ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master thesis Operations Research and Quantitative Logistics

Insights in an Adaptive Large Neighborhood Search-Path Relinking Algorithm for the Vehicle Routing Problem with Simultaneous Pick Up and Delivery, Time Windows and Multiple Trips.

Author:

Luuk Trouw

Student ID number:

458204

Supervisors:

Dr. T. Dollevoet (EUR)

R.A.O Salem (GreenRoutes)

Second Assessor:

Dr. R. Spliet (EUR)

November 13, 2021

Abstract

In this paper, we consider a practical waste container distribution problem which can be modelled as a Vehicle Routing Problem with Simultaneous Pick Up and Delivery and Time Windows (VRPSPDTW) with the allowance of multiple trips per vehicle. An Adaptive Large Neighborhood Search-Path Relinking (ALNS-PR) algorithm is developed to solve this problem. The performance of the algorithm, with emphasis on the PR component, is examined. We find that the algorithm is able to find solutions relatively close to an obtained lower bound. The PR component has a slight positive impact on the solution value as well as the speed of converging to high quality solutions. An iteration performed by the PR component is found to have higher chances to find improvements than an ALNS iteration. A proposed method of updating the elite set used for the PR component shows to perform well, which results in a well-performing PR component at each stage in the algorithm.

Contents

Nomenclature

Sets

N	The set of the depot and all customers.
C	The set of all customers.
V	The set of all vehicles.
A	The set of all arcs between nodes in N
A_C	The set of all arcs between customers in C
H	The set of all delivery container types.
F	The set of all pick up container types.

Indices

i, j	Indices representing a certain node from the set N.
(i, j)	Index representing a certain arc from the set A that starts at node i and ends at node j.
f	Index representing a certain pick up container type from the set F.
h	Index representing a certain delivery container type from the set H.

Variables

x_{ij}	Binary decision variable indicating whether a vehicle travels directly from node i to node j.
x'_{ij}	Binary decision variable indicating whether a vehicle ends a trip at node i, visits the depot, and starts its next trip with node j.
$y_{ij,f}$	Decision variable denoting the demand picked up from customers up to customer i and transported over arc (i, j), of container type f.
$z_{ij,h}$	Decision variable denoting the delivery demand still in the vehicle of container type h after visiting customer i which remains to be delivered and is transported over arc (i, j).
t_i	Decision variable denoting the starting time of serving customer i.

u Integer decision variable indicating the number of vehicles exceeding the maximum.

Parameters

T_{ij} The travel time from location i to location j.

Q The size capacity of a vehicle.

W The weight capacity of a vehicle.

S_i The service time of customer i.

S_0^+ (S_0^-) The loading (unloading) time of a vehicle at the depot at the beginning (ending) of a trip.

d_{if} The delivery demand of customer i of container type f.

p_{ih} The pick up demand of customer i of container type h.

a_i The starting time of the time window of customer i.

b_i The ending time of the time window of customer i.

a_0 The starting time of the time horizon.

b_0 The ending time of the time horizon.

w_f^p The weight of a pick up container of container type f.

w_h^d The weight of a delivery container of container type h.

l_f^p The size of a pick up container of container type f.

l_h^d The size of a delivery container of container type h.

τ The penalty factor per vehicle exceeding the maximum number of allowed vehicles.

λ The maximum number of solutions in the elite set.

ρ The maximum fraction of the initial non-overlapping arcs which may be created in PR.

ξ The number of iterations without new best after which the algorithm terminates (stopping criterion).

ξ^{reset} The number of iterations without new best after which the algorithm resets its solution.

ζ	The maximum proportion of all customers which can be deleted in an ALNS iteration.
π^N, π^-, π^+	Respectively the scores of number of customers to remove, removal operators and repair operators.
ι	The number of different customer removal sizes in an ALNS iteration.
χ^{graps}	The fraction of best possible moves considered in the GRASP repair operator.
ψ	The number of iterations at the beginning of the algorithm, which are free of score updating.
γ	The number of iterations after which a score update takes place.
σ^{best}, σ^{imp}, σ^{acc}	The score an operator receives if it finds a new best, better or worse accepted solution.
ϕ	The total scores an operator has collected before a weight update.
β	The number of times an operator is used before a weight update.
α	The sensitivity of score updates.

1 Introduction

Consider a waste container distribution network consisting of a single depot which distributes large construction waste containers. These containers are mostly used in the construction sector to throw away debris from demolished buildings. Since this type of waste is often bulky and heavy, mostly standardized containers are used. Transport is done using specialized trucks, that have specific handling equipment to pick up the containers. Customers can request the rental of empty containers, which they can fill up with large waste quantities during the rental. After the rental expires, the full containers are picked up to return them to the depot. For the placement of empty containers, multiple delivery customers are introduced. Additionally, pick-up customers are introduced, where full containers must be picked up to be emptied at the depot. Some customers may also wish to both receive empty and release full containers. This can happen, for example, if a container is full while there is still debris left. To serve customers, a homogeneous fleet of vehicles can be used to transport containers from and to the depot. At the depot, empty containers are stored in inventory. Furthermore, emptying of full containers also takes place at the depot, which is the only place to bring full containers. Each vehicle has a limited capacity in terms of quantity and weight. Before the vehicle planning is made, the customer demands are known. Additionally, each customer has a known time window in which it has to be served. There are several different types of containers in circulation, each with a specific size and weight. Since collected full containers take up a great deal of the capacity of a vehicle, it is presumably not possible to satisfy all customer demands with at most one trip per vehicle. Therefore, vehicles are allowed to make multiple trips from and to the depot to pick up or deliver new containers. When a vehicle is at the depot, it is allowed to wait before it starts its next trip. The latter can be useful to satisfy time window requirements of customers. Since the vehicles in use can be relatively big and therefore might cause traffic jams at customers, the depot is the only place where vehicles are allowed to wait.

For several waste collection companies this is a recognizable situation. Since these companies mostly have access to a limited fleet of special vehicles which have the proficiency to move containers, it is common that the same vehicles which deliver containers also pick up the containers. This problem also stretches to other logistical companies. In recent years, with the extending possibilities of the internet, smaller companies seek opportunities to deliver products. Thereby, the demand for online shopping has grown fiercely. However, not all stores have a warehouse to store products for a longer time. Another problem could be the requirement of having a delivery vehicle. Even if a store has a warehouse and a delivery vehicle available, if the online sales are insufficient, it might not be efficient to deliver the products themselves. Among

others, the above reasons led to an increasing demand for delivery companies. Delivery companies can efficiently serve multiple customers, by combining pick ups at multiple stores and deliveries to the customers. In contrast to waste collection companies, these delivery companies exploit their depot by using it as an intermediate station to store products, before delivering them to customers.

These companies all face complex logistical puzzles. To improve the competitive position, they try to keep their transportation costs as low as possible. Also, with the upcoming climate awareness, companies try to lower their emissions as much as possible. If companies are able to shorten routes, whilst meeting all constraints, they will save on emission, labour and gasoline costs. Therefore, it is important to plan routes as efficiently as possible. In particular, this paper is motivated by a practical application of a waste container distribution network provided by the company GreenRoutes. In this paper the aim is to develop a well-performing algorithm to find solutions to the specific problem stated above within reasonable time. The main objective is to minimize the total travel time. Another aim in this paper is to evaluate the performance of (specific parts of) the algorithm.

The remainder of this paper is structured as follows. In Section 2, a review of relevant existing literature is provided and the relation to the proposed problem is discussed. The problem is placed in its scientific field and the proposed contribution is mentioned. In Section 3, a formal mathematical problem definition is set out. In Section 4, the solution methods to solve the problem are proposed. Furthermore, Section 5 discusses the necessary data. In Section 6, the obtained computational results are presented and discussed. Finally, in Section 7 the main findings are summed up and in Section 8 possible further research is discussed.

2 Literature Review

The problem stated in the introduction can be seen as an adjusted form of the vehicle routing problem with simultaneous pick up and delivery and time windows (VRPSPDTW), which is a variant of the vehicle routing problem (VRP). This section reviews several relevant variants of the VRP and heuristics used to solve these variants. The VRP is a widely studied optimization problem, which was first described by Dantzig and Ramser [1959]. The original VRP considers a set of vehicles which have to supply the demand of a set of customers in such a manner that the total distance covered is minimized and all demands are satisfied. The VRP is computationally challenging due to the NP-hard nature of the problem. Therefore, researchers have been aiming to develop efficient algorithms to solve instances of the VRP.

2.1 VRP with time windows

In subsequent years, several variants of the VRP were introduced. The VRP with time windows (VRPTW) takes specific time windows in which customers must be visited into account. Kallehauge [2008] examined several exact methods to solve the VRPTW. Meanwhile, Bräysy and Gendreau [2005a] reviewed route construction heuristics and local search algorithms for this variant of the VRP. In their second part, Bräysy and Gendreau [2005b] also evaluated multiple metaheuristics developed for the VRPTW.

2.2 VRP with pick up and delivery

The VRP with pick up and delivery (VRPDP) considers the case that some customers do not only wish to receive goods, but also to deliver goods to the depot. We distinguish three main categories within the VRPDP, namely the VRP with backhauling (VRPB), the VRP with mixed pick ups and deliveries (VRPMPD) and the VRP with simultaneous pick ups and deliveries (VRPSPD). Wassan and Nagy [2014] further elucidated there three categories. Thereby, mathematical formulations are given to fit the three problems into perspective. In all cases, taking goods directly from one customer to another is not allowed. If this would be permitted, it would not be a proper extention of the VRP. This problem is considered as a pick up and delivery problem (GPDP). A GPDP is a generalization of the VRP which, for example, occurs often at taxi services. VRP is a special case of the GPDP, where all pick up locations are located at the depot, as Savelsbergh and Sol [1995] clarified.

In the VRP with backhauling (VRPB), it is required that pickups can only begin after all deliveries are done. Salhi and Nagy [1999] discussed several insertion heuristics for the VRPB, including a newly introduced cluster insertion heuristic. These insertion heuristics are useful as starting solutions for several meta-heuristics. The VRPMPD does not require pick ups to be performed after deliveries. Additionally, customers are only allowed to either have a pick up or delivery demand. The VRPSPD is very similar to the VRPMPD, in this case customers can have a pick up and a delivery demand simultaneously. It is noteworthy that the VRPMPD can be modelled as the VRPSPD by either adding a pickup or delivery demand of zero to each customer. Therefore, the VRPSPD is at least as hard as the VRPMPD. To construct a good initial solution for the VRPSPD, Chen and Wu [2006] proposed an insertion-based constructive heuristic. Bianchessi and Righini [2007] contributed by comparing multiple heuristic methods applied to the VRP-

SPD. They focus primarily on comparing the computational time, whilst also denoting the objective values. Multiple constructive heuristics are also investigated. Thereby, it compares local search algorithms and tabu search algorithms. In particular the performance of different neighborhoods, such as the RELOCATE and CROSS neighborhoods, are useful for designing algorithms to solve instances of the VRPSPD. An interesting method to solve the VRPSPD is discussed by Nagy and Salhi [2005]. It points out a four phase heuristic to optimize the problem, which allows *weakly feasible* solutions during the first phases. In the second and fourth phase, it improves the currently found solution using numerous suitable neighborhoods for vehicle routing problems. By doing this, Nagy and Salhi [2005] also described how well each combination of neighborhoods performs.

