Abstract

In this report, an optimisation problem for bunker supply operation is studied. Bunker supply involves delivering fuel, also known as bunker in the maritime industry to ships parked at anchorage. The operation is characterised as a Vehicle Routing Problem (VRP) with various constraints, such as profit maximisation objective, capacity, time windows, multi-compartment and so on. Two methods are proposed, namely a Mixed Integer Linear Program (MILP) and Adaptive Large Neighbourhood Search (ALNS). These methods are then tested using test data generated from the Solomon benchmark for test cases with 25, 50 and 100 customers. The results show that ALNS is a practical method to find good routes for bunker supply.

Acknowledgement

The author would like to express his sincere thanks and deepest appreciation to his final year project supervisor, Associate Professor Tai Kang for allowing him to undertake this project and giving him the independence to explore the project. Without his invaluable guidance and advice, the author would not be able to discover the different optimisation topics involved in this project.

The author would also like to thank his friends for their constant support and helpful inputs throughout the project, helping him build confidence in undertaking the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the maritime industry, bunker refers to the fuel used on ships. There are several types of bunker available for different applications [1]. In 2023, the Port of Singapore was the top bunkering port in the world with over 51 million metric tons [2] of bunker sold, supplying thousands of vessels which call at the port annually. Bunker is usually delivered to ships by a bunker barge. Typically, a bunker barge collects bunker at the oil terminal before delivering to multiple ships at the anchorage in a trip. As illustrated in Figure 1.1, bunker in the barge is transferred to customer ships through ship-to-ship transfer. When necessary, the barge returns to the oil terminal to top up its storage to service more customers. This process is known as bunkering.

1.2 Motivation

With 41 bunker suppliers [3] in the Port of Singapore, bunkering is a highly competitive industry. Hence, each supplier have incentive to find efficient routes for their bunker barges to deliver bunker to customer vessels. Improving route efficiency enhances competitiveness, reduces costs and boosts profitability of the supplier. This problem of finding the shortest routes while servicing the most customers is termed as the bunker supply problem and is the central problem to be studied in the this project.

Furthermore, to reduce environmental pollution and comply with international regulations, vessels have been using a wide range of bunker. As a result there has been an increase in complexity in bunkering operations with many different types of bunker.

Figure 1.1: Ship to ship transfer of bunker from barge to vessel

Current methods of routing are typically manual, relying on the experience of staff to determine an efficient route. Therefore, there is a need to automate this process, and find routes that are more efficient than those identified by human operators.

1.3 Objective and Scope

The aim of this project is to design an automated scheduling and routing system for bunkering operations. The routes generated will maximise revenue of bunker delivered and minimise travel distance of bunker barges.

To ensure that the routes generated are feasible, problem constraints like the capacity of each compartment within the barge and availability of each customer for delivery are considered. Following which, the performance of the system will be evaluated using benchmark datasets to demonstrate its effectiveness and functionality. With this system, the bunker supplier can expect to obtain efficient routes for their bunker barges, within a reasonable time frame.

1.4 Organisation of Report

The report is organised into 6 chapters.The respective focus of each chapter is briefly described below.

- Chapter 1 provides an introduction and motivation for the problem.
- Chapter 2 reviews current literature on routing problems, and introduce

methods to tackle those problems. From this review, the author decides to apply linear programming and heuristic methods to solve the problem.

- Chapter 3 gives the detailed mathematical formulation of the problem, including the assumptions adopted for the problem, notation of decision variables, problem constraints and the objective function of the problem.
- Chapter 4 describes the heuristic Adaptive Large Neighbourhood search (ALNS) used to solve the problem.
- Chapter 5 describes the implementation details of the 2 proposed methods and results from testing the effectiveness of the 2 methods.
- Chapter 6 discusses future improvements and conclusion of the report.

Chapter 2

Literature Review

In this chapter, we will discuss related studies which would help in the development of a routing system.

2.1 Vehicle Routing Problem

The vehicle routing problem (VRP) is a problem first formulated by George Dantizg in 1959 [4] and is a well-researched problem in operations research. The VRP aims to find the shortest route for a set of vehicles to visit a set of customers.

The VRP can be adapted to fit the bunker supply problem. The objective is modified to maximise profits, defined as the difference between total revenue from bunker delivered and total cost of travel. In the literature, this is known as the profitable tour problem. Another similar problem in literature is the orienteering problem, which aims to maximise revenue, while keeping route distances within a certain limit.

Furthermore, the following additional constraints have to be respected.

1. Capacity: Each bunker barge can only carry a limited amount of bunker.

2. Time Windows: Each vessel can only be visited within a specific time window.

3. Multi Trip: Each bunker barge is allowed to perform multiple trips in one planning window.

4. Multi Compartment: In each barge, there are multiple compartments for the different types of bunker.

5. Dynamic Service Time: Service time at the terminal to top up bunker is unknown beforehand as it is dependent on the amount of bunker delivered in the previous trip.

6. Continuous Planning Window: Due to the 24/7 operations, route planning would be done continuously in batches thus, it cannot be assumed that routes planned would start at the terminal.

The VRP is a NP-hard problem [5], which underlines the difficulty of finding an optimal route for large problem instances. It is further noted that the addition of time windows constraints makes it such that even finding a feasible route is NP-hard [6].

With these additional set of constraints placed on the VRP, it becomes equivalent to the bunker supply problem. Consequently, we can apply common methods to solve the VRP to solve the bunker supply problem. In existing literature, it is difficult to find papers which discuss exactly the same problem with the same constraints. Thus, modifications would be needed to apply proposed methods to solve the bunker supply problem.

In the VRP, concepts such as vehicle, node/customer and depot are used to describe various elements in the problem. In the bunker supply problem, these would be equivalent to barge, vessel and terminal respectively. These terms would be used interchangeably in this project.

2.2 Exhaustive Search Method

From research done on the bunker supply problem by Wang [7], the bunker supply problem is be solved using an exhaustive search method. The best route is found by iterating through all possible routes to find the best one. However, such a method is of factorial time complexity, which makes finding the optimal route for more than 10 customers impractical. Additional improvements have been made by Ho [8], to split the route of the different barges and have made finding an optimal route for up to 20 customers practical.

2.3 Linear Programming

The VRP can be modelled using linear programming (LP) methods. LP methods are flexible and can take into account the various constraints, such as multiple trips [9] or multiple commodities [10]. The problem can be modelled using math equations, before being solved using state of the art solver engines.

A standard LP formulation contains 3 parts, the decision variables, objective function and constraints. The decision variables represents decisions that should be made to solve the problem. The variables have to be set such that it fulfils the constraints of the problem. At the same time, the values of the decision variables minimises the objective function. For example, in the VRP, the decision variables binary variables to decide arcs to take between 2 nodes, with the objective function to minimise the cost of travel across arcs chosen. The constraints ensure that the arcs chosen follow a Hamiltonian path and only visit each node once. For the bunker supply problem, the decision variables, objective function and constraints would be modified to fit the problem. Once these are defined, the LP can be solved using the Simplex algorithm to obtain the optimal solution. For LP with integer decision variables, the integer constraint is first relaxed and the resulting LP solved. Subsequently, and branch and bound algorithm and cutting planes can be introduced to force the decision variables to be integer. This is known as the Branch and Cut.

This method is promising as LP methods have been used to solve a variety of optimisation problems beyond the VRP. Thus, advanced solver engines has been created to solve LPs, enabling us to find a solution quickly. We can thus leverage advancements in state-of-art solver engines to find the optimal solution. Notably, from a previous research study done at Piraeus Port in Greece [11], a Mixed Integer Linear Program (MILP) model has been developed to find an optimal routing for bunker supply barges. However, the model produced is inefficient and takes over 10,000 seconds to find an optimal route for 10 customers.

2.3.1 Branch Cut and Price

While direct linear programming methods described above are useful, they often struggle to find optimal solutions for large instances. The branch cut and price (BCP) can be used instead. Direct LP models can be decomposed into a master problem (MP). Instead of using arcs as decision variables, entire feasible routes are used as binary decision variables, with the objective function to chose the best route. This is the MP and for the VRP, and is also known as the set covering problem. However, in large problems, there are often too many feasible routes to chose from, thus directly solving the MP impractical. Hence, only a selected subset of routes is chosen, to form the restricted master problem (RMP), which would have its integer requirements relaxed and solved as a pure linear program.

This is done as it is easier to solve pure LP problems with a reduced set of decision variables. By solving the RMP, the duals can be obtained, which can be used to calculate the reduced cost of all other variables not in the RMP. By LP theory, variables that have a negative reduced cost would mean that addition of that variable to the RMP would improve the solution. However, there are many variables not in the RMP thus calculation of the reduced cost of each variable would take a long time. Instead, this calculation is converted into another LP problem, known as the pricing problem. Here, the objective is to find the variable with the lowest reduced cost.

For the VRP, the pricing problem is the resource constraint shortest path problem (RCSP), where we have to find the route which has the minimum reduced cost while respecting some resource constraints like time or capacity. Once routes with a negative reduced cost is found, they are added to the RMP, which is solved again, to generate another pricing problem which is solved again. This cycle continues until the solution of the pricing problem, the minimum reduced cost, is greater than zero. Thus, no addition of routes to the RMP would improve it's solution. Thus, the solution found by the RMP is the optimal solution to the MP.

As the RMP has it's binary constraints relaxed, a further branch and bound mechanism may be needed to find optimal binary solutions. To further accelerate the process, cutting planes are added to eliminate non-integer solution and reduce branching.

The BCP has been proven to work well to solve various types of VRPs, such as VRP with time windows [12] or multi-trip VRPs [13]. However, additional modification work has to be done to fit the bunker supply problem, mainly to fit the profit maximisation objective function and multiple different fuel types. However, implementation of the BCP is non-trivial. Furthermore, the state of the art implementations include further improvements to the basic BCP described above, further complicating the process of implementing the BCP method to the bunker supply problem. Instead of manually implementing the MP, RMP and Pricing Problem, a Dantzig-Wolfe Decomposition can be done on the original MILP formulation, which can create the MP, RMP and Pricing problems to be solved algorithmically [14]. Various projects are ongoing to automate the Dantzig-Wolfe decomposition [15], to make the implementation of the BCP easier. However, there are difficulties in detecting the underlying structure of the MILP to split into the MP and pricing problem. As such, current automated methods to convert the MILP to a BCP may not be as efficient as directly solving the

MILP using state-of-the-art solver engines.

2.4 Constraint Programming

Similar to Linear Programming, Constraint Programming (CP) allows us to translate the bunker supply problem into a set of logical constraints, which are then solved by constraint propagation, to find the optimal solution. However, unlike LP where the problem have to be modelled by a set of math equations, CP allows the model to be modelled using math equations and logical constraints. This allows the problem to be potentially modelled in a more efficient manner. CP is first used to solve transportation problems in [16]. More recently, CP has been applied to solve the team orienteering problem, which is similar to the VRP but with a revenue maximisation objective [17]. However, unlike LP, constraints that can be used to model the problem is dependent on the solver used. Also, compared to MILP, the literature is more sparse on CP methods. As such, CP is not considered to solve the bunker supply problem.

