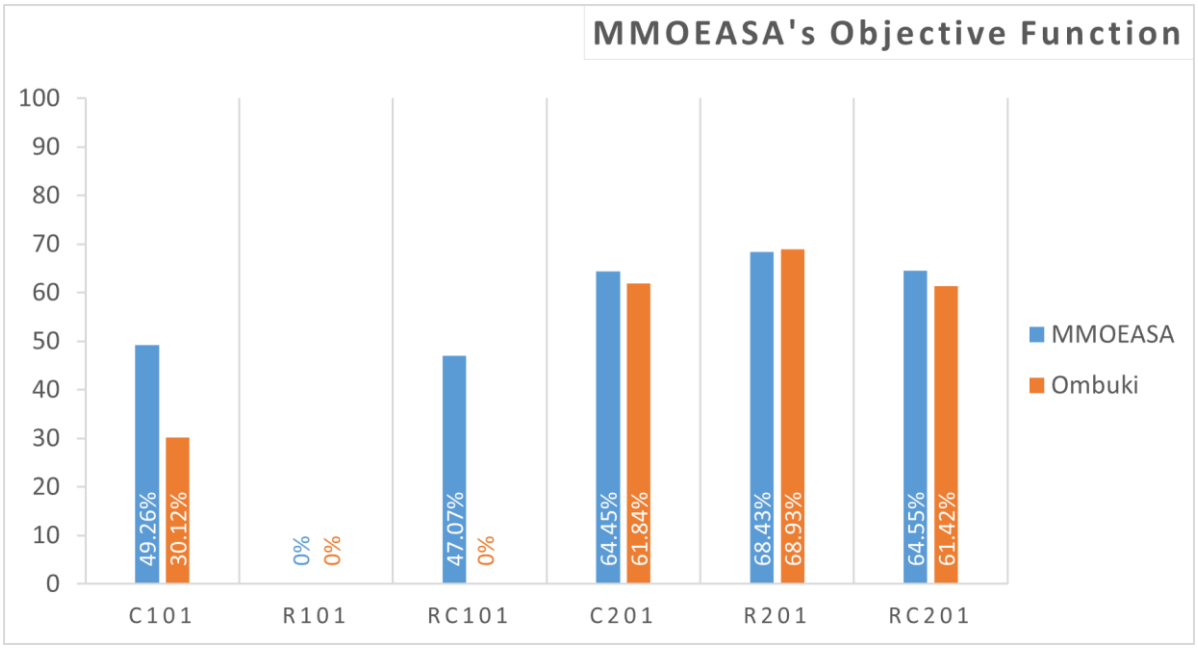


Figure 9: Bar graph representing MMOEASA's and Ombuki's Algorithm's median areas after solving MMOEASA's objective function.

It could be said that MMOEASA's objective function is more challenging to solve than Ombuki's. When considering the instances of problems C (clustered customers) specifically, it can be derived that the algorithms struggle to distribute the load. For example, it would be impossible to distribute cargo when one cluster of four customers has a small cargo demand and another cluster of seven customers has a high cargo demand if the clusters are positioned far apart.

As shown in Table 2 and Figure 9, the performance of both algorithms is slightly different. MMOEASA can solve the problem instance RC101, but Ombuki's Algorithm cannot. The reason Ombuki's Algorithm falls behind is likely due to the modified crossover and routing scheme operators. The modified operators likely do not honour Ombuki's Algorithm's original strategy.

MMOEASA generally performs better at solving its problem instance, but this can be expected as it is likely that MMOEASA was written to solve this objective function specifically. Such as how Ombuki's crossover operator finds feasible insertion points and, in the best scenario, eliminates an entire vehicle; this is an optimal effect in reducing the number of vehicles and solving Ombuki's objective function.

5.2.3 Comparing MMOEASA and Ombuki's Algorithm: Ombuki's Objective Function

Theoretically, Ombuki's objective function should be easier to solve than MMOEASA's. There should not be a situation where the number of vehicles is forced to be high. The only scenario where this would be true is when a vehicles' max capacity is low. The lowest vehicle capacity in the problem instances experimented with is only as low as 200, which is not high either but is workable. Ten customers, each with a demand of 20, could be serviced by a single vehicle.

Algorithm	Solomon's Problem Instance – Median Area Covered					
	C101	R101	RC101	C201	R201	RC201
MMOEASA	79.54%	0%	76.62%	85.54%	74.42%	77.99%
Ombuki	73.3%	0%	0%	72.89%	75.81%	67.32%

Table 3: Ombuki's Algorithm and MMOEASA at solving the CVRPTW with total distance and number of vehicles as the objective function.

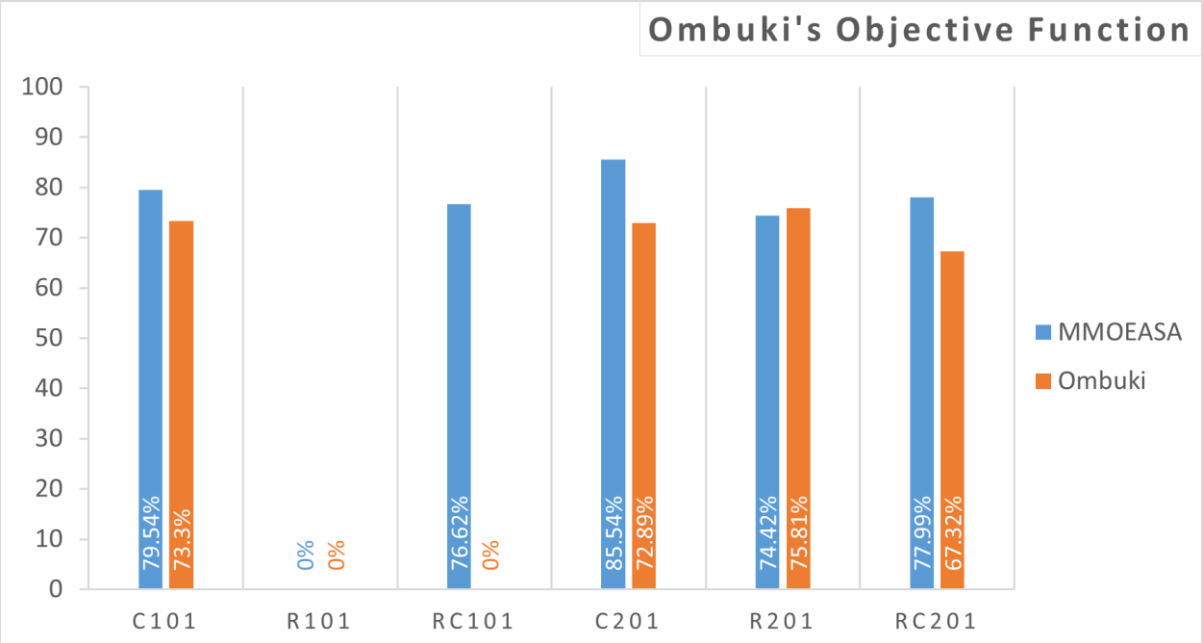


Figure 10: Bar graph representing Ombuki's Algorithm's and MMOEASA's median areas after solving Ombuki's objective function.

Ombuki's Algorithm appears to perform slightly better in problem instance R201, as per Table 2 and Table 3. Again as shown in Table 3 (and Figure 10), Ombuki's cannot solve RC101, and both algorithms failed to solve the problem instance R101. The issue may result from MMOEASA's crossover fix that is also applied to Ombuki's Algorithm. Table 1 represents the importance of this fix. However, before a feasible solution is found, crossovers are only given infeasible routes where destinations with no feasible insertion point. The already infeasible routes will force destinations to be appended to the end of a route, which likely does not support feasibility.

Although Ombuki's crossover distributes a route's destinations to other routes based on feasibility, feasibility-based insertions may be less effective if there is no feasible route, to begin with. The arrival time of nodes in infeasible routes will be high, which is often why time windows are violated; routes' distance between destinations is not low enough to arrive at the next destination within the time window. This will be tougher to solve in instance R101 due to the complexity of the problem; distant customers plus strict time windows.

It appears that Ombuki's objective function is easier to solve than MMOEASA's, but this is not directly comparable between objective functions as they both use different reference points to match the objective function. However, there is only one different value of X_{ref}: the cargo imbalance and number of vehicles axes are both the maximums from the problem instance. The other coordinate (total distance) is always 10,000, so this objective is comparable.

Objective Function	Average Total Distance
MMOEASA	2,566.261
Ombuki	2,110.024

Table 4: Median total distance of the non-dominated sets correlating to the median areas listed for MMOEASA – C101 in Table 2 and Table 3.

Because the total distance of the median graph coverage sets is lower for Ombuki's objective function, Table 4 partially indicates that it is easier to solve. However, the

objective functions are not directly comparable due to the different objectives. The objective function used depends on what the user is looking for in a solution, so it cannot be said that one is less effective than the other.

5.2.4 Comparing MMOEASA and Ombuki's Algorithm with FIGA

Table 2 and Table 3 show that MMOEASA produces higher quality solutions in the time given, which may or may not disprove the theory of problem-specific operators being more efficient; this does not prove the premise behind Ombuki's crossover is ineffective. It may reveal that Ombuki's crossover is less effective than it could be. Hence, FIGA's modified edition of Ombuki's crossover. The objective function used in FIGA is Ombuki's objective function: minimising the total distance and the number of vehicles.

	Solomon's Problem Instance – Median Area Covered					
Algorithm	**C101**	**R101**	**RC101**	**C201**	**R201**	**RC201**
FIGA	91.62%	82.75%	0%	94.06%	87.38%	86.3%
MMOEASA	79.54%	0%	76.62%	85.54%	74.42%	77.99%
Ombuki	73.3%	0%	0%	72.89%	75.81%	67.32%

Table 5: FIGA compared with MMOEASA and Obuki's Algorithm at solving the CVRPTW with total distance and number of vehicles as the objective function.

Table 5 also lists MMOEASA's and Ombuki's results from Table 3; for comparison. The results of FIGA from Table 5 and Figure 11 have proven to find better solutions in the same amount of time as both MMOEASA and Ombuki's Algorithm. FIGA was able to find the best-known solution for C101 and C201 after every execution; every area recorded was equal to 91.62% for C101 and 94.06% for C201. Figure 12 represents C101's solution and Figure 13 C201. In the other problem instances where results were found, solutions could sometimes challenge the best known but could not be determined as better as one objective was lower, but the other was higher.

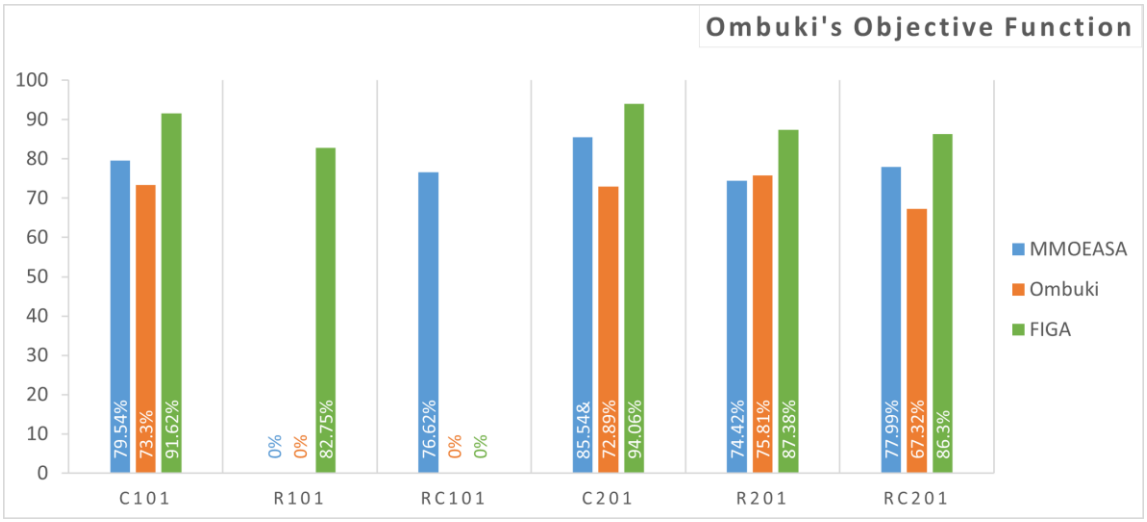


Figure 11: Bar graph representing FIGA's, MMOEASA's and Ombuki's Algorithm's median areas after solving Ombuki's objective function.

FIGA also struggled to solve problem RC101, which is similar to Ombuki's Algorithm. Because FIGA employs a modified edition of Ombuki's operator, the operator is likely the cause of the inability to solve the problem. Another indicator that the crossover operator is the cause is that both Ombuki's Algorithm and FIGA are operator-oriented with an 80% chance of crossover; the algorithms prioritise crossover. Other than fixing the crossover operator, a possible method of avoiding this would be to provide a mutation operator that swaps destinations not based on distance or time windows, unlike any of the mutations offered in FIGA. This could break free from feasibility-focused solutions and possibly find a solution.

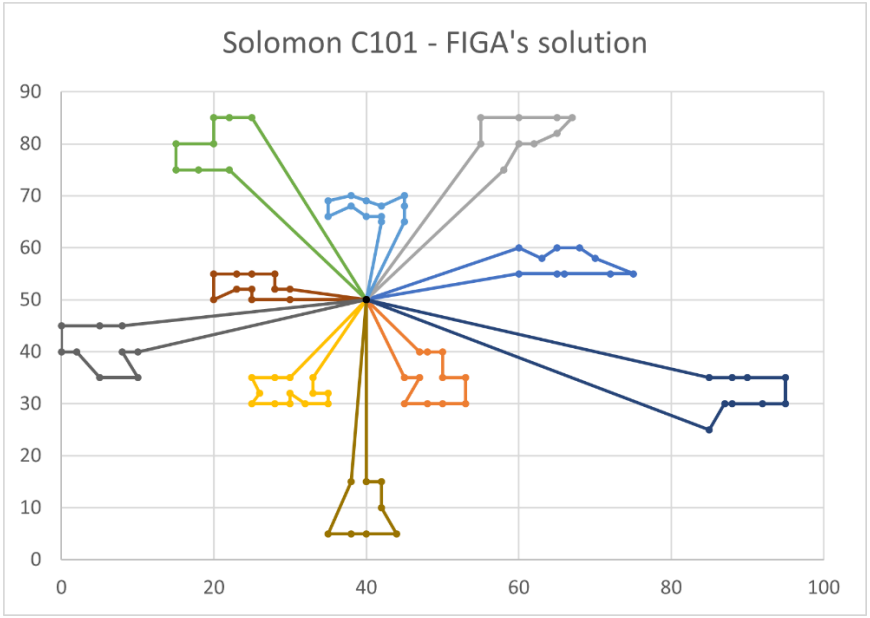


Figure 12: Visualisation of FIGA's solution to Solomon C101.