2.3 Roll on-roll off VRP and VRP with multiple trips

Another variant of the VRP is the Roll on-roll off VRP (RR-VRP). This problem requires vehicles to deliver products from the depot to customers and to collect products from customers and return it to the depot. Often multiple depots exist in this problem. In the RR-VRP it is often assumed for vehicles to have a capacity of only one or two products, which is common in the waste container collection industry. The vehicles will probably visit the depot multiple times. However, as for the GPDP, in the RR-VRP it is often allowed to relocate products between customers. Since RR-VRP problems usually do not contain more than fifteen customers, Raucq et al. [2019] developed a column generation algorithm that generates new columns using a heuristic method. However, if the number of customers grows, computational times of this solution method will most likely be very large. A heuristic approach was introduced by Wy et al. [2013]. As an addition time windows were considered. This heuristic is a large neighborhoods search (LNS) iterative heuristic consisting of a constructive heuristic and an improvement algorithm. The constructive heuristic constructs one route at the time by inserting as many customers in that route as possible. After construction, improvement is searched within several neighborhoods. Afterwards, destroy and insert methods form the base of the LNS part of the algorithm. The algorithm performs significantly better than the benchmark. Hauge et al. [2014] broadened the RR-VRP by considering the case in which a vehicle can transport up to eight containers. To solve this problem, a hybrid column generation procedure is proposed. To solve the pricing problem, a well performing fast tabu search is used. The column generation procedure finds high quality solutions in relatively small computational times. However, for this procedure it is necessary to solve a master problem multiple times.

An interesting aspect of the RR-VRP is the possibility to have multiple trips from and to the depot per vehicle. The VRP with multiple trips (MTVRP) in general relaxes the assumption that vehicles can only make one trip. A comprehensive overview of the structure of the MTVRP is given by Cattaruzza et al. [2016]. Several mathematical formulations of the problem are stated, with the two-index formulations from Koc and Karaoglan [2011] and Rivera et al. [2013] as most notable ones. These formulations stay very close to the wider known formulations for the VRP with single trips. Thereby, insertion heuristics are reviewed to build solutions. Also, several tabu search algorithms for the problem are discussed.

2.4 VRPSPD with time windows

If vehicles are capable of carrying multiple containers, the RR-VRP moves towards the VRPSPD in most aspects. With most RR-VRPs, the relocating of containers between customers is allowed, which remains to be a significant dissimilarity. When these movements are not allowed, and thereby we consider time windows, we can see the resulting problem as a VRPSPD with time windows (VRPSPDTW). Although this is a very specific variant of the VRP which is less investigated than other variants, some noteworthy research has been done in this area. As one of the first to study this variant, an exact algorithm for the VRPSPDTW was proposed by Angelelli and Mansini [2002]. This algorithm is based on a branch and price approach applied to a set covering formulation for the master problem. To obtain integer solutions, a branch and bound method is used. A limitation of this algorithm is that it is very time consuming and cannot obtain acceptable solutions in acceptable time. Wang and Chen [2012] proposed a co-evolution genetic algorithm to solve the VRPSPDTW. Further, benchmark instances are introduced, which are based on the influential benchmark instances introduced by Solomon [1987]. Next, a parallel simulated annealing method is developed by Wang et al. [2015]. Both heuristic approaches show improvement in the solution value compared to the commercial solver of CPLEX, within comparable computational times.

As it often is of primary importance to reduce the number of vehicles used rather than the distance traveled, Shi et al. [2020] developed a lexicographic-based two-stage algorithm to solve the VRPSPDTW. To start off, an initial solution is constructed using an adapted push forward insertion heuristic based on Solomon [1987] and Chen and Wu [2006]. In the first stage, the focus lies on minimizing the number of vehicles using a variable neighborhood search. In the second stage, a tabu search method is used to decrease the total distance traveled. The two-stage algorithm performs well in terms of decreasing the number of vehicles used, while keeping the computational times low. However, this is mostly at the expense of an

increase in distance traveled. On the VRPSPDTW, also multiple evolutionary algorithms are investigated. Belmecheri et al. [2009] and Belmecheri et al. [2013] developed respectively an ant colony optimization and a particle swarm optimization. Both optimizations are complemented with a local search procedure for each solution. In their optimizations, the possibility of a heterogeneous fleet is taken into account.

Finally, Hof and Schneider [2019] developed an innovative hybrid heuristic to solve a class of VRP-SPDs. An adaptive large neighborhood search (ALNS) procedure is combined with a path relinking (PR) procedure. After either an ALNS or PR iteration, local search is applied to the solution. The proposed heuristic appears to be able to accelerate the convergence rate of the search and to return competitive solution values in reasonable time.

2.5 Adaptive Large Neighborhood Search

For many VRP variations, Adaptive Large Neighborhood Search (ALNS) has shown to be a suitable algorithm. The main ingredient of ALNS is the destruction and building of the solution during optimization. The removal and insertion of parts of the solution are performed by altering multiple execution methods. Ropke and Pisinger [2006] developed an ALNS for several VRPSPD classes, which were able to compete with other algorithms. To solve a VRP with multiple trips, Azi et al. [2014] used an ALNS which also demonstrated benefits compared to known benchmarks. Furthermore, Hemmelmayr et al. [2012] also showed the succes ALNS algorithms can have in diversifying the search for more complicated two-echelon VRPs (2E-VRP). A 2E-VRP is a VRP in which products travel to an intermediate facility before arriving at their destination.

2.6 Path Relinking

In combination with solution methods for VRPs, path relinking (PR) can be used as a method to integrate intensification and diversification in the search. PR was originally proposed by Glover [1997]. It aims to discover promising solutions that are not reachable by performing local search. The procedure is based on the assumption that good solutions are likely to share characteristics. It juxtaposes the initial solution to a set of *elite* solutions. By creating a path of solutions between the initial and the guiding solution, the hope is to obtain improved solutions in the process. In the process, the initial solution is transformed into the guiding solution by stepwise incorporating characteristics of the guiding solution. The elite solutions are previously

found solutions with good solution values, which differ sufficiently from one another. The deviation between two solutions can, for example, be measured in terms of the number of arcs exclusively contained in one of both solutions. Ho and Gendreau [2006] showed that applying PR to a tabu search heuristic for solving a VRP significantly increases the results. Hashimoto and Yagiura [2008] and Rahimi-Vahed et al. [2013] both developed algorithms based on PR. The algorithms are used to solve a VRPTW and a multi-depot periodic VRP. Both algorithms show that PR contributes in the improvement of solution quality and computational efficiency.

To sum up, a lot of research has been done on VRPs. However, heuristic methods for instances of the VRPSPDTW are scarce. In the exact instance of the problem addressed in this paper, it is also allowed to carry out multiple trips per vehicle. Therefore, this problem is near the general assumptions of a RR-VRP with enlarged vehicle capacity as discussed by Hauge et al. [2014], although it is not allowed to relocate products between customers. Existing literature on heuristic methods to solve this specific variant of the VRP is limited. Nonetheless, such specific cases are of practical importance, which is also a motivation for the research. This research aims to develop an efficient heuristic method to this specific problem inspired by an adaption of the ANLS-PR procedure discussed by Hof and Schneider [2019]. This research contributes to existing literature by adapting the VRPSPDTW based procedure to the additional condition of having multiple trips per vehicle. Thereby, it contributes by evaluating the performance of specific components of the ALNS-PR algorithm. Further, the heuristic contains no elements of solving mathematical formulations of any kind.

3 Problem description

As stated in the literature section, the problem discussed in this paper can be seen as a special case of the VRPSPDTW. The problem can be modelled on a complete graph $G = (N, A)$, where $N = \{0, ..., n\}$ is a set of locations such that 0 represents the depot and $C = \{1, ..., n\}$ is a subset of N including the customers. Set A is defined as the set of the arcs in the graph, where $(i, j) \in A$ denotes the arc between location i and j for every $i, j \in N$, if $i \neq j$. The set A_C is a subset of A, containing all arcs which are not adjacent to the depot. Let T_{ij} denote the travel time from location i to j using arc (i, j). Furthermore, a fixed maximum number of vehicles is available to use, each with a weight capacity W and a size capacity Q. Let set V denote the set of all vehicles and accordingly let $|V|$ be the total number of vehicles.

Since each customer can have different types of demands, we introduce set H as the set of all types of delivery products and set F as the set of all types of pick up products. Note that, for example, a certain container type can be in both H and F, but is considered empty when it needs to be delivered and full if is needs to be picked up. The service time needed to (un)load demand at customer i is denoted by S_i and depends on the demand of the customer. When at the depot, S_0^+ (S_0^-) denotes the loading (unloading) time of a vehicle at the beginning (ending) of a trip. Furthermore, the delivery (pick up) demand of a certain container type f (h) of customer i is given by d_{if} (p_{ih}). Thereby, each customer may only be serviced within its prespecified time interval $[a_i, b_i]$, where a_i denotes the beginning time of the interval and b_i the ending time of the interval. More specifically, the complete process of starting the service and servicing a customer must be within that interval. The interval in which the all activities have to take place is defined as $[a_0, b_0]$. The different container types can have different sizes and weights. The weight and size of a pick up container of type f is given as w_f^p and l_f^p respectively, whereas for a delivery container of type h these are denoted as w_h^d and l_h^d respectively. The total weight and size of containers loaded on a vehicle is not allowed to exceed the weight and size capacity of the vehicle.