2.5 Heuristics

As VRP is NP-hard, exact methods described do not scale well to find optimal solutions for large problems. Instead, we can introduce methods which find approximate solutions in polynomial time. An example are heuristic algorithms which are known to scale better to large problem instances. From a practical perspective, a method which is able to find a good solution in a quick and robust manner are acceptable. Being able to find the optimal solution is usually of less importance.

Examples of heuristic methods are Genetic Algorithm (GA) or Adaptive Large Neighbourhood Search (ALNS) which has been used to solve VRPs. These methods are able to quickly converge onto a good solution in reasonable time. However, as compared to MILP methods, there is no guarantee that the optimal solution would be found.

GAs can trace its roots to the evolution process of living organisms as seen in nature. Application of evolution in computing was first proposed by Turing [18]. In GA, a candidate solution is represented as a chromosome. At the start, a set of chromosomes are generated and evaluated according to a fitness function. Subsequently, a selected number of chromosomes is chosen to undergo recombination. The chromosomes between 2 candidate solutions are mixed to create new candidate solutions also known as offsprings. These offsprings are then evaluated by the fitness function and the cycle repeats. During the creation of offsprings, some mutation may be added to randomly change some parts of the chromosomes. This added randomness can prevent the GA from being stuck in a local minima.

ALNS was first created to solve the VRP with pick-ups and deliveries [19]. ALNS is an extension of an earlier idea known as Large Neighbourhood Search (LNS) [20]. In LNS, the route is repeatedly destroyed and repaired using operators, until a good route is found. This works as it allows the algorithm to search repeatedly search different large neighbourhoods of the search space. allowing it to find a good solution across a large region of the search space. In ALNS, this idea is extended by selecting from a set of destroy and repair operators probabilistically based off their weights. The weight of each operator is adaptively adjusted based off the operator's previous performance. This makes it more likely that well-performing operators are chosen in subsequent iterations. Thus, the search becomes more efficient as good operators are chosen to search for a good solution.

As the name suggest, the ALNS is able to search a large neighbourhood space efficiently, allowing the heuristic to find a good solution. At the same time, due to different problem complexity and structure, it can be difficult to find a single destroy and repair operator that would work well for different problems. ALNS allows us to use a set of operators and selects the best ones to be used. This enhances the flexibility and robustness of ALNS for different problem types, making it an attractive solution method.

2.6 Machine Learning

With advancements in Machine Learning models, it is also possible to solve VRPs using machine learning methods. There are many different types of models under the machine learning umbrella such as attention models. Attention models are extremely powerful and is notably used in Large Language Models (LLM) which can provided generalised reasoning capabilities [21]. The first attention model used in routing problems involves training a pointer network [22] to solve the Travelling Salesman Problem (TSP), which is a simplified version of the VRP. More recently, attention models [23] has been used to solve VRP problems.

Attention models are used as these models are able to contextualise well between the inputs. This means that it is able to weigh the demands and time windows between the different input customers to determine which should be the next customer to service.

In an attention model, the problem information, such as customer location, time windows, demands, current existing route are taken as inputs into the input encoder layer. The encoder layer transforms the problem information into embeddings which would be used by the model.

Next, the embeddings are put through a series of attention layers each with trainable weights. Between each attention layer is an activation function. Once the embeddings are passed through the attention layer, it goes to the output decoder layer, which decodes the embeddings into the customer which should be visited next. This can be added into the route, and reused as the inputs into the attention model. By repeating this process until all customers are served or the end time window is reached, we would be able to obtain a good route.

As obtaining the ground truth is difficult, these models are trained by pitting it against itself. This allows the model to learn what weights are suitable in order to produce a good route.

These attention models are promising as most of the computational resources are invested beforehand during the training stage. Only a relatively small fraction of computational resources is used during inference, which is where the actual solving of a VRP problem occurs.

However, training and inference of attention models require use of Graphics Processor Unit, (GPU) which can be costly. Also, pre-trained models are usually designed to only take in a certain number of inputs. Thus, there is a limit imposed on the number of customers a model can solve for, when the model is trained.

2.7 Quantum Computing

Quantum Computers (QC) take advantage of the quantum properties of qubits to perform computation. This allows us to take advantage of quantum effects like superposition and entanglement to create speed-ups compared to a classical computer. Being able to out-perform state-of-the-art classical methods is known as the quantum advantage.

QCs are currently noisy intermediate-scale quantum (NISQ) devices [24]. This means that there is a limit to the number of qubits available (100s), and the qubits available are noisy within a limited coherence time. Thus, QCs are prone to errors, and quantum error correction (QEC) techniques are needed to ensure that calculations from QCs are accurate. As such, alogrithms can only employ quantum circuits of limited depths. An example of a suitable algorithm is Quantum Approximation Optimisation Algorithm (QAOA) [25] which have been employed to solve VRPs [26]. The QAOA is a hybrid algorithm, which solves classically difficult parts of the problem using a quantum computer and the remaining parts solved using a classical computer.

Given the current technological limitations, QC is unable to solve any meaningful size of VRP. Also, for NP-hard combinatorial optimisation problems like the VRP, QC is not expected to solve them in P-time. Despite these limitations, it is of research interest to find quantum methods that can find better approximate solutions quicker or cheaper than classical methods. It is an open problem if a quantum advantage exist for combinatorial optimisation problems [27].

2.8 Hybrid Methods

While the above methods have been separated into different types, in reality, they have be combined to form hybrid methods, taking advantage of the relative strengths of each method. For example, heuristics and machine learning [28] have been used to improve the branch and bound mechanism of LP methods. Similarly, LP [29] have been included as one of the operators in the heuristic, creating matheuristics methods. Also, in the BCP, the pricing problem can be solved using CP [30] or heuristics.

2.9 Research Gaps

Due to the NP-hard nature of the problem, the exhaustive search method proposed by [7], [8] would be infeasible for large problem instances. Thus, the author has decided to explore linear programming and heuristic methods to tackle the problem. Due to difficulty, Branch Cut and Price method is not used. Due to resource constraints, the author is unable to implement machine learning methods. The current developments of quantum computers make it impractical to use quantum methods beyond the theory stage.

Reviewing literature on linear programming methods, the author is only aware of one research study which fully considers all 6 constraints [11]. However, it's performance is sub-optimal and the author has identified inefficiencies in the math formulation. Research considering multiple trips [9] is the foundation for the math model developed. This is used as most of the constraints except for multi-compartments are already modelled.

However, limitations of LP methods has led the author to explore heuristic methods. The ALNS heuristic is employed as an alternative to the MILP model. ALNS is chosen as it is designed to solve VRP, and from existing literature, has good performance. This makes ALNS a promising candidate solution to the bunker supply problem. The ALNS implementation in this paper is guided mainly by research done by the original paper [19] and a modern implementation to solve a similar problem [31].

Chapter 3

Mathematical Formulation

3.1 Problem Description

In the bunker supply problem, the bunker supplier would have $\mathcal{K}$ number of barges with limited capacity Q for each type of bunker f. The barges would have to supply all potential customer vessels $\mathcal{D}$, each having different demands for bunker and bunker type. The barge visits vessels to supply bunker before returning back to the terminal to top-up. This is considered as a trip. A barge is expected to make multiple trips during a planning window. The time required to top-up at the terminal would be linearly related to the amount of fuel required to top-up the barge back to full capacity. The barge can only supply the vessel within the start and end time windows of the vessel. If the barge reaches the vessel early, it is permitted to wait, although this would imply a less efficient route. In most literature and this project, the end time window is the last time possible that the barge can visit the vessel. However, in practical applications, the end time window is usually the last time possible before the barge have to leave the vessel. As such, the end time window may have to be pushed forwards by the service time in order for the solutions generated to be feasible.

Routes would be planned on a rolling planning window basis. This means that barge routes would be frequently updated, as new vessels demand top-up of fuel. Hence, at the start of a new planning window, the barge may not be at the terminal and instead be out servicing or travelling, as planned by the previous planning window. As such, the model should allow the operator to input the starting conditions of the various barges. The barges may not necessarily start at the terminal and this would have to be handled appropriately.

The problem assumes that the barge would top up to full capacity every time it arrives at the terminal. It is also assumed that vessels are serviced in one visit and cannot be serviced multiple times (no split deliveries allowed).

We set up an undirected graph with each vertex i representing a customer vessel. Each edge on the graph is represented by x_{ij}^{kr} and would have a cost to travel represented by C_{ij}. A trip r is defined as a route which starts at the terminal, visits various vessels and ends at a terminal. During one planning window, a barge k can undergo $\mathcal{R}$ many trips, as long as it does not breach the time windows.

3.2 Parameters

T_{ij}: travel time required to travel from vessel i to vessel j
C_{ij}: cost to travel from vessel i to vessel j equivalent to $\zeta * T_{ij}$ where ζ is the cost per unit time to operate the vessel
S_i: time required to service vessel i
A_i: Start time window of vessel i
B_i: End time window of vessel i
Q_f^k: capacity to carry fuel type f on vessel k
N_{if}: fuel type f required by vessel i
R_{if}: revenue to supply fuel type f to vessel i equivalent to $N_{if} * Price_f$

3.3 Sets

$\mathcal{D}$: set of vessels i to supply
$\mathcal{N}$: set of vessels i to supply & bunker terminal equivalent to $\mathcal{D} \bigcup 0$
$\mathcal{M}$: set of vessels i to supply & bunker terminal & dummy end at bunker terminal equivalent to $\mathcal{D} \bigcup 0 \bigcup N+1$
$\mathcal{K}$: set of barges k
$\mathcal{R}$: set of trips r undertaken by barge k
$\mathcal{F}$: set of fuel types f

3.4 Decision Variables

x_{ij}^{kr}: 1 if barge k travels from vessel i to vessel j on trip r, 0 otherwise.
y_i^{kr}: 1 if vessel i is visited by barge k on trip r, 0 otherwise.
t_i^{kr}: Time when barge k arrives at vessel i on trip r.
s_0^{kr}: Service time at the depot at the start of trip r for barge k.

l_{if}^{kr}: Fuel load of type f on barge k on trip r at the start of the trip ($i = 0$) or at the end of the trip ($i = N + 1$).
u_{if}^{kr}: Amount of fuel of type f delivered to vessel i by barge k on trip r

3.5 Objective Function

Objective function is to maximise the sum of revenue generated from supplying bunker minus the cost of travel between vessels.

$$maximise \sum_{i \in \mathcal{D}} \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}} \sum_{f \in \mathcal{F}} y_i^{kr} * R_{if} - \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}} x_{ij}^{kr} * C_{ij} \tag{3.1}$$

3.6 Constraints

3.6.1 Route Constraints

$$\sum_{k \in \mathcal{K}} \sum_{r \in \mathcal{R}} y_i^{kr} \leq 1 \quad \forall i \in \mathcal{D} \tag{3.2}$$

$$\sum_{j \in \mathcal{N}} x_{ij}^{kr} = \sum_{j \in \mathcal{N}} x_{ji}^{kr} = y_i^{kr} \quad \forall i \in \mathcal{N},\, k \in \mathcal{K},\, r \in \mathcal{R} \tag{3.3}$$

Equation 3.2 ensures that each vessel is visited at most once.