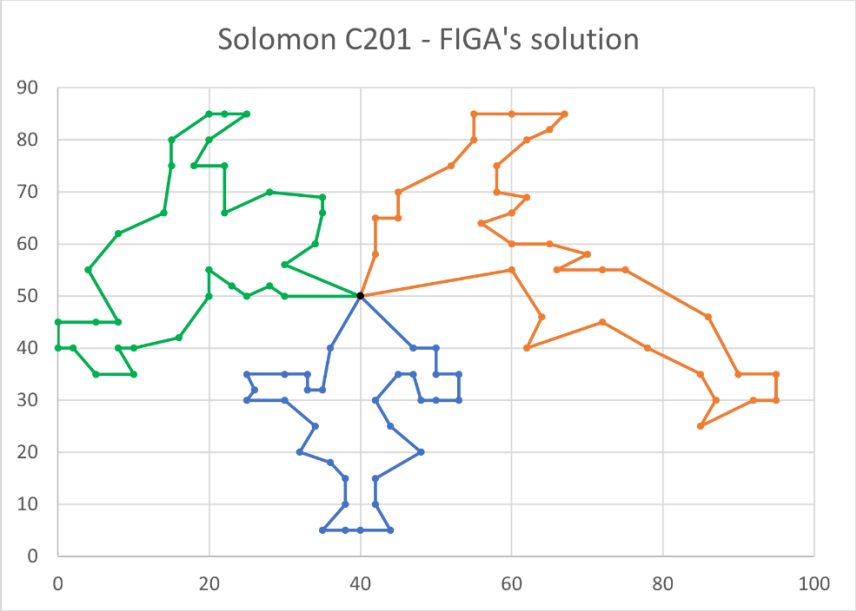


Figure 13: Visualisation of FIGA's solution to Solomon C201.

5.2.5 Best and Worst Areas For Each Algorithm and Objective

The best and worst areas of the acquired results, shown in Table 6 and Table 7, help gain more insight into the performance of the algorithms.

Algorithm	Case	Solomon's Problem Instance – Area Covered					
		C101	R101	RC101	C201	R201	RC201
FIGA	Best	91.62%	82.87%	0%	94.06%	87.59%	86.45%
	Worst	91.62%	82.66%	0%	94.06%	87.37%	86.17%
MMOEASA	Best	79.74%	0%	78.26%	87.18%	79.02%	79.05%
	Worst	78.77%	0%	72.26%	81.56%	73.69%	76.8%
Ombuki	Best	74.71%	0%	0%	85.52%	79.56%	68.68%
	Worst	71.51%	0%	0%	71.21%	73.69%	65.04%

Table 6: Each algorithm's best and worst areas after solving Ombuki's objective function.

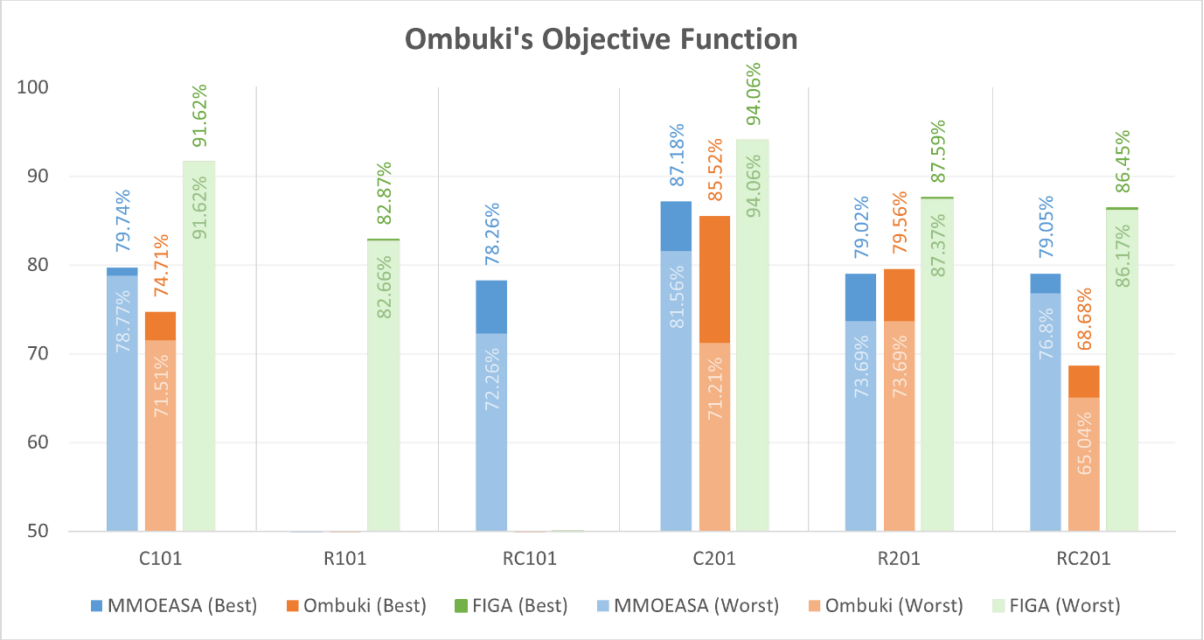


Figure 14: Bar graph representing the difference between each algorithm's best and worst areas to Ombuki's objective function.

Algorithm	Case	Solomon's Problem Instance – Area Covered					
		C101	R101	RC101	C201	R201	RC201
MMOEASA	Best	51.65%	0%	51.28%	72.24%	71.25%	68.39%
	Worst	45.55%	0%	40.09%	64.13%	66.4%	60.91%
Ombuki	Best	44.11%	0%	0%	65.11%	72.23%	66.26%
	Worst	0%	0%	0%	60.37%	67.43%	57.28%

Table 7: MMOEASA's and Ombuki's Algorithm's best and worst areas after solving MMOEASA's objective function.

The bar graphs in Figure 14 and Figure 15 highlight the difference between each algorithm's worst and best areas for each problem and objective function. The light areas of the bars represent the worst areas, and the more vivid areas on top represent how much better the best areas are.

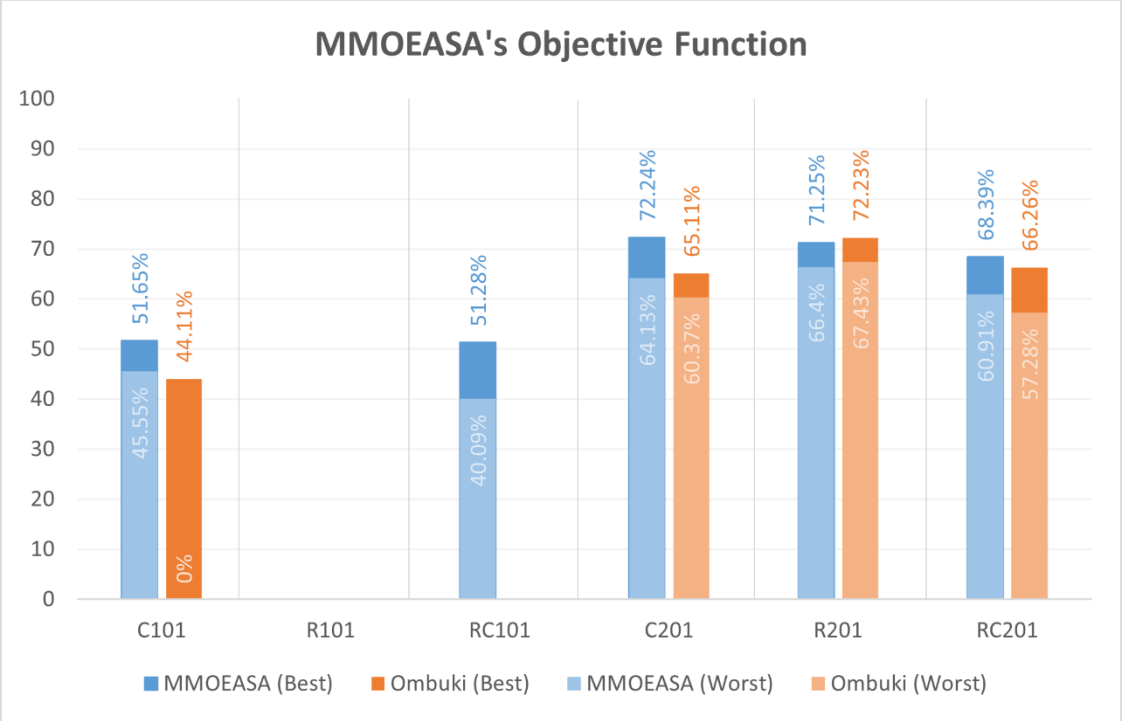


Figure 15: Bar graph representing the difference between MMOEASA's and Ombuki's Algorithm's best and worst areas to MMOEASA's objective function.

5.2.6 Custom Initialising Heuristic (DTWIH)

The function of an initialising heuristic is to create some form of solution to work from. With a time-based termination condition, this stage must finish quickly to begin genetic operations as soon as possible. However, a function of the DTWIH of FIGA is also to attempt the initialisation of a feasible population to stop the GA from having to wait until a feasible solution is generated before making feasible generations. Measurements are taken of how much time each algorithm's initialisation heuristic takes and how many feasible solutions they produce.

To measure the initialisation heuristics' statistics, each experiment was run separately from the previous experiments where the areas were acquired. This is to perform initialisation of equal population size in each algorithm, that size being thirty solutions. The population size needs to be consistent during this measurement to make comparisons fair. When initialisations' execution time is concerned, it would be unfair to Ombuki's Algorithm to expect it to perform 300 initialisations, as it originally does. At the same time, the others only have to do around a tenth of 300.

The initialisation heuristics of MMOEASA and Ombuki are not developed to create feasible solutions from the initialisation stage. Therefore, it is not significant that they do not do so (as seen in Table 8).

Algorithm's Initialisation Heuristic	Average Time Taken (milliseconds)	Solomon's Problem Instance – Avg. Feasible Initialisations					
		C101	R101	RC101	C201	R201	RC201
FIGA	**15.74**	0	0	0	30	28	30
MMOEASA	**24.645**	0	0	0	0	0	0
Ombuki	**31.062**	0	0	0	0	0	0

Table 8: Average time taken and average feasible initialisations of each algorithm's initialisation heuristic.

The DTWIH from FIGA cannot produce feasible solutions to the problem instances with narrow time windows (C101, R101, and RC101 in Table 8). These problem instances are hard to solve for this reason and take the GAs some time to produce a feasible solution, so this is predictable in these instances. A solution to this may be to remove the randomisation (Figure 5 (c.i), (d.i), (e.i), and (f.i)) during the insertion process. This would insert every destination in the sorted order. The drawback of doing so would be that every initialising solution would be identical, likely rendering crossovers ineffective until some mutations have been applied.

Table 8 shows that for each initialisation heuristic, the average time taken is different. In theory, MMOEASA's initialisation should be the fastest as it only generates one solution and makes n copies of it, where n is the population size (thirty in this experiment). It will be slower because of the copying process: performing thirty memory copies in Python is possibly a slower process than performing thirty distinct initialisations. Ombuki's initialisation, on the other hand, has a higher time complexity as it, like FIGA, performs thirty distinct initialisations.

5.3 Proving Results' Significance: Mann-Whitney Test

The results of FIGA, MMOEASA and Ombuki's Algorithm are measured for luck. For the significance test, the Significance Level is set to 1% (0.01), meaning that the

margin for a result to be considered lucky is strict in that any p-value higher than this will show there may be a degree of luck. However, a p-value higher than 1% but not by much does not necessarily determine that the results are guaranteed to be a result of luck. Where results fall below this level, they will be emboldened and underlined within the visualisation. Equal results will be hyphenated as, in these instances, there are no results to test whether they were acquired by luck; for example, where both FIGA and Ombuki's Algorithm could not solve RC101.

For the results of Ombuki's objective function, it is essential to remember that the algorithm with the highest median result from Table 5 is considered to have the highest results. For example, when analysing the significance of comparing MMOEASA's and Ombuki's Algorithm's results, because Ombuki's Algorithm's median area for R201 is higher than MMOEASA's, the Mann-Whitney test will determine the probability of Ombuki's median being higher due to luck. However, because MMOEASA's median area is higher for C101, the Mann-Whitney test for this problem will determine the probability of MMOEASA's median being higher due to luck. Therefore, Analysing the results of Ombuki's Algorithm and MMOEASA for R201 (in Table 5), for example, a p-value above 1% will show that Ombuki's Algorithm's median is probably higher by luck, and vice versa for C101.

Algorithms	p-value of results to different problem instances					
	C101	**R101**	**RC101**	**C201**	**R201**	**RC201**
FIGA and MMOEASA	0.3747%	0.3747%	0.3747%	0.3747%	0.5963%	0.3968%
FIGA and Ombuki	0.3747%	0.3747%	-	0.3747%	0.5963%	0.3968%
MMOEASA and Ombuki	0.3968%	-	0.3747%	**1.587%**	**54.17%**	0.3968%

Table 9: Mann-Whitney test of the areas gathered for each algorithm and problem instance when solving Ombuki's objective function.

According to the test results in Table 9, all areas gathered for FIGA are significant. The p-values represent less than a 1% chance of a FIGA's median being better by luck when its areas are compared to both MMOEASA's and Ombuki's Algorithm's

areas. However, analysing MMOEASA's and Ombuki's areas show that in problems C201 and R201, the algorithm with the highest median possibly achieved so by luck. For R201, there is a 54.17% chance that Ombuki's Algorithm scored higher due to luck.

For MMOEASA's objective function, the algorithms considered the best at each problem instance are those with the highest medians from Table 2. Again, if the p-value is higher than 1% (0.01), the areas and median may be higher by luck.

Algorithms	p-value of results to different problem instances					
	C101	**R101**	**RC101**	**C201**	**R201**	**RC201**
MMOEASA and Ombuki	0.5963%	-	0.3747%	**7.54%**	**84.52%**	**11.11%**

Table 10: Mann-Whitney test of the areas gathered for MMOEASA and Ombuki's Algorithm in each problem instance when solving MMOEASA's objective function.

Again, Table 10 shows that Ombuki's Algorithm's R201 results have a very high probability of being a result of luck. Also, in MMOEASA's objective function, C201 and RC201 are likely to result from luck. Therefore, the only significant results of the experiments conducted on MMOEASA's objective function are with C101 and RC101. The algorithms should be executed more than five times during experimentation to fix this issue by helping ensure results are more significant. Unfortunately, as discussed, further experiments are out of this dissertation's scope of time.

It is also worth noting that because, in theory, MMOEASA's objective function could be difficult to solve, both algorithms may be arriving at very similar results near the optimal solution to that objective function. Therefore, because the areas would be similar in this scenario, the Mann-Whitney test could easily determine the small differences in the algorithms' areas as lucky, as a result.

