Erasmus University Rotterdam

Erasmus School of Economics

MSc Thesis Operations Research & Quantitative Logistics

Different Heuristics for the Simultaneous Pickup and Delivery Problem with Time Windows

Abstract

The waste processing industry faces many logistical challenges, one of which is the distribution of waste containers using vehicles with limited capacities to a set of customers with known demand and time windows. This problem can be modelled as a Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows. A Solution Initialization Heuristic (SIH) is developed and combines a cheapest insertion algorithm with a greedy scheduling heuristic. Several refinement methods are developed, including a Variable Neighborhood Descent (VND), a Large Neighborhood Search (LNS) and a MIP-Start procedure. The combination of SIH and VND obtains near optimal results for one of the data sets and in combination with LNS, it provides very competitive solutions for the other data set. MIP-Start procedure is able to find optimal solutions for four out of the six data sets and is able to obtain good bounds for the other two. An analysis of the neighborhoods in VND shows that intra-route optimization is more applicable in this case compared to inter-route. The methods used within the LNS are analyzed and methods such as Cost Removal, Cluster Removal, Greedy Repair and κ-Regret Repair outperform other methods and show promising results. Overall, the results suggest that the proposed methodology is able to obtain competitive solutions fast.

Author:	**Supervisors:**	**Second Assessor:**
Tomas den Hertog	dr. W. van den Heuvel (EUR)	MSc M. Wagenvoort (EUR)
452523	R.A.O. Salem (GreenRoutes)	

July 15, 2022

Contents

Abbreviations

Table 1: List of abbreviations in alphabetical order

Abbreviation	**Meaning**
LNS	Large Neighborhood Search
MIP	Mixed Integer Programming
NP-hard	Non-deterministic Polynomial-time Hardness
PDPTW	Pickup and Delivery Problem with Time Windows
SIH	Solution Initialization Heuristic
TS	Tabu Search
TSP	Travelling Salesman Problem
VND	Variable Neighborhood Descent
VRP	Vehicle Routing Problem
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPPDTW	Vehicle Routing Problem with Pickup and Delivery and Time Windows
VRPSPDTW	Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Window
VRPTW	Vehicle Routing Problem with Time Windows

1 Introduction

Vast amounts of waste are produced yearly and the processing is a complex problem which involves many logistical challenges. Consider a distribution network with a single depot, which delivers and collects large waste containers to and from customers. These containers are mostly used for construction or renovation waste and their sizes are standardized. They are transported by specialized trucks, equipped with handling equipment to load and unload the containers. Customers order empty containers to be delivered to them, fill them with waste over a period of time and request a pickup afterwards. Full containers are picked up and returned to the depot, where they are emptied, cleaned and stored, after which they are available for delivery again. Now consider multiple customers, whom can have demand for delivery, pickup or both for different types of containers. Customers can request both a delivery and pickup for example, in case their container is full but there is still waste left or the construction is not done yet. The demand of customers is known in advance and additionally each customer has a known time window in which service must take place. In order to serve all requests a homogeneous fleet of trucks is available for transport from and to the depot. The trucks have a limited weight and size capacity and the length of their schedule is restricted by the opening hours of the depot. Service times for unloading and loading of containers both at the depot and at the customers are known and constant. If trucks arrive at a customer before the start of the time window, they are allowed to wait but arriving after the end of the time window is not allowed.

The purpose of this thesis is to construct a complete schedule for all vehicles such that each customer is visited and every demand is met while respecting the weight, size and time window constraints. In this schedule trucks are allowed to drive multiple routes from depot to customers and back to the depot, in order to unload the full containers and reload with empty containers for delivery. This problem is known in the literature as the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW).

The VRPSPDTW is a generalization of the well known Vehicle Routing Problem (VRP), which is a widely studied topic in the literature. The vehicle routing problem, and therefore the VRP-SPDTW, is a $\mathcal{NP}$-hard problem and this severely limits the application of exact methods and requires a heuristic approach for most real-world sized instances. The main goal of this thesis is to develop and implement a Solution Initialization Heuristic and several methods to refine the obtained solution and provide insights on the quality of this solution, while focusing on obtaining

competitive solutions within a short time span. The objective of the developed methods is to minimize the total travel time of all vehicles combined and secondarily minimize the number of vehicles used. The research in this thesis is motivated by GreenRoutes, a company that provides route optimization software to companies within the waste processing sector, as their customers face the problem described in this thesis.

This thesis starts with a review of the relevant literature in Section 2, where the VRPSPDTW and relevant variants of the VRP are discussed. Next, in Section 3, a Mixed Integer Programming formulation is given, which is implemented and used as a benchmark for comparison. Section 4 describes the data used in this thesis as it is provided by GreenRoutes. The solution methods developed and implemented in this thesis are discussed in Section 5. The parameters of these methods are tuned and computational results are obtained, which are discussed in Section 6. Lastly, Section 7 summarizes the thesis, draws conclusions and possible future research is discussed.

2 Literature Review

The problem introduced in the previous section can be seen as a Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW), also known as the Pickup and Delivery Problem with Time Windows (PDPTW), which is a variant of the Vehicle Routing Problem (VRP). This section reviews the formulations and heuristics for the VRP and relevant variants. The VRP was first described by Dantzig & Ramser (1959) and is a generalization of the Travelling Salesman Problem (TSP), which was introduced by Hassley Whitney in 1934 as stated in Flood (1956). The basic VRP consists of a set of customers with known demand, which have to be served by a homogeneous set of vehicles such that the total distance is minimized and all demands are satisfied. One of the main difficulties of the VRP is the fact that it is $\mathcal{NP}$-hard as it is a generalization of the TSP (Garey & Johnson (1979)) and hence all generalizations of the VRP are also $\mathcal{NP}$-hard.

Laporte & Nobert (1987) show that exact methods therefore only work for smaller instances and a recent review of exact methods by Baldacci et al. (2012) shows promising results, being able to solve instances of up to 75 customers. However, the methods require long running times and are hence not suited for real-life applications. Therefore, other methods such as (meta)heuristics, machine learning or evolutionary algorithms have been widely studied and surveys were conducted by Toth & Vigo (2002) and Golden et al. (2008).

In this section we will discuss methods for several extensions of the VRP. Firstly, we discuss the VRP with Time Windows (VRPTW), where each customer has to be served within a time window in Section 2.1. Followed by the VRP with Pickup and Delivery (VRPPD), where each customer either receives goods from a warehouse or returns goods to the warehouse, in Section 2.2. These two extensions are combined into the Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW) which is discussed in Section 2.3. Lastly, the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW), where customers are allowed to have demand for both pickup and delivery, is discussed in Section 2.4.

2.1 VRP with Time Windows

The VRP with Time Windows (VRPTW) is the problem of constructing a set of routes such that each customer is visited exactly once within a time interval by a homogeneous fleet of vehicles with respect to capacity constraints. Some real-world applications of the VRPTW are bank deliveries, postal deliveries, school bus routing and industrial refuse collection. The $\mathcal{NP}$-hardness of the VRPTW limits the use of exact solution methods for most real-life instances and favors the use of heuristic solution approaches instead.

Kallehauge (2008) reviews several exact formulations and analyzes several different algorithms to solve them. Bräysy & Gendreau (2005a) conducted a survey on route construction and local search algorithms for the VRPTW and in their second part, Bräysy & Gendreau (2005b), complement their previous work with a survey on metaheuristics. More recently, Kumar & Panneerselvam (2012) conducted a survey on the latest developments and concluded that future research should be more focused on hybrid methods, the combination of multiple existing methods.

A Multi Objective Problem (MOP) approach is proposed by Ombuki et al. (2006), who use it in combination with a Genetic Algorithm. The quality of the produced solutions is competitive but the most significant contribution is the interpretation of the VRPTW as MOP. It is argued that the MOP view of VRPTW is the most natural, as no unnecessary bias is introduced into the search and no objective is considered as more compared to the other objectives.

A hybrid method is proposed by Schneider et al. (2014), who combined a Variable Neighborhood Search with a Tabu Search (VNS/TS) heuristic for the Electric-VRPTW. The added difficulty of the E-VRPTW is the limited reach of the electric vehicles and the required charging along routes. The hybrid VNS/TS heuristic uses the diversification effect of the VNS and uses the TS heuristic to efficiently search the obtained solution space. The VNS/TS heuristic is tested in a numerical

study which shows the strong performance of the heuristic and the positive effect of combining the two methods.

2.2 VRP with Pickup and Delivery

The VRP with Pickup and Delivery (VRPPD) is the problem of constructing a set of routes such that each customer is visited once for either a pickup or a delivery by a homogeneous set of vehicles with respect to the capacity constraints. Real-world applications of the VRPPD are mainly in reverse logistics as more manufactures want control of their entire supply chain. On one hand for environmental reasons and on the other hand because of potential savings of combining pickups and deliveries.

Three different versions of the VRPPD are distinguished, similar to the survey by Wassan & Nagy (2014), the VRP with Backhauls (VRPB), the VRP with Mixed Pickups and Deliveries (VRPMPD) and the VRP with Simultaneous Pickups and Deliveries (VRPSPD). The VRPB requires all deliveries to be performed before any pickups can be made, the VRPMPD allows deliveries and pickups to be mixed but customers are only allowed to have one or the other, which is relaxed in the VRPSPD. Wassan & Nagy (2014) provide exact formulations for the VRPSPD and show that the formulation can be used to model the other two problems. The VRPMPD can be modelled as a VRPSPD by adding zero delivery demand to pickups and vice versa and the VRPB by adjusting the parameters of the formulation. It is important to note that taking goods from one customer to another is not allowed in all three cases, as this would not be a proper extension of the VRP. When the transportation of goods between customers is allowed the problem belongs to the class of General Pickup and Delivery Problems (GPDP), which is a generalization of the VRP, as stated by Savelsbergh & Sol (1995). A GPDP where all destinations or origins are the depot is a VRPPD.

The VRPB requires all deliveries to be performed before any pickups can be made. This assumption is caused by the fact that vehicles are rear-loaded and the rearrangement of the loads at delivery points is infeasible, as stated by Goetschalckx & Jacobs-Blecha (1989). They also provide an alternative three index formulation, in comparison to the two index formulation by Wassan & Nagy (2014) and performed an extensive computational analysis of multiple initial solution construction algorithms. Concluding that the class of greedy algorithms is capacity oriented and the K-median algorithms are distance oriented. The first exact algorithm is described by Toth & Vigo (1997) and shows promising results. The algorithm uses a Lagrangian lower bound, which

is improved by adding valid inequalities and is able to solve instances with up to 100 customers. However, because of the computational complexity of the exact algorithm the running times are considerably high.