Equation 3.3 are arc flow constraints to ensure the route goes in a loop starting and ending at the depot.

Sub-tour elimination

Sub-tours are routes which are cyclic, but do not start and end at the terminal. This is a problem as all barges are should start and end their routes at the terminal. To eliminate this, various sub-tour elimination constraints (SEC) has been proposed to remove sub-tours. The following is a non-exhaustive list of different methods to remove sub-tours.

1. Danzig–Fulkerson–Johnson (DFJ): This method iteratively adds constraints to connect the sub-tours which exist in the solution, and solve the problem with the added constraints [32].

2. Miller–Tucker–Zemlin (MTZ): This method adds an ordering continuous decision variable at each node, to order the visits. Thus, routes can be forced to start at the terminal and consequently, to form a cycle and end at the terminal [33].

3. Gavish–Graves (GG): This method adds a flow continuous decision variable at each arc. Constraints are added such that all flows must eventually reach back to the terminal. Thus, all routes have to end at the terminal [34].

In this formulation, MTZ method is a natural extension of the problem. As time windows of each customer have to be respected by the servicing barge, a decision variable to track visit time t_i^{kr} has to be added. Consequently, this variable also acts as the ordering variable in the MTZ method.

3.6.2 Time Constraints

$$t_0^{kr} + s_0^{kr} + T_{0j} \leq t_j^{kr} + M_{0j}(1 - x_{0j}^{kr}) \quad \forall j \in \mathcal{D},\, k \in \mathcal{K},\, r \in \mathcal{R} \tag{3.4}$$

$$t_i^{kr} + S_i + T_{ij} \leq t_j^{kr} + M_{ij}(1 - x_{ij}^{kr}) \quad \forall i, j \in \mathcal{D},\, k \in \mathcal{K},\, r \in \mathcal{R},\, i \neq j \tag{3.5}$$

$$t_i^{kr} + S_i + T_{i0} \leq t_{N+1}^{kr} + M_{i0}(1 - x_{i0}^{kr}) \quad \forall i \in \mathcal{D},\, k \in \mathcal{K},\, r \in \mathcal{R} \tag{3.6}$$

$$t_{N+1}^{kr} \leq t_0^{k,r+1} \quad \forall k \in \mathcal{K},\, r \in \mathcal{R} \setminus \{N-1\} \tag{3.7}$$

$$t_{N+1}^{kr} \leq T_H \quad \forall k \in \mathcal{K},\, r \in \mathcal{R} \tag{3.8}$$

$$A_i \leq t_i^{kr} \leq B_i \quad \forall i \in \mathcal{D},\, k \in \mathcal{K},\, r \in \mathcal{R} \tag{3.9}$$

$$s_0^{kr} = \beta(FC - \sum_{f \in \mathcal{F}} (l_{N+1,f}^{k,r-1}) \quad \forall k \in \mathcal{K},\, r \in \mathcal{R} \setminus \{0\} \tag{3.10}$$

$$s_0^{k0} = 0 \quad \forall k \in \mathcal{K} \tag{3.11}$$

where FC is the full capacity of the barge
where β is the rate of fuel transfer at the depot
where $M_{ij} = \begin{cases} B_0 + FC * \beta + T_{0j} - A_j & \forall j \in \mathcal{N}, \\ B_i + S_i + T_{ij} - A_j & \forall i \in \mathcal{D},\, j \in \mathcal{N} \end{cases}$

To ensure that the constraints are linear, the Big M formulation is used. The M coefficient is tightened to as small as possible to make the formulation more efficient.

Equation 3.4 ensures that the time start at the depot t_0^{kr} at the start of the trip is before the time the barge arrives at the first vessel i including servicing and travelling time. This is separate to account for dynamic service time s_0 at the start of the trip.

Equation 3.5 ensures that ensures that the time visited at the next vessel j is after the time the barge arrives at the previous vessel i including servicing and travelling time. It allows for the barge to wait at vessel j before arriving.

Equation 3.6 ensures that the time reached the depot t^{kr}_{N+1} at the end of the trip is after the time the barge arrives at the last vessel i including servicing and travelling time.

Equation 3.7 ensures that the time start of the next trip $r+1$ at the depot is after the barge has arrived at the depot at the end of the previous trip r.

Equation 3.8 ensures that the end of all trips is before the end time for the planning window.

Equation 3.9 ensures that start A_i and end B_i time windows of the vessel are respected.

Equation 3.10 ensures that service time at the start of the trip s^{kr}_0 is the difference between the full capacity of the barge and the amount of fuel left from the previous trip multiplied by the rate fuel can be transferred onto the barge.

Equation 3.11 ensures that service time at the for the depot for the first trip is 0. It is assumed that there would be no loading of fuel at the start.

Arcs Filtering

$$A_i + S_i + T_{ij} > B_j \tag{3.12}$$

If equation 3.12 is satisfied, the arc x^{kr}_{ij} is infeasible to travel as it would not satisfy time window constraints. Thus, x^{kr}_{ij} is removed from the formulation for all $\mathcal{K}, \mathcal{R}$. This reduces the number of decision variables, making the formulation more efficient. This is especially useful for problems with tight time windows, which would result in the removal of many arcs.

3.6.3 Load Constraints

$$l^{kr}_{0f} = FC_f \quad \forall k \in \mathcal{K},\ r \in \mathcal{R} \setminus \{0\},\ f \in \mathcal{F} \tag{3.13}$$

$$u^{kr}_{if} = y^{kr}_i * N_{if} \quad \forall i \in \mathcal{N},\ k \in \mathcal{K},\ r \in \mathcal{R},\ f \in \mathcal{F} \tag{3.14}$$

$$l^{kr}_{N+1,f} = l^{kr}_{0f} - \sum_{i \in \mathcal{D}} u^{kr}_{if} \quad \forall i \in \mathcal{N},\ k \in \mathcal{K},\ r \in \mathcal{R},\ f \in \mathcal{F} \tag{3.15}$$

Equation 3.13 ensures that at the start of the trip, the barge is at full capacity. Except for the first trip where the load would be dependent on the initial conditions.

Equation 3.14 ensures that the fuel delivered to the vessel is equal to the demand if the vessel is visited by the barge. Otherwise, fuel delivered to the vessel is 0 as it is not visited by the barge.

Equation 3.15 ensures that at the end of the trip, the load left in the barge is the difference between the starting initial load and the sum of all fuel delivered to vessels in that trip.

3.7 Initial Conditions

Due to the rolling planning windows, there is a need to define the initial conditions of the various barges.

If the plan is at the start of the planning window, there would be no route constraints, start time of all barges would be at $t = 0$ and all barges would be at the full load.

If the plan is planning for subsequent planning windows, the barge would be starting at the vessel which it is at, based off the previous planning window. Similarly, the time and load constraints would be the time and load of the barge currently.

However, due to flow conservation in the model, all routes planned would have to be in a loop. It is not possible to create a route which does not end at the same place where it started. To get around this, the author proposes to add a dummy visit from the depot to the starting location. This would allow for the model to create a loop for a route that starts and end at the depot. Consequently, starting time of the barge have to be adjusted to account for the dummy travelling time from the depot to the starting customer.

3.7.1 Route Constraints

$$x^{k0}_{0,SV^k} = 1 \quad \forall k \in \mathcal{K} \tag{3.16}$$

where SV^k is the starting vessel of each vehicle k

3.7.2 Time Constraints

$$t^{k0}_{0} = ST^k \quad \forall k \in \mathcal{K} \tag{3.17}$$

where ST^k is the starting time of each vehicle k

3.7.3 Load Constraints

$$l^{k0}_{0f} = SF^k_f \quad \forall k \in \mathcal{K},\, f \in \mathcal{F} \tag{3.18}$$

where SF_f^k is the starting fuel of each vehicle k for fuel type f

Chapter 4

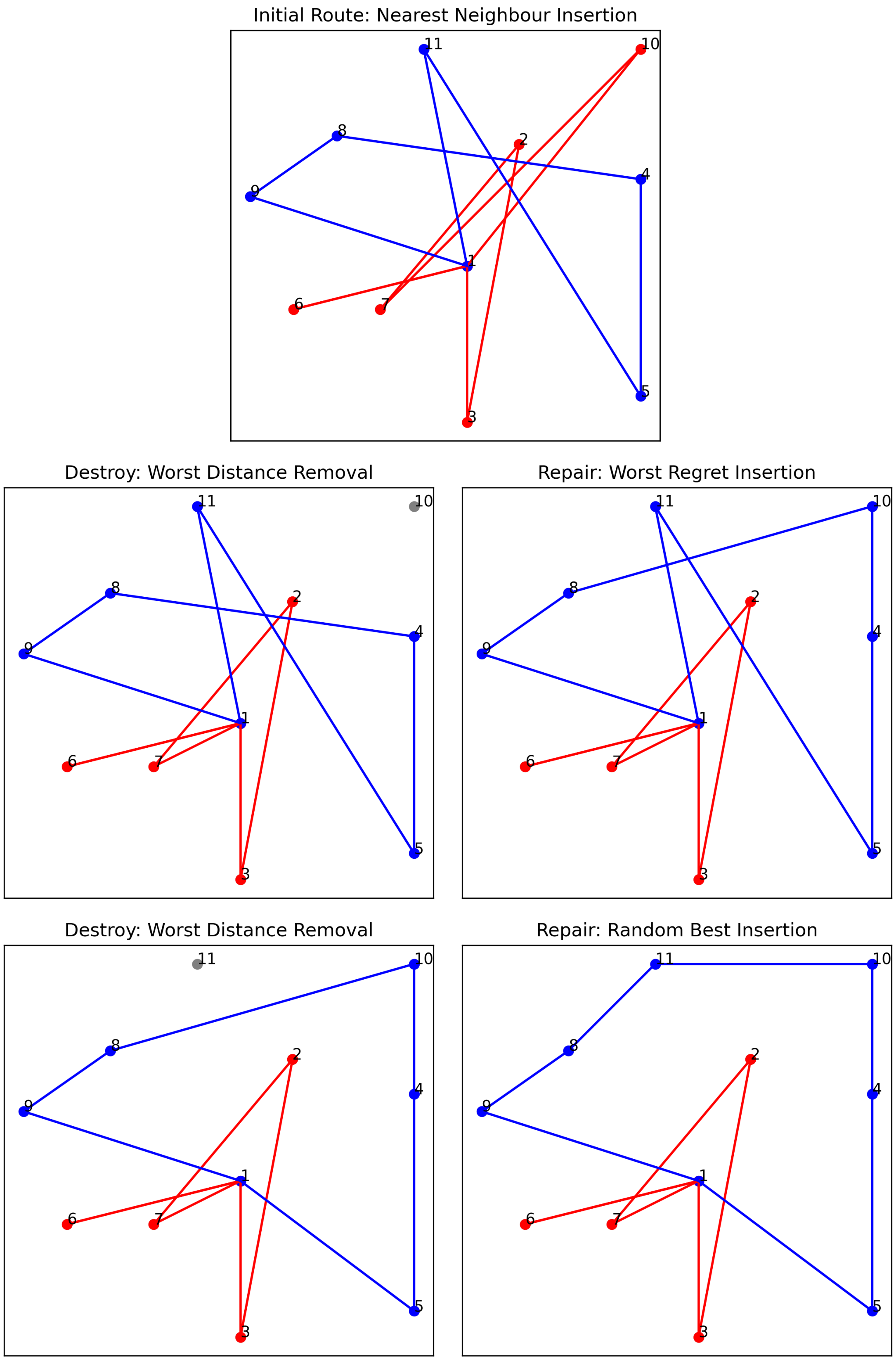
Adaptive Large Neighbourhood Search

The Adaptive Large Neighbourhood Search (ALNS) is an alternative method that can be used to solve the bunker supply problem. The outline for ALNS is defined in Algorithm 1.