6 Conclusion

The evaluations made in chapter 5 have proven that the algorithms in question can effectively satisfy the complex CVRPTW's constraints while producing quality solutions. However, the development of FIGA, in particular, proved to be more successful than the project overview interpreted. Not only was a custom GA developed within the scope of the project, but it is also able to produce higher quality solutions than the other two algorithms in the same length of execution time.

The experimentation of algorithms developed by other researchers led to interesting theories about what may and may not be efficient. Based on the research, GAs that contain operators which respect the CVRPTW's constraints would theoretically perform better. The Simulated Annealing process employed by MMOEASA, on the other hand, was thought to be unimportant, as discussed by the rejected ideas. However, the evaluation process disproved the theory as although MMOEASA does not perform crossovers with the hope of maintaining feasibility, it outperformed Ombuki's Algorithm.

Keeping in mind that Ombuki's Algorithm had to be modified due to an anomaly in the original paper, something is missing from the implementation of Ombuki's Algorithm. Therefore, MMOEASA cannot claim to outperform Ombuki's Algorithm. Nonetheless, the research revealed that GAs are effective methods of solving the CVRPTW, and hybrid-heuristic and problem-specific crossovers in GAs further enhance their abilities, as shown by the algorithms' differing performances.

Concluding explicitly on the FIGA research, the theories and approaches taken were successful because the implementation is effective. The GA can arrive at exceptionally high-quality solutions and the best-known solution in some cases. It is expected that the wait time reduction functionality of FIGA is a significant factor in its success. Because significant wait times are detrimental to feasibility, eliminating wait times by using them to visit another destination appears to be effective. Nonetheless, the FIGA is still flawed: advancements can be made on the initialisation heuristic and operators implemented in FIGA. Therefore, improvements and further research are noted for future work.

7 Project Fulfilment

The initial project overview was to discover the most effective evolutionary algorithms written to solve the Capacitated Vehicle Routing Problem with Time Windows and compare them. The first stage of reviewing scientific research discovered that the problem does not have one or two “best” algorithms that can solve the CVRPTW because the problem is much more complex than initially interpreted.

7.1 Evaluation

The algorithms were not chosen because they are the best but because the research revealed they produce high-quality solutions to a likely unsolvable problem. They were also chosen because, while they are both Genetic Algorithms, they are somewhat different in functionality. By comparing a GA-heuristic hybrid and another GA with a genetic crossover mindful of the CVRPTW’s constraints, the results collected from the two algorithms were significant enough to be compared and concluded on what makes their methods efficient. Therefore, both aims of comparing different algorithms and identifying why they are efficient were met.

Implementing FIGA was a difference between the initial objectives and the project outcome. The initial overview did not specify a custom algorithm implementation because developing a custom solution appeared out of the project’s scope. It was out of scope as the time frame of implementing even one GA could not be interpreted before having written one for the first time. Implementation of a custom algorithm and two existing algorithms appeared infeasible with the project's time scope.

The project exceeded expectations because the outcome of implementing researched algorithms was successful, and the implementation and experiments were done with two algorithms written by other researchers and FIGA. FIGA alone also exceeded expectations as it provided better solutions to the problem instances experimented on in the same amount of time as the other two algorithms and, in some instances, found the best-known solution. Although the results are meaningful, the results acquisition stage of the algorithms implemented could have been more thorough. As discussed in the evaluation of the results, each algorithm was only executed five times for each problem instance, with MMOEASA and Ombuki’s Algorithm being executed on each

problem an additional five times (so ten) for each problem instance using MMOEASA's objective function.

The decision to develop FIGA was also made to solidify understanding of the CVRPTW and explore questions and theories that arose during the implementation of the other researchers' algorithms. Because FIGA was implemented, understanding the CVRPTW's scope and complexity was greatly enhanced. During the implementation of the other researchers' algorithms, it became clear why the problem is complex and how to solve NP(-Hard) problems. A concept that was not known during the initial project overview stage.

Overall, decisions made during the project led to meaningful results. However, due to their differing objective functions, the decision to use MMOEASA and Ombuki's Algorithm was a risk. It was likely that the algorithms' crossover and mutation operators or heuristics were developed to solve their original objective functions. Solving one algorithm's objective function with another algorithm could have failed for this reason, and migrating an objective function may have required drastic changes to an algorithm. For example, MMOEASA's Hypervolume-based Metropolis function had to be implemented in Ombuki's Algorithm to experiment with MMOEASA's objective function in Ombuki's Algorithm. If the Hypervolumes were used in any other part of MMOEASA, for example, in the crossover operator, this would mean that MMOEASA's crossover would also need to be applied to Ombuki's Algorithm so that the Hypervolumes for MMOEASA's objective function. Of course, doing so would make research into Ombuki's Algorithm at MMOEASA's objective function insignificant.

Finally, the project satisfied all aims and objectives as the outcome:

- Presents a new algorithm that can satisfy the objective function with high quality, based on an understanding of the problem,
- Expands upon existing research as FIGA is also developed based on theories derived from MMOEASA and Ombuki's Algorithm,
- Implements the algorithms researched and FIGA in a manner that can provide meaningful results for comparison,
- Critically reviews algorithms by identifying the major functions of each algorithm, then represents their effectiveness from the algorithms' results.

7.2 Self Appraisal

This section contains a critical analysis of personal work carried out and decision making; the researcher will be referred to in the first person.

I believe the project was a success as the evaluation stage revealed results more significant than what was hoped. Initially, the project aimed to compare two existing algorithms. However, thanks to prior research and the implementation stage, critical theories of the algorithms' functionality arose. I devoted much time to learning the problem and the function of GAs. As a result, developing FIGA was feasible within the project's scope; implementation went smoothly because I understood how GAs are written, and I knew which theories I wanted to investigate.

I made certain decisions to save time throughout the project, decisions of which did so. The decision to use Python was the most important. While experimentation of the algorithms would be more effective in C or C++ (to perform more operations in less time), Python made writing the algorithms and debugging them much more straightforward. Also, using the existing C source code of MMOEASA's objective function to check solutions' validity saved much time. MMOEASA's checker was used as MMOEASA was the first algorithm implemented, so I ensured its solutions were valid before developing other algorithms. If I committed time finding an external validity checker, I would have to spend additional time formatting one of MMOEASA's solutions so that the external validator could load it, so this was avoided. The decision to use Hypervolumes previously calculated by MMOEASA's researchers was also made to save time. The Hypervolumes are necessary for MMOEASA's acceptance criterion, and as discussed in the Technical Justification, they are time-consuming to calculate. So, pre-existing values were used, despite the disadvantages discussed in the Technical Justification.

A decision that I would change during a revisit to the project regards the algorithms selected for experimentation as those that were chosen are written to solve different objective functions. The differing objective functions were resolved in this instance as a solution was possible by modifying the acceptance criteria. However, modifying the acceptance criterion is a poor solution as it can impact the original intention of an algorithm. Despite this, the algorithms still performed optimally.

Another decision I may change would be to implement a test plan. Unit Testing was not implemented because the algorithms are stochastic, making random changes hard to test. Also, because there are many operators present in each algorithm, testing them individually is tedious because of the randomised effect they have. The most feasible form of Unit Testing would be to ensure that every solution is formatted correctly after every generation; all vehicles should start and end at the depot, and there should be no unvisited/duplicate destinations. I did these tests while debugging the algorithms but did not make a significant, note-worthy plan due to the algorithms' complexity. As previously discussed, the finalisation of the Literature Review was postponed for the same reason. Therefore, ensuring that these conditions were met was an iterative process that occurred after the implementation of each operator. This reinforced my assurance that the algorithms could be implemented within the project's scope and, therefore, no real test plan was developed.

As a whole, the project's time management could have been more concise, but there is justification for why there is no detailed allocation of time to tasks. Initially, I was given credible advice that a comprehensive time visualisation technique, such as a Gantt Chart, would conceivably be incorrect when developed in the project preparation stage and, thus, diverted. Alternatively, I developed a Product Flow Diagram (see Appendix 3) that illustrates the flow of tasks in the project but without time allocated to said tasks. Instead, time management relied on the weekly progress meetings assigning achievable tasks in a coherent quantity. This proved successful due to the project's success: the additional implementation and evaluation of FIGA were completed on top of the original aims.

Post-completing the project showed me that time allocation via a Gantt Chart or other means would have been incorrect and completed retrospectively to represent the time allocations. The first discovery of this was completing the background research: because of the project's time scope, I did not want to waste time researching algorithms that I may have failed to implement. As a result, the Literature Review completion had to be delayed until I was sure I could implement the chosen algorithms, which was unforeseen during the project planning stage.

7.3 Future Work

With the newly developed Genetic Algorithm, additional research could be carried out on the functionality it contains. Notably, it would be interesting to see what effects deterministic methods (with no stochastic elements) could have that focus on eliminating wait times as the FIGA's crossover does. However, as speculated in the Technical Justification, likely, this method could not be used in a deterministic algorithm. Wait times have no immediate signification of the best combination of destinations. It is effective in the GA because after trial and error of replacing long wait times with a destination, the best replacement may be found.

Further details during evaluation would also be meaningful. Further details include the statistics of operators' success rates. For MMOEASA, it would be complicated to meaningfully show how effective Simulated Annealing, Multi-Start, and MO Metropolis perform without simply comparing the non-dominated set's quality, as this dissertation does. On the other hand, it is more straightforward to show the success rate of the crossover and mutation operators by calculating how many dominating solutions they produced before the best-known solution is found. Such information would reveal what functions of the algorithms are efficient. Knowing which functions are effective would allow for further utilisation of their effects in other operators.

As per the purpose of the initialisation heuristic, the results acquired show that, in the case of narrow-time-windowed problem instances, it is unable to fulfil its purpose of initialising the population with feasible solutions. Feasible initialisations are aimed to give the algorithm a head start at meaningful genetic computations by working with already-feasible solutions. Although the wide-time-windowed problem instances were given feasible solutions, because the FIGA struggled to solve the narrow-time-windowed problem RC101, it could be said that feasible initialisations in this instance would also have helped FIGA solve the problem.

The instances that the FIGA could not solve are also an investigation area. It is believed that the FIGA was unsuccessful in the problem instance R101 because it implements functionality from Ombuki's crossover. Because Ombuki's Algorithm is also unable to solve the same problem instance, the crossover may cause this. Therefore, adjustments could be made to the crossover to prove this theory.

References

Bang-Jensen, J., Gutin, G., & Yeo, A. (2004). When the greedy algorithm fails. *Discrete Optimization*, *1*(2). https://doi.org/10.1016/j.disopt.2004.03.007

Baños, R., Ortega, J., Gil, C., Márquez, A. L., & de Toro, F. (2013). A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. *Computers & Industrial Engineering*, *65*(2), 286–296. https://doi.org/10.1016/j.cie.2013.01.007

Bjorndal, M. H., Caprara, A., Cowling, P. I., della Croce, F., Lourenço, H., Malucelli, F., Orman, A. J., Pisinger, D., Rego, C., & Salazar, J. J. (1995). Some thoughts on combinatorial optimisation. *European Journal of Operational Research*, *83*(2). https://doi.org/10.1016/0377-2217(95)00005-B

Bräysy, O., Hasle, G., & Dullaert, W. (2004). A multi-start local search algorithm for the vehicle routing problem with time windows. *European Journal of Operational Research*, *159*(3), 586–605. https://doi.org/10.1016/S0377-2217(03)00435-1

Chang Wook Ahn, & Ramakrishna, R. S. (2003). Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, *7*(4). https://doi.org/10.1109/TEVC.2003.814633

Chen, J., & Chen, S. (2008). Optimization of Vehicle Routing Problem with Load Balancing and Time Windows in Distribution. *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, 1–4. https://doi.org/10.1109/WiCom.2008.1527

Chen, S., & Smith, S. F. (1999). Improving Genetic Algorithms by Search Space Reductions (with Applications to Flow Shop Scheduling). *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, *1*, 135–140.

Cohen, J. (1979). Non-Deterministic Algorithms. *ACM Computing Surveys*, *11*(2). https://doi.org/10.1145/356770.356773

Cook, S. (2000). The P versus NP problem. In *Clay Mathematical Institute; The Millennium Prize Problem*.

Cook, S. A. (1971). The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing - STOC '71*, 151–158. https://doi.org/10.1145/800157.805047

Crowder, H., Johnson, E. L., & Padberg, M. (1983). Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research*, *31*(5). https://doi.org/10.1287/opre.31.5.803

Dantzig, G. B. (1951). Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation (T.C. Koopmans, ed.)* (pp. 359–373). Wiley.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, *6*(1). https://doi.org/10.1287/mnsc.6.1.80

Dempster, M. (1982). *Deterministic and Stochastic Scheduling* (M. A. H. Dempster, J. K. Lenstra, & A. H. G. Rinnooy Kan, Eds.). Springer Netherlands. https://doi.org/10.1007/978-94-009-7801-0

Desaulniers, G., Madsen, O. B. G., & Ropke, S. (2014). Chapter 5: The Vehicle Routing Problem with Time Windows. In *Vehicle Routing*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611973594.ch5

Dorigo, M., Maniezzo, V., & Colorni Alberto. (1991). *Positive Feedback as a Search Strategy*.

Eiben, A. E., Raué, P.-E., & Ruttkay, Zs. (1994). *Genetic algorithms with multi-parent recombination*. https://doi.org/10.1007/3-540-58484-6_252

Figliozzi, M. (2010). Vehicle Routing Problem for Emissions Minimization. *Transportation Research Record: Journal of the Transportation Research Board*, *2197*(1). https://doi.org/10.3141/2197-01

Floyd, R. W. (1967). Nondeterministic Algorithms. *Journal of the ACM*, *14*(4), 636–644. https://doi.org/10.1145/321420.321422

Garey, M. R., Johnson, D. S., & Stockmeyer, L. (1974). Some simplified NP-complete problems. *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing - STOC '74*. https://doi.org/10.1145/800119.803884

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. The MIT Press. https://doi.org/10.7551/mitpress/1090.001.0001

Hoos, H. H., & Stützle, T. (2005). *Stochastic Local Search*. Elsevier. https://doi.org/10.1016/B978-1-55860-872-6.X5016-1

Jozefowiez, N., Semet, F., & Talbi, E.-G. (2008). Multi-objective vehicle routing problems. *European Journal of Operational Research*, *189*(2), 293–309. https://doi.org/10.1016/j.ejor.2007.05.055

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Knuth, D. E. (1974). Postscript about NP-hard problems. *ACM SIGACT News*, *6*(2). https://doi.org/10.1145/1008304.1008305

Kumar, S. N., & Panneerselvam, R. (2012). A Survey on the Vehicle Routing Problem and Its Variants. *Intelligent Information Management*, *04*(03), 66–74. https://doi.org/10.4236/iim.2012.43010

Ladner, R. E. (1975). On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, *22*(1), 155–171. https://doi.org/10.1145/321864.321877

Mann, H. B., & Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, *18*(1), 50–60. https://doi.org/10.1214/aoms/1177730491

Marler, R. T., & Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, *41*(6), 853–862. https://doi.org/10.1007/s00158-009-0460-7

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114

Ombuki, B., Ross, B. J., & Hanshar, F. (2006). Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows. *Applied Intelligence*, *24*(1). https://doi.org/10.1007/s10489-006-6926-z

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving.*

PK, F. A. (1984). What is Artificial Intelligence? *Success Is No Accident. It Is Hard Work, Perseverance, Learning, Studying, Sacrifice and Most of All, Love of What You Are Doing or Learning to Do*, 65–65.