The assumption made in VRPB is relaxed in the VRPMPD, hence pickups and deliveries are allowed to be performed in any order on a vehicle route but customers are only allowed to have either a pickup or a delivery. This problem is not widely studied in the literature as methods designed for the VRPSPD, where customers have both a pickup and delivery, can be used for the VRPMPD, as stated by Wassan et al. (2008). VRPMPD instances can be transformed into VRPSPD instances by adding a zero demand pickup to deliveries and a zero demand delivery to pickups. To obtain a good initial solution Chen & Wu (2006) proposed an insertion-based procedure. The procedure is initiated by choosing a random customer as first customer, calculating all values of the insertion criterion and inserting the customer with the lowest insertion criterion. This procedure is repeated until no more customers can be inserted into the route, after which a customer is randomly chosen from the remaining and the procedure is repeated. Montané & Galvao (2006) designed a Tabu Search heuristic which uses four types of movements, relocation, interchange, crossover and 2-opt, to obtain new solutions. Their methods obtain good results which are close to optimality within a considerably low running time. Another Tabu Search algorithm was developed by Wassan et al. (2008), which uses a reactive Tabu Search that is able to check feasibility of proposed moves quickly and found several new best solutions to benchmark problems.

2.3 VRP with Pickup and Delivery and Time Windows

The VRP with Pickup and Delivery and Time Windows (VRPPDTW) is the combination of the two extensions previously discussed, also known in the literature as the Pickup and Delivery Problem with Time Windows (PDPTW). Early research, conducted by Dumas et al. (1991), provides a formulation and an exact algorithm, namely a column generation scheme to solve the PDPTW. Their experiment showed that the time windows and the distribution of load demands are the parameters that have the most significant impact on running times besides the size of the instance. More recently Cordeau et al. (2007) conducted a survey of exact algorithms and heuristics for the VRPPDTW. They also provided a more compact formulation of the VRPPDTW. One of the heuristics mentioned is the Tabu Search heuristic developed by Nanry & Barnes (2000). They constructed new benchmark problems for the VRPPDTW based on Solomon's benchmarks (Solomon (1987)) and tested their Reactive Tabu Search. The heuristic consistently returns good solutions

with very little computational effort. Li & Lim (2003) used a tabu-embedded simulated annealing algorithm to solve the VRPPDTW. Their method was the first efficient approach to solve large multiple-vehicle VRPPDTW instances and is easily adaptable to generalizations of the VRPPDTW. Simulated annealing was also used in the first stage of the two-stage hybrid algorithm proposed by Bent & Van Hentenryck (2006). The first stage reduced the total number of routes and the second stage, which uses a Large Neighborhood Search (LNS), is meant to decrease the total travel cost. They concluded that the two-stage approach, in comparison to only using one of the two methods, improves the solution quality significantly.

Recently, more research has been conducted into exact methods for the VRPPDTW. Ropke et al. (2007) defined a new formulation for the problem and introduced new valid inequalities which are used in a branch and cut algorithm. The algorithm is tested on several instances and can solve relatively large instances, of up to 200 nodes, to optimality. Following their previous work, Ropke & Cordeau (2009) improved the algorithm by introducing a branch-and-cut-and-price algorithm. Baldacci et al. (2011) presented an exact algorithm based on a set partitioning formulation which is strengthened by Subset-Row inequalities. The method is shown to be more effective and is also able to solve 15 instances previously unsolved by an exact method.

2.4 VRP with Simultaneous Pickup and Delivery and Time Windows

In comparison to the VRPPDTW, where customers require either a pickup or a delivery, the VRPSPDTW allows customers to have both a pickup and a delivery at the same time. Any VRPPDTW instance can be transformed into a VRPSPDTW, if a customer requires a pickup the delivery demand is set to zero and vice versa. By the same logic any VRPSPDTW instance can be transformed into a VRPPDTW, by splitting the pickup and delivery demand into two requests. However, this may lead to a solution in which two requests by the same customer are served by a different vehicle. The VRPSPDTW is a generalization of the VRP and is not investigated as widely as the other variants discussed in this section, although some noteworthy research has been done. Angelelli & Mansini (2002) proposed an exact algorithm, implementing a branch and price approach on a set covering formulation.

Because of the $\mathcal{NP}$-hardness of the problem, more research has been done into (meta)heuristics. A genetic algorithm using a variant of the cheapest insertion algorithm is proposed by Wang & Chen (2012). They modified the Solomon (1987) benchmark problems for VRPTW and conducted a computational study. The genetic algorithm is compared to CPLEX software and is shown to

provide better solutions with less computational effort. Another genetic algorithm is proposed by Liu et al. (2013), who also propose a Tabu Search algorithm. The genetic algorithm is on a permutation chromosome with a split procedure complemented with a local search. These methods are compared in a computational study and their performance is of the same level but the Tabu Search requires more computational time, but both methods outperform CPLEX. Wang et al. (2015) proposed a Multiple Objective Local Search (MOLS) and a Multiple Objective Memetic Algorithm (MOMA) approach. Their formulation has an objective made out of five components, the number of vehicles, the total travel distance, the travel time of the longest route, the waiting time and the delay time, which are all minimised. A parallel simulated annealing algorithm is proposed by Wang et al. (2015) and they have shown the effectiveness of their method by improving on multiple objectives on the benchmark instances made by Wang & Chen (2012). Another Tabu Search is proposed by Shi et al. (2018), who developed a method that shifts between two tabu searches. Tabu Search I rapidly decreases the objective but is sensitive to local optima, Tabu Search II is a slower method but is very capable of escaping local optima. The effectiveness of their method is shown by comparing them to the benchmark instances as proposed by Wang & Chen (2012) and they improve several best known objective values.

Based on this literature review a Solution Initialization Heuristic is implemented which uses an insertion criterion, whereof the effectiveness was shown by Solomon (1987). To refine this solution a Variable Neighborhood Search, a Tabu Search and a Large Neighborhood Search are implemented as Gendreau et al. (1994), and Ropke & Pisinger (2006) showed the effectiveness to the class of Vehicle Routing Problems.

3 Problem Description

Inspired by the three index formulations of Azi et al. (2010) and Hernandez et al. (2014), this problem is modelled on a complete graph $G = (N, A)$. Where $N = \{0, \ldots, n+1\}$ is the set of all locations, such that 0 represent the start and $n+1$ the destination of each route, which are denoted by a different index but may be the same geographical location. Let $C = \{1, \ldots, n\}$ denote a subset of N that represents all customers. The set A denotes the set of all arcs, where $(i, j) \in A$ is an arc between location i and j, for every $i, j \in N, i \neq j$ and with each arc (i, j), traveling times T_{ij} are associated. The set of vehicles is denoted by V and consists of $|V|$ number of homogeneous trucks, with size capacity S and weight capacity W. The working day of a vehicle consists of a sequence

of routes, all starting and ending at the depot. The set of all routes is denoted by R, and the total number of routes $|R|$ is sufficient enough to accommodate the maximum number of routes the fleet V can possibly perform, these routes are allowed to be empty. It is assumed that routes performed by the same vehicle are numbered in increasing order, meaning that a vehicle only performs route s after r if $r < s$. Customers are allowed to have demand for different types of products for delivery and for pickup as well, hence the set F for all types of pickup demand and the set G for all types of delivery demand and are introduced. Each product in F (G) has weight w_p^f and size s_p^f for $f \in F$ (w_d^g and s_d^g for $g \in G$), where the subscript p (d) indicates a pickup (delivery). Furthermore, with each customer $i \in C$, time window $[a_i; b_i]$, pickup demand p_i^f, delivery demand d_i^g and service time σ_i are associated. Time windows $[a_i; b_i]$, where a_i denotes the beginning of the time interval and b_i the ending, are imposed for each customer and service must take place within this interval. The pickup (delivery) demand of containers of type $f \in F$ ($g \in G$) of a customer $i \in C$ is denoted by p_i^f (d_i^g). The time needed to service a customer based on the specific demand(s) of customer i are denoted by σ_i. Start of service at customer i plus the service time σ_i, may not exceed the end of the time window b_i. The depot also has a time window $[a_0; b_0]$ and $[a_{n+1}; b_{n+1}]$ which represents the business hours of the depot and all activities must take place within this interval. Furthermore, the service times at the depot for loading (unloading) trucks is denoted by σ_0^L (σ_0^U) and takes place at the beginning (ending) of a route.

Next, multiple decision variables are introduced, which in combination with the graph G and the parameters defined provide a mixed-integer linear programming formulation. Firstly, let binary variable x_{ij}^r indicate if arc (i, j) appears in route r, note that if $x_{0(n+1)}^r = 1$, route r is empty. Next, let binary variable y_i^r indicate if customer $i \in C$ is served by route r. In order to keep track of time, let t_i^r denote the time at which service starts at customer $i \in C$ if it is served by route r and when customer i is not served by route r the value is meaningless. The variable t_0^r (t_{n+1}^r) denotes the start (end) time of route r, for each $r \in R$. For each pair of routes $r, s \in R$, where $r < s$, binary variable z_{rs} indicates if route r is immediately followed by route s by the same vehicle. Furthermore, the loads inside the trucks are tracked by the variables P_{ij}^f and D_{ij}^g. Variable P_{ij}^f indicates the amount of demand picked up to customer i of type f and transported over arc (i, j), and D_{ij}^g indicates the amount of delivery demand still in the truck after visiting customer i of type g and transported over arc (i, j). Finally, variable v indicates the number of vehicles used, beyond the available fleet of size $|V|$ which is penalized by factor γ. The problem can be formulated using the following MIP formulation:

$$\min \quad \sum_{(i,j)\in A} T_{ij} \sum_{r\in R} x_{ij}^r + \gamma v \tag{1}$$

s.t.