Using the graph G, a mixed-integer linear programming formulation can be provided for the problem. To do so, multiple variables are introduced. First, let binary variable x_{ij} indicate whether a vehicle travels directly from node i to node j. Additionally, binary variables x'_{ij} are introduced to detect when vehicle finishes a trip with customer i, visits the depot, and starts its next trip with customer j ($x'_{ij} = 1$). The usage of this variable is inspired by Koc and Karaoglan [2011]. Let variable t_i denote the starting time of serving customer i for all customers in C. Finally, variables to keep track of the load are introduced. Variable $y_{ij,f}$ indicates the demand picked up from customers up to customer i and transported over arc (i, j), of container type f. Variable $z_{ij,h}$ indicates the delivery demand still in the vehicle of container type h which remains to be delivered after visiting customer i and is transported over arc (i, j). Finally, let u be defined as an integer variable denoting the number of vehicles used exceeding the maximum. Each of those vehicles is penalised with penalty factor τ. The problem can be formulated using the following mixed-integer linear program (MIP):

$$\min \quad \sum_{i\in N}\sum_{j\in N, j\neq i} T_{ij}x_{ij} + u\tau \tag{1}$$

st.

$$\sum_{i\in N\setminus\{j\}} x_{ij} = 1 \qquad \forall j \in C \tag{2}$$

$$\sum_{i\in N\setminus\{j\}} x_{ij} = \sum_{i\in N\setminus\{j\}} x_{ji} \qquad \forall j \in C \tag{3}$$

$$\sum_{j\in C: j\neq i} x'_{ij} \leq x_{i0} \qquad \forall i \in C \tag{4}$$

$$\sum_{i\in C: i\neq j} x'_{ij} \leq x_{0j} \qquad \forall j \in C \tag{5}$$

$$\sum_{j\in C} x_{0j} - \sum_{(i,j)\in A_C} x'_{ij} \leq |V| + u \tag{6}$$

$$t_i + T_{ij} + S_i \leq t_j + M(1-x_{ij}) \qquad \forall (i,j) \in A_C \tag{7}$$

$$t_i + T_{ij} + S_i + M(1-x_{ij}) \geq t_j \qquad \forall (i,j) \in A_C \tag{8}$$

$$t_i + S_i + T_{i0} + S_0^- + T_{0j} + S_0^+ \leq t_j + M(1-x'_{ij}) \qquad \forall (i,j) \in A_C \tag{9}$$

$$a_i \leq t_i \leq b_i - S_i \qquad \forall i \in C \tag{10}$$

$$T_{0j} + S_0^+ + a_0 \leq t_j \qquad \forall j \in C \tag{11}$$

$$t_i + S_i \leq b_0 - S_0^- - T_{i,0} \qquad \forall i \in C \tag{12}$$

$$\sum_{i\in N} y_{ji,h} - \sum_{i\in N} y_{ij,h} = p_{jh} \qquad \forall j \in C, \forall h \in H \tag{13}$$

$$\sum_{i\in N} z_{ij,f} - \sum_{i\in N} z_{ji,f} = d_{jf} \qquad \forall j \in C, \forall f \in F \tag{14}$$

$$\sum_h l_h^p y_{ij,h} + \sum_f l_f^d z_{ij,f} \leq Q x_{ij} \qquad \forall (i,j) \in A \tag{15}$$

$$\sum_h w_h^p y_{ij,h} + \sum_f w_f^d z_{ij,f} \leq W x_{ij} \qquad \forall (i,j) \in A \tag{16}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{17}$$

$$x'_{ij} \in \{0,1\} \qquad \forall (i,j) \in A_C \tag{18}$$

$$y_{ij,h}, z_{ij,f} \in \mathbb{N} \qquad \forall (i,j) \in A, \forall f \in F, \forall h \in H \tag{19}$$

$$t_i \in \mathbb{R} \qquad \forall i \in C \tag{20}$$

$$u \in \mathbb{N} \tag{21}$$

The objective function (1) is the total traveling time of all vehicles combined, added with the total penalty costs $u\tau$ for all vehicles exceeding the maximum number of vehicles. Constraints (2) make sure

every customer is visited exactly once, whereas constraints (3) make sure the incoming vehicle will also leave the customer. Constraints (4) and (5) prevent variables x'_{ij} from being artificially set to one. They are needed to make sure constraint (6) gives the actual number of used vehicles on the left-hand side. Constraints (7)-(12) ensure that the time variables are set correctly. More specifically, constraints (7) and (8) make sure the service times at successive customers are correctly specified and no waiting between customers is implemented. Constraints (9) establish the service times of successive customers if the depot is visited in between, whilst allowing for waiting time at the depot. Constraints (10) are implemented to make sure the complete service process of a customer is performed within its time window. Constraints (11) and (12) make sure the beginning and ending of the routes happens within the full time horizon. Constraints (13) and (14) make sure the demands of the customers are satisfied, whereas constraints (15) and (16) take the capacity constraints into account. The domains of all variables are specified by constraints (17)-(21).

4 Methodology

In this section, the solution method to solve the instances of the VRPSPDTW is described. The solution method will be a heuristic procedure based on the ALNS-PR procedure in Hof and Schneider [2019]. The search procedure consists of four main operations. The objective is to minimize the total distance driven. Thereby, for every vehicle exceeding the number of allowed vehicles, a penalty cost of τ is added to the objective value. To have a clear view on the heuristic, some definitions are made. A *subroute* is a set of customers visited by a vehicle in a particular order, where the vehicle departs and ends at the depot without visiting the depot in between. A *route* consists of all subroutes a vehicles performs in the complete schedule. If a route consists of multiple subroutes, the depot is visited in between.

First, an initial solution is constructed. In this paper, a push forward insertion heuristic will be used, which aims at inserting a customer on the *cheapest* place in the existing routes respecting all constraints (Section 4.1). Secondly, local search is performed each time a new solution is found. The local search consists of searching several small neighborhoods using variable neighborhood descent (VND) in order to find improvements in little time (Section 4.2). In the main phase of the algorithm, two kind of iterations can be done, an adaptive large neighborhood search (ALNS) iteration or a path relinking (PR) iteration. The idea of combining the two operators is based on Hof and Schneider [2019]. A PR iteration needs a so called elite set ($\mathbb{S}^e$), which consists of solutions with relatively good characteristics. This set is initialised empty and can have a maximum size of λ solutions. When the elite set is not completely filled or the current solution has

been visited before, an ALNS iteration is performed. To diversify the search, each ALNS iteration destroys a part of the solution using one out of a set of destroy methods. To rebuild the destroyed part, it uses one out of a set of repair methods (Section 4.3). After several iterations are performed, the elite set of solutions is compiled, consisting of the best solutions which have sufficient deviation in characteristics between them (Section 4.5.1). Once the elite set is filled, Path Relinking (PR) will be applied to solutions which have not been found before. Its main goal is to intensify the search. The PR procedure is extensively explained in Section 4.4.

After each iteration, the Simulated Annealing (SA) based acceptance decision (Section 4.5) decides how to update the new solutions. After ξ iterations without finding a new best solution, the search is truncated and the final all time best solution is returned. Thereby, after ξ^{reset} iterations without finding a new best solution, the current solution is reset to the best solution found so far. The pseudocode of the complete heuristic can be found in Algorithm 1. Each specific part of the algorithm will be explained in the following sections.

Algorithm 1 ALNS-PR algorithm

$S \leftarrow$ *ConstructInitialSolution*()
$S \leftarrow VND(S)$
Initialize best solution $S^* \leftarrow S$
while ξ iterations no new best solution found **do**
    if S already visited $\vee$ elite set not completely filled **then**
        $\hat{S} \leftarrow ALNS(S)$
        $\hat{S} \leftarrow VND(\hat{S})$
    else
        $\hat{S} \leftarrow PR(S, \mathbb{S}^e)$
        $\hat{S} \leftarrow VND(\hat{S})$
    if acceptdecision($\hat{S}, S$) **then**
        $S \leftarrow \hat{S}$
        if $\hat{S}$ is the new best **then**
            $S^* \leftarrow \hat{S}$
    UpdateSolutions(S)
    UpdateParameters()
return S^*

To be able to obtain a benchmark on the heuristic, the GUROBI solver will be used to compute bounds on the solution value of the MIP formulation. This can be done by solving the LP-relaxations and using branch and bound methods. With the lower and upper bound we are able to compare the results obtained by the heuristic.

4.1 Solution initialization

The first step of the algorithm is to build an initial solution. A push forward insertion heuristic (PFIH) will be used to construct the solution. The heuristic is inspired by the insertion-based procedure described by Chen and Wu [2006]. The problem of Chen and Wu [2006] differs from the problem in this paper. Among others, time windows and the possibility to visit the depot during a route require the elaboration of a new PFIH. In the first step of the heuristic, a seed customer is chosen. Since the PFIH cannot push back the departure times of customers or the depot in a subroute, the beginning of the time window of the selected seed customer will always be as early as possible. Otherwise, if a customer which has a relatively late beginning of its time window is selected as seed customer, the vehicle will wait at the depot before starting its trip to the seed customer. Since the PFIH fixes the depature time of the depot after initialization of a subroute, this might lead to issues with the placement of customers with early time windows. For example, if a vehicle departs at the depot at $t = 14400$ and waits at the depot until then, a customer of which the time window ends before $t = 14400$ can never be placed in that subroute anymore. Due to the push forward technique, the arrival time of the customer under consideration will be the depot departure time plus the travel time to this customer, which results in infeasibility. Thus, selecting a seed customer with the earliest time window ensures the placement of other customers with early time windows. Among the customers with the earliest starting time windows, the furthest customer from the depot will be selected. The first initial subroute will only visit the seed customer.

Among the unrouted customers, the cost of inserting the customer into all feasible positions in the current subroute is calculated. The arrival time of the inserted customer will be the departure time of the preceding customer plus the travel time from the preceding customer to the inserted customer. All customers which are placed after the inserted customer in the current subroute will be pushed forward in time with the extra travelling and service time which takes place before them. If a position does not respect a time window or capacity constraint of any customer in the subroute, it is not considered. The cost of inserting customer i into

a specific position between j and k is given in formula (22). j and k can be all combinations of consecutive locations in the current subroute, therefore j or k can also be the depot. The cost comes down to the extra distance traveled in the subroute minus the cost of making a single retour trip to customer i.

$$cost(i,(j,k)) = T_{ji} + T_{ik} - T_{jk} - T_{0i} - T_{i0}. \tag{22}$$

Afterwards, the insertion move with the lowest costs will be performed in the subroute, if the cost is below 0. All customers which come after the new customer will be pushed forward in time. This process will be repeated until it is not possible to insert a customer in the current subroute anymore. Then, the possibility of returning to the depot and beginning a new subroute within the same route is considered. In this case, it is allowed for a vehicle to wait at the depot. Among all customers for which this is possible, the new seed customer is selected the same way as mentioned above. In this new subroute, the same process is followed. If no new subroute can be formed, a new seed customer will be selected to start a complete new route in the same way as the first route. This process is repeated until all customers are routed. During the heuristic, the arrival times and weight and size capacity are tracked and updated after each insertion. These parameters are defined at each customer in the route. The waiting time of a subroute is defined by the time a vehicle waits at the depot before departing for the subroute. The pseudocode of the heuristic can be found in Algorithm 2. To save computational time, the costs of a certain move are calculated before the computationally harder feasibility check. If the costs do not improve on the best move, the feasibility check is not necessary anymore. After the solution initialisation, exceeding the maximum number of vehicles is punished by penalty τ per vehicle exceeding.