Price, K. v., Rainer, S. M., & Lampinen, J. A. (2006). *Differential evolution: a practical approach to global optimization.* Springer Science & Business Media. https://doi.org/10.1007/3-540-31306-0

Russell, S., & Norvig, P. (2002). *Artificial intelligence: a modern approach.*

Schonlau, M., Welch, W. J., & Jones, D. R. (1998). *Global versus local search in constrained optimization of computer models* (pp. 11–25). https://doi.org/10.1214/lnms/1215456182

Solomon, M. M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*(2), 254–265. https://doi.org/10.1287/opre.35.2.254

Solomon, M. M., & Desrosiers, J. (1988). Survey Paper—Time Window Constrained Routing and Scheduling Problems. *Transportation Science*, *22*(1), 1–13. https://doi.org/10.1287/trsc.22.1.1

Thangiah, S. R., Nygard, K. E., & Juell, P. L. (1991). GIDEON: a genetic algorithm system for vehicle routing with time windows. *[1991] Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application*. https://doi.org/10.1109/CAIA.1991.120888

Toth, P., & Vigo, D. (2001). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics.

Tversky, A., & Kahneman, D. (1973). Availability: A heuristic for judging frequency and probability. *Cognitive Psychology*, *5*(2). https://doi.org/10.1016/0010-0285(73)90033-9

Varian, H. R. (1976). Two problems in the theory of fairness. *Journal of Public Economics*, *5*(3–4), 249–260. https://doi.org/10.1016/0047-2727(76)90018-9

von Neumann, J., & Morgenstern, O. (2007). *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press. https://doi.org/10.1515/9781400829460

Young, P. (1983). Some structural properties of polynomial reducibilities and sets in NP. *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing - STOC '83*, 392–401. https://doi.org/10.1145/800061.808770

Zelany, M. (1974). A concept of compromise solutions and the method of the displaced ideal. *Computers & Operations Research*, *1*(3–4), 479–496. https://doi.org/10.1016/0305-0548(74)90064-1

Zitzler, E., Brockhoff, D., & Thiele, L. (2007). The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration. In *Evolutionary Multi-Criterion Optimization* (Vol. 4403, pp. 862–876). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-70928-2_64

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257–271. https://doi.org/10.1109/4235.797969

Appendix 1 Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Investigating the performance of multiple algorithmic and heuristic solutions to the Capacitated Vehicle Routing Problem with Time Windows

Overview of Project Content and Milestones

To research the Capacitated Vehicle Routing Problem with Time Windows and compare the efficiency of different evolutionary algorithms, heuristics, and possibly hybrids that find solutions to this variant of the problem. The algorithms' solution's statistics will determine an algorithm's efficiency. To assess efficiency, I need an objective function that I'll investigate, such as the cost of the solutions produced by the algorithms (via a combination of fuel required for the solution found and vehicles and drivers) compared to the cost of the best-known solution. The main milestones of the project are to:

1. Select which variant of the Vehicle Routing Problem will be used to compare the algorithms and heuristics.
2. Decide which evolutionary algorithms and heuristics (that can solve the problem selected) will be the most important to compare; which existing methods produce the most efficient results to the objective function I choose to investigate.
3. Build a software application that gives instances of the chosen problem to the solutions selected. The application should output the statistics of the methods used once finished.
4. Collect and visualise the data (statistics) of each evolutionary algorithm and heuristic used when solving the problem.
5. Use the data to critically compare the evolutionary algorithms and heuristics to conclude on their efficiencies.

The Main Deliverable(s):

The most important deliverable is the software application written to solve the Capacitated Vehicle Routing Problem with Time Windows using various algorithms and heuristics. The application should provide the statistics of each algorithm/heuristic used when solving the problem; this will allow me to visualise and compare the results of each algorithm's/heuristic's ability. The application itself may be able to make some comparisons.

The dissertation will contain all of the research collected from the application. The statistics given by the application will be interpreted, represented and discussed here. I will likely use graphs and some other figures to help visualise the data

returned from each solution, along with explanations of what each figure is showing proof of and comparisons to other researchers' findings.

The Target Audience for the Deliverable(s):

People in the service industry who provide delivery services may want a faster system that offers solutions. For example, suppose a company has many deliveries to make and many vehicles to take them. In some cases, a specific algorithm may take a while to develop a solution to the problem. The Capacitated Vehicle Routing Problem with Time Windows' constraints may also make algorithms slower or faster; for example, some algorithms may perform better at accounting for a vehicle's capacity.

Other researchers may be interested in any new findings. Previously undiscovered statistics that prove some algorithms/heuristics are better/worse at solving the specific problem may inspire changes to existing methods of solving the Capacitated Vehicle Routing Problem with Time Windows.

Of course, both of the target audiences mentioned will be interested in an algorithm/heuristic that proves better accuracy to the best solution than those they are currently using.

The Work to be Undertaken:

I must carefully consider which algorithms and heuristics to compare, as many can be applied to solve the problem. I must consult existing literature that proposes methods of solving the problem and determine which are currently the strongest (and, therefore, the most important.) To analyse the algorithms/heuristics, I will, of course, need to successfully write their code in my application, which is another task likely to take some time.

Given that the project investigates the statistics of different evolutionary algorithms and heuristics, I will need to collect data about each algorithm and heuristic used to solve the selected problem. Once I've gathered the statistics of each method used, I will also need to visualise and analyse the data to understand what information is given to critically compare each algorithm's/heuristic's abilities.

It will be wise to test every evolutionary algorithm and heuristic that I implement within the application. I need to confirm that each implemented method of solving the Capacitated Vehicle Routing Problem with Time Windows can do so correctly, as incorrect/buggy implementations are likely to give either bad or no results. Therefore, the debugging process may reveal bugs that take time to fix.

It would also be wise to evaluate the data returned by what I think is a correctly implemented algorithm/heuristic to ensure the values are correct. But, I'll need to be careful as I cannot assume the results returned will be what I predicted, yet the implementations may still be correct.

Additional Information/Knowledge Required:

- I need to learn the problem variant's dataset to know how the application should handle information. Therefore, I also need to understand how the algorithms are going to put the data to use.
- I plan on using a high-level programming language to speed up development, although this will lower the performance of the algorithms. I have considerable experience with Python, so I plan on using it to develop the application.
- The evolutionary algorithms and heuristics used are likely to be intricate; I'll need to take time to understand them and implement them correctly to ensure there are no errors in the results.
- To visualise the data that I get from the algorithms' statistics, I need to figure out the best way to represent it. The visualisation of the data needs to be easy to read and understand, yet also meaningful and comparable. I may need to learn how to use Python or my experience with R to visualise the data.

Information Sources that Provide a Context for the Project:

- The Truck Dispatching Problem - original paper that proposed the Vehicle Routing Problem, by G. B. Dantzig, J. H. Ramser
- Bio-inspired Algorithms for the Vehicle Routing Problem - by F. B. Pereira, J. Tavares
- Lin-Kernighan Heuristic - Wikipedia
- Evolutionary Algorithms for the Vehicle Routing Problem with Time Windows - by O. Bräysy, W. Dullaert, M. Gendreau
- Vehicle Routing Problem Instances - Networking and Emerging Optimisation

The Importance of the Project:

Comparing the different methods of solving the Capacitated Vehicle Routing Problem with Time Windows may reveal that some evolutionary algorithms or heuristics are better than others. A solution to the problem variant will be considered better if it produces a higher quality solution in terms of the total cost taken required to perform it (after accounting for the number of drivers, vehicles, and fuel required).

The Key Challenge(s) to be Overcome:

- Finding the most important methods of solving the Capacitated Vehicle Routing Problem with Time Windows is essential. It will lay the project's path, so I need to use the most meaningful algorithms and heuristics for my study to make comparing them worthwhile.
- The quality of the evolutionary algorithms and heuristics implemented must be high. As previously mentioned, bugs may give inadequate data, which would cause my results to be incorrect. Solutions will need to be implemented carefully and tested afterwards.

- Visualising the data so I can compare it must also be done with care. An incorrectly drawn graph will cause misinterpretation of the actual results, causing my conclusion to be incorrect.

Appendix 2 Second Formal Review Output

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: COURTNEY, Taylor

Supervisor: PAECHTER, Ben

Second Marker: CHERUBIN, Stefano

Date of Meeting: 15 November 2021

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

Please comment on the progress made so far

Background research & literature review to be ready in 3 weeks.

Solution implementation started 2 weeks ago.

Final solution design scheduled to start within 12-15 weeks.

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.

Compare different methods of solving a constrained delivery problem.

Ideal achievement is to reproduce results from scientific literature.

Possibility of devising a new solution.

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

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

Product flow diagram designed at early stages.

Recommendation is to include any kind of time management system.

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

If yes then please comment otherwise write down your suggestions.

Optimization of cost function as obtained by solutions provided by several algorithms.

Cost function include parameters like road distance, and number of vehicles.

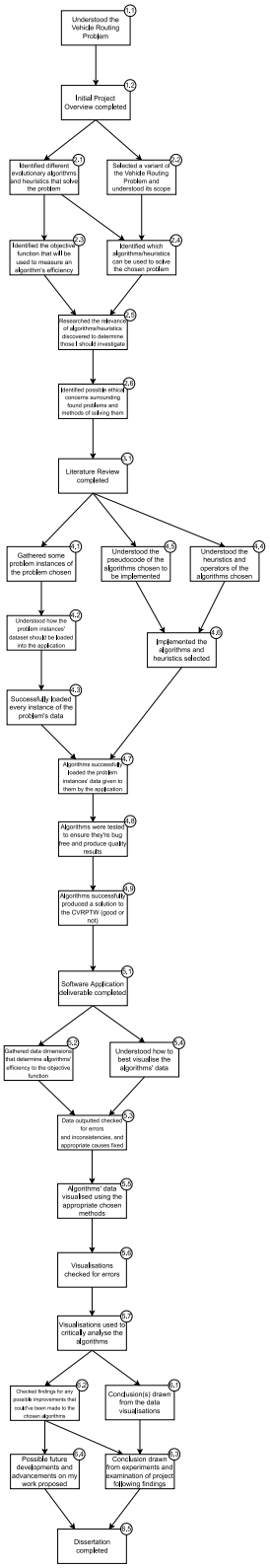
Any other recommendations as to the future direction of the project

Non critical: improve management of time.

Signatures: Supervisor Ben PAECHTER Second Marker Stefano Cherubin

Student Taylor Courtney

Appendix 3 Diary Sheets (or other project management evidence)



EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 14/09/2021 **Last diary date: N/A**

Progress:

- Is it possible to solve multiple variants of the VRP with the same algorithm, or is there too big a difference between different variants?
- Would it be a better idea to compare Genetic Programming with Evolutionary Algorithms or simply compare Evolutionary Algorithms and heuristics? Also, the 4th paper listed in your IPO mentions Hybrids, a combination of EAs and heuristics: what about these?
- Decided to use Python as the primary programming language
- Decided to compare Evolutionary Algorithms and heuristics (and possibly Hybrids?)

Objectives:

- Investigate which problem to solve and which solutions to use (EAs/Heuristics/Hybrids, but remember Ben suggested to consider whether or not implementing three different types of solutions may take too much time to understand and implement, as opposed to two)
- Could start to look into the literature: depends on how much time is available after looking into problem instances and their solutions
- Get advice from Pritam on how to complete a Product Flow Diagram

Supervisor's Comments:

- Stick to one type of problem and focus on one that relates to the real world as it is. For example, time windows are probably more important now
- Keep track of every decision you make and use this documentation in your dissertation to weigh the pros and cons of the decisions you've made; such as why you've chosen to use Python
    - "Why compare evolutionary algorithms and heuristics?" Me: "Heuristics may be faster than AI as heuristics are written specifically to solve this type of problem"
- What's going to be done for next week? Me "Investigate which problem to solve and which algorithms and heuristics to use"
- Chose to use Python as the programming language; dissertation could explain why not use C++ for the reasons Taylor suggested ("Although C++ is fast, development is slow as it's a tricky language")
- Ben confirmed you could look into Hybrid algorithms as well, but you need to think about the time frame of implementing all the different algorithms you'll use

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 21/09/2021 **Last diary date: 14/09/2021**

Progress:

- Clarify:
  - What are the variables that determine the efficiency of a method used to solve the problem?
  - Would it be best to visualise the data returned using R or Python? (Likely to be personal preference, but one might be faster; R is likely easier to use, but Python could automatically visualise it as soon as the problem is solved)
- Ben suggested using the most realistic problem; which is most realistic: "Capacitated VRP with Pick-ups, Deliveries and Time Windows" or just "Capacitated VRP with Time Windows"? (As customers tend to use post boxes to return items as opposed to driver "Pick-ups")
- Algorithm/heuristics discovered so far:
  - NSGA-II (Genetic Algorithm)
  - Lin-Kernighan Heuristic
  - HMEDA (Hybrid) - paper discovered proved this to be better than NSGA-II
  - Figliozzi Heuristic - HMEDA found this one to be a competitor
  - The GRASP Heuristic appears to be popular (and Figliozzi utilises it), but its original paper is inaccessible via Napier (GRASP x ELS Hybrid)

Objectives:

I was unwell this week. Ben was unable to reschedule the meeting, due to his schedule, so we agreed this meeting was cancelled.

Follow on from last week's incomplete objective; look into literature about existing solutions to your chosen Vehicle Routing Problem (Capacitated VRP with Time Windows.)

Supervisor's Comments:

When we agreed to cancel the meeting (see the Objectives section as to why), Ben suggested I continue working for next week as last week's objectives were pretty broad.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 28/09/2021 **Last diary date: 21/09/2021**

Progress:

- Decided on solving the Capacitated Vehicle Routing Problem with Time Windows
- More algorithms/heuristics discovered:
  - Multi-Objective Genetic Algorithm - "Minimizing the number of vehicles affects vehicle and labour costs while minimizing distance affects time and fuel resources. Therefore, the VRPTW is intrinsically a MOP (Multi-objective Optimization Problem) in nature…"
  - (Lin-Kernighan competitor?) Optimization by Simulated Annealing: an Experimental Evaluation
  - Heuristic methods - 2001, analyses 2-interchange, simulated annealing, tabu search, and genetic algorithms
- "... a local search method is almost mandatory to achieve results of high quality." (A. Abraham, p. 387)
  - From the same source, p. 393: "... 2-Opt and λ-interchange (two well-known local search methods for VRP) …"
  - From the second algorithm above's paper: "A combinatorial optimization problem can be specified by identifying a set of *solutions* together with a *cost function* that assigns a numeric value to each solution. An *optimal* solution is a solution with the minimum possible cost (there may be more than one such solution). Given an arbitrary solution to such a problem, local optimization attempts to improve on that solution by a series of incremental, local changes."
- Began dissertation introduction;
  - Made an attempt at giving the project background. Explaining: the VRP and why I chose the CVRPTW, NP-hardness, and how to determine a solution's validity.
  - Should the introductions have subsections? For example, 1.1 Background, 1.2 Aims and Objectives, …

Post-meeting (amendments made after the meeting occurred; noted here instead of in the progress notes for next week):

- We discussed the idea of an objective function. It's important to know what you're aiming to analyse from the experiments as, Ben pointed out, there are multiple ways in which the solutions you get can be interpreted.
- Throught this week's research, I found an objective function that appears to be common: the average cost of an algorithm's solutions. Existing research papers derive the cost from aspects such as the total amount of drivers required for a solution, and their vehicle and fuel requirements.