$$\sum_{r\in R} y_i^r = 1 \qquad \forall i \in C \tag{2}$$

$$\sum_{j\in N} x_{ji}^r = y_i^r \qquad \forall i \in N, r \in R \tag{3}$$

$$\sum_{j\in N} x_{ij}^r = y_i^r \qquad \forall i \in N, r \in R \tag{4}$$

$$t_i^r + T_{ij} + \sigma_i \leq t_j^r + M(1 - x_{ij}^r) \qquad \forall i \in N, j \in C, r \in R \tag{5}$$

$$t_i^r + T_{i(n+1)} + \sigma_0^U \leq t_{n+1}^r + M(1 - x_{i(n+1)}^r) \qquad \forall i \in C, r \in R \tag{6}$$

$$t_{n+1}^r + \sigma_0^L \leq t_0^s + M(1 - z_{rs}) \qquad \forall r, s \in R, r < s \tag{7}$$

$$t_{n+1}^r \leq b_0 - \sigma_0^U \qquad \forall r \in R \tag{8}$$

$$t_0^r \geq a_0 + \sigma_0^L \qquad \forall r \in R \tag{9}$$

$$a_i y_i^r \leq t_i^r \qquad \forall i \in C, r \in R \tag{10}$$

$$b_i y_i^r \geq t_i^r \qquad \forall i \in C, r \in R \tag{11}$$

$$|R| - \sum_{r\in R} \sum_{s\in R, r<s} z_{rs} \leq |V| + v \tag{12}$$

$$\sum_{r\in R, r<s} z_{rs} \leq 1 \qquad \forall s \in R \tag{13}$$

$$\sum_{r\in R, r<s} z_{sr} \leq 1 \qquad \forall s \in R \tag{14}$$

$$\sum_{j\in N} P_{ij}^f - \sum_{j\in N} P_{ji}^f = p_i^f \qquad \forall i \in C, f \in F \tag{15}$$

$$\sum_{j\in N} D_{ji}^g - \sum_{j\in N} D_{ij}^g = d_i^g \qquad \forall i \in C, g \in G \tag{16}$$

$$\sum_{f\in F} s_p^f P_{ij}^f + \sum_{g\in G} s_d^g D_{ij}^g \leq S x_{ij}^r \qquad \forall (i,j) \in A, r \in R \tag{17}$$

$$\sum_{f\in F} w_p^f P_{ij}^f + \sum_{g\in G} w_d^g D_{ij}^g \leq W x_{ij}^r \qquad \forall (i,j) \in A, r \in R \tag{18}$$

$$x_{ij}^r \in \{0,1\} \qquad \forall (i,j) \in A, r \in R \tag{19}$$

$$y_i^r \in \{0,1\} \qquad \forall i \in N, r \in R \tag{20}$$

$$P_{ij}^f \in \mathbb{R} \qquad \forall (i,j) \in A, f \in F \tag{21}$$

$$D_{ij}^{g} \in \mathbb{R} \qquad \forall (i,j) \in A, g \in G \tag{22}$$

$$t_{i}^{r} \in \mathbb{R} \qquad \forall i \in C, r \in R \tag{23}$$

$$v \in \mathbb{N} \tag{24}$$

The objective function (1) minimizes the total travel time and the number of vehicles used beyond the available fleet. Constraints (2) ensure that each customer is visited exactly once, where Constraints (3) ensure a predecessor and Constraints (4) a successor for each customer. The Constraints (5) to (11) ensure the feasibility of the time schedule. Constraints (5) guarantee sufficient time for travel and service between customer i and j. Adequate time to unload/load at the depot after/before a route is ensured by Constraints (6) and (7) respectively. The first and last service of routes must take place within the opening hour of the depot which is ensured by Constraints (8) and (9). Constraints (10) and (11) ensure that service at customer i takes place within the time window. The number of vehicles used is determined by Constraint (12) and ensures that variable v is set to the correct value. Artificially setting z_{rs} values to one is prevented by Constraints (13) and (14). The satisfaction of pickup and delivery demand is ensured by Constraints (15) and (16) respectively. Constraints (17) and (18) ensure that the load of a truck is feasible in terms of size and weight respectively and force P_{ij}^{f}, D_{ij}^{g} to be zero when needed. Finally, the domains of all variables are given by Constraints (19) to (24).

4 Data

Real-world data is provided by GreenRoutes, a company that supplies route optimization for waste collectors. Their customers face the problem as described in this thesis and therefore their data can be used to construct a real-world data set. The data set contains 500 customers and to investigate the performance of the heuristics the set is split into different sized instances, a small, middle and large data set containing 50, 100 and 200 customers respectively. In order to calculate the travel time, T_{ij}, between customers i and j Open-Source Routing Machine (OSRM) is used. OSRM is a system that solves the shortest path problem between two coordinates using the road network, while taking maximum speeds, weight and height restrictions into account.

4.1 Data description

The data provided by GreenRoutes will be divided into two different data sets from which instances of different sizes are obtained. The initial data set contains customers with demand for both big containers (6m^3 and 10m^3) and small containers (2.5m^3 and 1m^3). Conveniently big and small containers are not handled by the same vehicles and the data set naturally divides into two, one set containing demand for big containers and the other only for small containers. However the structure of the two sets is identical and for each customer the following data is available:

- Location: consisting of the full address, longitude and latitude, this data is used to calculate the travel time, T_{ij}, to other customers or the depot.
- Time window: defined by a starting time a_i and an ending time b_i, service must take place within this time window.
- Demand: pickup, delivery or both, including the type of container and the quantity, denoted by P_{ij}^f and D_{ij}^g and is used to calculate the service time, σ_i, for customer i.

For the depot the following data is available:

- Location: consisting of the full address, longitude and latitude, used to calculate the travel time to customers, T_{0i} and $T_{i(n+1)}$.
- Business hours: for all instances the business hours are 7am to 3pm and as the time horizon is defined in seconds, $t = 0$ corresponds to 7am and to a_0, where $t = 28800$ corresponds to 3pm and b_0.
- Fleet: data about the different kinds of trucks, consisting of:
    - Size capacity, S
    - Weight capacity, W
    - Number of available trucks, $|V|$
    - Service times: unloading and loading at a customer i, σ_i, or at the depot σ_0^U and σ_0^L.

First, we focus on the big containers of 6m^3 and 10m^3. These are transported using special trucks, sometimes called skip trucks or skip lorries, who can carry either one or multiple empty containers or one full container. The maximum number of empty 6m^3 containers a truck can carry is five and for the 10m^3 the number is three. The size, s_d^g, of a empty smaller 6m^3 container is set to 1 and for the bigger 10m^3 to $1\frac{2}{3}$ and their weight, w_d^g, is 900kg and 1100kg respectively. For

a full container, either $6m^3$ or $10m^3$, the size, s_p^f, is 5 and the weight, w_p^f, 12000kg. Trucks able to handle these containers have a size capacity, S, of 5 and a weight capacity, W, of 12000kg and hence can handle only one full container. Any combination of empty $6m^3$ and $10m^3$ is allowed. Unloading and loading, or delivering and picking up, both take 10 minutes. Based on the demand of a customer the service times are calculated, for the depot the service times are the same no matter the number of containers, unloading and loading both take 15 minutes.

Furthermore, the smaller containers of $1m^3$ and $2.5m^3$ are transported using trucks also known as boom trucks or lorry loaders. The smaller $1m^3$ containers are used to deliver soil or gravel to customers and do not require a pickup as the package is disposable. These trucks have six spots and per spot can either carry:

- Up to three empty $2.5m^3$ containers
- One full $2.5m^3$ container
- Up to two full $1m^3$ containers
- One empty $2.5m^3$ and one full $1m^3$ container.

A single spot is given a capacity of 1 and hence, full $1m^3$ containers have size $\frac{1}{2}$, empty $2.5m^3$ size $\frac{1}{3}$ and full $2.5m^3$ size 1. The weight of a full $1m^3$ container is 1500kg, an empty $2.5m^3$ 300kg and a full $2.5m^3$ 3000kg. Trucks of this type have a size capacity of 6 and a weight capacity of 15000kg. Service time per container are 10 minutes, empty or full and at the depot 15 minutes for unloading and for loading.

5 Methodology

In this thesis, a *route* is a set of customers visited by a vehicle which starts and ends at the depot, without visiting the depot in between. A *tour* is made up of all routes performed by a vehicle and if it consists of multiple routes the depot is visited in between. All tours for a day are combined into a *schedule*. The objective is to minimize the total travel time and secondarily the number of vehicles or equivalently the number of tours in a schedule, used. In this section, the methodology to solve the VRPSPDTW instances is described. Firstly a solution is initialized by the Solution Initialization Heuristic as described in Section 5.1. This solution is refined by a Variable Neighborhood Descent as described in Section 5.2. Section 5.3, describes a Tabu Search and Section 5.4 a Large Neighborhood Search. Both methods are used to either improve the solution produced

by SIH or further refine the solution from the VND. Lastly, a MIP-Start procedure is described in Section 5.5 which uses the MIP formulation as described in Section 3.

5.1 Solution Initialization Heuristic

The Solution initialization Heuristic (SIH) works in two stages, firstly routes will be initialized using a cheapest insertion heuristic inspired by Solomon (1987) and thereafter the routes will be combined into tours in a greedy manner. In the first stage, routes are constructed by inserting unrouted customers into existing routes at the cheapest feasible position. The heuristic takes the customer, container, vehicle and depot data as input, which it uses to construct the routes and their corresponding arrival times, weights and capacities at each customer on a route. An outline of the heuristic can be found in Algorithm 1. All customers are put into the set $UnroutedCustomers$ and an empty set for all routes, R, is defined. The first route is defined, as the set R is still empty and thus the Boolean $CustomerInserted$ will be false, by the Lines 12 to 14. In Line 12 an arbitrary customer, $RandomCustomer$, from the set $UnroutedCustomers$ is chosen by the method $GetRandomCustomer()$. Next, in Line 13, a new route is made with $RandomCustomer$, also the corresponding arrival times, weights and capacities at each moment on the route are created. Finally, in Line 14, the $RandomCustomer$ is removed from the set of unrouted customers as it is now on a route. In the next iteration, the set R of all routes is not empty and the for loop in Line 5 is entered in order to execute Lines 6 to 10. For each route r in R, the best customer to insert is calculated by the $BestCustomerPerRoute()$ method, hence per iteration multiple customers can be inserted. This method takes a route and the set of unrouted customers and finds the cheapest feasible, if existent, customer to insert. The method loops through all customers in $UnroutedCustomers$ and calculates the cost of insertion on each place in the route for each customer. Simultaneously, the method tests if the insertion at the particular place is feasible and if the insertion is both feasible and currently the cheapest, it is stored. After all places on the route are considered for each customer, the cheapest feasible customer, if existent, is returned. The cost of insertion is defined by the following measure, where i and j are customers on route r and c is the inserted customer:

$$cost(i,j,c) = \alpha * dur_{add}(i,j,c) + (1-\alpha) * dist_{add}(i,j,c), \quad 0 \leq \alpha \leq 1 \tag{25}$$

$$dur_{add}(i,j,c) = T_{ic} + T_{cj} - T_{ij} - T_{0c} - T_{0(n+1)} \tag{26}$$

$$dist_{add}(i,j,c) = Dist_{ic} + Dist_{cj} - Dist_{ij} - Dist_{0c} - Dist_{0(n+1)} \tag{27}$$

The cost, given by Equation (25) is defined by the weighted sum of the added duration and the added distance of inserting customer c between customer i and j. The added duration, Equation (26), is defined as the time it takes to visit customer c in between i and j, $T_{ic}+T_{cj}$ minus the duration from i to j, T_{ij}, and the duration of visiting c from the depot on its own, $T_{0c} + T_{c(n+1)}$. The final part favors customers further away from the depot over customers closer, as a new route containing a customer further away would be more costly. The formula for added distance, Equation (27), is constructed in a similar manner. After consideration of different insertion criteria, i.e. euclidean distance, 'as the crow flies' distance and difference in arrival times, Equation (25) was the most effective and hence implemented.