Algorithm 2 Push Forward Insertion Heuristic

$S \leftarrow \emptyset$
$SeedCustomer \leftarrow DetermineSeedCustomer()$
$S \leftarrow StartNewRoute(SeedCustomer)$
$UpdateParameters()$
$RemainingCustomers \leftarrow C \setminus \{SeedCustomer\}$
while $RemainingCustomers \neq \emptyset$ **do**
    $BestInsertion \leftarrow None$
    for i in $RemainingCustomers$ and j in insertion places in current subroute **do**
        if $Insertion(i, j)$ improves on $BestInsertion$ **then**
            if $Insertion(i, j)$ feasible **then**
                $BestInsertion \leftarrow Insertion(i, j)$
    if $BestInsertion \neq None$ **then**
        $S \leftarrow PerformInsertion(BestInsertion)$
    else
        $BestNewSubrouteSeedCustomer \leftarrow None$
        for i in $RemainingCustomers$ if i is feasible after visiting depot **do**
            if i qualifies above $BestNewSubrouteSeedCustomer$ **then**
                $BestNewSubrouteSeedCustomer \leftarrow i$
        if $BestNewSubrouteSeedCustomer \neq None$ **then**
            {Extend current route with a new subroute}
            $S \leftarrow StartNewSubroute(BestNewSubrouteSeedCustomer)$
        else
            {Start whole new route with a new subroute}
            $SeedCustomer \leftarrow DetermineSeedCustomer()$
            $S \leftarrow StartNewRoute(SeedCustomer)$
    $UpdateParameters()$
    $RemainingCustomers = RemainingCustomers \setminus \{InsertedCustomer\}$
Return: S

4.2 Variable Neighborhood Descent

Each time a new solution is found, local search will be performed. As local search element we will use Variable Neighborhood Descent (VND). The VND searches six neighborhoods in a particular order. The first neighborhoods are smaller and therefore quicker to search. In each neighborhood, the move with the best improvement is searched. If this yields an improvement to the current solution, this move will be implemented and the search will return to the first neighborhood. If there is no improvement in a neighborhood, the search will continue in the next neighborhood. The later neighborhoods will be larger and computationally harder. These are used to escape local optima from the smaller neighborhoods. Since they are placed at the end, they will be searched less frequently than the smaller ones. This way, the VND can find fast improvements without getting stuck in local optima too often. The pseudocode of the VND can be found in Algorithm 3.

Algorithm 3 Variable Neighborhood Descent

$S \leftarrow$ input
$k \leftarrow 1$
while $k \leq |K|$ **do**
    {Find the best move in the k^{th} neighborhood}
    $S' \leftarrow N_k^*(S)$
    if $z(S') < z(S)$ **then**
        $S \leftarrow S'$
        $k \leftarrow 1$
    else
        $k \leftarrow k + 1$
Return: S

The first neighborhood is the Or-Opt neighborhood introduced by Or [1976]. In this neighborhood, a single customer is relocated in its own subroute. The second neighborhood is the Or-Opt*. In this routine a customer is removed from its own subroute and inserted into a location in another subroute. As third neighborhood, the 2-Opt neighborhood is used. This operator removes two arcs from a single subroute, say (a,b) and (c,d) and replaces them by two new ones, (a,c) and (b,d), whilst reversing the order of customers between the customers b and c. Thereafter comes the Swap neighborhood. This neighborhood involves swapping the positions of two customers from the same subroute or two different subroutes. After searching

the Swap neighborhood, the 2-Opt^* neighborhood will be searched. This is an Inter-subroute variant of 2-Opt. It removes one arc from two distinctive subroutes and connects the first part to the second part of the other subroute. The final neighborhood is the exchange neighborhood. This neighboorhood is an extension to Swap. Two customers are simultaneously inserted into each others' subroute, but are not required to be placed into the exact location of the other customer. The two selected customers can be from the same or different subroutes. Table 1 shows characteristics of all neighborhoods.

Table 1: Characteristics of all Neighborhoods contained in the VND

Priority	Neighborhood	Move	Complexity
1	Or-Opt	Relocate a customer within a certain subroute	$O(n^2)$
2	Or-Opt*	Relocate a customer from one subroute to some place in another subroute	$O(n^2)$
3	2-Opt	Destroys arcs (a,b) and (c,d) (of the same subroute) and forms arcs (a,c) and (b,d)	$O(n^2)$
4	Swap	Swaps the position of two customers	$O(n^2)$
5	2-Opt*	Splits two subroutes into two parts and reconnects the non-corresponding parts	$O(n^2)$
6	Exchange	Swaps customers of two different subroutes and places them in the best place of their new subroute	$O(n^4)$

4.3 Adaptive Large Neighborhood Search

Once the current solution has been found before or the elite set is not yet filled, an ALNS iteration is performed. This consists of destroying and rebuilding a part of the solution. Destroying and rebuilding the solution is done with a chosen destroy and repair method. After each iteration, the used methods are evaluated, and are given scores based on their performance. A certain operator is randomly selected based on these scores. This way, the best methods will be used more frequently. The scores are based on the solution quality arising from an ALNS iteration. Inspiration of the used methods will be based on, among others, Hof and Schneider [2019] and adapted to the specific case of our problem. Seven destroy methods and three repair methods can be used in each iteration.

The first step of an ALNS iteration is selecting a removal and a repair operator to use. The probability of selecting operator i is given as $\frac{\pi_i}{\sum_j \pi_j}$, where π_i denotes the current weight of operator i. The scores of all destroy methods are given with vector π^-. Afterwards, the number of customers to be removed is determined. This number is given by n^-, and is selected from a given set N^-. The maximum value in this set equals $\zeta|C|$. The complete set is made up of ι values determined in equal steps upon the maximum

value: $N^- = \{\lfloor \frac{1}{t}\zeta|C|\rfloor, \lfloor \frac{2}{t}\zeta|C|\rfloor, ..., \lfloor \zeta|C|\rfloor\}$. The different values in N^- will also be evaluated into scores given as π^N. Once the customers are removed, a repair operator is randomly selected based on the scores of the repair operators, π^+. The pseudocode of one ALNS iteration can be found in Algorithm 4.

Algorithm 4 Adaptive Large Neighborhood Search

1: $S' \leftarrow S$
2: $n^- \leftarrow selectNumberOfCustomersToBeRemoved(\pi^N)$
3: $destroyMethod \leftarrow selectDestroyMethod(\pi^-)$
4: $repairMethod \leftarrow selectRepairMethod(\pi^+)$
5: $S' \leftarrow removeCustomers(destroyMethod, S', n^-)$
6: $S' \leftarrow repairCustomers(repairMethod, S')$
7: Return: S'

4.3.1 Removal and repair operators

To remove n^- customers from the routes, the following destroy operators are used. All methods only allow customers to be removed if the remaining routes are feasible with respect to time windows, weights and size constraints. If removing a customer results in dissolving a subroute, the waiting times of the consecutive subroute are updated. If a complete route dissolves and the maximum number of vehicles used was exceeded, the penalty costs are reduced.

Costs removal removes the customer which has the highest costs of serving it. This customer is determined by the savings in objective value if it is removed. The savings of removing customer i can be calculated as follows: $s = T_{i-1,i} + T_{i,i+1} - T_{i-1,i+1}$. Note that $i-1$ and $i+1$ are the customers before and after customer i in the current subroute. This operator aims to remove customers which appear to be placed undesirable. It continues to remove the customers with the highest savings until n^- customers are removed.

Random removal randomly removes customers from the routes until n^- customers are removed.

Route removal removes the smallest complete route with all its subroutes. The smallest route is determined as the route which serves the lowest number of customers. By doing so, the aim is to dissolve small subroutes and distribute the removed customers in existing routes. This way the number of vehicles used can be reduced.

Cluster removal selects a random seed customer and compiles a sorted list of the closest customers to it. This list is based on the travel distance $T_{seedcustomer,i}$ for each customer i. Afterwards, it continues to remove the first feasible customer of the list until n^- customers are removed. This operator aims at removing

customers close to each other.

Relatedness removal aims at removing customers which have similar characteristics and therefore are interchangeable. This operator also selects a seed customer at random. Afterwards, a sorted list of customers based on their relatedness to the seed customer is compiled. The relatedness scores are based on the absolute difference in distance, demands and time windows between the two customers. Each characteristic is normalized by dividing by the extreme values of that characteristic in the whole data set. For distance, the score becomes: $r^T_{i,j} = \frac{T_{i,j}}{\max_{k \in C}(T_{i,k})}$. Since the weight and sizes of the pick up demand are closely related, the relatedness score for pickup becomes the average of both: $r^p_{i,j} = \frac{1}{2}\frac{|p^w_i - p^w_j|}{\max_{k\in C} p^w_k - \min_{k\in C} p^w_k} + \frac{1}{2}\frac{|p^s_i - p^s_j|}{\max_{k\in C} p^s_k - \min_{k\in C} p^s_k}$, where p^w_i (p^s_i) denotes the total weight (size) of the pick up demand of customer i. The same holds for the delivery demand: $r^d_{i,j} = \frac{1}{2}\frac{|d^w_i - d^w_j|}{\max_{k\in C} d^w_k - \min_{k\in C} d^w_k} + \frac{1}{2}\frac{|d^s_i - d^s_j|}{\max_{k\in C} d^s_k - \min_{k\in C} d^s_k}$, where d^w_i (d^s_i) denotes the total weight (size) of the delivery demand of customer i. Note that if a customer only has a pick up (delivery) demand, its delivery (pick up) demand equals zero. As to time windows, also the average between the difference of the beginning and ending of the time window is considered: $r^{tw}_{i,j} = \frac{1}{2}\frac{|a_i - a_j|}{\max_{k\in C} a_k - \min_{k\in C} a_k} + \frac{1}{2}\frac{|b_i - b_j|}{\max_{k\in C} b_k - \min_{k\in C} b_k}$. Then, the complete relatedness score between the seed customer i and customer j is given as the sum of relatedness scores:

$$r_{i,j} = r^T_{i,j} + r^p_{i,j} + r^d_{i,j} + r^{tw}_{i,j} \tag{23}$$

After compiling the relatedness list, the same procedure of removing n^- based on a list as in the Cluster removal is applied, starting with the most related customer.

Departure time removal also aims at removing comparable customers. The measure used in this operator is the departure time (similar to the service time t_i) of leaving a customer in the current solution. In this case, after randomly drawing the seed customer, the list is compiled based on the absolute difference in current departure times with the seed customer, with the narrowest departure times differences having preference. The underlying idea behind this, is that when customers which have the same departure times are being removed in the same iteration, interchangeable insertion positions are created. The same list removal procedure as described above is used.