Objectives:

- Submit IPO and let your second marker know that it's submitted

- Look into papers that explore the same problem and objective function as you (also, make sure you've decided on an objective function). Based on Ben's comments, this will help knowing if an algorithm is worthwhile looking into as researcher's "improvements" on algorithms that have very few citations are likely to not be worthwhile
- Try to decide which algorithms you're going to use by the next meeting in order to make some sort of start next week
- Also, decide which set of problem instances to use (Solomon's appears to be popular)

Supervisor's Comments:

- Ben's IPO review:
    - Don't discuss the space and time complexity of algorithms as they're not important.
    - The point should be to analyse one measurement of success (objective function - more on this below), one of (depending on what a customer would want the solution to do, Ben suggested):
        - How good the solution is
        - How many solutions it can produce
        - How fast solutions are produced
        - If algorithms can get a solution in a certain time
        - …
- It's good to use the same problem instances that other researchers have used to get an exact comparison. Also:
    - Look at the **objective function** that other researchers have used; an objective function is a formula that determines success. For example, will you add a penalty for every time window that is missed? Will you take into account the amount of fuel used?
    - Ben recommended looking into existing objective functions as, unless you're interested in getting a specific finding, creating a new one yourself would be unnecessary extra work
    - Papers that explore the same objective function as the one you're looking into will save a lot of time as they're confirmed to have the same qualities as what you're looking for (and already have some figures that you can use, which saves time as well)
- Implementing and researching evolutionary algorithms is difficult enough, but doing so for things like Multi-Objective GAs is even more difficult.
- Ben mentioned that there's an existing program that takes a solution that's been produced by an/your algorithm and analyses the score of the solution you've acquired to that problem instance. He suggested that using this may help save time in analysing algorithms' success.
    - Programs that do this may have different objective functions from what you've gone with. If this is the case, you may need to ask around to see if any of the universities' researchers in this area know of any (Ben's suggestion).

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 28/09/2021 **Last diary date: 05/10/2021**

Progress:

- Deciding on algorithms (that I've discovered) via their objective functions, and what those functions are:
  - To minimize the number of vehicles and each route's length:
    - (HMEDA) A hybrid estimation of distribution algorithm for the vehicle routing problem with time windows - ScienceDirect
    - (VRPTW as a Multi-Objective Optimisation Problem) Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows | SpringerLink
    - (Local Neighbourhood Search heuristic) Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems
    - (Figliozzi heuristic) Vehicle Routing Problem for Emissions Minimization - as the name suggests, this one prioritises emissions minimization
    - (MMOEASA) A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows - ScienceDirect - "a Pareto-based multi-objective evolutionary algorithm … using simulated annealing …"
  - To minimize the penalty of routes that violate constraints and their cost:
    - (Lin-Kernighan for VRP) An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems
  - To minimize route lengths and route imbalances (imbalances in the distances travelled by vehicles and their loads):
    - (Simulated Annealing) A Simulated Annealing-based parallel multi-objective approach to vehicle routing problems with time windows - ScienceDirect
- During this week, I emailed Ben about:
  - How I considered adjusting research focus from comparing multiple evolutionary algorithms and heuristics to comparing evolutionary algorithms with genetic programming
  - Using old datasets and research as they're relatively old as it was a concern of mine (as I was thinking some of this research may no longer be relevant and I couldn't tell if it was or wasn't)
  - Clarifying what he meant when he briefly mentioned implementing your own algorithm (in the last meeting, this wasn't noted because it was so briefly mentioned); is it ok to continue with your plan of comparing existing CVRPTW solutions or is it better to create your own?
- Decided on Solomon's datasets to test the CVRPTW solutions
- I'm sure I've decided on my objective function being the first from the above list: "To minimize the number of vehicles and each route's length".
  - (Retrospect comment, made on 24/10/21) what I also meant by this was using all 5 of the algorithms from that objective function, but Ben later recommended

sticking to 2-3 as 5 will likely be too many to try. Following this, I planned on using HMEDA, MMOEASA and the Multi-Objective GA - this changed on 24/10/21.

Objectives:

- Begin literature review
- Try to begin some coding as well, as per Ben's reasoning for starting both now
    - You'll need to decide on which algorithms you're going to use for this
    - Get a hold of datasets, understand how you're going to read them in and how your algorithms will work and use the datasets provided
    - Look into the possibility of libraries that contain the algorithms you choose

Supervisor's Comments:

- Don't worry about datasets/research being old as it may just mean they're good enough that researchers have struggled to improve on them (metaheuristics.org is a site that does some comparisons of older methods)
- If you're researching and comparing solutions to the problem, you may find yourself some ways of improving methods along the way, but comparing solutions alone is fine (as you've specified from the beginning, if you find you want to improve a solution then it's not necessary that your initial aims need to be changed for this)
- Genetic programming is likely to be too complicated to try and implement for this kind of problem (at an undergraduate level), what you're researching is certainly fine
- You're probably at the stage where you should start some writing and probably even writing some code
    - Ben suggested that it's probably a good idea to do both as if you do the review then find the code you've reviewed doesn't do well, then you have a problem
    - Because you've looked at what problems you're going to solve and what evaluations to make, you're likely in a good place to start
- The five algorithms that you've found for your objective function may be too many:
    - You could set a goal of trying to implement 3
    - Then later, you may not make that goal and drop to two, or surpass that goal and implement all five
- "I'm not quite sure where to start with the literature review…" - Taylor
    - Could begin by discussing the CVRPTW, then discuss the datasets you're going to use, the objective function you're planning on solving, the algorithms/heuristics you're going to use (though remember what a literature review is: you're giving the reader enough knowledge to know as much as you do about the problem and solutions before they start reading your investigation further; e.g. you could explain what evolutionary algorithms are in the LR)
- Will you use libraries to implement the algorithms you've chosen (if possible), or will you write them yourself?

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 12/10/2021 **Last diary date: 05/10/2021**

Progress:

- Began literature review: drafted CVRPTW explanation (literature review section one):
  - What the problem entails;
  - Why it's difficult to reach an optimal solution;
  - Why a traditional search (linear) isn't feasible - trying to lead onto NP-hardness literature section from here
- It has been difficult to make progress this week due to having been given new coursework from both of my other modules
- I'm pretty sure libraries that provide the algorithms you've chosen are going to be out of the question as it's unlikely that any libraries will have them

Objectives:

- Continue the literature review
- As per Ben's suggestion, try your hardest to write some code
  - At this stage, it's probably best to get a dataset and try loading it in

Supervisor's Comments:

- Try to organise your time so that you don't forget your project, try to do something for your project every week
- You're likely slightly below the average progress made at this point, judging by your progress from previous weeks
- Try not to go into too much detail discussing NP-hardness as it's a rather complicated topic; and although a problem type may be NP-hard, it may not take long to solve, for example, if you had one customer and one vehicle
- Ben would like you to try some coding ASAP to figure out whether you can implement what you're planning to or not
  - Ben's recommendation was to do whichever, either the literature review or coding, you feel most up to doing at any moment you want to do some work
  - You may find that you're tired of writing reports for other coursework and want to do some code or vice versa
- It's certainly worthwhile looking into libraries with your chosen algorithms as they'll save time
  - Although, also consider whether or not any libraries found will allow you to parameterise them (condition them just like they were conditioned in the research papers you've found) to get the same results those researchers got
- The literature review soft-deadline in week 9 is still feasible:
  - The main focus of the literature review should not be the length of it but the quality and whether you've discussed everything that's relevant to your project
  - You'll likely find that, after you've discussed everything you want to discuss, that you've written a lot more than you thought you would

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 19/10/2021 **Last diary date: 12/10/2021**

Progress:

Code progress:

- Understood how Solomon's dataset is structured
- Implemented a file reader that loads the datasets specified by the command line arguments;
    1. Amount of customers - can be either '25', '50' or '100'
    2. Type of problem - can be either 'C', 'R', or 'RC'
    3. Problem set - can be either '1' or '2'
    - An example command is: "main.py 25 RC 2"
    - If I'm correct in my understanding of Solomon's dataset then because there are multiple variants of the datasets that fall into each of the combinations above, I can load each dataset (usually 7 or 8) and solve them all in one run?
- Created classes that can be used to store the Problem Instances and Destinations (customers and the depot)

Literature review progress:

- (Possibly) finished the CVRPTW definition section (literature review section 1)
    - Added a brief explanation as to why the problem is NP-hard
- Began literature review section 2: NP-Hardness
    - Wrote an explanation (following my understanding) of what Non-deterministic Polynomial-time is and how the VRP is such a problem
    - Planned out how to explain why the VRP is NP-hard and not NP or NP-complete

Objectives:

Literature review:

- Discuss the algorithms you've decided to use (as per Ben's suggestion - the last point in the Supervisor's Comments)
    - You'll likely need to discuss the foundations of the algorithms first; for example, if an algorithm you're reviewing is a Genetic Algorithm (GA), you'll need to cover what AI is, then what Evolutionary Algorithms are and then go on to explain GAs
    - If you get to review your chosen algorithms, you could show the algorithm's pseudocode from the research papers you've found
        - You need to explain what the algorithm does; pseudocode, with an explanation of it, can help with this

As far as the code for this week is concerned, Ben suggested writing most of the literature review first to help you understand what the algorithms are doing; you're now at the stage where you need to write the algorithms, so do the literature review part of them first

Supervisor's Comments:

- Most people tend to load only one file at a time when reading so that you can test using a specific dataset, but the way you've done it is ok;
    - It doesn't really matter which way you're going to load it, as long as you get the evaluations you need
    - Taylor: "it may be easier to do it this way, rather than individually entering each dataset into the command line and keeping track of which ones I've tested."
- Ben's explanation of NP-hardness was similar to yours; "there are far too many combinations of data to examine, so you need to basically sample them."
- Next, for the literature review, look at papers that solve the problem you're solving
    - In particular, look at ones that solve Solomon's instances and see if you can maybe improve on them in any way
    - They don't need to evaluate Solomon's solutions in the papers, but it would be helpful; you can try to replicate the results they got and, if you don't, you probably have a bug (whether it's with the parameters or your code)
    - It'll speed up the testing of your implementations to have papers that use Solomon's instances
- It's a good idea to do lots of testing as it's hard to notice if an algorithm isn't behaving correctly
- You don't need to discuss P versus NP because A: it's easy to get wrong and B: it's not really relevant to your dissertation
- Discuss every decision that you've made as it helps to give your dissertation meaning; dissertations need to discuss decisions to show professionalism and understanding of the context

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 26/10/2021 **Last diary date: 19/10/2021**

Progress:

- I've briefly considered using my findings to also compare them with some of the strongest existing methods (using the data of other researchers); these methods may be old, and the algorithms I'm planning on using are more recent as I would like to compare algorithms with newer advancements
    - Nevermind; this won't be possible as there are issues surrounding it, such as:
        - My PC doesn't have the same specs as the other researchers.
        - Other researchers likely didn't take the same measurements as I will, or that each other did.
- After reviewing the algorithms I previously selected, from those that I'd found in the week 28/09/21-05/10/21, I discovered that:
    - HMEDA might not be worth an investigation; I need to look into its relevance compared to other techniques, keep in mind it's new and won't have many citations. However, it appears to use multiple objective functions which may make it hard to know if it's good at solving my objective function.
    - There isn't much diversity between the other two algorithms; they're both hybrid multi-objective algorithms. Maybe consider investigating an Ant Colony Optimisation model or Tabu Search.
    - Other possible algorithms found:
        - An ant colony optimization model: The period vehicle routing problem with time windows - the problem with this one is that it solves the PVRPTW, not the VRPTW.
        - A parallel iterated tabu search heuristic for vehicle routing problems
        - (Yet to find more…)
- For the literature review, I:
    - Completed the NP-hardness section.
    - Possibly completed the Evolutionary Algorithms and Heuristics definition section.
    - Discovered that there are problems with the three algorithms I chose to experiment with a few weeks ago:
        - HMEDA is built to account for more variables in the objective function than I will (8 to be exact).
        - The other two are both multi-objective GAs; it'd be good to have a bit of a range of solutions, so I can pick one of them and discard another.