If there is no cheapest feasible customer, the $BestCustomerPerRoute$ returns empty and the next route $r \in R$ is considered. After all routes are considered and no customer has been inserted, Lines 12 to 14 are called and a new route is made with an arbitrary customer. This is repeated until all customers are routed and the set of routes, R, is returned.

Algorithm 1: Cheapest Insertion Heuristic

Input: Customer, container, vehicle and depot data

Output: $R :=$ Routes with corresponding arrival times, weights and capacities

```
1  UnroutedCustomers ← C
2  R ← ∅
3  while UnroutedCustomers ≠ ∅ do
4      CustomerInserted ← False
5      for r in R do
6          BestCustomer ← BestCustomerPerRoute(r, UnroutedCustomer)
7          if BestCustomer ≠ ∅ then
8              r ← PerformInsertion(BestCustomer)
9              UnroutedCustomers = UnroutedCustomers \ BestCustomer
10             CustomerInserted ← True
11     if CustomerInserted = False then
12         RandomCustomer ← GetRandomCustomer(UnroutedCustomers)
13         R ← StartNewRoute(RandomCustomer)
14         UnroutedCustomers = UnroutedCustomers \ RandomCustomer
```

The routes constructed using Algorithm 1 are combined into tours by Algorithm 2. The first tour is initialized by the Lines 11 to 13, as the set of tours T is empty. In Line 11, the method $GetRandomRoute()$ is called, which returns a random route from a subset of all routes. This subset

contains all routes which start at the earliest time, $t = 0$, or if no such routes are left, the route with the earliest starting time. A new tour is constructed in Line 12 and the corresponding route is removed from the set of routes. In the next iteration, the set of Tours, T, is not empty and the Lines 4 to 9 are executed. For all $t \in T$, the method $AddFeasibleRoute()$ searches for a feasible route to extend the current tour. The remaining routes are considered in descending order in terms of duration, which favors the relatively longer routes to be scheduled before the shorter ones. If a feasible route is found, the route is added to the tour in Line 7 and removed from the set of routes to consider in Line 8. In one iteration multiple tours can be extended and in the case that all $t \in T$ are considered and no tour was extended, a new tour is constructed by Lines 11 to 13. This is repeated until all routes are scheduled and the output is a set of tours, also called a schedule, with corresponding arrival times, weights and capacities.

As Algorithm 1 uses a random element in Line 12, different solution can be constructed in different runs of the algorithm. In order to obtain results, the algorithm is run 25 times per instance and averages are taken. Secondly, in order to optimize performance, the sets containing routes or tours in Algorithm 1 and 2 are kept as small as possible. In the implementation, routes or tours for which all remaining option are considered and no feasible addition found, are removed from the corresponding set.

Algorithm 2: Greedy Tour construction heuristic

Input: $R :=$ Routes with corresponding arrival times, weights and capacities
Output: $T :=$ Tours with corresponding arrival times, weights and capacities

```
1  T ← ∅
2  while R ≠ ∅ do
3      RouteInserted ← False
4      for t in T do
5          RouteToAdd ← AddFeasibleRoute(t, R)
6          if AddedRoute ≠ ∅ then
7              t ← PerformAddedRoute(RouteToAdd)
8              R = R \ RouteToAdd
9              RouteInserted ← True
10     if RouteInserted = False then
11         RandomRoute ← GetRandomRoute(R)
12         T ← StartNewTour(RandomRoute)
13         R = R \ RandomRoute
```

5.2 Variable Neighborhood Descent

Secondly, to improve the solution initialized as described in Section 5.1, a Variable Neighborhood Descent (VND) is implemented. The aim of the VND is to find a improved set of tours T' with respect to the current set of tours T, an improvement is defined as decrease in the objective, $z(T') < z(T)$. The VND searches for a better solution in the neighborhood of the current solution, where a neighborhood is defined as a subset of solutions. Let X denote the total solution space and the current solution by $T \in X$. The neighborhood $N(T) \subseteq X$ is composed of all solutions that are in some sense close to T and $T' \in N(T)$ is a neighbor of T. The VND starts searching for a new solution in the first neighborhood N_1, if an improvement has been found the new solution T', is accepted and the VND continues searching in the solution space of the new current solution. However, if no improvement has been found, the VND searches in the second neighborhood N_2, if this yields an improvement the VND starts the search over in the first neighborhood with this improved solution. If no improvement was found the search is continued in the next neighborhood, if no improvement was found in the largest neighborhood, the VND is finished and returns the current solution. An outline of the VND can be found in Algorithm 3.

Algorithm 3: Variable Neighborhood Descent

Input: $T :=$ Tours with corresponding arrival times, weights and capacities

Output: $T :=$ Tours with corresponding arrival times, weights and capacities

```
1 k ← 1
2 while k ≤ k_max do
3 |   T' ← N_k^*(T)
4 |   if z(T') < z(T) then
5 |   |   T ← T'
6 |   |   k ← 1
7 |   else
8 |   |   k ← k + 1
```

Where k_{max} is defined as the index of the last neighborhood and the following neighborhoods are defined:

N_1: Or-Opt, Relocate a single customer within a route which has a neighborhood size of $O(n^2)$.

N_2: Or-Opt*, Relocate a single customer from one route to a place within another route, a variant of Or-Opt with a size of $O(n^2)$.

N_3: Swap, which swaps the position of two customers and has a neighborhood size of $O(n^2)$.

N_4: 2-Opt, Destroy arcs (a, b) and (c, d) (from the same route) and form arcs (a, c) and (b, d), which has a neighborhood size of $O(n^2)$.

N_5: 2-Opt*, splits two routes into two parts and reconnects the first half of the first route to the second half of the second route and vice versa, with a neighborhood size of $O(n^2)$.

The difference between neighborhoods who strictly make changes only within routes (inter-route) and strictly between routes (intra-route) is important to note. Inter-route changing neighborhoods are N_1 and N_4 and focus on improving current routes, while N_2 and N_5 are intra-route changing neighborhoods and focus on moving customers between routes. Neighborhood N_3 can search both the inter-route and intra-route solution space.

5.3 Tabu Search

Furthermore, a Tabu Search (TS) is implemented, which is a method capable of escaping local optima. Gendreau et al. (1994) first showed its effectiveness to the VRP and both Nanry & Barnes (2000) and Chen & Wu (2006) use a form of Tabu Search in their approaches. Similar to VND, Tabu Search starts from a solution T and aims to find the next solution T' in the neighborhood of T, in contrast to VND T' may be accepted even if the objective value is worse if no improving move is available. This occurs if the search is in a local optimum or all improving moves are on the tabu list. The tabu list is a list containing (partial) potential solutions which can not be selected for a number of iterations and are marked as tabu. This is done in order to move the search into unexplored areas of the solution space and hence attempt to escape potential local optima. An outline of the Tabu Search can be found in Algorithm 4.

The input is comprised of a set of tours and three parameters, the neighborhood index κ_{tabu}, the length of the tabu list Λ_{tabu} and the maximal number of iterations $\mathcal{M}_{tabu}$. The neighborhoods used in the tabu search are the same as the VND, as described in Section 5.2 and the exact neighborhood is defined by κ_{tabu} which takes a value from 1 to 5. The search is performed solely in this neighborhood, although multiple tabu searches can be performed sequentially. The length of the tabu list, Λ_{tabu}, defines the maximum number of potential solutions which are prohibited from being evaluated in the next iteration. The maximum number of iterations, $\mathcal{M}_{tabu}$, defines the maximum number of iterations the Tabu Search is allowed to explore without finding an improvement. The Tabu Search is initialized in Lines 1 to 5, $TabuList$ as empty, $NumIteration$ at zero,

the *BestObjective* as the objective value of the input and the *BestSolution* as the input. While $NumIteration$ is less than $\mathcal{M}_{tabu}$, the method $BestNeighbor()$ searches for the best neighbor of T in neighborhood N_k which is not on the tabu list $TabuList$ in Line 7. The potential solution found is added to the tabu list in Line 8 and if the objective value of the potential solution is less than the current best known objective Lines 10 to 12 are excecuted. In Line 10 the best known objective is updated, next the *BestSolution* is updated and subsequently the $Numiteration$ is set to zero in Line 12. If the objective is not higher $NumIteration$ is updated and next in Line 15 the tabu list is updated by the $UpdateTabuList()$ method. The method takes the tabu list and Λ_{tabu} as input, if the length of the tabu list exceeds Λ_{tabu} the potential solutions which are on the list for the most iterations are removed. Lastly, in Line 16 the solution to be evaluated in the next iteration is set to the solution found in this iteration. When $NumIteration$ is equal to $\mathcal{M}_{tabu}$, the Tabu Search returns the *BestSolution* and the corresponding *BestObjective*.

Algorithm 4: Tabu Search

Input: T, $\kappa_{tabu}, \Lambda_{tabu}, \mathcal{M}_{tabu}$

Output: *BestSolution*, *BestObjective*

```
1  TabuList ← ∅
2  NumIteration ← 0
3  k ← κ_tabu
4  BestObjective = z(T)
5  BestSolution ← T
6  while NumIteration < M_tabu do
7      T' ← BestNeighbor(N_k, T, TabuList)
8      TabuList ← TabuList ∪ T'
9      if z(T') < BestObjective then
10         BestObjective ← z(T')
11         BestSolution ← T'
12         NumIteration ← 0
13     else
14         NumIteration ← NumIteration + 1
15     TabuList ← UpdateTabuList(TabuList, Λ_tabu)
16     T ← T'
```

5.4 Large Neighbourhood Search

Another method capable of escaping local optima, Large Neighborhood Search (LNS), is implemented. Ropke & Pisinger (2006) and Emeç et al. (2016) all show its effectiveness to the VRP-SPDTW and use a diversified set of methods in their LNS. In LNS a solution T is partially destroyed by a destroy method and this partial solution T' is subsequently repaired by a repair method. In this section an overview of the LNS is given and pseudo code is provided, Section 5.4.1 reviews the seven destroy and four repair methods and Section 5.4.2 describes the mechanism for destroy/repair method selection.