In line 1, an initial solution is first constructed before the main ALNS algorithm is applied to find a good solution. The Nearest Neighbour heuristic used to create an initial route $s_{initial}$. This would be set as the candidate route s.

Subsequently, the candidate route s would undergo destroy and repair operations to create a new route s'. A destroy operator d is chosen from a set of destroy operators $d()$. An operator is chosen probabilistically from its weights w_d, with higher weighted operators having a higher likelihood of being chosen. At the start, weights of all operators would be equal. The same procedure is done to chose repair operator r from the set of repair operators $r()$ using weights w_r. The new route s' is then evaluated against s and accepted to be the next candidate route according to the acceptance criteria. Otherwise the new route s' is discarded. This cycle repeats until the stopping criteria is reached. During this process, the best found route so far is recorded as s_{max}. The creation of an initial route and 2 iterations of the repair and destroy process is illustrated in Figure 4.1.

The weights for the destroy and repair operators are updated according to their performance, controlled by the learning rate l_r parameter. It is assumed that past performance is indicative of future performance. As such weights of well performing operators are increased to make it more likely for them to be chosen in subsequent iterations, improving the performance of the overall algorithm. Such a mechanism is in place as it is unknown which operator would work best for a given problem structure (tight time windows/ many unserved customers etc.).



Initial Route: Nearest Neighbour Insertion
Destroy: Worst Distance Removal
Repair: Worst Regret Insertion
Destroy: Worst Distance Removal
Repair: Random Best Insertion
1
2
3
4
5
6
7
8
9
10
11

Figure 4.1: Creation of initial route and repair and destroy process

Thus, several operators are implemented and chosen from to allow the overall algorithm to have good performance across different problem structures.

A range of destroy and repair operators are implemented such that there is a balance of diversifying and intensifying operators. Diversifying operators are those which allow the solution to explore new regions in the search space while intensifying operators focus on improving the solution in a local region.

This process continues until the stopping criteria is met. The stopping criteria is reached when the temperature $\mathcal{T}$ reaches 0 or a certain number of non-improving steps is reached.

Algorithm 1: Adaptive Large Neighbourhood Search

```
input  : ProblemIinstance $i$
output: Best Solution $s_{max}$
1  $s \leftarrow$ InitialSolution($i$);
2  $s_{max} \leftarrow s$;
3  while stopping criteria not met do
4      select destroy method $d()$ and repair method $r()$ from $w_d, w_r$;
5      $s' \leftarrow r(d(s))$;
6      if accept($s, s'$) then
7          $s \leftarrow s'$;
8          if $c(s) < c(s_{max})$ then
9              $s_{max} \leftarrow s$;
10         end
11     end
12     update $w_d, w_r$;
13 end
```

4.1 Acceptance Criteria

The acceptance algorithm, Simulated Annealing (SA), is inspired from a physical process in metallurgy known as annealing. Temperature, $\mathcal{T}$, is used to control the randomness allowed during the search. At the start of the iteration, the process would start at a high temperature $\mathcal{T}$ value, which reduces according to the cooling schedule for each subsequent iteration. During each iteration, a new route s' is created from the candidate route s. If s' is better than s, it is accepted as the new candidate route. However, if s' is worse, it may be accepted as the new candidate route according to the Metropolis criteria as defined by equation 4.1.

$$e^{\frac{f(s')-f(s)}{\mathcal{T}}} \tag{4.1}$$

Using the current temperature and the difference of the objective function of the new and candidate route, a threshold can be generated using equation 4.1. A random number between 0 and 1 is generated and compared to the threshold. If the random number is lower than the threshold, the poorer new route is accepted. Otherwise it is discarded and there is no change to the candidate route.

As the temperature decreases, the threshold value also decreases. This decreases the probability that a poorer route is accepted and the search would converge to the local best route.

Thus, in high temperature regimes, the acceptance criteria acts like a random walk, where almost all solutions are accepted. In a low temperature regime, the criteria acts like a hill climb, with only improving solutions accepted.

Other alternative criteria may be used such as a pure hill climbing criteria, which only accepts a solution if it is better than the candidate solution. However, SA is chosen as it allows the algorithm to explore poorer solutions early in the iteration, which may have the potential to create better solutions after refinement. This prevents the algorithm from being stuck in a local maxima.

4.1.1 Cooling Schedule

For each iteration, a temperature variable $\mathcal{T}$ is recorded and updated for use in the SA acceptance criteria. $\mathcal{T}$ is initialised at a starting maximum temperature $\mathcal{T}_{max}$ and the algorithm ends when it reduces to below $\mathcal{T}_{min}$. At each iteration of the algorithm, $\mathcal{T}$ is adjusted according to:

$$\mathcal{T}_{i+1} = \alpha \mathcal{T}_i \tag{4.2}$$

where i refers to the iteration number and α is the pre-set cooling rate.

As such, the number of iterations $MaxIter$ needed for the algorithm to terminate is as follows:

$$MaxIter = \log_{\alpha} \frac{\mathcal{T}_{min}}{\mathcal{T}_{max}} \tag{4.3}$$

Other than the exponential cooling schedule proposed, alternative schedules can also be implemented, such as the linear or logarithmic schedule, which would result in lesser or more exploration at later stages of the algorithm respectively. The exponential schedule is chosen as it is the most popular and provides a good balance between early exploration of good regions at the start of the algorithm and later explorations of the local minima towards the end of the algorithm.

4.2 Operator Selection

The destroy $d()$ and repair operators $r()$ are chosen using weights w_d, w_r respectively using the roulette wheel mechanism. The probability of an operator i being chosen is given by

$$P(i) = \frac{w(i)}{\sum_{j=1}^{k} w(ij)} \tag{4.4}$$

At the start, weights of all operators are initialised to 1. After each iteration, the weights are awarded a score σ according to the Table 4.1. To ensure the scores are sensible, $\sigma_1 > \sigma_2 > \sigma_3 > \sigma_4$.

Score	**Description**
σ_1	The last iteration resulted in a new global best solution.
σ_2	The last iteration resulted in a solution that has not been accepted before. The cost of the new solution is better than the cost of the current solution.
σ_3	The last iteration resulted in a solution that has not been accepted before. The cost of the new solution is worse than the cost of the current solution, but the solution was accepted.
σ_4	The last iteration resulted in a solution has been rejected. The cost of the new solution is worse than the cost of the current solution.

Table 4.1: Score Adjustment Parameters

Subsequently, the weights of the operators used are adjusted every N_{update} iterations. N_{update} is defined using a θ_{update} parameter which sets N_{update} as a fraction of $MaxIter$.

$$N_{update} = \theta_{update} * MaxIter \tag{4.5}$$

At every N_{update} iterations, the weights are updated as follows.

$$w(i) = w(i)(1 - l_r) + l_r \frac{\pi_i}{\beta_i} \tag{4.6}$$

l_r is a pre-set learning rate factor $0 < l_r < 1$. A value of 0 results in no adjustments of weights according to performance while a value of 1 would mean only the performance of the previous iteration is considered. π_i is the sum of all scores σ

accumulated by the operator and β_i is the number of times the operator is used during N_{update} iterations.

4.3 Nearest Neighbour Search

In the Nearest Neighbour Search (NNS) the initial route is created by are adding customers which are closest from the previous visit. If addition of a customer exceeds the capacity for the trip, the customer is added to the next trip instead. This process ends when no more customers can be added to the route without exceeding the time windows. This is done until all vehicles are assigned a route.

In this implementation, the NNS is modified with the seed customer chosen to be furthest from the terminal, instead of closest to the terminal. This is done so that the route created would be moving closer towards the terminal, allowing for quick top-up of fuel if the barge needs to return back to the terminal.

Apart from NNS, other heuristics may be used to create an initial route. However, it is noted that the algorithm to produce an initial route is not intended to create a particularly optimised route. Speed and simplicity are prioritised over being able to create a good route.

Algorithm 2: Nearest Neighbour Search

```
input  : Problem Instance $i$
output: Initial Solution $s_{initial}$
1  for $k \leftarrow 0$ to $N_{barges}$ do
2      $s_{initial} \leftarrow i$ select furthest customer $i$;
3      repeat
4          $j \leftarrow$ closest customer from last customer $i$ in $s_{initial}$;
5          $s' \leftarrow s_{initial} + j$ ;
6          if FeasibleCheck($s'$) then
7              $s_{initial} \leftarrow s'$
8          end
9      until unfeasible;
10 end
```

4.4 Destroy Operators

The destroy operators take in a route and destroys it by removing customers. The number of customers removed is a random value between 1 and a pre-set maximum degree of destruction $\mathcal{D}$. To allow the algorithm to search a large neighbourhood, the maximum degree of destruction is usually set to a relatively large number. By destroying the route, we allow the removed customers to be reinserted later during the repair phase in a more favourable position or to remove inefficient customers and allow other customers to be served instead. Random Removal, Worst Distance Removal and Shaw Removal are 3 destroy operators that can be chosen.

4.4.1 Random Removal

The Random Removal (RR) operator randomly removes customers from a random barge and trip from the route. This allows for some randomness in the heuristic, to prevent the heuristic becoming too deterministic and only exploring locally.

4.4.2 Worst Distance Removal

The Worst Distance Removal (WDR) operator removes the customers which takes the longest to travel from the previous customer and to travel to the next customer. By removing the worst customer, travel distance in the solution would be reduced, allowing for other customers to be inserted in place instead.

4.4.3 Shaw Removal

Shaw Removal (SR) is named after it's creator [20] and removes customers which are closely related. The relatedness between two customers can be determined using a relatedness equation 4.7. a and b are parameters to be set. The equation measures the distance between two points and the difference between their start and end time windows. A lower score would indicate that the 2 customers are closely related. This can be seen as opposite to WDR which would remove customers with the greatest distance.

$$a * distances[i, j] + b * (\Delta starttimewindow + \Delta endtimewindow) \tag{4.7}$$

To start the procedure, a random visited customer is chosen. Next, the relatedness score of all other customers is calculated with reference to the chosen customer. Lastly, the customers with the lowest relatedness score, along with the initial chosen customer is removed, up to the degree of destruction. By removing closely related customers, these would allow them to be reinserted during the repair step, where they are visited in succession.