Objectives:

- Make your research more focused and figure out what you're going to implement; don't keep trying to explore new areas of research
    - Once you've done so, you can start discussing the areas discovered in your literature review
    - If you find any other areas of research that you think is worth mentioning, you can still do so, but don't get caught up in them

- Implement something;
    - Make an attempt at implementing one of the algorithms you've chosen
    - This is to ensure that you can actually implement an algorithm
    - It doesn't have to be a completely successful implementation and ready for next week, but, of course, that would be ideal

Supervisor's Comments:

- Most algorithms you find are likely to be multi-objective anyway, so the two you've found are likely to be suitable as they may use different methods from each other
- Algorithms that are somewhat similar will likely be easier to implement;
    - There's going to be a lot of understanding and implementation, so trying to implement too much may be tough
    - Ben recommended starting writing code also, to get a feeling of how the implementation is going to work
    - He also recommended trying to implement at least one algorithm
    - The next point is to focus on whether you are actually able to implement a solution to the problem, nevermind a very good one
- It doesn't really matter if you implement solutions that are new or old;
    - Newer versions of old algorithms will be more efficient at getting solutions as the only reason people publish them is that they are improvements
- If you investigate algorithms that are similar, it may be easier to come up with your own solution as you'll have a better understanding of what makes them good
- Ben confirmed it is difficult to compare algorithms using others' data and is probably not a good idea for the reasons you've listed
- If others provide you with source code, it is perfectly fine to copy it;
    - You're not marked on the work that others have done, you're marked on yours
    - So the graphs that you create from their finding, or your own algorithm, is what you'll be marked for
    - It's likely a good idea to make use of others' code to make it easier on yourself when implementing the algorithms
- The paper on a Parallelised Tabu Search Algorithm that you found;
    - It could be parallelised to either simply reduce the time complexity of the algorithm or to create more time to examine more solutions
    - Parallelisation is a whole other topic that would take a while to research; you could maybe write about it but don't try to implement it
- You should, at this point, be focusing your work and research down to manageable topics rather than spreading it out and discovering newer topics; otherwise, you're going to struggle to make the project manageable

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 02/11/2021 **Last diary date: 26/10/2021**

Progress:

- Updated choice of algorithms:
  - (MMOEASA) A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows - ScienceDirect
  - (Ombuki's Algorithm) Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows | SpringerLink
- Progressed with the literature review:
  - Discussed other methods of finding solutions, but that I don't plan on researching:
    - Ant Colony Optimisation
    - Branch-and-Cut algorithms
  - Planned on discussing, in my next section (where I write about the solutions I'm going to use)
    - Genetic Algorithms (already discussed, but I'll explain further how they solve the VRP)
    - Simulated annealing
    - Multi-objective (+ local search)
    - …
  - I'll also need to discuss evaluation strategies. Many papers seem to have implemented a Pareto-based ranking system alongside their GAs fitness function, so this is likely to be the main topic of discussion
- I tried to code MMOEASA and ran into some problems:
  - The pseudocode provided for it is pretty high level and, therefore, it is easy to make mistakes in understanding/implementing it
  - Some of the input values the researchers used are unspecified, so if I cannot find a way of getting/calculating them, I'll struggle to implement the algorithm and may have to choose another
  - I ran into Ben on campus and he recommended that I contact the researchers, who wrote the paper, for advice; I contacted two of them and one's email address is no longer registered at the institute listed on the paper, and the other didn't reply

Objectives:

- In your literature review, begin the next section where you write about functions of the algorithms that you're going to use; likely:
  - Simulated Annealing
  - Multi-Objective
  - Multi-Start
  - Local Search
- Could maybe start the evaluation strategies? (Pareto-based and Weighted Sum)

- Try to get in touch with the MMOEASA developers again to see if they can offer any directions as to how they initialised their algorithm's parameters

Supervisor's Comments:

- You can Google the researcher who changed institute to find where they are now and contact them that way
    - The site should have their previous papers listed; so you'll know it's them
- You should discuss the Pareto-based ranking system
    - Read about it and if it doesn't make sense, Ben offered to help with the understanding of it
    - I asked if I could also discuss other ranking systems to (sort of) compare them - Ben said it is ok as there is only really two: Pareto and a Weighted Sum system
- Ben's explanation of Local Search: "We go from one point in the solution, that we're currently at, and make a small change in the current situation to try and improve the solution we've already got"
    - "The search you make may be the functions of whichever algorithm you implement. So a Genetic Algorithm may perform a crossover to get a better solution"
    - I asked "If we had a hybrid Genetic Algorithm with heuristics, then may the Local Search employ both of the properties of a GA and a heuristic"
    - Ben confirmed that "Yes, it can"
- If the researchers who wrote the MMOEASA paper don't respond, you could look into simulated annealing yourself to maybe understand what they did or do experiments yourself to work the numbers out

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 09/11/2021 **Last diary date: 02/11/2021**

Progress:

- Found the active email address the other MMOEASA developer, who I couldn't get a hold of last week, and asked about his algorithm
    - To no avail; I didn't get a response
- Literature Review progress:
    - Researched evaluation strategies of the algorithms, as per my thoughts from papers I've read that mention Pareto-based evaluation
    - Completed a new LR section, "Evaluation Strategies" which highlights two main strategies: Pareto and, as per Ben's suggestion, Weighted Sum
    - I decided not to begin research for the final section yet; the section where I review the algorithms I'm going to experiment with
        - The reason is that if I fail to implement an algorithm after I've completed its section in the LR and have to find a new algorithm, then that LR section will be deleted
        - Studying and reviewing the algorithms I want to use will also take time to understand and write about as they utilise complex heuristics, such as Simulated Annealing, so I want to make sure I'm definitely going to use them before researching them extensively
- Code progress
    - I tried again to implement MMOEASA further;
        - I believe I was able to use most of the pseudocode successfully, although there are a couple of lines towards the end I struggled with (I believe they were lines 16 and 17)
        - I've had a look at the additional functions which the algorithm uses, described in the paper, and I believe this algorithm is going to take quite a bit of time to implement and debug, even if I knew the input parameters they were using, so I may need to find a way to either get the value the researchers used
            - For example, the Genetic Algorithm's Mutation function utilises 10 different operators for mutation which will take a while to implement
- I realise it's a bit of a loop that I'm caught in because I'm not researching the necessary topics too much as it may waste time if my implementation is unsuccessful and I have to change algorithm, but that may be exactly what's holding me back as I don't have the detailed understanding that I need to have to implement MMOEASA

Objectives:

- Continue coding and, if necessary, research the relevant topics in more detail to try to understand the input parameters of MMOEASA and so you can have a better understanding of how to implement the additional functions, such as Calculate_cooling()

- I tried not to study the topics too much for now in case I cannot implement the algorithms that they're relevant to (such as Simulated Annealing; read this week's progress)

Supervisor's Comments:

- Considering how large the MMOEASA algorithm is (I mentioned to Ben that the Mutation function has been implemented using 10 different mutation operators);
  - If one operator is too difficult to implement, you can maybe skip it, but it's up to you
- Remember that you're trying to replicate other researchers' results, but this means a couple of things:
  - If there is an anomaly/inconsistency in their paper, you can write about that in your dissertation then say you did your best to work around it
  - The papers your copying from may not be perfect themselves, and may even have bugs that turned out to work in their paper's evaluation anyway
- I asked Ben if it'd be a good idea, if I can't implement MMOEASA, to switch algorithms at this point, while I still have time
  - He recommended continuing with MMOEASA because you'd likely be able to at least implement something and get some results from it
- Ben offered to try and get a hold of one of the MMOEASA developers to get the advice I've needed the past couple of weeks

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 16/11/2021 **Last diary date: 09/11/2021**

Progress:

- I got an email response from the main MMOEASA developer who gave me a lot of information about how the algorithm was implemented
    - Therefore, I should be able to go ahead with implementing MMOEASA
    - He explained how their specialised Simulated Annealing heuristic works and their Pareto-optimisation technique
    - He also explained the input parameters of the algorithm; the first hurdle I encountered when implementing the algorithm that I noted in week 02/11/2021
- Literature Review progress:
    - I began to explain MMOEASA and the heuristics it uses but realised there were a couple of topics I had missed in the LR prior to this stage;
    - MMOEASA uses local search methods and I hadn't explained what they are and why they're helpful, especially for the VRP
    - Multi-Objective ranking systems could've been made clearer in the Evaluation Strategies
- Code progress:
    - With the information given by the MMOEASA developer, I believe I have made progress implementing the Simulated Annealing heuristic
    - The ranking system that the algorithm uses (MO_Metropolis) was also explained; he showed how the Metropolis calculation is performed in a multi-objective case with multiple solutions

Objectives:

- The main objective is to get a result and verify it is ok
    - You can either find a tool to evaluate it for you or do it by hand yourself
    - Maybe also make a problem instance that only has 2/3/4 customers to make it easier for you to evaluate

Supervisor's Comments:

- Try to get an algorithm that will at least produce a result, before implementing everything perfectly
    - You'll likely find that solutions produced at this stage are from a buggy algorithm which you will need to fix, so finding bugs now would be useful
- Is there an application that evaluates results, against the objective function, for you?
    - Raul may have used one for this, but if there isn't a community-built one that evaluates results for you then you could maybe build one yourself
    - This is to ensure that your program isn't giving false results and the solutions you're getting are actually good results, and may even save you time in having to evaluate the results yourself

        - For example, how are you going to measure the total distance of the routes? An existing program may be able to work this out for you
        - There may also be bugs in your evaluation function, for example, your MO_Metropolis function implementation, that doesn't evaluate the results correctly and shows that you're getting a better result than you are
            - Ben's example was if you accidentally evaluate the distance of *N - 1* routes instead of *N*
- It'd be a good idea to see if the implementation of MMOEASA you were sent contains its own objective function evaluation so you can compare yours to theirs (to try to find any bugs in yours)
- Ben's keen for you to get a result and validate it and be confident your algorithm implementation works
    - It'd be sensible to do this before implementing a lot of additional operators; for example, only using one of the 9 MMOEASA mutation operators

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 23/11/2021 **Last diary date: 16/11/2021**

Progress:

- I have run into another problem when implementing MMOEASA - it uses a figure named Hypervolume, which I haven't heard of, and the implementation I was sent by an MMOEASA developer doesn't contain "fronts" (as they're called) to calculate Hypervolumes
    - I may need to ask the MMOEASA developer for more info
    - I decided that for now, while I figure out what to do about this, I can use the hard-coded values that he used to try and get a result
        - There's a hard-coded value for each problem instance that the developer used and they're all assigned at runtime based on the name of the text file (problem instance file) that is loaded
        - If I am to follow this implementation then I will be restricted to only using the same problem instance files the MMOEASA developer used
- Code progress - I've potentially implemented (awaiting first execution attempt to confirm):
    - The Customer Random Reallocation mutation operator, following Raúl's implementation
        - The mutation operators require additional operators; for example, Customer Random Reallocation requires functions that shift all the destinations for a vehicle either to the left or the right, depending on the index of the destination that is modified by the mutation operator
    - The MO (Multi-Objective) Metropolis function, following Raúl's implementation
    - The crossover operator and its additional functions
    - The Hypervolume calculation, based on the dataset loaded; I previously mentioned that I'm unsure of what Hypervolumes are, but later in the week:
        - I contacted the MMOEASA developer who explained Hypervolumes; he mentioned that they're estimations of the objective functions
        - Therefore, I managed to implement calculations of the exact Hypervolume values for the functions I'm going to use, but I need to confirm whether exact values are better or worse than generalised estimations

Objectives:

- My MMOEASA is ready for a test run, but there are with Python's imports that I need to fix first
    - It's likely that I will run into more errors and bugs once I've fixed this, therefore, I should continue fixing them and then try again to get a result

Supervisor's Comments:

Ben was unable to attend today's meeting

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 30/11/2021 **Last diary date: 23/11/2021**

Progress:

- All progress made this week was made with the implementation of MMOEASA:
  - I've been fixing runtime problems with the code that caused exceptions
  - Once I got the algorithm to run I wasn't receiving a result because the crossover and mutation operators weren't applying their changes
  - Once I fixed the operators not modifying solutions, I noticed there were more runtime errors with them which I've been fixing
  - I've also been fixing logic errors with the Simulated Annealing "Calculate_cooling()" and the "MO_Metropolis()" functions
- I've also decided to heavily focus on other coursework as of late as I'm nearing their deadlines and I figured I can continue with my project over Christmas if I need to

Objectives:

- I will continue to try to correct my algorithm implementation and debug any problems in an attempt to get a valid solution

Supervisor's Comments:

- Ben recommended that if I need more information about Hypervolumes that I could speak to Kevin Sim
  - Ben said he thinks that Hypervolumes are objective function values of the most optimal solution, but he's not entirely sure
- Ben was concerned that because I've done a lot of coding that I may now have a lot of code to debug

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 07/12/2021 **Last diary date: 30/11/2021**

Progress:

- I've spent most of this week completing other coursework as I have two deadlines on the 10th, but I have made some progress
- Debugging is going very well, I think; most of the fixed I've applied to the operators I've added seem to be successful
    - Of course, I will need to refer to the official implementation of MMOEASA to make sure that what I expect this to do is correct
- Reminder: ask Ben if he would like access to this Google Drive folder as he mentioned it a while ago

Objectives:

- Continue debugging
- Try to get the literature review finished over the holiday

Supervisor's Comments:

- Ben recommended to try and work on the literature review when you'd like a break from the coding
- We decided to take a break over the holiday and only arrange to discuss things if I need advice
    - This decision was made as I figured I'd mostly be debugging the algorithm, which isn't meeting-worthy
- Ben's response to needing access to the diary was only to see how I'm doing it and that I am in fact keeping a diary

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 04/01/2022 **Last diary date: 07/12/2021**

Progress:

- Continued bug fixing with the MMOEASA implementation
    - The algorithm now no longer crashes at any point, but it is unable to find any feasible solutions; the solutions are currently being marked as infeasible due to the time window and/or capacity violations and, therefore, results "theoretically" aren't obtained
    - In other words, results are being produced but none of them do not violate the time window or cargo constraints

Objectives:

- Fix the bugs mentioned in the "Progress" section
- Continue the objectives from the last diary entry

Supervisor's Comments:

N/A (no meeting was organised; check the previous diary entry, explaining why)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 11/01/2022 **Last diary date: 04/01/2022**

Progress:

- Feasible results are now being produced;
  - This week I decided to implement more of the mutation operators as I figured the reason that the objective function was declaring solutions infeasible was because the algorithm was not generating a diverse range of solutions and, therefore, every solution was similar to the TWIH (Time-Window-based Insertion Heuristic) solution, which is always infeasible
  - Once I began implementing more operators, specifically once MMOEASA's third mutation operator was implemented, feasible results started being produced, proving my theory
  - I've also completed the implementation of the mutation operators and have been implementing remaining bug fixes and optimising some parts of the code following the completion

Objectives:

- Try to find a solution validator (as Ben suggested)
  - Did Raul use one?
- Perform a final check for bugs and possible optimisations for MMOEASA; if this is finished:
  - Begin looking into the next algorithm you've chosen
  - Continue the literature review if you have time this week; you could now write the information you need to in the literature review about the algorithms

Supervisor's Comments:

N/A (still on holiday; as mentioned in previous diaries, no unnecessary meetings will be scheduled)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 18/01/2022 **Last diary date: 11/01/2022**

Progress:

- Finished MMOEASA bug fixing and optimisations;
    - Despite my notes from previous meetings explaining why I wouldn't (it will restrict me to only using the problem instances that Raul uses; Solomon's 100 destination problems) I decided to use Raul's Hypervolume values as they play a massive role in the MO_Metropolis acceptance criterion
        - This is because to generate Hypervolumes, I need to execute the algorithm a few times on a single problem instance, then get the objective functions of the most optimal solutions found in each execution to calculate that one solution's Hypervolumes - and repeat this process for every problem instance - which would take a lot of time
        - I figured that since the other algorithm I've chosen to implement also uses Solomon's instances, it will also use the 100 destination problem variants, so the restriction to these problem instances wouldn't make a difference when trying to reach the original paper's results
- Finished Literature Review;
    - I covered the heuristics, search optimisations, and any other special implementations of the algorithms I've chosen; such as MMOEASA's Simulated Annealing
    - Again (as mentioned in previous diaries), the reason the LR wasn't completed until now is because I hadn't implemented MMOEASA successfully until now

Objectives:

- Now is the time to find a solution validator to ensure your solutions are correct and feasible (again, so that I can prove my algorithm's solutions are valid; there may be a bug in my objective function that gives false positives, or something to that effect)
    - Because, once this is done, you can try to replicate Raul's results
- If possible, once Raul's results have been replicated, make a start with the dissertation

Supervisor's Comments:

- I asked ben about how to write the sections:
    - Lit review
    - Technical review may be sensible
    - How you wrote the code
        - Show software engineering principles; how you made the code optimal, followed standards, etc.
    - What experiments you're going to do
        - And how you're going to evaluate them
    - Results
    - Conclusions

    - Evaluation of the project
- Ben suggested using the university's computers to run multiple experiments simultaneously (since you mentioned that Python is causing the algorithm to run slower than it could)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 25/01/2022 **Last diary date: 18/01/2022**

Progress:

- I decided to use the solution validator from the original MMOEASA source code to check the solutions
    - I chose to use this validator as I already have the source code for it and when I was searching for solution validators online, I was unable to find any solution validator for Solomon's problem instances; so this will likely time
    - I emailed Ben to confirm whether doing this is wise as using the original algorithm's objective function means I would need to trust that it, too, is bug-free (which was the reason for using a second validator as well as my own)
- I managed to successfully implement the original MMOEASA solution validator with my data structures (in C)
    - Can confirm that the solutions generated by my implementation of MMOEASA and my objective function is acquiring the same results of a solution as Raul's original objective function
- Made amendments to the dissertation Introduction and Literature Review, and worked on the Technical Review
    - Confirm with Ben the TR is a review of technologies required to use the app
    - … and not technical applications of the prototype to the real world, or is it?