An outline of the LNS can be found in Algorithm 5, the input is comprised of a set of tours, T and a set of parameters $parameters_{LNS}$. In Lines 1 to 4 the LNS is initialized, $NumIteration$ is set to zero, the input tour is set as the current $BestSolution$ and its corresponding objective value as the $BestObjective$. The set $parameters_{LNS}$ contains all parameters for the destroy and repair methods as well as the weight updating factors, $\delta^{+}_{destroy}$, $\delta^{-}_{destroy}$, δ^{+}_{repair}, δ^{-}_{repair}, the initial weights π^{-}, π^{+} and the maximum number of iterations, $\mathcal{M}_{LNS}$, which are initialized in Line 4. While the $NumIteration$ is lower than $\mathcal{M}_{LNS}$, the $ChooseDestroyMethod()$ draws a destroy method from the available set based on the weights of all destroy methods, π^{-} in Line 6. Next in Line 7, $DestroySolution()$ uses this method in combination with $parameters_{LNS}$ to partially destruct the solution T into T'. In Line 8 and 9 the same procedure is repeated for selecting and applying a $RepairMethod$, these steps result in a new set of tours, T" which is evaluated in Line 10. If this new solution has a lower objective value than the current $BestObjective$ it is stored as the new current best in Lines 11 to 13 and in Line 14 the $NumIteration$ is set to zero. However if this is not the case $NumIteration$ is increased by one and finally in Line 17 the method $UpdateWeights()$ is used to update the weight, which is further elaborated in Section 5.4.2. When $NumIteration$ is equal to $\mathcal{M}_{LNS}$ the LNS returns the $BestSolution$ and the corresponding $BestObjective$.

Algorithm 5: Large Neighborhood Search

Input: T, $parameters_{LNS}$

Output: $BestSolution, BestObjective$

$NumIteration \leftarrow 0$
$BestObjective = z(T)$
$BestSolution \leftarrow T$
$\mathcal{M}_{LNS}$, $\delta^+_{destroy}$, $\delta^-_{destroy}$, δ^+_{repair}, δ^-_{repair}, π^-, $\pi^+ \leftarrow parameters_{LNS}$
while $NumIteration < MaxNumIterations$ **do**
    $DestroyMethod \leftarrow ChooseDestroyMethod(\pi^-)$
    $T' \leftarrow DestroySolution(DestroyMethod, T, parameters_{LNS})$
    $RepairMethod \leftarrow ChooseRepairMethod(\pi^+)$
    $T" \leftarrow RepairSolution(RepairMethod, T', parameters_{LNS})$
    if $z(T") < BestObjective$ **then**
        $BestObjective \leftarrow z(T")$
        $BestSolution \leftarrow T"$
        $T \leftarrow T"$
        $NumIteration \leftarrow 0$
    else
        $NumIteration \leftarrow NumIteration + 1$
    $\pi^-, \pi^+ \leftarrow UpdateWeights(\delta^+_{destroy}, \delta^-_{destroy}, \delta^+_{repair}, \delta^-_{repair}, \pi^-, \pi^+)$

5.4.1 Destroy and Repair methods

To remove customers from the current solution, the following seven destroy methods are defined. If removing a customer results in an empty route or tour, the corresponding route or tour is deleted from the schedule and all relevant variables are updated.

Random Removal: Randomly removes customers from the current schedule until n^-_{rand} customers are removed while remaining a feasible schedule. The variable n^-_{rand} is defined by $parameter_{LNS}$ set and is given as a proportion, p_{rand} of the total number of customers $|C|$.

Smallest Tour: Removes the tour, including all corresponding routes, with the least customers from the current schedule. This method aims to reallocate these customers to the other tours and thereby reducing the total number of tours and thus the total number of trucks.

Smallest Route: Removes the route with the least customers from the current schedule. Similar to the previous method, the aim is to allocate the customers to other routes and tours,

ultimately in order to reduce the number of tours.

Longest Tour: Removes the tour with the most customers from the current schedule. The aim of this method is to destroy a large part of the solution which is more difficult to destroy by the other methods.

Longest Route: Removes the roue with the most customers from the current schedule. The aim of this method is similar to the previous method.

Cost Removal: Removes the n^-_{cost} customers who have the highest cost of serving them. The cost of serving, cs_j, customer j is defined by $cs_j = T_{ij} + T_{jk} - T_{ik}$, where i and k are the customers before and after customer j. Similar to Random removal the variable n^-_{cost} is defined by the $parameter_{LNS}$ set and is given as a proportion p_{cost} from the total number of customers $|C|$. The aim of this method is to remove customers at expensive, undesirable places and move them to more desirable, cheaper places.

Cluster Removal: Selects a random customer i, then removes the $n^-_{cluster}$ which are closest to customer i. The measure which determines the closeness between two customers i and j is the travel distance T_{ij} between the two. The variable $n^-_{cluster}$ is determined in a similar manner as Random and Cost Removal by a proportion $p_{cluster}$ of the total number of customers $|C|$.

To reinsert the customers who have been removed the following four repair methods are used. Each methods prioritizes the insertion of customers into existing tours and routes but if no feasible places are left for customers, new routes and tours are allowed to be created.

Greedy: This method reinserts each customer into its cheapest place. During each iteration a customer is inserted at the insertion place, where the cost of insertion defined as in Equation (25), is the cheapest.

Random: This method reinserts customers into random feasible place.

GRASP: This method constructs a list of n_{GRASP} cheapest, feasible insertion places per customer and randomly chooses one of them to insert the customer into. The number n_{GRASP} is defined by the parameter p_{GRASP} times the total number of customer $|C|$.

κ-Regret: The methods previously described have a greedy component which favors the insertion of customers with low insertion cost over those with high insertion cost. However postponing

the insertion of these customers may lead to even higher insertion cost. The κ-Regret method tries to resolve this by looking "ahead". For each customer a list of insertion places, sorted in descending order in terms of insertion cost is composed. The κ-Regret value is then defined as the difference between the κ-th value and the first value in the list. Each iteration the method inserts the customer which has the highest κ-regret value. This method is inspired by Emeç et al. (2016).

5.4.2 Weight Updating

The weight of each method is based on the performance of that method in the previous iterations: if a method has performed well the weight is increased, if not, the weight is decreased. All methods start with equal weights and hence have the same probability to be chosen, as the LNS progresses weights shift and improving methods are favored. After each iteration if a better solution has been found the weights of the corresponding destroy and repair method are increased by factor $\delta^+_{destroy}$ and δ^+_{repair}, if not the weights are decreased by factors $\delta^-_{destroy}$ and δ^-_{repair}. The increasing factors $\delta^+_{destroy}$ and δ^+_{repair} are always greater than one and the decreasing factors $\delta^-_{destroy}$ and δ^-_{repair} are always lower than one. For a high value of δ^+ and a low value of δ^- the LNS favors improving methods over diversifying methods and for low values of δ_+ and high values for δ^- vice versa. The selection of values for the parameters is a trade-off between intensification and diversification, these parameters are tuned in Section 6.1.

5.5 MIP-Start

The problem in this thesis and as described by the MIP formulation from Section 3 is $\mathcal{NP}$-hard, hence will not solve in polynomial time. However, solutions obtained by the methods described in Sections 5.1 to 5.4 can be used as a starting position for the MIP formulation from Section 3. A starting position is defined by values for (part of) the decision variables and in this case can be obtained from the schedule. The number of vehicles v, the start time of service for at each customer y^r_i with $i \in C, r \in R$ and the variables x^r_{ij} for $i, j \in N, r \in R$ and z_{rs} for $r, s \in R, r < s$ can directly be obtained from a schedule. The variables which keep track of the load, P^f_{ij} for $i, j \in C, f \in F$ and D^g_{ij} for $i, j \in C, g \in G$ are implied by the aforementioned. The solver starts looking from this starting position and if this starting position is reasonably good, the solver might find the optimal solution in a limited amount of time. This procedure is called MIP-Start and is applied to schedules which can not be improved further by the methods as described in previous sections.

6 Computational Results

In this section the parameters of the algorithm are tuned and the corresponding computational results of the algorithms are discussed. All methods are implemented in **Python** version 3.8.9 and the MIP formulation and the MIP-Start procedure both use the solver **Gurobi** version 9.5.1. All numerical results are obtained by running the experiments on a computer with a 2.2 GHz Quad-Core Intel i7 and 16 GB RAM. In Section 6.1 the parameters are tuned and in Section 6.2 the computational results of the algorithms are discussed.

6.1 Parameter Tuning

The Solution Initialization, Tabu Search and Large Neighborhood Search contain several parameters and the tuning is discussed in this section. Firstly, the tuning of α, the parameter in the SIH is discussed in Section 6.1.1. Next, the tuning of the parameters used in the TS, κ_{tabu}, Λ_{tabu}, $\mathcal{M}_{tabu}$, is discussed in Section 6.1.2. Lastly, all parameters of the LNS are tuned and discussed in Section 6.1.3.

6.1.1 Solution Initialization Heuristic

The Solution Initialization Heuristic has one parameter named α, which effects the insertion criterion, Equation (25). As the value of α approaches one, the insertion criterion has an emphasis on added duration over added distance and vice versa for α approaching zero. As can be seen in Figure 1, the different values of α do not have a large, for that matter significant, impact on the total number of tours or the total duration. Which can be explained by the correlation between distance and duration, as distance increases duration increases and vice versa. Only for smaller distances, the duration may not decrease with the same factor, for example if the shortest path (in a straight line) crosses a river or a large park without roads, however this occurs rarely in this particular data set. As the number of customers increases, the total duration is slightly lower for higher values of α, which can be explained by the emphasis on the total duration instead of the total distance. Hence, the value of α is set to 0.9, in order to minimize total duration while still taking the distance into account.

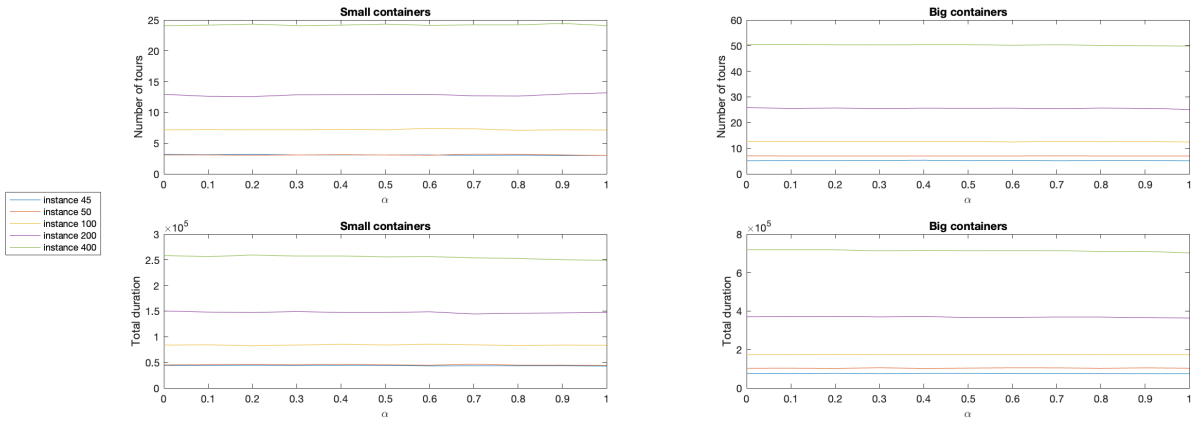


Figure 1: Solution Initialization Heuristic results for different values of α.