Time window removal removes customers which are closest to violating a time window constraint. This operator aims at creating more insertion possibilities by removing customers which likely would be the reason for infeasibility. A sorted list is created of customers who are closest to violating a time window constraint, after which the same list removal procedure as described above is used.

To reinsert the removed customers, the following repair methods are used. Each methods priority is to insert a customer into existing subroutes using the *push forward* technique as described in the constructive

heuristic. Once no customers can be placed into existing subroutes using this technique, inserting the customer as first customer in a subroute is tried. This is done using *pushing backward* of the depot in time. This technique allows shorthening the waiting time before subroutes. If this is not possible, creating a new subroute in an existing route is tried. If that is also not possible, a whole new route is created with the furthest customer among the customers with the earliest beginning time windows.

Greedy repair is very similar to the constructive heuristic. It inserts some customer into the best place possible. This is determined by the increase in objective value of inserting a customer at the certain place.

GRASP repair is a more diversified variant of the greedy repair operator. It composes a sorted list of best insertion moves and chooses randomly between those. The list is of size $x^{grasp}|C|$. Once a move has been randomly selected, only that move is executed. After each insertion, a new list is composed to ensure feasibility of the moves.

Random repair inserts a random customer at a random feasible place.

4.3.2 Adaptive evaluation of operators

The weight update of each adaptive element is based on scores obtained throughout the search. However, during the first ψ iterations of the algorithm no scores are given, to obtain a reasonable starting point to start judging operators. After these iterations, every γ iterations a weight update is performed. This update is based on the scores each operator obtained in the last γ iterations.

After each ALNS iteration, local search is performed with the VND algorithm (Section 4.2). Afterwards, the solution quality is judged. Whether or not the new solution is accepted is based on Simulated Annealing (Section 4.5). The new solution can be of four different kinds, it can be: the best one yet, better than the current solution, worse than the current solution but accepted and worse than the current solution and not accepted. Only in the latter case the solution is not accepted. If a solution is not accepted, no scores will be awarded to adaptive elements. For the other cases, scores of respectively σ^{best}, σ^{imp} and σ^{acc} are given to each of the used elements, if the new solution is never found before. This score is added to the total score ϕ_i of element i. Further, β_i keeps track of the number of times element i is used. After γ iterations, the weight of each element is updated as follows,

$$\pi_i = \pi_i(1-\alpha) + \alpha\frac{\phi_i}{\beta_i} \tag{24}$$

where α is a parameter to control the sensitivity of old and new scores. After the weights are updated, β_i and ϕ_i are set to zero for all adaptive elements.

4.4 Path Relinking

Once the local search returns a solution which had not been found before and the elite set is filled, PR is performed. The PR step juxtaposes the current solution to an elite solution. The current solution is transformed into the guiding solution by stepwise incorporating characteristics, in the form of arcs, of the elite solution. With PR, the hope is to obtain improved solutions by inheriting characteristics from the elite solutions, which cannot be found in the local search. Hof and Schneider [2019] showed this procedure can speed up the search significantly. The deviation between solutions will be measured using the number of arcs two solutions have in common.

In a complete PR iteration, Path Relinking is used between each solution in the elite set and the current solution. The maximum size of the elite set is given as λ. The procedure of PR between an elite solution and the current solution is as follows. The first step is to determine which arcs from the elite solution are contained in the current solution and which are not. Since the goal is to transform the current solution towards the elite solution, all mutual arcs will be fixed. I.e. if arc (i, j) is used in both the elite and current solution, it is not allowed to remove arc (i, j) in the transformation of the current solution. By doing this, *fixed* sequences of customers can be identified. The sequence ξ_i^+ is defined as the sequence of fixed arcs after customer i and the sequence ξ_i^- is defined as the sequence of fixed arcs before customer i.

The next step is to insert arcs from the elite solution into the current solution. The set of arcs which are contained in the elite solution, but are not in the current solution is defined as $A^{\neq}$. If an arc $(i, j) \in A^{\neq}$ is created by rearranging the customers, the fixed sequences ξ_i^- and ξ_j^+ have to stay intact. Thereby, the rest of the routes stay the same. Then, to create an arc (i, j), five different situations can be distinguished. The characteristics of each situation can be found in Table 2.

Situation 1 occurs when both ξ_i^- and ξ_j^+ do not contain the depot. In this situation arc (i, j) can be created by either inserting ξ_i^- in the subroute of customer j, before ξ_j^+, or the other way around. In this case both options are considered. The subroute from where the fixed sequence is removed remains the same, except for the removed customers. If either of the two fixed sequences contains the depot and the other does not, *situation 2* takes place. In this situation it is only possible to move the sequence which is not connected to the depot, since moving the sequence with the depot would result in a subroute which does not start or end at the depot. Another option is that ξ_i^- starts at the depot and ξ_j^+ ends at the depot. In

Table 2: Different situation to create an arc in the PR iteration

Situation	Occurance	Move
1	Both ξ_i^- and ξ_j^+ do not contain the depot	Relocate sequence ξ_i^- before j or relocate sequence ξ_j^- after i
2	Either ξ_i^- or ξ_j^+ contains the depot and the other does not	Relocate the sequence that does not contain the depot to the one that does
3	Both ξ_i^- and ξ_j^+ contain the depot and i and j are in the same subroute	Remove all customers in between the sequences and use the Greedy repair operator
4	Both ξ_i^- and ξ_j^+ contain the depot and i and j are not in the same subroute	Use the $2-opt^*$ operator to create a new subroute consisting of ξ_i^- and ξ_j^+
5	Either i or j is the depot	Try relocating the sequence which is not the depot to the beginning/ending of each subroute

this case two situations can happen. If both i and j are in the same subroute, the only way to create the arc (i, j) is to remove all customers in between. Once they are all removed, they are reinserted with the *Greedy repair* operator from the ALNS (Section 4.3) whilst respecting fixed sequences. This is referred to as *situation 3*. However, if i and j are in different subroutes, the arc is created with the $2-opt^*$ operator. I.e. the sequences ξ_i^- and ξ_i^+ are placed after each other making a new subroute, while the remaining parts of their old subroutes also form a subroute. This case is defined as *situation 4*. This situation brings two new subroutes as replacement for two old subroutes. The old subroutes had distinctive places in their routes. Since the new two subroutes can be placed in either places, both options are considered. Finally, *situation 5* occurs when either i or j is the depot and the other fixed sequence does not contain the depot. It this case the arc can be formed in multiple ways. Namely if i is the depot, the ξ_i^+ can be placed at the beginning of all subroutes, vice versa if j is the depot. Therefore, in this situation, all options are considered.

Which arc to insert from the set $A^{\neq}$ into the current solution is determined by the lowest costs. The costs of forming each arc in $A^{\neq}$ will be calculated, where each arc can be created by one of the above mentioned situations. The arc with the lowest costs will then be created in the current solution, with the requirement of the solution to stay feasible. After the current solution has been modified, the complete same process is repeated. Each newly created arc is fixed if it is contained in the elite solution. This goes on until a fraction ρ of the initial $A^{\neq}$ is formed, or no feasible arc can be created. Afterwards, the transformed current solution will be returned.

Path Relinking is performed between the initial current solution and each of the solutions in the elite set. After obtaining all transformed initial solutions, the best of the resulting solutions is returned as output from

the complete PR iteration. A pseudocode of a complete PR iteration can be found in Algorithm 5.

Algorithm 5 Path Relinking

$S \leftarrow currentS$
$S' \leftarrow \emptyset$
for $S_e \in \mathbb{S}^e$ **do**
    $S_c \leftarrow S$
    $A^{\neq} \leftarrow FindDifferingArcs(S_c, S_e)$
    $I^A \leftarrow |A^{\neq}|$
    while $x_{best} \neq \emptyset \wedge A^{\neq} \geq (1-\rho)I^A$ **do**
        $x_{best} \leftarrow \emptyset$
        for $(i,j) \in A^{\neq}$ **do** $x \leftarrow FindBestMoveToCreate((i,j))$
            if x better than $x_{best} \vee x_{best} = \emptyset$ **then**
                $x_{best} \leftarrow x$
        if $x_{best} \neq \emptyset$ **then**
            $S_c \leftarrow CreateBestMove(x_{best})$
            $A^{\neq} \leftarrow FindDifferingArcs(S_c, S_e)$
    if $S' = \emptyset \vee S_c$ improves on S' **then**
        $S' \leftarrow S_c$
Return: S'

4.5 Solution Updating

After an iteration of the ALNS-PR algorithm, the newly found solution, S', is evaluated. Whether it is accepted as new current solution is based on Simulated Annealing (SA). If S' has a better objective value than the old current solution (S), it will always be accepted as new solution. However, if it has a worse objective value, it still has a chance of being accepted. This is to escape local optima. The chance of a worse solution being accepted is equal to $e^{-(f(S')-f(S))/T}$, where $f(x)$ denotes the objective value of solution x and T is a temperature parameter determining the sensitivity of the probability of accepting the solution. At the beginning of the search, T is initialized as T^{init}. T^{init} is the temperature for which a $\delta\%$ deterioration to the initial constructed solution has 50% chance of being accepted. Thus, T^{init} can be calculated with the following formula.

$$T^{init} = \frac{S^{init} - S^{init}(1+\delta)}{\ln(0.5)} \tag{25}$$

To converge to the best solutions, T is decreased every iteration. The decrease is gradually done in n^{SA}

steps towards zero. For the purpose of diversification, T is set back to T^{init} every time a new best solution is found. During the complete algorithm, the best solution found will be saved and updated every time a better one is found.

4.5.1 Admission to Elite Set

To keep a well performing Path Relinking component, it is necessary to update the elite set of solutions. The newly accepted solution can be admitted to the elite set in three ways. Firstly, if the elite set is not completely filled yet, the solution will always be accepted. Secondly, if it is the best found so far, it is always accepted and it replaces the worst solution in the set. This is needed to guarantee the quality of the solutions in the elite set. However, it is important to have a diversified elite set of solutions. If several best solutions are found consecutively, they presumably have a lot in common. This would result in an elite set with solutions which all have a relatively good objective value, but do not differ enough from each other. Then, transforming an initial solution with all solutions from the elite set would most likely result in comparable outcomes. Therefore, if a solution is not the best so far, it still has a possibility to be admitted.