4.5 Repair Operators

The repair operators take in a destroyed route and repairs it by adding customers into the route until no more customers can be added feasibly. This is allows the route to be quickly improved until no further improvements can be made (local minima). Greedy Best Insertion, Random Best Insertion and Worst Regret Insertion are 3 repair operators that can be chosen.

4.5.1 Greedy Best Insertion

In Greedy Best Insertion (GBI) the best customer from the pool of unserved customers is inserted into the best position. The route is then updated, and the best customer is removed from the pool. The next best customer is then inserted and the process continues until no customer is available to be added such that the solution would improve. This greedy operator allows for the solution to be improved quickly, guiding the search to quickly converge on a solution. Psuedocode can be found in Algorithm 3.

In the algorithm, each unserved customer x is inserted into all possible positions in the given route s and the increase in objective function is recorded. This list is sorted and the insertions are checked to be feasible, starting from the insertion that results in the largest increase in objective function. Once a feasible solution is found, the newly inserted customer is removed from the unserved customer list. This process repeats until there are no more customers which can be feasibly inserted into the route.

Algorithm 3: Greedy Best Insertion

```
input  : Destroyed Candidate Solution $s$
output: Repaired New Solution $s'$
1  repeat
2      $\mathcal{S} \leftarrow$ ObjDiff(for all unserved customers in all positions in $s$) ;
3      Sort $\mathcal{S}$ in increasing order;
4      for $s' \leftarrow 0$ to $\mathcal{S}$ do
5          if FeasibleCheck($s'$) then
6              remove $x$ from unserved customers;
7              $s \leftarrow s'$;
8              break
9          end
10     end
11 until no unserved customers;
```

4.5.2 Random Best Insertion

In Random Best Insertion (RBI) a random customer from the pool of unserved customers is inserted into the best position. This is in contrast to GBI where the best customer is inserted into the best position. The route is then updated, and the random customer is removed from the pool. This process continues until no customer is available to be added such that the solution would improve. This operator is implemented as an improvement to GBI, which is myopic inserts in the best customer in earlier insertions and sacrifice subsequent insertions. To avoid this, a random customer is selected to be inserted instead, avoiding short term insertions.

4.5.3 Worst Regret Insertion

For Worst Regret Insertion (WRI) the customer with the highest Regret Value is added to the solution. The regret value of a customer is calculated as the difference between the best and second best insertion position of the customer. As such, a customer with low regret would mean that there are at least 2 positions where it can be inserted with similar increase in objective function. In contrast a customer with high regret would mean that if we do not insert in the customer in the current iteration, it would be more difficult to insert the customer subsequently. Thus, priority is given to customers with the highest regret value. This continues until no improving insertions can be made. Similarly, insertions are done until there are no more customers which can be feasibly inserted.

4.5.4 Feasibility Check

During the repair process, the algorithm would have to check feasibility of the routes for each possible insertion location of each customer, for both capacity and time constraints. For capacity constraints, this is easily checked as it is a simple addition of all demands for all visited customers in a trip. For time constraints, in a naive approach, a simulation of the route is done, to check if visits are within time windows and to update the time accordingly if it is before the start time window. This takes $O(n)$ time where n refers to the number of customers served.

To speed up this process, the idea of time slacks has been introduced by Nagata et al. [35]. Time slacks is the amount of time available at an insertion location, such that a delay would not make the route infeasible. For instance, for customer x which is directly after insertion location i, the barge arrives at x at 100 time units before the end time window, before any insertion of any customers at i. Thus, the time slack ts_x available is 100, as we can arrive up to 100 time units later at x without breaching time windows. As the barge can wait before serving customers, this waiting time wt_x also needs to be taken into consideration. If we arrive 100 time units early at x, the total time slack available at customer x would be 200.

Similarly, at the next vessel y, ts_y is 50 and wt_y is 100. As such, the total time slack available would be the addition of the time slack and waiting time at y and the waiting time of x, for a total of 250. What this means is that we can insert in a customer at position i before customer y such that it causes a time delay of at most 250 time units before it is infeasible to visit y. However, do note that this means that customer x would become infeasible to visit, as it can only accept a time delay of at most 200. Hence to ensure that the route remains feasible, the time slack TS_i^r of an insertion location is the minimum of all total time slacks of customers after i and is given as the equation 4.8.

$$TS_i^r = min([ts_x + \sum_{i=i}^{x} wt_i]) \tag{4.8}$$

When a customer a is inserted into location i, this results in a delay for all subsequent customers, as they can only be served later. This time delay, TD_r, can be calculated according to equation 4.9. a refers to the inserted vessel, while x, y refers to the vessel before and after the inserted vessel a. W_a refers to the waiting time before the inserted vessel is served.

$$TD_r = S_a + T_{xa} + T_{ay} - T_{xy} + W_a \tag{4.9}$$

However, by inserting customer a, additional servicing time is needed at the terminal to top up the additional bunker delivered before the next trip starts. Thus, the time delay, TD_{r1} for all subsequent trips is given by equation 4.10.

$$TD_{r1} = TD_r + \beta S_a \tag{4.10}$$

Lastly, there is an edge scenario where there is enough time slack in the original trip to cover the delay due to the insertion of an additional customer. However, there is not enough time slack to cover the delay due to the additional time needed to refuel the barge at the terminal.

With the concept of time slacks, the time slacks of a route can be calculated for each insertion location beforehand. The time delay due to insertion of a customer at a location can then be compared with the pre-calculated time slack values for the route to check if there is sufficient time slack to insert in this new potential customer. This prevents the need to simulate each route for every inserted position to determine if it is feasible to insert the customer into a particular location. The time complexity to check if insertion of a customer into a route is feasible is constant, greatly reducing computational time.

4.6 Stopping Criteria

As mentioned in section 4.1, after every iteration, the temperature would be reduced by the cooling rate α. When the temperature reaches the minimum temperature T_{min}, the algorithm would end. This is the primary method for terminating the algorithm.

4.6.1 Non-Improvements

During the search process, the solution may be stuck in a local optima and become unable to escape. Thus, a restart procedure is included to help escape this local optima. After a number of non-improvement iterations N_{nonImp} , the candidate solution s would be reset to the best recorded solution s_{max}. N_{nonImp} is calculated using equation 4.11 and is controlled by parameter θ_{nonImp} which sets N_{nonImp} as a fraction of $MaxIter$.

$$N_{nonImp} = \theta_{nonImp} * MaxIter \tag{4.11}$$

As the acceptance criteria may accept worse solutions, the candidate solution is usually not the best recorded solution. With the restart procedure, the algorithm can restart it's search from the best route found, allowing it to hopefully find better routes. However, if the algorithm is still unable to find better solutions even after the restart, the algorithm would terminate prematurely. This improves the efficiency of the algorithm, as such a scenario would suggest that the algorithm has already found the best possible route.

Chapter 5

Methodology & Results

5.1 Test Problem Generation

The Solomon Benchmark [36] contains problems random (R), clustered (C) and randomly clustered (RC). These describes how the customers are scattered around the terminal. These problems can be further broken down into type 1 and type 2 problems. Type 1 problems have a short scheduling horizon, allowing for only a few customers to be serviced by one vessel. In contrast, Type 2 problems have long scheduling horizon allowing many customers to be serviced by a vessel. Lastly, the problems are numbered from 1 upwards. Smaller numbered problems have customers with tight time windows, and larger numbered problems have time windows which are large. For example for problem R201, the first digit means that the customers are randomly scattered. The second digit represents that it is problem type 2 and the last 2 digits represent that the time windows are very tight for this problem.

Only type 2 problems are used for testing as the long scheduling horizon allows the barges to take multiple trips. To ensure problems are representative, only those with very tight or very loose time windows are used for testing.

However, the Solomon dataset is not able to fully replicate the bunker supply problem. It only has one demand type. To remedy this for each customer, fuel demand is randomly assigned to 1 of 5 possible fuel types. Also, service time is taken to be dependent on the total amount of fuel demanded by the vessel, instead of being a constant value for all vessels.

5.2 Implementation

Both the mathematical formulation and Adaptive Large Neighbourhood Search (ALNS) heuristic is implemented in the Python Programming Language. Both algorithms are run on a personal computer with CPU Intel i7-10510U. Table 5.1 gives values for the problem parameters.

Parameter	Description	Value
ζ	Cost per unit time to operate vessel	0.1
$Price_f$	Price of each fuel type f	$[1, 1, 1, 1, 1]$
β	Rate of fuel transfer at depot compared against when at vessel	0.2
$\mathcal{K}$	Number of barges available	3
Q_f	Capacity of each compartment in barge carrying fuel type f	$[50, 50, 50, 50, 50]$

Table 5.1: Problem Parameters

5.2.1 MILP Implementation

The mathematical formulation is implemented using a solver agnostic library Pyomo. A solver agnostic library is chosen as it would allow for testing of various different solvers to find the best one. Gurobi version 12.0.0 is chosen as the solver as preliminary tests show it to perform the best, amongst a list of commercial and open source solvers, such as CPLEX, HiGHS and GLPK. This is within expectations as commercial solvers are known to be more performant than open source ones. To ensure fairness when comparing against ALNS, the solver is limited to using only one thread of the CPU. To ensure reasonable computing times, the solver is set to a time limit of 1 hour or 3600s. All other solver settings are kept at the default.

5.2.2 ALNS Implementation

Due to the large number of parameters, it is difficult and time consuming to tune all the parameters. Thus, values for various parameters for ALNS are chosen according to previous literature [31]. Table 5.2 provide a summary of all the parameters used for the ALNS algorithm. Due to randomness present in ALNS, results would differ slightly for repeated runs on the same problem. Thus, the average solution and time taken for 5 runs of the algorithm is reported.

Parameter	**Description**	**Value**
$\mathcal{T}_{max}, \mathcal{T}_{min}$	Starting and Ending Temperature for SA	$[50, 10^{-3}]$
α	Cooling rate for SA	0.995
l_r	Learning rate	0.1
$\mathcal{D}$	Maximum Degree of Destruction	0.3
$\sigma_1, \sigma_2, \sigma_3, \sigma_4$	Scores for updating operator weights	$[1, 0.5, 0.25, 0]$
θ_{update}	Number of iterations to update operator weights	0.005
θ_{nonImp}	Number of non-improving iterations	0.2
a, b	Shaw parameters	$[1, 1]$

Table 5.2: ALNS Parameters

5.3 Experiment Design

The quality of the solution would be based on a few metrics, mainly, objective function, solution time and Gap from best bounds (for MILP).

To ensure that the developed methods are effective, the methods would be put through 3 sets of experiments.

1. Methods are tested on a set of 6 different test cases, each test case hand crafted to test a particular constraint.

2. Methods are tested on the modified version of the Solomon Benchmark as described in Section 5.1. The formulation is tested on small, medium and large instances with 25, 50 and 100 customers respectively.