6.1.2 Tabu Search

The Tabu Search as described in Section 5.3 has three parameters, κ_{tabu}, Λ_{tabu} and $\mathcal{M}_{tabu}$. The neighborhood in which the Tabu Search is performed is determined by κ_{tabu} and takes a value from 1 to 5. The parameter Λ_{tabu} determines the length of the tabu list, the longer this list, the further backwards the search is able to go in a neighborhood. Having Λ_{tabu} to large leads to undesirable large running times and Λ_{tabu} to small complicates escaping local optima as we can not search back far enough. The different values for Λ_{tabu} tested are 100, 250, 500 and 1000. The maximum number of iterations we are allowed to search without improvement is determined by $\mathcal{M}_{tabu}$. The value of $\mathcal{M}_{tabu}$ is a trade-off between time-efficiency and comprehensive searching. The lower $\mathcal{M}_{tabu}$, the faster the search is completed at the risk of missing a better solution and for higher $\mathcal{M}_{tabu}$ vice versa. $\mathcal{M}_{tabu}$ is tested for the values 50, 100, 200, 500 and 1000. All neighborhoods are combined with each possible pair of Λ_{tabu} and $\mathcal{M}_{tabu}$, the results are given in Table 2.

Table 2: Average improvement and standard deviation of Tabu Search over Solution Initialization per neighborhood for all combinations of tabu list length and maximal number of iterations

	κ_{tabu}	1	2	3	4	5
Small containers	Average improvement	4.96%	12.84%	4.44%	5.98%	8.26%
	St. Dev improvement	0.64%	1.49%	1.16%	0.85%	1.35%
Big containers	Average improvement	0.98%	8.18%	8.54%	1.16%	11.73%
	St. Dev improvement	0.26%	1.28%	0.90%	0.44%	1.19%

As can be seen from Table 2, for the small container instances neighborhood two, Or-Opt*, results in an average improvement of 12.84% which is significantly more than the others. The first, third and fourth neighborhood underperform in comparison to the other two, which can be explained

by the inter-route nature of these neighborhoods as opposed to the intra-route nature of the other two. The room for improvement within routes is smaller as routes have been greedily constructed without regard to the other routes and hence there is more room for improvement between routes. For the big container instances neighborhood five, 2-Opt*, results in an average improvement of 11.73% which is significantly more than the others. Neighborhood one and four result in small improvements for the same reason as the small container instances. However neighborhood three, which swaps two customers, seems to perform well in comparison to its performance for small container instances, which can be explained by the structure of the solution for big container instances. In this case when a pickup is performed the truck must return to the depot, hence swapping two pickup customers can result in big improvements, while for the small container instances this is not true. With the best neighborhood known for both small and big container instances, the average improvement and running times for different values of Λ_{tabu} and $\mathcal{M}_{tabu}$ are shown in Table 3 and 4.

Table 3: Average improvement and running times of Tabu Search after Solution Initialization in the best neighborhood for different lengths of the tabu list over all different values for the maximal number of iterations

	Λ_{tabu}	100	250	500	1000
Small containers	Average improvement	12.52%	13.01%	12.63%	13.20%
	Average time (s)	17.66	29.10	37.58	44.96
Big containers	Average improvement	12.50%	11.00%	11.53%	11.90%
	Average time (s)	74.34	81.71	98.30	101.76

Table 4: Average improvement and running times of Tabu Search after Solution Initialization in the best neighborhood for different number of maximal iterations over all different values for the tabu list length

	$\mathcal{M}_{tabu}$	50	100	200	500	1000
Small containers	Average improvement	12.39%	12.05%	12.59%	13.40%	13.77%
	Average time (s)	3.44	6.50	16.06	37.50	98.13
Big containers	Average improvement	11.99%	11.29%	10.88%	12.73%	11.75%
	Average time (s)	13.08	23.69	50.53	127.73	230.11

As can be seen from Table 3, for both the the small and big container instances the difference in performance is minimal for different values of Λ_{tabu}, while the difference in running times is significant. Hence, a lower value for the length of the tabu list Λ_{tabu} is chosen. For the maximal number of iterations, as can be seen from Table 4, this trend continues, hence a lower value for $\mathcal{M}_{tabu}$ is set. Performance measures for each combination of tabu list length and maximal number of iterations can be found in Appendix A. In conclusion, the final values for the parameters of the Tabu Search can be found in Table 5.

Table 5: Final values for the Tabu Search parameters for both small and big container instances

	κ_{tabu}	Δ_{tabu}	$\mathcal{M}_{tabu}$
Small containers	2	100	50
Big containers	5	100	50

6.1.3 Large Neighborhood Search

The parameters of the LNS as described in Section 5.4 are tuned in three phases. Firstly, the destroy and repair specific parameters are tuned, thereafter the weight updating factors are tuned and lastly the maximum number of iterations. In Table 6 all different values for the parameters in the first phase are displayed with in bold the final values used in the LNS. The three parameters p_{random}, p_{cost} and $p_{cluster}$ determine the proportion of customers to be removed from the solution. As the aim of the Random Removal method is to diversify the solution and the average improvement is similar for all values, see Appendix A, the value is set to 0.25. For both p_{cost} and $p_{cluster}$ the value is set to 0.15, as this value has significantly better performance than the low value, see Appendix A, comparable performance to the high value while not being as destructive. For the repair methods parameters p_{GRASP} and κ_{regret}, the final value is the one that yields the best improvement on average, see Appendix A for details.

Table 6: All the different values used in LNS parameter tuning, with in bold the final value, using the small container instance with 50 customers

Component	Parameter	Value's			
Destroy	p_{random}	0.075	0.15	**0.25**	-
	p_{cost}	0.075	**0.15**	0.25	-
	$p_{cluster}$	0.075	**0.15**	0.25	-
Repair	p_{GRASP}	**0.05**	0.1	0.15	0.2
	κ_{regret}	2	3	**4**	5

In the second phase of the LNS parameter tuning the weight updating factors, $\delta^+_{destroy}$, $\delta^-_{destroy}$, δ^+_{repair}, δ^-_{repair} are tuned. In Table 7 the results of the parameter tuning are shown, with in bold the final values for each parameter. All combinations of $\delta^+_{destroy}$, $\delta^-_{destroy}$, δ^+_{repair}, δ^-_{repair} are tested and are sorted from most improving to least improving. The final values as in Table 7 are the most frequent values in the top 50, 100 and 200 of this list.

Lastly, in the third phase the maximum number of iterations parameter is tuned. The results of can be found in Table 8, as the maximal number of iterations increase the average improvement does as well. As the LNS is allows more iterations without improvement, more possible solution are constructed and hence better objectives are found but at the cost of computation time. A length

of 100 is chosen as this yields a relative high improvement with manageable running times.

Table 7: All the different values used in the second phase of LNS parameter tuning, with in bold the final value, using the small container instance with 50 customers

Parameter	Values					
$\delta^+_{destroy}$	1.05	1.1	1.2	1.5	**1.75**	2
$\delta^-_{destroy}$	0.9	0.95	**0.975**	0.99	1	
δ^+_{repair}	1.05	1.1	1.2	1.5	1.75	**2**
δ^-_{repair}	0.9	0.95	**0.975**	0.99	1	

Table 8: Results for the third phase of LNS parameter tuning, with in bold the final value, using the small container instance with 50 customers

$\mathcal{M}_{LNS}$	50	75	**100**	200	500
Average improvement	7.53%	9.18%	13.47%	13.49%	17.67%
Runtime (s)	1.48	2.66	4.15	7.04	20.15

6.2 Algorithm Results

When the parameters have been set to their final value, the performances of the algorithms are evaluated. All methods are run for all instances of both the small and the big container data sets. The MIP formulation and MIP-Start procedure are run with a time limit of 30 minutes to obtain a lower and upper bound, an overview of the computational results can be found in Table 9 and 10. Optimality gaps are calculated with respect to the best known bound, which in all cases is obtained by MIP-Start procedure.

For the small container instances, Table 9, the SIH is able to obtain a first solution within a small amount of time for all instance sizes with. This solution is greatly improved by applying VND, as this yields in a decrease of 12.69%, 18.93% and 18.71% of the optimality gap, for the 50, 100 and 200 customer instance respectively. Using this solution, Tabu Search, Tabu Search plus LNS and LNS are performed in order to improve the solution further. Tabu Search is a computationally heavy component as can be seen from the respective running times, with a decrease in optimality gaps of 0.51%, 0.48% and 0.26%. Applying LNS to this solution results in another decrease in optimality gap of 2.31%, 4.49% and 2.93% which results in the lowest optimality gaps, excluding the MIP-Start. When the Tabu Search is skipped and only LNS is applied after VND, the resulting optimality gaps are similar to not skipping Tabu Search. For the instance with 50 customers, MIP-Start procedure results in a optimal solution within half an hour of 33028.7. Whereas for the 100 and 200 customer instance an optimality gap of 2.08% and 13.36% remains but the lower bound is

improved in comparison to the bound obtained by the MIP.

Table 9: Results of all methods for the small container instances

Method	Instance size	UB	LB	Optimality gap	Runtime (s)
MIP		90744	32797.3	63.86%	1800
SIH		43451.2	-	23.99%	0.09
SIH + VND		37237.3	-	11.30%	0.95
SIH + VND + TS	50	37025.2	-	10.79%	10.12
SIH + VND + TS + LNS		36090.1	-	8.48%	12.24
SIH + VND + LNS		36107.6	-	8.53%	3.07
MIP-Start		33028.7	33028.7	0.00%	1233.11
MIP		141912	51066.4	64.02%	1800
SIH		76462.4	-	32.37%	0.38
SIH + VND		59734.0	-	13.44%	12.42
SIH + VND + TS	100	59409.0	-	12.96%	57.19
SIH + VND + TS + LNS		56368.7	-	8.27%	64.84
SIH + VND + LNS		56730.7	-	8.85%	20.07
MIP-Start		52806.5	51708.12	2.08%	1800
MIP		354743	79760.7	77.52%	1800
SIH		122876.0	-	33.61%	1.50
SIH + VND		95861.7	-	14.90%	102.43
SIH + VND + TS	200	95572.6	-	14.64%	335.87
SIH + VND + TS + LNS		92401.3	-	11.71%	372.95
SIH + VND + LNS		92852.8	-	12.14%	37.08
MIP-Start		94156.9	81579.2	13.36%	1800

The results for the big container instances can be found below in Table 10. The SIH obtains a solution in less than a second for all instances and the constructed solutions are at optimality gaps of 13.33%, 15.59% and 13.15% for 50, 100 and 200 customers respectively. The SIH on big container instances outperforms the SIH on small containers instances, this can be explained by the capacity restrictions of the big containers instances. With the big container instances, once a vehicle has performed a pickup full capacity of that truck is used and it must return to the depot. This results in considerably shorter routes, for which the greedy approach of the SIH is better suited.Performing a Variable Neighborhood Descent on these solutions yields in a decrease of 11.94%, 14.52%, 12.36% in the optimality gap. The Tabu Search, Tabu Search plus LNS and LNS alone are not able to close the remaining optimality gap. Using the MIP-Start procedures optimal values for all three instances are found, within considerable low times for the 50 and 100 customer instances and for the 200 customer instance in little over 15 minutes.