This is done in a convenient way to increase the diversity between the elite solution whilst not giving in too much on the objective values. The candidate solution is first compared to the worst solution, in terms of objective value, from the elite set. If the candidate solution is accepted with the SA acceptance decision, with temperature T^{init}, and it does not decrease the average diversity between the solutions in the elite set, it is admitted and replaces the worst solution. This diversity is measured in number of non-overlapping arcs. If the candidate solution is not accepted at the cost of the worst solution, the process is repeated with the second worst solution of the elite set. This procedure is continued upon the second best solution of the elite set, so that the best solution is never removed. An admission of this kind will never decrease the diversity of the set, and because of the SA acceptance decision will never give in too much on the objective value. Therefore, updating the elite set with this method will result in an elite set which balances the diversity and quality of its solutions.

5 Data

Since the problem has a practical nature, real world data will be used. The company GreenRoutes provides route optimization for waste collection companies. GreenRoutes handles customers which face the problem as described in this paper. Therefore, they are able to provide a real world data set. A data set with 250

customers is provided. To investigate the properties of the algorithm for different instance sizes, the larger instance will be split in multiple smaller instances. Therefore, it is possible to obtain computational results for different instances and instance sizes. Three different sizes of instances will be investigated, the *small*, *medium* and *large* instances contain 45, 100 and 200 customers respectively. The system OSRM is used to obtain the distance and travel times between each location. OSRM (open-source routing machine) is a system which returns the shortest path in the road network for any two coordinates. This system solves the shortest path problem for each arc (i, j) between customers i and j.

5.1 Data description

Each instance will contain one depot and multiple customers of which the locations are specified in geographic coordinates. In total there are four kinds of demand, two types of pick up demand and two types of delivery demand. There are two types of containers, namely the $2.5m^3$ container and the *big-bag*. Each customer can have a pick up or delivery demand of each of the two container types. Since every single demand is denoted as a separate customer, the customers with the same coordinates are merged. By doing so, customers can have all sorts of demands with mixed container types. If a container needs to be delivered, it is considered empty. Empty containers can be stacked, which saves up space. If a container is picked up it is considered full with debris, a full container can not be stacked anymore. A big-bag can only be stacked if it is placed on top of one empty container, otherwise it can never be stacked. This leads to container specific sizes and weights. Each vehicle has six places for $2.5m^3$ containers, therefore the size of a full $2.5m^3$ container is considered 1. Since three $2.5m^3$ empty containers can be stacked up on top of eachother, the size of an empty $2.5m^3$ container is defined as $\frac{1}{3}$. The big-bags take up half of a $2.5m^3$ container space, and therefore have a size of $\frac{1}{2}$. The weights of a big-bag (full and empty), a full $2.5m^3$ container and an empty $2.5m^3$ container are 1500, 3000 and 300 kilograms respectively. Every vehicle has a total size capacity of 6 and a total weight capacity of 15000 kilograms. Note that for this data the size requirements can be incorporated using the capacity conception mentioned above. In general this might not be possible. This conception can be used for all size requirements for which the composition of the load does not influence the filled truck space, for example with tank wagons. However, if the filled truck space depends on the composition of the products in the load, the usability of this conception depends of the specific problem. For example, if a big-bag is not allowed to be placed on top of a empty container, a load composition of five full containers, one empty container and one big-bag does not meet the size requirement. However, it does not exceed the maximum size standard of 6, namely $5\frac{5}{6}$. For the problem in consideration, the usage of this conception

encompasses all requirements.

The drivers are not permitted to start earlier than at 7am. Eight hours later, at 3pm, all drivers must have finished at the depot. The time horizon will be defined in seconds, with $t = 0$ corresponding to 7am and $t = 28800$ corresponding to 3pm. Whenever a vehicle departs from the depot, a standard preparation time of 15 minutes is taken into account to load containers and prepare the vehicle. If a vehicle arrives at the depot after a trip, also 15 minutes are included to unload the vehicle. The loading and unloading of containers is a difficult operation to perform. At a customer, picking up one container of any kind takes 10 minutes, delivering one container takes the same amount of time. The service time is defined as the sum of the times of all containers operations which need to be executed. The servicing of a customer needs to be done completely within the time window of the customer. If a customer has no specified time window, the time window is given as $[0, 28800]$.

6 Computational Results

In this section the parameters of the algorithm and model modifications are tuned, thereby computational results to evaluate the performance of the algorithm are discussed. All methods are implemented in **Python** version 3.8. Furthermore, the MIP formulation uses the commercial solver **Gurobi** version 9.1.1. All experiments are run on an computer with a 2.7 GHz Intel Core i5 and 8 GB RAM. Since the ALNS and SA components of the algorithm contain multiple operators which use randomness, 50 runs are performed for all computational results obtained with the use of ALNS or SA. Note that when performing a feasibility check, for most moves only a fraction of a subroute in the entire solution changes, while the rest stays the same. Therefore, one only has to check the relevant changing values for a feasibility check to save running time. In Section 6.1 the parameters needed for the algorithm are tuned, in Section 6.2 several model modifications are considered and in Section 6.3 computational results of the algorithm are given and its performance is evaluated.

6.1 Parameter Tuning

The ALNS-PR algorithm includes a total of 16 parameters. During the development of the algorithm, several parameter values have been observed as well-performing. In addition, parameter values used in literature are used to set a base value for each parameter. By doing this, an initial parameter setting is formed as a starting

point for the parameter tuning. For each parameter the base value, as well as a value above and below, is considered. Thus, three values need to be evaluated for all parameters. In total this leads to 3^{16} possible parameter settings. Considering all possibilities will take too much computational time, therefore the tuning of different parameters is done in in sequence. However, the performance of some parameters have a lot of influence among each other. These parameters are tuned simultaneously. The sequence of parameters to be tuned is an important aspect of the parameter tuning. If one chooses to tune a certain parameter at the beginning and tunes another parameter which has influence on that parameter later on, it can be possible that the parameter is wrongly tuned. Therefore, the parameters which are presumed to have no influence on the tuning of the remaining parameters are tuned first. Firstly, the parameter of a specific removal operator, χ^{grasp}, is tuned. Secondly, the remaining ALNS parameters are tuned, followed by the PR parameters. Lastly, the penalty costs, acceptance desicion parameters and stopping criterion are tuned. The complete sequence of parameters can be found in Table 3.

Table 3: Sequence of parameters in the parameter tuning

Order	1	2	3	4	5	6	7	8	9	10
Parameter	χ^{grasp}	ζ, ι	ψ	$\sigma^{best}, \sigma^{imp}, \sigma^{acc}$	α, γ	ρ	λ	τ	δ	ξ, ξ^{reset}, n^{sa}

Once a parameter is tuned, its new value will be fixed for rest of the parameter tuning. The value is determined based on the average obtained objective value, the average running time and the average iterations performed per run. The latter two are related, however due to possible inefficiencies in the programming of either an ALNS or PR iteration, only looking at the running time could give a deceived view. In general, the objective value will be used to decide the value. However, if either the running time or the number of iterations has strongly increased while the objective values are similar, a faster converging algorithm is preferred. All parameter tuning is performed on the instance of 45 customers. Prior to the parameter tuning, the implementation of the Exchange neighborhood is examined. Since this neighborhood has $O(n^4)$ as complexity, its running time can be significantly larger than the other neighborhoods. Table 4 shows the performance of the ALNS-PR algorithm with and without the Exchange neighborhoods.

Although Exchange increases the performance in terms of objective value, the running time becomes unacceptably high. To keep reasonable running times for the larger instances, this neighborhood is further neglected. Thus, the parameter tuning is done with the first five neighborhoods in the local search. Table 5 shows the parameters with their considered values as well as their final value after parameter tuning.

Table 4: Performance measures of the ALNS-PR algorithm with a local search consisting of 5 or 6 neighborhoods using the base values for all parameters and an instance size of 45 customers

Method	UB	SD	Time	Iterations
ALNS-PR, VND without Exchange	28788.63	1.35%	20.36s	161.06
ALNS-PR, VND with Exchange	28630.67	0.41%	1011.03s	160.33

Table 5: Results of the parameter tuning of the ALNS-PR algorithm using an instance size of 45 customers, with the final value of each parameter written in bold

Component	Parameter	Low Value	Base Value	High Value
ALNS-PR	ξ	50	**75**	125
	ξ^{reset}	$\frac{1}{3}\xi$	$\frac{2}{3}\xi$	*None*
	τ	3600	14400	**28800**
ALNS	α	0.1	**0.2**	0.3
	σ^{best}	6	**9**	15
	σ^{imp}	3	**6**	9
	σ^{acc}	1	**3**	5
	γ	10	**20**	50
	χ^{grasp}	0.2	**0.3**	0.4
	ψ	5	20	**50**
	ζ	0.15	0.25	**0.35**
	ι	3	5	**7**
PR	ρ	**0.5**	0.7	0.9
	λ	5	**10**	15
SA	δ	0.0025	**0.01**	0.05
	n^{sa}	25	**100**	125

As for the parameters ι and ζ a trade-off between objective value and running time is made, where higher parameter values increase the running time, but improve the objective value. Since the objective value is of greater importance, the higher parameter values are chosen. Logically, increasing stopping criterion parameter ξ will result in larger running times, with presumably better objective values. Therefore, for step 10 in the parameter tuning sequence, the objective value is not decisive and a trade-off between the objective value and running time needs to be made. Note that in practical usage this trade-off decision can differ if one prefers a faster algorithm at the cost of worse objective value. The computational results of all steps in the parameter tuning process can be found in Appendix A.

6.2 Model Tuning

To test other possible model settings, modifications are implemented to the PR component and tested on the instance with 100 customers. From all transformations to elite solutions from the elite set, only the best transformation in terms of objective value undergoes local search. An option worth considering is to perform local search on all transformations to obtain more possible solutions, and selecting the best of those (referenced as *Method b*). This approach results in better objective values than the general algorithm, but it does take more time to reach those as can be seen in Table 6. Thus, considering all transformations while performing local search is useful for the intensification, but logically goes at the costs of performing more local searches which costs extra time. Another possible modification of the algorithm is to adjust the way the solution reset is done after the reset criterion is reached. Instead of returning to the best solution, one could destroy the whole solution and rebuild it using the GRASP repair operator (referenced as *Method c*). Since the elite set is already formed, this method could prove to be valuable by applying PR to solutions which have no relations to the previous solutions. In the same table we find that this modification does not result in better objective values, but decreases the running time. The latter can be explained by the inability to find new best values in the remaining iterations, because the newly build solution needs to improve relatively much from its input objective value to beat the best solution. The remaining iterations can be too few to accommodate this improvement.