3. ALNS is compared against best known solutions (BKS) from a similar problem. They are tested on small, medium and large instances with 25, 50 and 100 customers respectively.

The T_B column in MILP refers to the time taken to find the best solution, while T is the time taken for the algorithm to terminate. Gap_o refers to the gap in the solution between the ALNS and MILP methods. A negative Gap_o would indicate that the solution found by ALNS is worse than MILP. Gap_o is calculated such that it is comparable to Gap found by MILP. As such Gap_o should always be lower than Gap, as it is impossible to be better than upper bound solution found by MILP.

5.4 Experiment 1

The objective of this experiment is to prove that both methods are able to provide good solutions for simple test cases. The 6 test cases are described in Table 5.3.

Test Case	Name	Description
1	Travelling Salesman Problem (TSP)	To test if 1 barge can find the shortest route to visit 6 customers arranged in a circle around the terminal
2	Vehicle Routing Problem (VRP)	To test if 3 barges are able to find the optimal route to 18 customers
3	Multi-Trip Vehicle Routing Problem (MTVRP)	To test if 1 barge is able to perform multiple trips to 18 customers
4	Multi-Commodity Vehicle Routing Problem (MCVRP)	To test if 3 barges are able to deliver to customers with different fuel types
5	Vehicle Routing Problem with Time Windows (VRPTW)	To test if 3 barges are able to deliver to customers with tight time windows, which define the route
6	Greedy Vehicle Routing Problem (GVRP)	To test if 1 barge is able to select the best customers to deliver to when it is not possible to service all customers

Table 5.3: Experiment 1 Test Cases

It is expected that both methods are able to quickly solve all 6 test cases, as they are set-up to be relatively simple. The results are given in Table 5.4 and the solutions found by ALNS of the test problems is in Figure 5.1.

It can be seen that both MILP and ALNS methods are able to given similar results for the test cases. MILP is able to prove optimality solution for 3 out of the 6 test cases. The results prove that both methods are able to handle the constraints in the bunker supply problem appropriately.

5.5 Experiment 2

The objective of this experiment is to test the relative performance between the 2 methods. This would allow us to compare the 2 methods and determine the

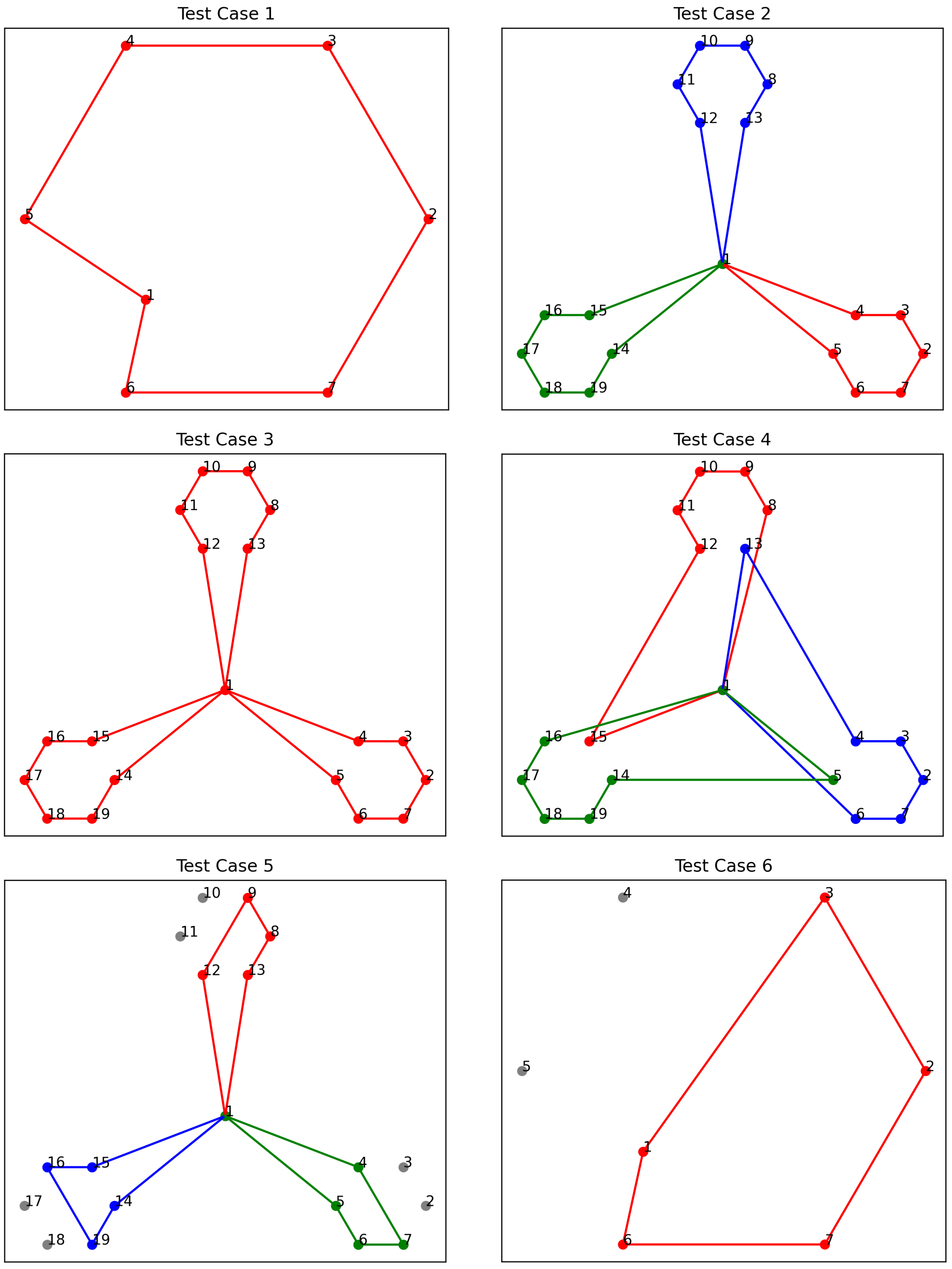


Figure 5.1: Experiment 1 routes obtained from ALNS

Test	**MILP**				**ALNS**		
Case	Solution	T_B(s)	Gap(%)	T(s)	Solution	T(s)	Gap_o(%)
1	56.94	0.02	0.00	0.02	56.94	0.61	0.00
2	163.12	1	0.00	61.79	163.12	1.95	0.00
3	163.12	2	2.80	180	163.12	1.72	0.00
4	155.70	15	5.04	180	155.80	2.85	0.06
5	104.59	56	61.15	180	104.59	2.35	0.00
6	37.52	0.04	0.00	0.04	37.52	0.71	0.00

Table 5.4: Results for Experiment 1

strengths and weakness between the 2 approaches. The results are given in table 5.5 and a sample of the results obtained using ALNS for small test cases are shown in Figure 5.2.

Test	**N**	**MILP**				**ALNS**		
Case		Solution	T_B(s)	Gap(%)	T(s)	Solution	T(s)	Gap_o(%)
C201	25	432.81	3	0.00	77.66	432.81	3.26	0.00
C208	25	432.88	377	1.43	3600	432.88	4.13	0.00
R201	25	284.34	2	0.00	163.04	282.97	4.13	−0.48
R211	25	295.46	2259	1.71	3600	294.61	3.78	−0.29
RC201	25	490.51	87	0.00	555.46	487.10	5.07	−0.70
RC208	25	494.95	1640	6.65	3600	496.55	4.21	0.32
C201	50	795.13	3430	2.76	3600	796.62	13.31	0.19
C208	50	796.05	3349	3.60	3600	803.97	15.01	1.00
R201	50	615.14	3069	4.05	3600	627.58	12.69	2.02
R211	50	647.15	3420	4.38	3600	657.48	12.21	1.60
RC201	50	734.09	2774	21.21	3600	794.32	18.65	8.21
RC208	50	799.39	2978	18.65	3600	884.22	14.31	10.61
C201	100	1360.83	3561	28.22	3600	1593.14	85.25	17.07
C208	100	1253.37	3560	39.99	3600	1681.41	68.87	34.15
R201	100	830.81	3140	61.76	3600	1040.57	87.83	25.25
R211	100	776.26	3130	79.78	3600	1328.46	77.23	71.14
RC201	100	853.46	3378	86.77	3600	1035.33	87.96	21.31
RC208	100	801.67	3556	107.58	3600	1338.63	94.36	66.98

Table 5.5: Results for Experiment 2

For small test instances ($N = 25$) both methods are able to find optimal or close to the optimal solution however, solution time is much more stable for ALNS, varying between 3.26 and 5.07 seconds. In contrast MILP takes betwen 77.66 and 3600 seconds to terminate.

For medium test instances ($N = 50$) ALNS in general slightly outperforms MILP method, being able to find slightly better solutions for all test cases in

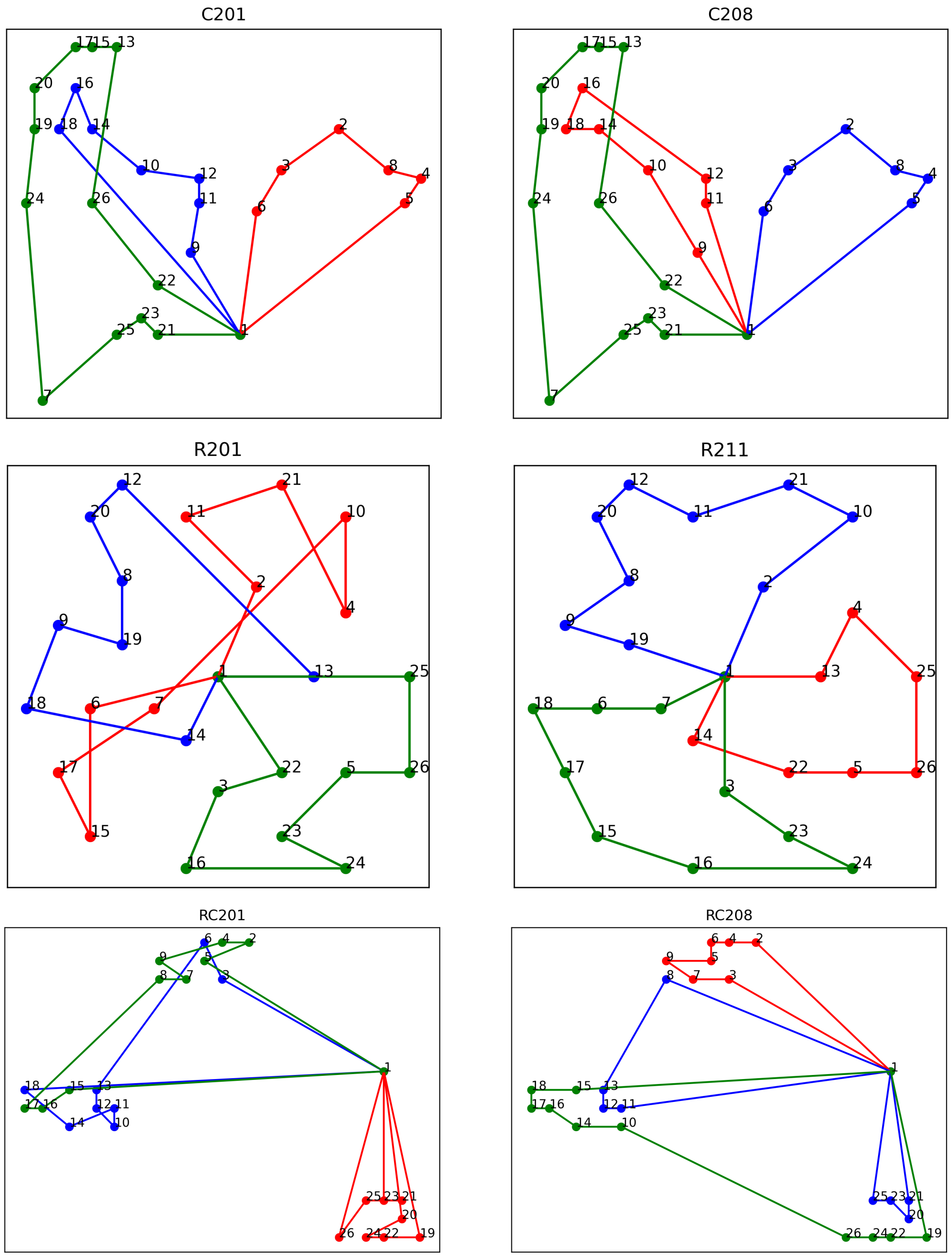


Figure 5.2: Experiment 2 routes obtained from ALNS for small test cases

much quicker time. However, MILP is able to show that the solutions found are still relatively close to optimal for at least 4 of the 6 test cases.