Table 10: Results of all methods for the big container instances

Method	Instance size	UB	LB	Optimality gap	Runtime (s)
MIP		227027	82758.2	63.55%	1800
SIH		95792.8	-	13.33%	0.01
SIH + VND		84194.7	-	1.39%	1.79
SIH + VND + TS	50	83875.3	-	1.01%	10.13
SIH + VND + TS + LNS		83861.1	-	0.99%	11.58
SIH + VND + LNS		84194.7	-	1.39%	3.24
MIP-Start		83026.7	83026.7	0.00%	20.48
MIP		481390	164590.3	65.81%	1800
SIH		194985.7	-	15.59%	0.03
SIH + VND		166373.7	-	1.07%	16.51
SIH + VND + TS	100	166138.6	-	0.93%	51.27
SIH + VND + TS + LNS		166114.3	-	0.92%	58.08
SIH + VND + LNS		166261.4	-	1.01%	23.33
MIP-Start		164590.3	164590.3	0.00%	70.43
MIP		952892	310560.9	67.41%	1800
SIH		357648.3	-	13,15%	0.18
SIH + VND		313104.2	-	0,79%	184.29
SIH + VND + TS	200	312805.3	-	0,70%	400.96
SIH + VND + TS + LNS		312805.3	-	0,70%	445.71
SIH + VND + LNS		313104.2	-	0,79%	229.04
MIP-Start		310627.7	310627.7	0.00%	1075.09

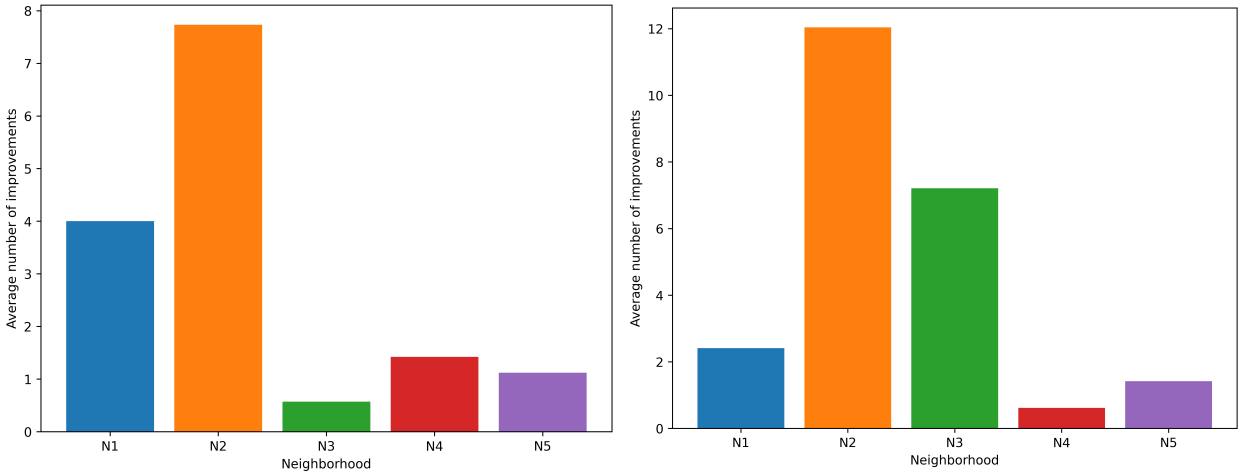
6.2.1 VND Analysis

As can be seen from Table 9 and 10 the VND reduces the optimality gap the most, hence in this section its performance is analyzed in more detail. Below in Figure 2a the average number of improvements found per neighborhood for the small container instances are visualised and in Figure 2b similarly for the big container instances. For both the small and big container instances the most improvements are found in the second neighborhood, Or-Opt*. The SIH has a greedy nature and is thus focused on constructing the shortest route with respect to itself but not with respect to other routes. Hence, the intra-route nature of Or-Opt* and its position as the second neighborhood explains its performance.

For the small container instances neighborhood one, Or-Opt, is the second best performing neighborhood and significantly outperforms the third, fourth and fifth neighborhoods. On the one hand this can be explained by the fact that it is the first neighborhood and is thus visited every iteration and on the other hand by the inter-route nature. As the routes for small container instances generally speaking contain more customers in comparison to big container instances, where

this neighborhood is outperformed by the third, the room for improvement is bigger. The greedy nature of the SIH only focuses on the best next customer and not on the customers thereafter, hence improvements can also be found within each route. The neighborhoods, N_3, N_4 and N_5, further refine the solution but do not find improvements as frequently.

For the big container instances the third neighborhood, Swap, is the second best performing. It is significantly better in comparison to neighborhood N_1, which can be explained by the capacity restrictions for the big container instances. Moving a customer from one route to another route may result in an infeasible solution. For instance moving a pickup customer to a route with already a pickup customer is always infeasible, as is moving a delivery customer after a pickup customer. The number of feasible moves is very restricted in comparison to swapping two customers, as swapping two pickup customers, or two delivery customers, usually does not violate the capacity restriction. Hence, more improvements are found in neighborhood N_3 and it outperforms N_1, N_4 and N_5. Note that the intra-route neighborhoods N_2 and N_5 outperform the inter-route neighborhoods N_1 and N_4, also because of the tight restrictions on capacity.



(a) Small container

(b) Big container

Figure 2: Average number of improvements over 100 runs found by the VND per neighborhood

6.2.2 LNS Analysis

Lastly, the performance of the different destroy and repair methods used within the LNS are analyzed. In Figure 3 the average weight per method when running the LNS 100 times on the 50 customer small container instance are displayed. In the top figure the weights for the destroy methods are displayed, where the performance of the Cluster Removal, Cost Removal and Random

Removal are most notable. The Cost Removal performs well from beginning to the end, this can be explained by the focus on individual customers who have been placed undesirably and are moved to a more preferable position in the schedule. The Cluster Removal method performs extremely well in the beginning but its performance declines towards the end. Later in the search a greater share of the customers is in a desirable position in the schedule and the cluster removal method is too destructive and removes too many customers. In comparison to early in the search when entire clusters of customers might be placed on a less convenient tour. Notably, the effectiveness of diversifying the search by using the Random Removal method is shown, also the Longest Route removal method has some affect. The inferior performance of the Smallest Route, Tour and the Longest Tour methods stands out and omitting these methods might results in a better overall performance of the LNS.

The performances of the repair methods are displayed on the bottom of Figure 3. Both the Greedy and the κ-Regret repair method perform very well throughout the entire search. For the Greedy method this can be explained by its focus on inserting each customer individually at its most desired position in the schedule. While for the κ-Regret method it can be explained by its ability of looking "ahead", and inserting customers at desirable positions in the schedule with respect to the remaining customers. The performance of the GRASP method is fine but on the other hand the performance of the Random Repair method is not as desired. Once again, omitting this method from the LNS might result in better overall performance of the LNS.

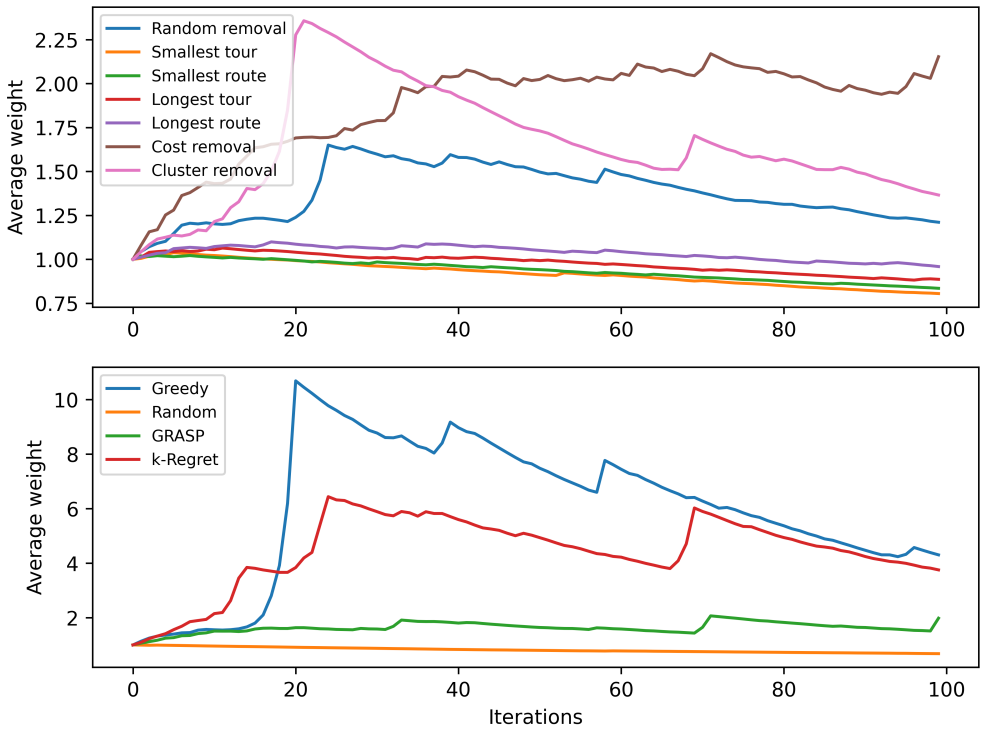


Figure 3: Average weight per destroy (top) and repair (bottom) method over 100 runs of the LNS on the 50 customer small container instance

7 Conclusion

In this thesis, the goal was to develop a solution construction heuristic and several refining methods which could obtain competitive solutions in a short time frame and obtain insights into the performance of these methods. The problem at hand is the distribution of different waste containers to a set of customers using a homogeneous fleet of vehicles. Customers are to be served within a time window and their demand is known in advance and may consist of both a pickup and a delivery or either. The vehicles used to deliver the containers have limited weight and size capacity and must comply to the operating hours of the depot. This is the problem faced by GreenRoutes, a company that supplies route optimization software within the waste processing sector. In the literature this problem is known as a Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows, which belongs to the widely studied class of Vehicle Routing Problems.