Of crucial importance for having a well functioning PR component is the formation of the elite set. To examine the performance of the elite set in this paper, two other formation methods are used. First, in Hof and Schneider [2019] the elite set is formed by always admitting the best solution at the costs of the worst solution. Thereby other solutions have the chance to be admitted if the solution is SA accepted against the worst solution in the elite set and admitting this solution does not decrease the average diversity of the elite set (referenced as *Method d*). As can be seen in the table, this method fails to improve on the method used in this paper. The main difference is that in the method in this paper, new solutions have extra chances to be admitted to the elite set by replacing any of the solutions (except the best) with the criterion mentioned above, whereas in Hof and Schneider [2019] only replacing the worst is possible. This could lead to admitting several new best solutions in a row which have a lot in common which harms the diversity of the set. With the method in this paper this situation can be repaired if those solutions are replaced by new ones, while the undiversified sequence can remain intact for a long time if only the worst solution can be replaced. Thus a reasonable shortcoming of the considered formation method could be the lack of diversification between the

solutions in the elite set. Another adjustment to the formation method of the elite set could be to not let the best solution necessarily replace the worst solution in the elite set, but replace the solution which results in the highest average diversity increasement (referenced as *Method e)*. From the table we find that this method also performs worse. Removing the solutions which result in the higher diversification can be at the expense of the solution quality of the elite set. This in turn would result in worse PR transformations and a worse performing PR component. Altogether, the ALNS-PR algorithm as described in this paper outperforms all modifications when considering objective value, running time and number of iterations. Therefore, this algorithm is used to obtain further computational results.

Table 6: Performance measures of modifications of the PR component with the ALNS-PR algorithm and 100 customers. Method b performs VND on all PR transformations. Method c rebuilds the solution using GRASP after a reset. Method d forms the elite set with the method used in Hof and Schneider [2019]. Method e lets the best solution in the elite set replace the one which results in the most diversity in the set.

Method	UB	SD	Time	Iterations
ALNS-PR	44304.59	1.23%	155.69s	245.06
Method b	44042.36	1.34%	247.39s	213.08
Method c	44620.48	1.70%	141.94s	195.92
Method d	44431.3	1.25%	176.02s	275.1
Method e	44402.24	1.16%	161.87s	223.96

6.3 Algorithm Results

Once all parameters are set to their final value, the performance of the algorithm can be evaluated. The complete ALNS-PR algorithm is run on all three instances. The MIP formulation is run for a maximum of 30 minutes to obtain a lower and upper bound. The computational results of the constructive heuristic (PFIH), the constructive heuristic with local search (PFIH + VND), the ALNS-PR algorithm, the ALNS algorithm (ALNS-PR algorithm where the PR component is omitted) and the MIP are shown in Table 7. The bounds of the methods are visualized in Figure 2

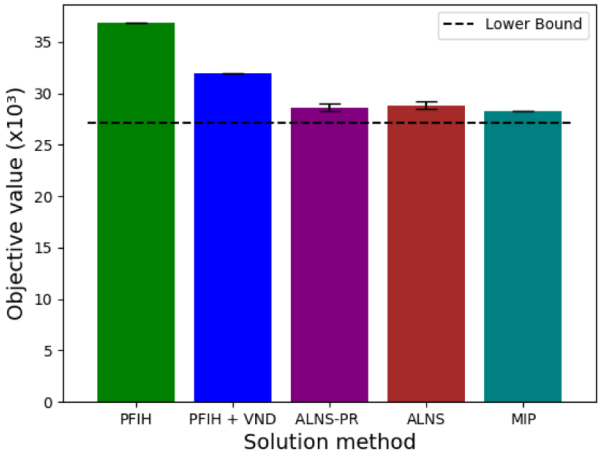
It can be seen that the PFIH used to construct an initial solution is able to obtain a first objective value in relatively little time for all instance sizes. But after performing local search on the initial solution, the objective value decreases notably. The quadratic size of the neighborhoods results in rapidly growing running times for one local search move if the instance size increases. Performing VND on the initial solution yields a shrinkage of 18.05%, 9.96% and 23.90% of the optimality gap, for the small, medium and large instance

Table 7: Performance measures of the PFIH, PFIH with VND, ALNS-PR, ALNS and MIP on all instance sizes

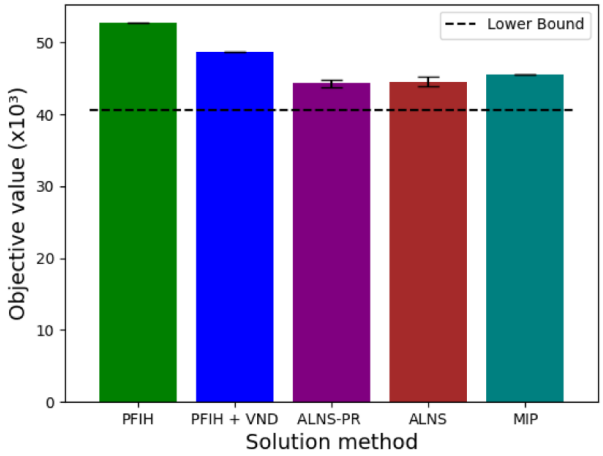
Method	Instance Size	UB	SD	LB	Optimality Gap	Time	Iterations
PFIH	45	36848.40	N/A	N/A	35.95%	0.07s	N/A
PFIH + VND		31956.62	N/A	N/A	17.90%	0.35s	N/A
ALNS-PR		28622.04	1.21%	N/A	5.60%	18.94s	154.25
ALNS		28851.46	1.31%	N/A	6.45%	20.88s	163.86
MIP		28272.78	N/A	27103.84	4.31%	1800s	N/A
PFIH	100	52706.90	N/A	N/A	29.96%	0.14s	N/A
PFIH + VND		48667.82	N/A	N/A	20.00%	0.62s	N/A
ALNS-PR		44304.59	1.23%	N/A	9.24%	155.69s	243.28
ALNS		44568.15	1.47%	N/A	9.89%	132.66s	226.12
MIP		45543.82	N/A	40557.1	12.30%	1800s	N/A
PFIH	200	114934.0	N/A	N/A	52.42%	0.31s	N/A
PFIH + VND		96913.20	N/A	N/A	28.52%	7.80s	N/A
ALNS-PR		84599.89	1.90%	N/A	12.19%	896.45s	317.64
ALNS		84971.69	1.77%	N/A	12.69%	951.71s	283.44
MIP		139319.6	N/A	75405.6	84.76%	1800s	N/A

respectively. The solution after PFIH and VND is called the input solution from now on. For all instance sizes the ALNS-PR algorithm improves on the input objective values. The running time of the ALNS-PR algorithm increases when the instance sizes increases. Thereby, the number of iterations until convergence also increases with the instance size. From the input solution, ALNS-PR is able to reduce the optimality gap with on average 12.30%, 10.76% and 16.33% for the instance sizes. Notable are also the standard deviations between the runs, they show that the solution is subject to the randomness of the components in the algorithm. The MIP reaches competitive solutions for the small and medium instance size. For the large instance size it is not able to reach a solution with good quality. The MIP is also able to retrieve a helpful lower bound on the objective values. The MIP uses *big M* constraints in the formulation. Therefore, the LP relaxations used to solve the MIP are unlikely to result in a feasible solution. Thus, it should be noted that the MIP is unlikely to close the complete gap within reasonable running time. With the obtained lower bounds, we observe that the ALNS-PR algorithm is able to close most of the optimality gap from the input solution.

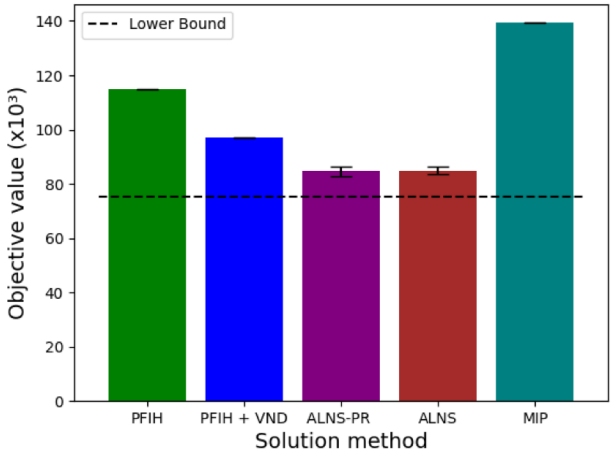
When compared to the ALNS algorithm, where the PR component is omitted, the ALNS-PR algorithm finds better objective values on average for all instances sizes. However, the ALNS is competitive with optimality gaps that are only 0.85%, 0.65% and 0.50% larger for the respective instance sizes. These gaps can be the result of the inability of ALNS operators and local search neighborhoods to reach improvement when stuck at a current best solution within 75 iterations. While using PR on the other hand, combining the characteristics of old solutions with these neighborhoods can prevent getting stuck at tough solutions and



(a) Instance size of 45 customers



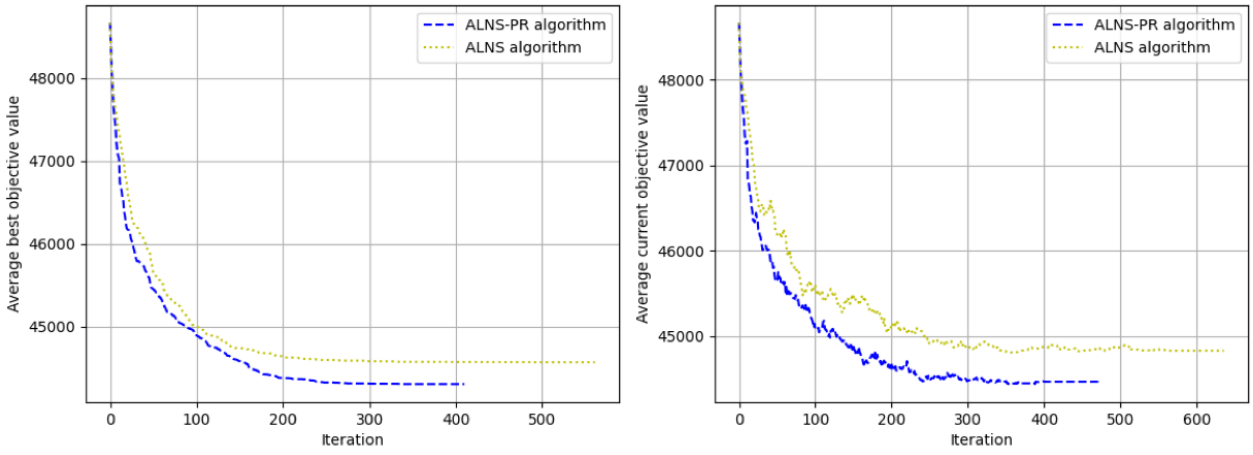
(b) Instance size of 100 customers



(c) Instance size of 200 customers

Figure 2: Visualization of the obtained upper and lower bounds of the PFIH, PFIH with VND, ALNS-PR, ALNS and MIP on all instance sizes. The methods are defined on the x-axis.The standard deviation for each objective value (upper bound) is given with error bars

reaching the stopping criterion. The standard deviations of the solutions is comparable, which is presumably due to the fact that randomness is mainly implemented in the ALNS component. The running times and iterations performed do not differ a lot between the ALNS and ALNS-PR algorithm. This leads us to examine the performance of the two algorithms during the execution. For the instance with 100 customers Figure 3a shows the average best objective value found during the search at each iteration for both algorithms, whereas Figure 3b shows the average current objective value at each iteration. The other instance sizes can be found in Appendix B.