For large test instances ($N = 100$) ALNS has a clear advantage over MILP in terms of both solution quality and solution time. However, due to the large upper bound gap produced by MILP, it is difficult to ascertain if the solutions produced by ALNS is close to the optimal solution.

These results show that ALNS is the superior method with its advantage increasing as the test problem size increases.

5.6 Experiment 3

The objective of this experiment is to test the performance of ALNS against the best known solutions (BKS) [37] to evaluate its effectiveness. Given the results from Experiment 2, the ALNS method is proven to be superior and is thus used as a point of comparison against Best Known Solutions (BKS). Specifically, published results from the Capacitated Multi-Trip Vehicle Routing Problem with Time Windows - Loading Times (CMTVRPTW-LT) problem is used. This problem is chosen as a benchmark as it most closely resembles the bunker supply problem that the author has found. However, due to the novel set of constraints in the bunker supply problem, there are differences between the published problem and the bunker supply problem. The differences are compared in Table 5.6.

Bunker Supply Problem	**Published Problem**
Objective function to maximise profit, implying that not all customers may be visited due to constraints	Objective function to minimise travel distance, this assumes that all customers must be visited
Multiple different fuel types demanded by vessels	Only 1 demand type from customers
Service time at vessel is dependent on amount of fuel demanded	Service time is the same for all customers

Table 5.6: Differences between bunker supply problem and published problem in Experiment 3

To encourage the ALNS method in the bunker supply problem to visit all customers, the price of all bunker fuels is increased to 10. This ensures that the optimal solution should be the route which visits all customers. To ensure a fair comparison, solutions produced by ALNS is checked to ensure that it visits all customers, before the distance travelled is calculated and admitted as a valid solution. Thus, the comparison of the route's distances from both methods is fair. For the bunker supply problem, the demand for the first fuel type is the demand and the rest of the fuel types is 0. This reduces the multiple different fuel types to just one. as service time at the vessel is calculated beforehand, the calculation is changed to be the same fixed value as the published problem. Following the parameters in the published problem, capacity of the first compartment is 100, and 2, 4 and 8 barges are used for problems with 25, 50 and 100 customers respectively.

The results are presented in Table 5.7. Note that the objective is to minimise total distance travelled to visit all 100 customers. As such, the lower the value of the solution, the better the solution. Consequently, a positive Gap_o would mean that the ALNS solution is worse than the BKS.

Test Case	**N**	**BKS**			**ALNS**		
		Solution	T(s)	Gap(%)	Solution	T(s)	Gap_o(%)
C201	25	380.8	2.1	0.00	387.14	6.02	1.66
C208	25	360.9	5.9	0.00	380.34	5.00	5.39
R201	25	554.6	5.8	0.00	567.30	5.06	2.29
R211	25	400.1	4.0	0.00	405.56	4.84	1.36
RC201	25	660.0	1.0	0.00	745.50	5.98	12.95
RC208	25	506.4	5.0	0.00	518.78	5.55	2.44
C201	50	714.2	152.9	0.00	737.58	26.52	3.27
C208	50	688.6	132.6	0.00	719.22	23.55	4.45
R201	50	909.8	65.8	0.00	946.56	23.24	4.04
R211	50	716.8	10 800.4	0.73	747.00	23.45	4.21
RC201	50	1096.6	4.8	0.00	1208.40	25.91	10.20
RC208	50	916.8	266.2	0.00	937.42	23.84	2.25
C201	100	1509.5	712.7	3.62	1576.06	114.61	4.41
C208	100	1451.9	1850.8	0.00	1541.98	112.56	6.20
R201	100	1403.1	1384.2	0.00	1485.96	164.80	5.91
R211	100	1178.4	2325.7	2.64	1260.94	176.45	7.00
RC201	100	1809.5	95.6	0.00	1960.00	196.42	8.32
RC208	100	1572.7	10 800.4	0.59	1748.18	165.17	11.16

Table 5.7: Results for Experiment 3

From the results, the implemented ALNS method performs poorly in all test cases and in the worst case performs 13% poorer that the BKS, with it performing the worst for RC problems. This can be due to the ALNS having to search a larger search space since it does not assume the need to visit all customers. However, as ALNS is non-deterministic, in one of the golden runs for solving a large C201 problem, ALNS is able to find a solution that improves upon BKS. Details for this solution can be found in Appendix A. ALNS performs slower than BKS for small instances, while being faster for medium and large instances. For BKS, the small and medium instances are solved using similar hardware as ALNS, but for the large instance, the BKS uses significantly more powerful hardware to run their algorithm. As such, comparison of time taken for both algorithms is not a strong indication of their relative speeds.

5.7 Discussion

Results from experiment 1 shows that both methods are viable to produce good routes for simple bunker supply problems. Results from experiment 2 suggests that in practical applications, the ALNS method is more suitable as it is able to quickly find good or even optimal solutions in reasonable time. Solution time for the ALNS method is also has a smaller variance between problem types, making it more reliable. However, the MILP method still has its applications in checking the performance of the ALNS method, especially for small and medium instances, as MILP is able to provides a reasonable upper bound to the best possible solution. In practical applications, MILP can be used to retrospectively evaluate solutions generated by ALNS, to ensure that they are good. Overall, the results obtained are within expectations, as heuristic methods are generally known to be faster than exact methods, at the sacrifice of being able to prove optimality.

However, experiment 3 suggest that there are still areas for improvement in the ALNS method. Comparing against BKS, ALNS is generally able to find a solution quicker than BKS. However, solution quality can be lacking, especially for RC problem types. Perhaps better destroy and repair operators can be developed for RC problem types. Alternatively,a similar BCP method can be implemented to solve the bunker supply problem. It is likely that it would be able to produce better solutions and prove that they are optimal. In many practical applications, proving optimality is secondary. However, as the bunker supplier is likely in competition with many other suppliers, there is likely some interest in finding the most efficient route, to ensure that they are at least as competitive if not more competitive that the competition. As such, the BCP method could be explored, as it is able to proof optimality while keeping solution times relatively reasonable. However, the search space that ALNS is searching is fundamentally larger than that of the BKS, as it accepts a larger combination of customers to be visited and has to account for multiple compartments in the barge. Hence, the relative lack of performance by ALNS is not unsurprising. Thus, conclusions drawn from experiment 3 should be considered as preliminary.

Chapter 6

Summary & Conclusion

6.1 Conclusion

The methods presented show that they are effective in solving the bunker supply problem. The MILP method proposed is shown to be able to find good solutions for small and medium problems, but is only able to do so in reasonable time for small problems. In contrast, the ALNS method is both effective and practical for all problem sizes tested. By being able to adequately solve problem sizes of up to 100 customers, this project is a significant advancement from previous students' work which is only able to solve for up to 20 customers. Furthermore, the proposed ALNS method is able to improve upon one of best known solutions from the benchmark. While the ALNS method is generally superior to MILP, it comes at the cost of being able to prove optimality. As such, while ALNS can be used for practical daily planning, MILP can be used to retroactively check routes planned by ALNS, to ensure that they are close to optimal.

6.2 Future Work

However, other novel methods, such as machine learning or even quantum methods may also provide speed-ups in solving the problem. However, according to the no-free-lunch principle, these other methods may have other drawbacks, and it may not be immediately clear which of them would be the most suitable.

Additionally, the author has identified a few ways to improve on the proposed methods.

1. Use of more efficient MILP formulations of the VRP, such as the 3 index or 2 index formulations in [9], which can reduce the number of decision variables and improve solve times.

2. Use of advanced exact methods like Branch Cut and Price to solve the problem.

3. Use of hybrid methods like matheuristics to solve the problem. Combining both linear programming and heuristic methods can be done which may result in a better method. For instance, one of the repair operators may involving using the MILP engine to find an optimal solution, for a reduced sub-problem.

4. Further experimentation of alternative destroy and repair operators, such as destroying routes by Kruskal clusters to find operators which are more performant.

5. Hyperparameter tuning of ALNS, to find better parameters to suit the problem type bunker suppliers are facing.

6. More efficient algorithmic implementation of ALNS or implementation in faster compiled programming languages like Rust.

More research can be done into these other methods to determine if they are suitable. The work done in this report in the previous sections can be used as a basis of comparison to determine their performance.

Bibliography

[1] "Products from petroleum, synthetic and renewable sources — Fuels (class F) — Specifications of marine fuels," May 2024. [Online]. Available: https://www.iso.org/standard/80579.html

[2] "Cautious Optimism for 2024 Amidst New Highs in Vessel Arrival Tonnage, Container Throughput, Bunker Sales and Tonnage Registration in 2023," Jan. 2024. [Online]. Available: https://www.mpa.gov.sg/docs/mpalibraries/media-releases/for-immediate-release-mpa-media-release_2023-maritime-singapore-performance_final.pdf?sfvrsn=9dbccd9a_0#:~:text=51.82%20million%20tonnes%20of%20bunker,community%20as%20a%20bunkering%20hub.

[3] "List of licensed bunker suppliers in the Port of Singapore," Aug. 2024. [Online]. Available: https://www.mpa.gov.sg/docs/mpalibraries/mpa-documents-files/oms/bunkering/bunkering-services-providers/list-of-licensed-bunker-suppliers---1-aug-2024.pdf?sfvrsn=68bc4ab6_0

[4] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, no. 1, pp. 80–91, Oct. 1959. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/mnsc.6.1.80

[5] J. K. Lenstra and A. H. G. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, Jun. 1981. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/net.3230110211

[6] M. W. P. Savelsbergh, "Local search in routing problems with time windows," *Annals of Operations Research*, vol. 4, no. 1, pp. 285–305, Dec. 1985. [Online]. Available: http://link.springer.com/10.1007/BF02022044

[7] J. Wang, "Optimization for Scheduling of Bunker Suply Operations," 2019.