Objectives:

- Try to begin working on the second algorithm
    - This will help you get a feel for how much time it may take to implement
- Start planning how you're going to evaluate the results (and continue trying to replicate Raul's results; an objective from last week)

Supervisor's Comments:

- The technical review is a review of the decisions you made
    - These will be any decision made relating to the project (such as justification of your: chosen language, datasets used, time management, etc.)
- You should figure out how you'll do the experiments
    - Ben recommended that you research Statistical Significance; the results generated by the algorithm will likely always be different and, therefore, you'll need to come up with some method of giving meaningful statistics
    - The reason is that you need to be able to differentiate from actual figures and executions of the algorithm that are considered as "lucky" as the algorithm is stochastic; results will be different every run
- Ben asked what I plan to do next now that I have a successful implementation:
    - I said "I plan on implementing the second algorithm, but no more than two as I imagine it'd be better to write my own" to which he said it'd be far better to try

to come with you're own changes to the algorithm than writing a third algorithm from research
- Ben also recommended that if you have to choose between writing the second algorithm and doing your own that it'd be better to do your own (to show your understanding of the project); this depends on how much time you have available

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 01/02/2022 **Last diary date: 25/01/2022**

Progress:

- This week's progress has been slow due to family issues, but following this point is a list of what I did manage to get done
- Continued writing the Technical Review; now renamed to Technical Justification
    - Ask Ben if you should include decisions you made regarding choices such as: what VRP type to investigate, what objective function to investigate, etc.
- Decided on the evaluation strategy
    - MMOEASA's evaluation strategy will be used as it is considerate of the fact that GAs are stochastic
    - I figured that I'd be faster finding a suitable evaluation strategy in the papers I've chosen to research that by researching multiple strategies
    - However, I'd need to confirm that it's ok to use MMOEASA's method with the other algorithm, based on how MMOEASA's evaluation works (using the TWIH)
- I noticed that the algorithms chosen investigate different objective functions
    - Ask Ben if it'd still be ok to research them and compare the effectiveness of different objective functions

Objectives:

- Determine whether or not you're going to use either:
    - The second algorithm with MMOEASA's objective function,
    - Or MMOEASA alone;
    - This has to be done if the objective function of the second algorithm cannot be changed; Raul's cannot be changed due to me using his external Hypervolume approximations for the MO_Metropolis function
- Continue the Technical Justification and trying to implement the second algorithm (continuing from last week)

Supervisor's Comments:

- For Raul's evaluation strategy, you can work out the area each Hypervolume occupies by working them out as rectangles; the width is the distance along x from the original HV to the next and y being the distance from the HV to the x-ref's y-coordinate
- You can use the x-ref (that MMOEASA's evaluation uses) for the other algorithm as well because all it's doing is trying to create a (constant) large offset point for each problem instance to compare the ND solutions to
    - If you find that the next algorithm creates worse solutions than the Hypervolume of the x-ref point, then all you would need to do to fix that is to triple, quadruple, etc. the x and y coordinates instead of doubling it, but you must then do the same for MMOEASA's evaluation

- As for the differing objective functions, they need to be the same in order to be able to make a fair comparison in your results
    - To fix this, you could change the objective function of one algorithm to the other's objective function, or experiment with both objective functions in each algorithm

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 08/02/2022 **Last diary date: 01/02/2022**

Progress:

- Implemented the calculation that MMOEASA's paper uses to calculate the efficiency of the algorithm
    - Mention that the TWIH ref point is smaller than it should be (in your data)
- Determined that I may not be able to use the objective function from Ombuki's algorithm during tests as MMOEASA relies on the Hypervolumes to evaluate solutions
    - This may not be the case as I could use Ombuki's Pareto-rank algorithm; that doesn't require Hypervolumes
- Began implementing algorithm 2 (which I named Ombuki's algorithm after the researcher; as they didn't explicitly name the algorithm themselves)
    - I've been able to implement the population initialisation functions, feasible-izer, and the Pareto-rank function
        - Need to confirm with Ben if my understanding of the Pareto-rank function is correct

Objectives:

- Ideally, I'd like to finish these for next week so I can move to the evaluation stage:
    - Technical Justification
    - Algorithm 2 implmentation

Supervisor's Comments:

- As long as your metrics in the evaluation function (TWIH x and y coordinates multiplied by two) are the same as Raul's then it will be fine to use them as you'll get similar results
- It looks like the Pareto-ranking function is comparing a single solution with every solution in the population; to check if it's not dominated (not just comparing it with another, single solution; the way you're currently doing it)
- Following up from your missed question from last week:
    - It would be wise to discuss why you researched things like Solomon's instances and the VRPTW specifically

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 15/02/2022 **Last diary date: 08/02/2022**

Progress:

- I'm pretty sure I've finished the implementation of Ombuki's Algorithm
  - However, it is not generating feasible solutions
  - I believe this is because of the initialiser; the crossover and mutation operators are not powerful and don't often modify the first vehicle, so solutions are often infeasible because the first destination is likely to have a high ready time
- Likely finished, for now, the Technical Justification in the dissertation;
  - One section remains: the Custom Solution justification
  - This is where I justify my own solution. So, as I'm unsure exactly of what I'll implement, it's yet to be written
- In the last meeting, I determined that I cannot use Ombuki's objective function in MMOEASA as MMOEASA requires Hypervolumes in its acceptance criterion (which I don't have for objectives other than total distance, distance disbalance, and cargo disbalance)
  - But this may not be true as I could also swap the acceptance criteria, as well as the objective functions

Objectives:

- If you're confident you've implemented the other two algorithms then you can make a start on your own
  - You could try making your own changes to the two algorithms you have so far to see if they perform any better with your changes
  - If you do have the two algorithms implemented then you could begin evaluations and comparisons, as per Ben's suggestion

Supervisor's Comments:

- Maybe create a graphic of the initialising heuristic you've come up with to help explain it better in your dissertation
- It may turn out that when you try to use MMOEASA's objective function in Ombuki's Algorithm, or vice versa, you may find that they both worst best with their own objective function as that's what they've been written to solve
  - It may be the case that additional aspects of the algorithms, such as the Crossover and Mutation operators, are implemented to improve that algorithm's objective functions
- Ben says you should compare results as soon as you can; despite not having your own solution yet

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 22/02/2022 **Last diary date: 15/02/2022**

Progress:

- Implemented each algorithm's objective function and acceptance criterion in both algorithms, for evaluation
    - Also implemented MMOEASA's evaluation strategy (graph) for Ombuki's Algorithm
- Considered possible optimisation techniques that I could use in my own algorithm and I managed to come up with my own:
    - Initialisation heuristic
    - Crossover operator
    - Multi-threaded mutation invocation
- Did some comparisons with MMOEASA's objective function
    - I was unable to do more as I kept being asked to leave D02 due to classes
    - I also noticed that my implementation of accepting MMOEASA solutions into the non-dominated set was incorrect: bad solutions were being kept in the set and weren't removed if they had been dominated

Objectives:

- Continue with evaluations after MMOEASA non-dominated set correction
- Try to create a graphic of your initialisation heuristic idea to explain it better

Supervisor's Comments:

- Multi-threading may be interesting as it could cut down the elapsed time (as I thought: it would allow for more mutation operations to be carried out in the same amount of time)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 01/03/2022 **Last diary date: 22/02/2022**

Progress:

- Gathered more results
  - During this process, an experiment with Solomon's instance R101 revealed a bug in MMOEASA: the re-insertion of infeasible customer placements during crossover would create a new vehicle when the amount is at its limit, exceeding the limit of vehicles if all existing 25 (in the problem instance) couldn't occupy the unvisited customer
  - This was fixed quickly, but the bug appears to be present in the original MMOEASA source code, also
- Began implementing my own solution
  - Created a visual representation of my custom insertion heuristic
    - I also implemented my insertion heuristic successfully: it appears to be fulfilling its purpose of creating a feasible solution at the initialisation stage
  - Began implementing a crossover operator that uses a decision tree to decide on the fittest location for every destination to be inserted
    - This proved infeasible as the decision tree is a very demanding process; the technical justification in the dissertation now covers why
  - Mutation operators have been planned out and also implemented: the operators implemented are separate from both existing algorithms, but there are multiple mutators (like MMOEASA) that also specialise in feasibility (like Ombuki's Algorithm)
  - Decided that to evaluate the custom algorithm, I'll also need to gather statistics on the effectiveness of MMOEASA's and Ombuki's operators and compare them with statistics from my custom GA

Objectives:

- Experiment on your own algorithm
- Start writing up the results of the experiments with the other two algorithms

Supervisor's Comments:

- Ben said that initialisations are supposed to be quick as to save time performing evolutionary functions, so it'd be good to compare the run time of your initialising heuristic to the execution time of others'
  - Ben also suggested that you could slightly randomise your insertion heuristic: consider figure (d), the insertion of "10, 6, 4" could be shuffled to become "6, 4, 10", for example
- Recommended that you experiment of the mutation operators individually to get a feel for what works well and what does not

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 08/03/202 (postponed to 10/03/2022) **Last diary date: 01/03/2022**

Progress:

- I discovered a big inconsistency in my gathered results: I hadn't been using the same termination condition for each algorithm, but instead I'd been using the termination conditions from the original research papers
    - I also noticed that I had applied a solution to a scenario that Ombuki's paper never mentioned: during crossovers, when no feasible insertion point is found in the parent being merged into, I was inserting into the fittest infeasible position, but Ombuki does not state that this should happen and only states that a new vehicle should be created to accommodate infeasible destinations
        - With my implementation reverted to match Ombuki's paper, the algorithm is no longer giving feasible solutions, so should I keep this change or bring re-implement my own fix?
    - Therefore, I had to redo my results of MMOEASA and Ombuki's algorithm
    - This worked out ok and I also found a better way to evaluate my results using a method that provides higher accuracy of results' quality
- Finished the implementation of my own algorithm
    - The operators have shown to be effective, with the crossover being around 10% effective in C101 and mutations being about the same; the numbers may be lower than expected (are they?) because:
    - So far, as tested, the algorithm is able to provide the most optimal known solution to Solomon's 100-customer instance C101 in under a minute
- Began writing up the evaluation stage of the dissertation

Objectives:

- Update Ombuki's implementation and finish gathering of results, then continue with evaluation

Supervisor's Comments:

- It'd be better to measure the effectiveness of the operators using the final result of the algorithm as that's all that is important: can your algorithm produce a better result in the same amount of time as the other algorithms?
- If you write up about the result of both of your Ombuki implementations (the original insertion of infeasible vehicles and your alternative), then that's ok as they're both results
    - It's ok to say that an algorithm wasn't able to produce any results to a problem as it's still research and won't look like a bug because the other algorithms are working
    - Don't forget to mention that there's an anomaly in the paper; the reason why you're having to experiment using your own understanding of what their work originally did and your fix (alternative) to the anomaly

- Because they haven’t covered in their paper what happens in the infeasible insertion scenario, there’s no way you can replicate their results, so explain that in your dissertation

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 15/03/2022 **Last diary date: 08/03/2022**

Progress:

- Experiments finished
  - Following last week's meeting, I implemented both versions of Ombuki's Algorithm and ran experiments; one version with the original crossover and one with MMOEASA's infeasible insertion fix
  - I concluded that the issue with the unsolvable problem instance (in my algorithm) is a result of Ombuki's crossover as Ombuki's Algorithm also struggles to solve the problem instance
- Possibly wrote the evaluation section; yet to make a conclusion
  - Should the evaluation be submitted for review?