A Solution Initialization Heuristic is developed which combines a cheapest insertion heuristic, inspired by Solomon (1987), to construct routes with a greedy heuristic which constructs tours from these routes. The obtained schedule consists of a set of tours of one or multiple routes and is further refined by different heuristics. Firstly, a Variable Neighborhood Descent with five neighborhoods is

constructed to search the solution space close to the initial solution. As this search might get stuck in a local optima, two methods capable of escaping local optima are developed, a Tabu Search and a Large Neighborhood Search. The effectiveness of these methods on the class of Vehicle Routing Problems has been shown by Gendreau et al. (1994) and Ropke & Pisinger (2006) respectively. Tabu Search allows moves which worsen the objective and previously visited solutions are prohibited for a number of iterations, in order to move away from a local optima into unexplored parts of the solution space. In the Large Neighborhood Search a set of destroy methods deconstruct part of the solution and a set of repair methods is used to reinsert the removed customers. Each method in the set of destroy methods has its own purpose, from diversifying the search to removing undesirable customers and routes. The repair methods aim to reinsert the customers in the most desirable place, either in a greedy way or in a more sophisticated way. Lastly, a MIP-Start procedure is implemented which takes the best known solution and starts an exact search using the **Gurobi** solver from that point, which increases the odds of finding the optimal solution dramatically.

The Solution Initialization Heuristic in combination with the Variable Neighborhood Descent is shown to be really effective for the big container instances of all sizes. Within the VND, the intra-route oriented neighborhood Or-Opt*, is shown to be the best performing neighborhood. A near optimal solution can be found within minutes and the MIP-Start procedure is able to find the optimal solution for all three of the big container instances. For the small container instances the SIH with VND finds competitive solutions which are further refined by the Tabu Search and the Large Neighborhood Search. The LNS is shown to be more effective in comparison to the Tabu Search. A deeper analysis of the LNS shows the effectiveness of the Cost, Cluster and Random Removal methods, while the Greedy and κ-Regret are the most effective repair methods. Concluding, the advice to GreenRoutes is to use a combination of SIH, VND and LNS for the small container instances and for the big container instances the SIH and VND combination to obtain competitive results in a reasonable time.

The methods discussed in this thesis contain some limitations, the Tabu Search and the Large Neighborhood Search parameters are both tuned only on the 50 customer instances. More research can be done on the relation between the instance size and these parameters, as some destroy methods might be too destructive on the larger instances. Also the reactiveness of the LNS can be further researched, as in the current weight updating scheme weights can diverge a lot and specific good/bad combinations are not rewarded/punished. Within the LNS, other destroy and repair methods might be considered which focus on for example reducing the total number of

vehicles used or removing customers with similar characteristics. The use of a local search or a Tabu Search on a competitive solution found by the LNS might be useful to refine the solution further. Some neighborhoods used in the VND rarely find improvements and might not be worth the computational time. As VND is the most effective method developed in this thesis, research into omitting or replacing these neighborhoods could be considered for future research. Lastly, more research can be done into the insertion criterion used in the SIH, for example to include a looking "ahead" feature such as in the κ-Regret method in this thesis.

References

Angelelli, E., & Mansini, R. (2002). The vehicle routing problem with time windows and simultaneous pick-up and delivery. In *Quantitative approaches to distribution logistics and supply chain management* (pp. 249–267). Springer.

Azi, N., Gendreau, M., & Potvin, J.-Y. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, *202*(3), 756–763.

Baldacci, R., Bartolini, E., & Mingozzi, A. (2011). An exact algorithm for the pickup and delivery problem with time windows. *Operations research*, *59*(2), 414–426.

Baldacci, R., Mingozzi, A., & Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, *218*(1), 1–6.

Bent, R., & Van Hentenryck, P. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, *33*(4), 875–893.

Bräysy, O., & Gendreau, M. (2005a). vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, *39*(1), 104–118.

Bräysy, O., & Gendreau, M. (2005b). vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation science*, *39*(1), 119–139.

Chen, J.-F., & Wu, T.-H. (2006). Vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Operational Research Society*, *57*(5), 579–587.

Cordeau, J.-F., Laporte, G., Potvin, J.-Y., & Savelsbergh, M. W. (2007). Transportation on demand. *Handbooks in operations research and management science*, *14*, 429–466.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management science*, *6*(1), 80–91.

Dumas, Y., Desrosiers, J., & Soumis, F. (1991). The pickup and delivery problem with time windows. *European journal of operational research*, *54*(1), 7–22.

Emeç, U., Çatay, B., & Bozkaya, B. (2016). An adaptive large neighborhood search for an e-grocery delivery routing problem. *Computers & Operations Research*, *69*, 109–125.

Flood, M. M. (1956). The traveling-salesman problem. *Operations research*, *4*(1), 61–75.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability* (Vol. 174). freeman San Francisco.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science*, *40*(10), 1276–1290.

Goetschalckx, M., & Jacobs-Blecha, C. (1989). The vehicle routing problem with backhauls. *European Journal of Operational Research*, *42*(1), 39–51.

Golden, B. L., Raghavan, S., & Wasil, E. A. (2008). *The vehicle routing problem: latest advances and new challenges* (Vol. 43). Springer Science & Business Media.

Hernandez, F., Feillet, D., Giroudeau, R., & Naud, O. (2014). A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. *4or*, *12*(3), 235–259.

Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, *35*(7), 2307–2330.

Kumar, S. N., & Panneerselvam, R. (2012). A survey on the vehicle routing problem and its variants.

Laporte, G., & Nobert, Y. (1987). Exact algorithms for the vehicle routing problem. In *North-holland mathematics studies* (Vol. 132, pp. 147–184). Elsevier.

Li, H., & Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, *12*(02), 173–186.

Liu, R., Xie, X., Augusto, V., & Rodriguez, C. (2013). Heuristic algorithms for a vehicle routing problem with simultaneous delivery and pickup and time windows in home health care. *European Journal of Operational Research*, *230*(3), 475–486.

Montané, F. A. T., & Galvao, R. D. (2006). A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, *33*(3), 595–619.

Nanry, W. P., & Barnes, J. W. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, *34*(2), 107–121.

Ombuki, B., Ross, B. J., & Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, *24*(1), 17–30.

Ropke, S., & Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, *43*(3), 267–286.

Ropke, S., Cordeau, J.-F., & Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks: An International Journal*, *49*(4), 258–272.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, *40*(4), 455–472.

Savelsbergh, M. W., & Sol, M. (1995). The general pickup and delivery problem. *Transportation science*, *29*(1), 17–29.

Schneider, M., Stenger, A., & Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation science*, *48*(4), 500–520.

Shi, Y., Boudouh, T., & Grunder, O. (2018). An efficient tabu search based procedure for simultaneous delivery and pick-up problem with time window. *IFAC-PapersOnLine*, *51*(11), 241–246.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, *35*(2), 254–265.

Toth, P., & Vigo, D. (1997). An exact algorithm for the vehicle routing problem with backhauls. *Transportation science*, *31*(4), 372–385.

Toth, P., & Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Wang, & Chen. (2012). A genetic algorithm for the simultaneous delivery and pickup problems with time window. *Computers & industrial engineering*, *62*(1), 84–95.

Wang, Zhou, Y., Wang, Y., Zhang, J., Chen, C. P., & Zheng, Z. (2015). Multiobjective vehicle routing problems with simultaneous delivery and pickup and time windows: formulation, instances, and algorithms. *IEEE transactions on cybernetics*, *46*(3), 582–594.

Wang, C., Mu, D., Zhao, F., & Sutherland, J. W. (2015). A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering*, *83*, 111–122.

Wassan, N. A., & Nagy, G. (2014). Vehicle routing problem with deliveries and pickups: modelling issues and meta-heuristics solution approaches. *International Journal of Transportation*, *2*(1).

Wassan, N. A., Wassan, A. H., & Nagy, G. (2008). A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries. *Journal of combinatorial optimization*, *15*(4), 368–386.

A Parameter Tuning

Table 11: Results parameter tuning Tabu Search for small container instance with 50 customers in N_2

Δ_{tabu}	$\mathcal{M}_{tabu}$	Improvement	Runtime (s)
100	50	13.13%	3.09
100	100	13.11%	6.26
100	200	12.39%	10.99
100	500	11.22%	23.48
100	1000	12.76%	44.50
250	50	12.89%	3.59
250	100	10.46%	5.96
250	200	13.02%	16.26
250	500	15.84%	37.86
250	1000	12.85%	81.82
500	50	9.76%	4.13
500	100	12.45%	7.59
500	200	11.75%	19.83
500	500	14.79%	36.33
500	1000	14.39%	120.03
1000	50	13.78%	2.95
1000	100	12.19%	6.19
1000	200	13.19%	17.15
1000	500	11.76%	52.34
1000	1000	15.07%	146.18

Table 12: Results parameter tuning Tabu Search for big container instance with 50 customers in N_5

Δ_{tabu}	$\mathcal{M}_{tabu}$	Improvement	Runtime (s)
100	50	11.94%	14.54
100	100	11.77%	27.73
100	200	12.52%	51.07
100	500	13.82%	118.26
100	1000	12.44%	160.09
250	50	11.71%	10.61
250	100	10.98%	20.65
250	200	9.44%	44.48
250	500	11.52%	117.85
250	1000	11.33%	214.98
500	50	11.99%	13.08
500	100	10.86%	27.16
500	200	11.72%	54.26
500	500	13.38%	137.33
500	1000	9.68%	259.69
1000	50	12.30%	14.11
1000	100	11.56%	19.24
1000	200	9.86%	52.34
1000	500	12.20%	137.46
1000	1000	13.56%	285.66

Table 13: Average improvement and standard deviation of LNS over Solution Initialization for different values of destroy parameters on the 50 customer small containers instance

Parameter	Value	Average	St. Dev
p_{random}	0.075	11.03%	4.65%
	0.15	11.38%	4.46%
	0.25	11.41%	4.52%
p_{cost}	0.075	9.80%	4.44%
	0.15	11.75%	4.39%
	0.25	12.27%	4.43%
$p_{cluster}$	0.075	10.91%	4.39%
	0.15	11.40%	4.60%
	0.25	11.51%	4.64%

Table 14: Average improvement and standard deviation of LNS over Solution Initialization for different values of repair parameters on the 50 customer small container instance

Parameter	Value	Average	St. Dev
p grasp	0.05	12.22%	4.30%
	0.1	11.36%	4.37%
	0.15	10.84%	4.55%
	0.2	10.68%	4.81
k-regret	2	10.76%	4.56%
	3	11.44%	4.50%
	4	11.52%	4.62%
	5	11.37%	4.48%