[8] N. T. Ho, "Development of Optimal Scheduling System for Bunker Supply Operations."

[9] D. Cattaruzza, N. Absi, and D. Feillet, "Vehicle routing problems with multiple trips," *Annals of Operations Research*, vol. 271, no. 1, pp. 127–159, Dec. 2018. [Online]. Available: http://link.springer.com/10.1007/s10479-018-2988-7

[10] W. Gu, C. Archetti, D. Cattaruzza, M. Ogier, F. Semet, and M. G. Speranza, "Vehicle routing problems with multiple commodities: A survey," *European Journal of Operational Research*, vol. 317, no. 1, pp. 1–15, Aug. 2024. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0377221723008731

[11] M. Christiansen, K. Fagerholt, N. P. Rachaniotis, I. Tveit, and M. V. Øverdal, "A Decision Support Model for Routing and Scheduling a Fleet of Fuel Supply Vessels," in *Computational Logistics*, F. Corman, S. Voß, and R. R. Negenborn, Eds. Cham: Springer International Publishing, 2015, vol. 9335, pp. 46–60, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-24264-4_4

[12] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck, "A Generic Exact Solver for Vehicle Routing and Related Problems," in *Integer Programming and Combinatorial Optimization*, A. Lodi and V. Nagarajan, Eds. Cham: Springer International Publishing, 2019, vol. 11480, pp. 354–369, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-030-17953-3_27

[13] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud, "Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 249, no. 2, pp. 551–559, Mar. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S037722171500795X

[14] G. B. Dantzig and P. Wolfe, "Decomposition Principle for Linear Programs," *Operations Research*, vol. 8, no. 1, pp. 101–111, Feb. 1960. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/opre.8.1.101

[15] G. Gamrath and M. E. Lübbecke, "Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs," in *Experimental Algorithms*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C.

Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and P. Festa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 6049, pp. 239–252, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-13193-6_21

[16] J.-F. Puget, "Object oriented constraint programming for transportation problems," in *IEE Colloquium on Advanced Software Technologies for Scheduling*, 1993, pp. 4/1–413.

[17] R. Gedik, E. Kirac, A. Bennet Milburn, and C. Rainwater, "A constraint programming approach for the team orienteering problem with time windows," *Computers & Industrial Engineering*, vol. 107, pp. 178–195, May 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360835217301134

[18] A. M. Turing, "Computing Machinery and Intelligence," *Mind, New Series*, vol. 59, no. 236, pp. 433–460, 1950. [Online]. Available: http://www.jstor.org/stable/2251299

[19] S. Ropke and D. Pisinger, "An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows," *Transportation Science*, vol. 40, no. 4, pp. 455–472, Nov. 2006. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/trsc.1050.0135

[20] P. Shaw, "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems," in *Principles and Practice of Constraint Programming — CP98*, G. Goos, J. Hartmanis, J. Van Leeuwen, M. Maher, and J.-F. Puget, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, vol. 1520, pp. 417–431, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/3-540-49481-2_30

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Aug. 2023, arXiv:1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762

[22] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," Jan. 2017, arXiv:1506.03134 [stat]. [Online]. Available: http://arxiv.org/abs/1506.03134

[23] W. Kool, H. v. Hoof, and M. Welling, “Attention, Learn to Solve Routing Problems!” Feb. 2019, arXiv:1803.08475 [stat]. [Online]. Available: http://arxiv.org/abs/1803.08475

[24] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, Aug. 2018, arXiv:1801.00862 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1801.00862

[25] E. Farhi, J. Goldstone, and S. Gutmann, “A Quantum Approximate Optimization Algorithm,” Nov. 2014, arXiv:1411.4028 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1411.4028

[26] U. Azad, B. K. Behera, E. A. Ahmed, P. K. Panigrahi, and A. Farouk, “Solving Vehicle Routing Problem Using Quantum Approximate Optimization Algorithm,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 7564–7573, Jul. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9774961/

[27] B. Barak and K. Marwaha, “Classical algorithms and quantum limitations for maximum cut on high-girth graphs,” *LIPIcs, Volume 215, ITCS 2022*, vol. 215, pp. 14:1–14:21, 2022, arXiv:2106.05900 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2106.05900

[28] L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai, and L. Shi, “Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends,” Nov. 2021, arXiv:2111.06257 [cs]. [Online]. Available: http://arxiv.org/abs/2111.06257

[29] V. Schmid, K. F. Doerner, R. F. Hartl, and J.-J. Salazar-González, “Hybridization of very large neighborhood search for ready-mixed concrete delivery problems,” *Computers & Operations Research*, vol. 37, no. 3, pp. 559–574, Mar. 2010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0305054808001330

[30] S. Gualandi and F. Malucelli, “Constraint Programming-based Column Generation,” *Annals of Operations Research*, vol. 204, no. 1, pp. 11–32, Apr. 2013. [Online]. Available: http://link.springer.com/10.1007/s10479-012-1299-7

[31] V. F. Yu, N. Y. Salsabila, A. Gunawan, and A. N. Handoko, “An adaptive large neighborhood search for the multi-vehicle profitable tour

problem with flexible compartments and mandatory customers," *Applied Soft Computing*, vol. 156, p. 111482, May 2024. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1568494624002564

[32] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a Large-Scale Traveling-Salesman Problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, Nov. 1954.

[33] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," *Journal of the ACM*, vol. 7, no. 4, pp. 326–329, Oct. 1960. [Online]. Available: https://dl.acm.org/doi/10.1145/321043.321046

[34] B. Gavish and S. Graves, "THE TRAVELLING SALESMAN PROBLEM AND RELATED PROBLEMS," Jul. 1978.

[35] Y. Nagata, O. Bräysy, and W. Dullaert, "A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows," *Computers & Operations Research*, vol. 37, no. 4, pp. 724–737, Apr. 2010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0305054809001762

[36] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research*, vol. 35, no. 2, pp. 254–265, Apr. 1987. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/opre.35.2.254

[37] Y. Yang, "An Exact Price-Cut-and-Enumerate Method for the Capacitated Multitrip Vehicle Routing Problem with Time Windows," *Transportation Science*, vol. 57, no. 1, pp. 230–251, Jan. 2023. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/trsc.2022.1161

Appendix A

Detailed Solutions

Test Case: C201
Number of Customers: 100
Number of Vehicles: 8
Capacity: 100
Route Cost: 1502.2
Vehicle 1: [1, 2, 100, 96, 8, 4, 5, 90, 1, 92, 89, 85, 84, 83, 91, 1]
Vehicle 2: [1, 68, 64, 63, 1, 101, 98, 93, 95, 99, 1, 56, 58, 41, 45, 47, 1, 16, 18, 14, 10, 1]
Vehicle 3: [1, 21, 25, 30, 33, 34, 1, 66, 50, 55, 54, 88, 1]
Vehicle 4: [1, 94, 6, 76, 3, 1, 46, 52, 51, 53, 48, 44, 43, 42, 49, 1]
Vehicle 5: [1, 75, 73, 62, 65, 67, 70, 1, 69, 57, 59, 61, 60, 1, 19, 20, 17, 15, 13, 1]
Vehicle 6: [1, 7, 32, 36, 38, 37, 35, 1, 77, 72, 71, 74, 81, 80, 97, 1]
Vehicle 7: [1, 39, 40, 29, 27, 24, 1, 87, 86, 82, 79, 78, 1]
Vehicle 8: [1, 23, 28, 31, 26, 12, 11, 1, 9, 22, 1]

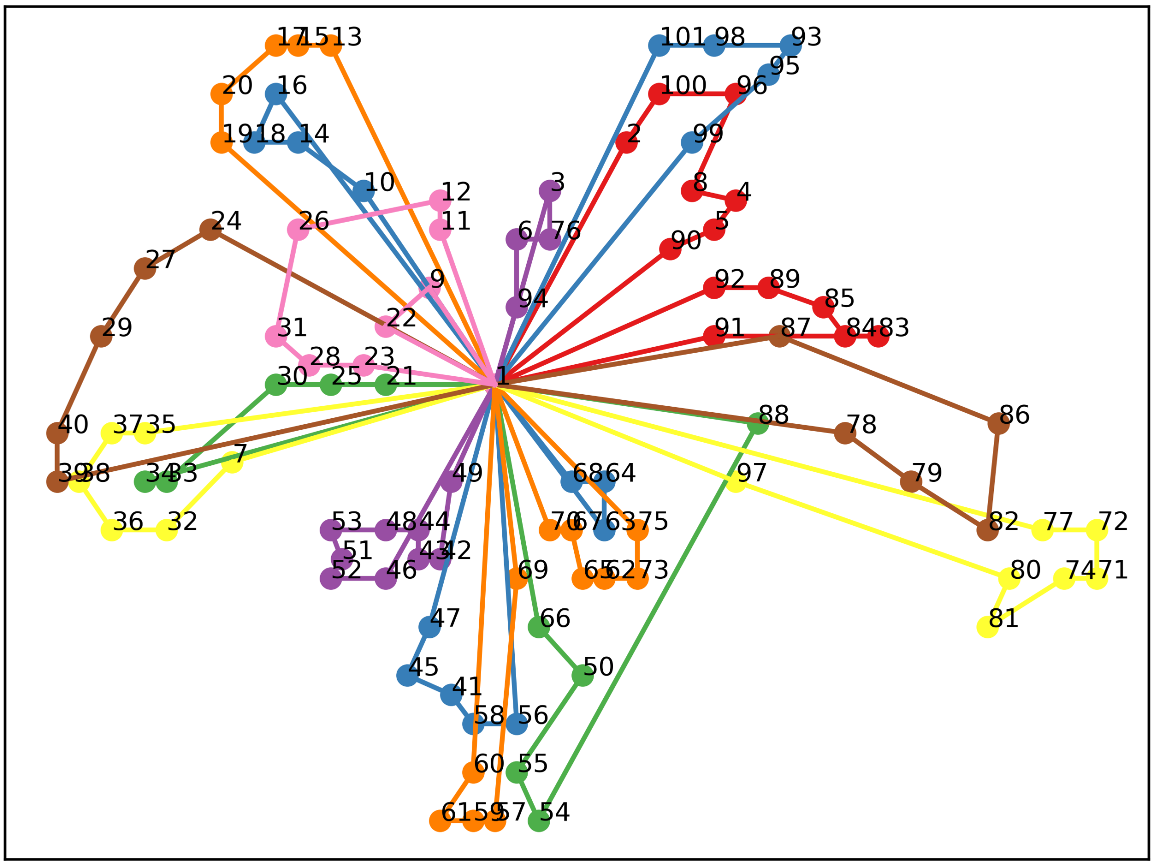


Figure A.1: Solution to C201