Objectives:

- Begin project- and self-evaluation

Supervisor's Comments:

- Don't forget to write in your evaluation about the project as a whole;
  - How the end compares to your initial overview of the project; why is the initial idea different from the end, what decisions changed that?
  - Self-evaluation: how you feel you performed in certain scenarios, decisions you made and what you learned
- Make it clear in your abstract, introduction, and conclusion

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 22/03/2022 **Last diary date: 15/03/2022**

Progress:

- Proceeded with project and self evaluation; may be finished
    - I've proofread it a couple of times and compared it with other dissertation's evaluations
- Added pseudocode of my custom GA's main algorithm and acceptance to the dissertation
- Wrote internal comments explaining unclear functionality in the algorithms
- Added diary sheets and other required figures to the appendix

Objectives:

- Implement suggested changes in the dissertation

Supervisor's Comments:

- FInd out how Raul was able to have confidence in his results showing that one algorithm is better than another; as Ben said, you can't be confident in results from stochastic algorithms as they may be lucky or unlucky
    - Let Ben know if there's any issues or email Raul directly
- Add bar graphs of the tables in your evaluation so we can see which results were better
- Join both table 3 and table 5 to make the results easier to read
- Because you used elapsed time as the termination condition, state that you ran the tests independently on multiple PCs of equal spec (i.e. not in the background while the PC was playing a game or something to that effect)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Taylor Courtney **Supervisor: Ben Paechter**

Date: 29/03/2022 **Last diary date: 22/03/2022**

Progress:

- Made the suggested changes specified in last week's meeting
- Spent this week making corrections to the dissertation in preparation for submission

Objectives:

- Implemente Ben’s suggested changes

Supervisor’s Comments:

- Ben recommended that you do the Mann-Whitney analysis on your results so that you can ensure your runs are not lucky
- In your Future Work section, Ben wasn’t sure what you meant about writing a deterministic algorithm based on wait time modification; it may help to explain that you mean a deterministic algorithm with no stochastic elements that tries to eliminate wait times
- The algorithm name “Custom GA” isn’t very clear; try to come up with a better identifier

Appendix 4 and following

Spreadsheet containing the results of MMOEASA's graph occupancies when solving MMOEASA's objective function:

	A	B	C	D	E	F
1	C101	R101	RC101	C201	R201	RC201
2	51.65	0	47.07	68.88	66.76	68.39
3	49.26	0	51.28	64.34	66.4	60.91
4	48.74	0	44.56	64.13	71.25	64.55
5	49.67	0	40.09	64.45	68.43	62.57
6	45.55	0	48.21	72.24	68.65	66.57

Spreadsheet containing the results of Ombuki's Algorithm's graph occupancies when solving MMOEASA's objective function:

	A	B	C	D	E	F
1	C101	R101	RC101	C201	R201	RC201
2	30.12	0	0	61.84	72.23	66.26
3	37.87	0	0	64.35	69.34	64.33
4	0	0	0	60.37	67.43	61.42
5	0	0	0	65.11	68.93	57.28
6	44.11	0	0	61.54	67.78	60.85

Spreadsheet containing the results of MMOEASA's graph occupancies when solving Ombuki's objective function:

	A	B	C	D	E	F
1	C101	R101	RC101	C201	R201	RC201
2	79.54	0	77.95	85.54	74.42	77.99
3	79.62	0	76.62	86.34	74.26	79.05
4	79.05	0	78.26	85.39	73.69	78.46
5	79.74	0	72.26	87.18	78.24	76.8
6	78.77	0	74.54	81.56	79.02	77.12

Spreadsheet containing the results of Ombuki's Algorithm's graph occupancies when solving Ombuki's objective function:

	A	B	C	D	E	F
1	C101	R101	RC101	C201	R201	RC201
2	73.3	0	0	85.52	73.69	67.59
3	74.71	0	0	72.89	74.19	68.68
4	74.24	0	0	81.5	77.33	67.32
5	72.5	0	0	71.21	75.81	65.04
6	71.51	0	0	71.36	79.56	65.8

Spreadsheet containing the results of the custom GA's graph occupancies when solving Ombuki's objective function:

	A	B	C	D	E	F
1	C101	R101	RC101	C201	R201	RC201
2	91.62	82.84	0	94.06	87.37	86.3
3	91.62	82.66	0	94.06	87.37	86.38
4	91.62	82.71	0	94.06	87.38	86.22
5	91.62	82.75	0	94.06	87.51	86.45
6	91.62	82.87	0	94.06	87.59	86.17

Spreadsheet containing the execution time and feasible initialisations of the custom GA's initialisation heuristic:

8	problem	initialiser_e	feasible_init
9	C101	15.648	0
10	C101	15.613	0
11	C101	15.662	0
12	C101	19.969	0
13	C101	15.661	0
14	R101	19.956	0
15	R101	15.66	0
16	R101	15.659	0
17	R101	15.655	0
18	R101	15.659	0
19	RC101	15.651	0
20	RC101	19.931	0
21	RC101	15.659	0
22	RC101	15.658	0
23	RC101	15.666	0
24	C201	15.661	30
25	C201	15.66	30
26	C201	15.655	30
27	C201	15.658	30
28	C201	1.181	30
29	R201	19.907	27
30	R201	15.626	27
31	R201	15.656	27
32	R201	15.654	30
33	R201	15.656	28
34	RC201	15.613	30
35	RC201	15.63	30
36	RC201	15.661	30
37	RC201	15.656	30
38	RC201	15.628	30

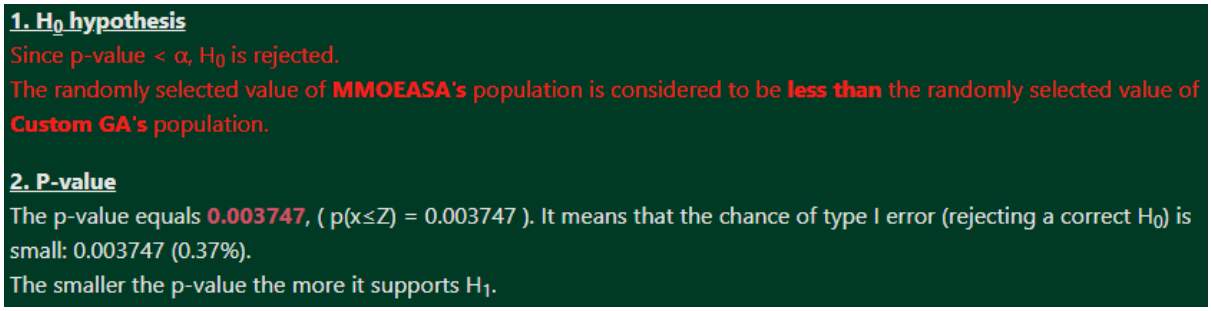
Spreadsheet containing the execution time and feasible initialisations of MMOEASA's initialisation heuristic:

8	problem	initialiser_e	feasible_ini
9	C101	31.234	0
10	C101	31.171	0
11	C101	15.601	0
12	C101	31.204	0
13	C101	15.628	0
14	R101	19.842	0
15	R101	19.915	0
16	R101	31.279	0
17	R101	15.624	0
18	R101	31.251	0
19	RC101	15.66	0
20	RC101	15.659	0
21	RC101	34.489	0
22	RC101	31.248	0
23	RC101	15.658	0
24	C201	31.282	0
25	C201	15.655	0
26	C201	15.654	0
27	C201	31.284	0
28	C201	19.939	0
29	R201	31.251	0
30	R201	31.444	0
31	R201	31.276	0
32	R201	31.268	0
33	R201	15.655	0
34	RC201	31.237	0
35	RC201	31.276	0
36	RC201	19.754	0
37	RC201	15.661	0
38	RC201	31.254	0

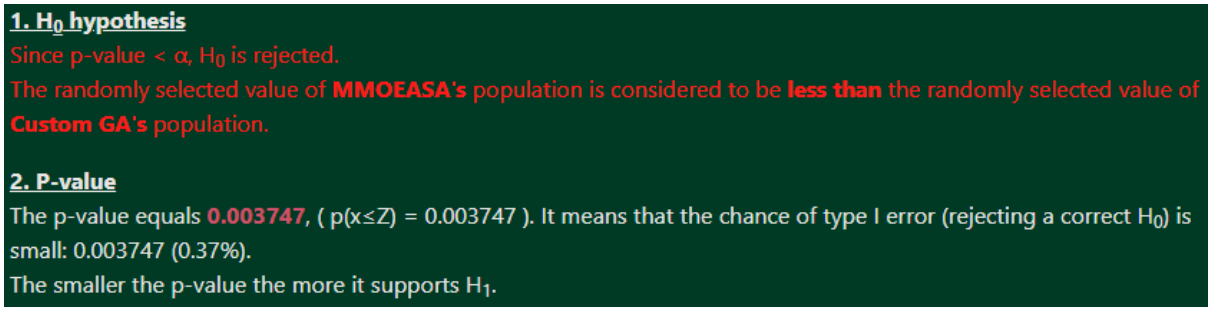
Spreadsheet containing the execution time and feasible initialisations of Ombuki's Algorithm's initialisation heuristic:

8	problem	initialiser_e	feasible_ini
9	C101	31.285	0
10	C101	35.533	0
11	C101	35.581	0
12	C101	31.279	0
13	C101	31.278	0
14	R101	31.279	0
15	R101	31.282	0
16	R101	31.266	0
17	R101	31.252	0
18	R101	33.435	0
19	RC101	31.283	0
20	RC101	35.573	0
21	RC101	31.279	0
22	RC101	35.565	0
23	RC101	35.622	0
24	C201	31.279	0
25	C201	19.886	0
26	C201	31.281	0
27	C201	31.252	0
28	C201	35.598	0
29	R201	31.25	0
30	R201	31.279	0
31	R201	31.283	0
32	R201	19.849	0
33	R201	31.249	0
34	RC201	19.82	0
35	RC201	31.281	0
36	RC201	31.252	0
37	RC201	31.282	0
38	RC201	31.25	0

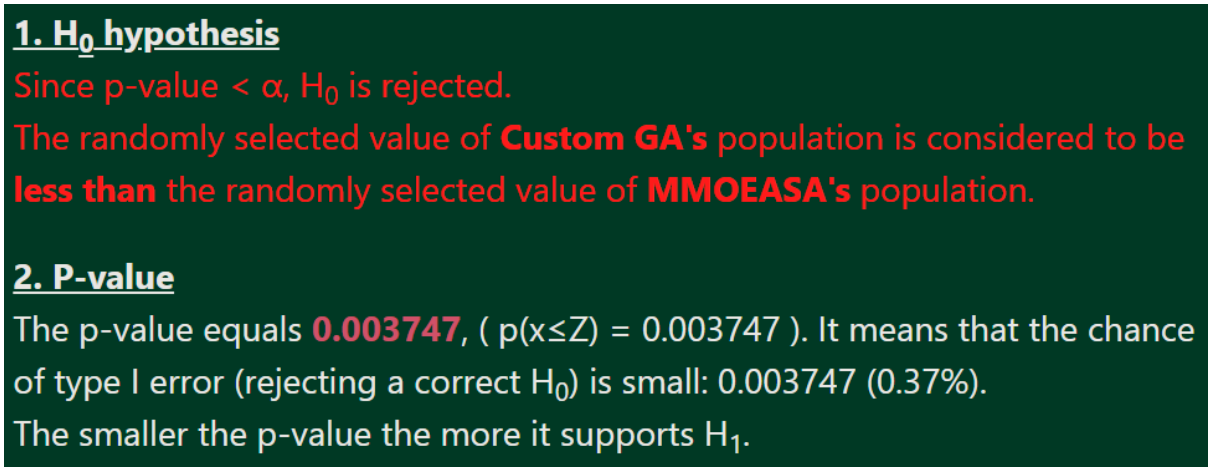
Ombuki's objective function, C101 p-value of the custom GA's results versus MMOEASA:



Ombuki's objective function, R101 p-value of the custom GA's results versus MMOEASA:



Ombuki's objective function, RC101 p-value of the custom GA's results versus MMOEASA:



Ombuki's objective function, C201 p-value of the custom GA's results versus MMOEASA:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **MMOEASA's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value

The p-value equals **0.003747**, ($p(x \leq Z) = 0.003747$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003747 (0.37%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, R201 p-value of the custom GA's results versus MMOEASA:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **MMOEASA's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value

The p-value equals **0.005963**, ($p(x \leq Z) = 0.005963$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.005963 (0.6%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, RC201 p-value of the custom GA's results versus MMOEASA:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **MMOEASA's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value

The p-value equals **0.003968**, ($p(x \leq Z) = 0.003968$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003968 (0.4%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, C101 p-value of the custom GA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value

The p-value equals **0.003747**, ($p(x \leq Z) = 0.003747$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003747 (0.37%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, R101 p-value of the custom GA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value

The p-value equals **0.003747**, ($p(x \leq Z) = 0.003747$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003747 (0.37%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, RC101 p-value of the custom GA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **Custom GA's** population.

2. P-value

The p-value equals **1**, ($p(x \leq Z) = 1$). It means that the chance of type I error, rejecting a correct H_0, is too high: 1 (100%).

The larger the p-value the more it supports H_0.

Ombuki's objective function, C201 p-value of the custom GA's results versus Ombuki's Algorithm:

1. H_0 hypothesis
Since p-value < α, H_0 is rejected.
The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value
The p-value equals **0.003747**, (p(x≤Z) = 0.003747). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003747 (0.37%).
The smaller the p-value the more it supports H_1.

Ombuki's objective function, R201 p-value of the custom GA's results versus Ombuki's Algorithm:

1. H_0 hypothesis
Since p-value < α, H_0 is rejected.
The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value
The p-value equals **0.005963**, (p(x≤Z) = 0.005963). It means that the chance of type I error (rejecting a correct H_0) is small: 0.005963 (0.6%).
The smaller the p-value the more it supports H_1.

Ombuki's objective function, RC201 p-value of the custom GA's results versus Ombuki's Algorithm:

1. H_0 hypothesis
Since p-value < α, H_0 is rejected.
The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **Custom GA's** population.

2. P-value
The p-value equals **0.003968**, (p(x≤Z) = 0.003968). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003968 (0.4%).
The smaller the p-value the more it supports H_1.

Ombuki's objective function, C101 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.003968**, ($p(x \leq Z) = 0.003968$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003968 (0.4%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, R101 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **1**, ($p(x \leq Z) = 1$). It means that the chance of type I error, rejecting a correct H_0, is too high: 1 (100%).

The larger the p-value the more it supports H_0.

Ombuki's objective function, RC101 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.003747**, (p(x≤Z) = 0.003747). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003747 (0.37%).

The smaller the p-value the more it supports H_1.

Ombuki's objective function, C201 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.01587**, (p(x≤Z) = 0.01587). It means that the chance of type I error, rejecting a correct H_0, is too high: 0.01587 (1.59%).

The larger the p-value the more it supports H_0.

Ombuki's objective function, R201 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.5417**, ($p(x \leq Z) = 0.5417$). It means that the chance of type I error, rejecting a correct H_0, is too high: 0.5417 (54.17%).

The larger the p-value the more it supports H_0.

Ombuki's objective function, RC201 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.003968**, ($p(x \leq Z) = 0.003968$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003968 (0.4%).

The smaller the p-value the more it supports H_1.

MMOEASA's objective function, C101 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.005963**, ($p(x \leq Z) = 0.005963$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.005963 (0.6%).

The smaller the p-value the more it supports H_1.

MMOEASA's objective function, R101 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **1**, ($p(x \leq Z) = 1$). It means that the chance of type I error, rejecting a correct H_0, is too high: 1 (100%).

The larger the p-value the more it supports H_0.

MMOEASA's objective function, RC101 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value < α, H_0 is rejected.

The randomly selected value of **Ombuki's** population is considered to be **less than** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.003747**, ($p(x \leq Z) = 0.003747$). It means that the chance of type I error (rejecting a correct H_0) is small: 0.003747 (0.37%).

The smaller the p-value the more it supports H_1.

MMOEASA's objective function, C201 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.0754**, ($p(x \leq Z) = 0.0754$). It means that the chance of type I error, rejecting a correct H_0, is too high: 0.0754 (7.54%).

The larger the p-value the more it supports H_0.

MMOEASA's objective function, R201 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.8452**, ($p(x \leq Z) = 0.8452$). It means that the chance of type I error, rejecting a correct H_0, is too high: 0.8452 (84.52%).

The larger the p-value the more it supports H_0.

MMOEASA's objective function, RC201 p-value of MMOEASA's results versus Ombuki's Algorithm:

1. H_0 hypothesis

Since p-value > α, H_0 cannot be rejected.

The randomly selected value of **Ombuki's** population is assumed to be **greater than or equal to** the randomly selected value of **MMOEASA's** population.

2. P-value

The p-value equals **0.1111**, ($p(x \leq Z) = 0.1111$). It means that the chance of type I error, rejecting a correct H_0, is too high: 0.1111 (11.11%).

The larger the p-value the more it supports H_0.