(a) Average best solution value

(b) Average current solution value

Figure 3: Average best and current solution found per iteration with the ALNS and ALNS-PR algorithm and 100 customers

As can be seen in Figure 3, during the search the ALNS-PR algorithm is ahead of the ALNS algorithm in means of average best objective value found and current objective value. The ALNS-PR algorithm follows a steeper line, which means it needs on average less iterations to decrease the solution quality. Also, the ALNS-PR graph continues to go down later on in the process. This can be explained by the property of PR to update the elite set during the search. Thus, later in the process the elite set will have increased in quality, which in turn could result in findings better solutions later on.

To obtain insight in the performance of the PR and ALNS component within the ALNS-PR algorithm, the iterations of each component are examined. Table 8 shows the number of times each iteration resulted in an improvement of the current solution or a new all time best.

If the instances size grows, a larger proportion of all iterations performed is executed by the PR component. This is due to the ability of larger instances to have multiple solutions with good objective values. Thus, the probability of finding a solution, which has never been found before (and afterwards performing a PR iteration) and which has an objective value which will be accepted by the SA component, is larger for bigger instances. As for smaller instance sizes the number of undiscovered solutions with a good objective value is smaller. Therefore, the newly accepted solution is more likely to be found before, resulting in relatively more ALNS iterations. For all instance sizes, the ALNS component accounts for the most iterations

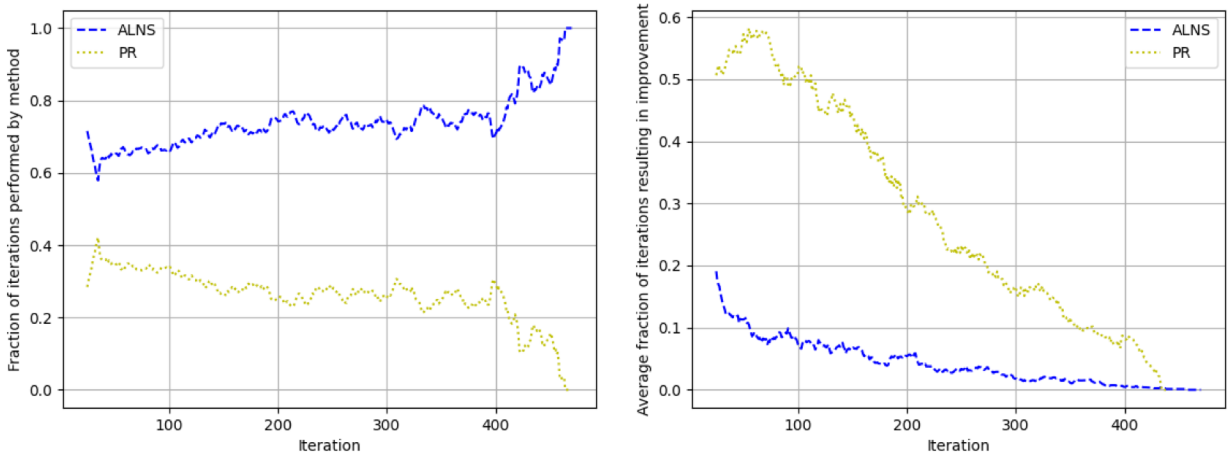
Table 8: Average number of iterations performed, number of improvements found and number of new best solutions found by the ALNS and PR component for all instance sizes

Component	Instance Size	Total Iterations	Improvement Iterations	New Best Iterations
ALNS	45	129.38	10.76	6.02
PR		24.86	11.98	2.22
ALNS	100	173.72	15.12	7.44
PR		69.56	37.14	7.86
ALNS	200	220.78	20.66	11.12
PR		96.86	46.70	13.02

performed in the complete algorithm. However, the number of times an iteration leads to an improvement of the current solution is higher for a PR iteration. On average, approximately half of the PR iterations results in an improvement to the current solution, whereas this is less than 10% for the ALNS iterations. The number of new best solutions found is also proportionally higher for the PR component, but this effect is less than for the improvement iterations. This can be the cause of the intensification aspect of the PR component when trying to escape a solution, i.e. if a newly found worse solution is accepted, the PR iteration might push the solution back to the previous one if that one was in the elite set. This results in an improvement iteration, but not in a new best solution. This property of the PR part could also have a negative impact on the diversification of the whole search.

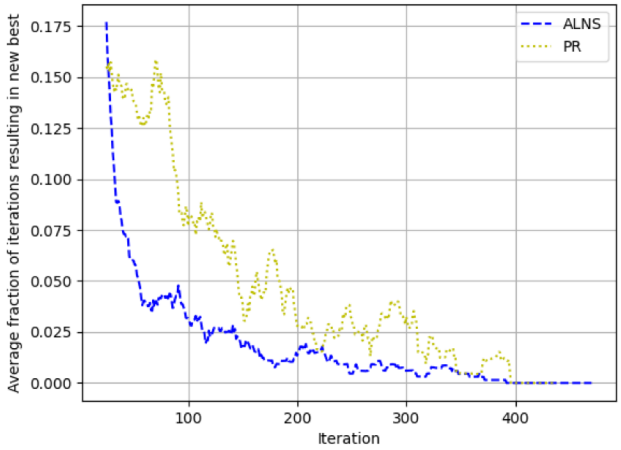
The performance of each component might fluctuate over time. To create insight in this aspect, average performance measures per iteration are plotted in Figure 4. To smoothen the curve all plots use a moving average over the last 25 iterations.

As more iterations are performed, the proportion of ALNS iterations gradually increases. This can be explained by the fact that after more iterations, more different solutions have been found. This will result in finding less new solutions, thus less PR iterations and more ALNS iterations. At the start of the algorithm, a PR iteration results more than half of the time in an improvement of the current solution. ALNS iterations results less frequently in improvements at every stage in the algorithm. As more iterations are performed, both methods find improvements less often. Naturally, it is harder to find an improvement when the current solution better, which mostly is the case after more iterations. However, the ALNS component moves faster to a fraction close to 0, while the PR component is longer able to maintain a higher fraction as iterations increase. This can be caused by the improvement of the elite set as more iterations have been performed. The same pattern can be seen for the fraction of iterations which result in new best solutions. In



(a) The average fractions of ALNS and PR iterations performed

(b) The average fraction of iterations resulting in an improvement



(c) The average fraction of iterations resulting in a new best

Figure 4: Average fraction of iterations performed, average fraction of iterations resulting in an improvement and and average fraction of iterations resulting in a new best solution by ALNS or PR for each stage in the ALNS-PR algorithm with 100 customers

this case, both ALNS and PR start off strong, and decrease over time. The strong start can be caused by the collaboration of the ALNS and PR component. In the first iterations, only ALNS is performed. In this stage only a few solutions have been found, thus the chance of finding a new best solution in an ALNS iteration is relatively large. By combining the first findings of the ALNS, PR is able to increase the current solution. As iterations increase, the same pattern as for the fraction of iterations which result in improvements can be observed. Figures of the total number of iterations performed, iterations resulting in improvements of the current solution and iterations resulting in new best solutions can be found in Appendix B.

Since the ALNS iterations must ensure the diversification of the algorithm, the performance of the removal operators is scrutinized. Figure 5 shows the average removal operator scores per iteration of the ALNS-PR algorithm and an instance size of 100 customers. The complete route removal operator performs well in the first iterations, but decreases afterwards. This can be because at the beginning of the search, unfilled and unfortunate routes consisting of remaining customers can still exist. Removing those routes and relocating their customers in other existing routes can be useful. Later on in the search, the other existing routes will be tighter optimized and therefore have less possibilities to include new customers. Whenever removed customers can not be placed in existing routes, new routes are created. This could turn out to have a bad effect on the objective value if the newly formed routes will be unfilled or unfortunate routes. Further, the bad performance of the highest costs and departure time removal operators stand out. Thus, the omission of either of these operators might increase the performance of the complete algorithm.

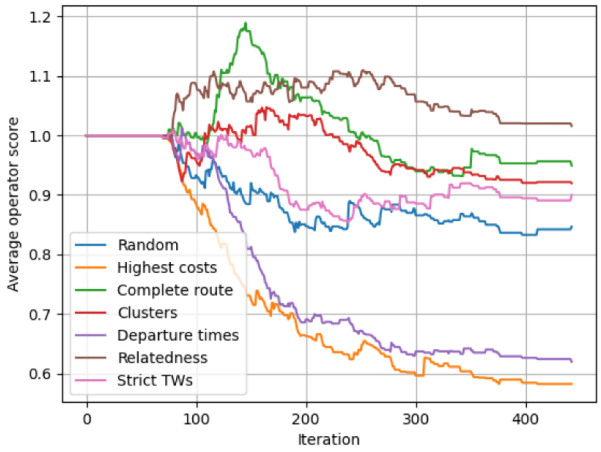


Figure 5: Average removal operator scores throughout the search with the ALNS-PR algorithm and 100 customers

Table 9 states the performances of omitting a certain ALNS segment compared to the complete algorithm. Omitting the highest costs removal operator does not give an improvement or deterioration for the objective value or running time. This operator can be seen as an operator which removes the customers which are the furthest from their predecessor and successor. Therefore, the customers which are naturally far away from all customers will always be removed. Thus, this operator may not succeed in selecting the customers which are wrongly placed. This can result in a bad operator performance. This could be improved by dividing the travel times in its measure by the average travel time to all other customer from the customer in consideration. This results in a more standardized measure, which can improve the operator performance.

Dropping the departure time removal operator on the other hand results in a worse upper bound as well as more running time. This leads us to believe that using this operator occasionally improves the algorithm.

Table 9: Performance measures of modifications of ALNS component with the ALNS-PR algorithm and 100 customers

Method	UB	SD	Time	Iterations
ALNS-PR	44304.59	1.23%	155.69s	245.06
Highest costs removal omitted	44322.97	1.38%	159.52s	241.92
Departure times removal omitted	44418.26	1.32%	175.69s	246.40

7 Conclusion

In this paper, the aim was to develop a well-performing algorithm for a practical waste container distribution problem and obtain insights in the performance of (specific parts of) the algorithm. An ALNS-PR algorithm is developed for the VRPSPDTW with the additional condition of allowing multiple trips per vehicle. The algorithm is able to obtain solutions with a relatively small optimality gap in reasonable time, with respect to the lower bound obtained by the MIP. The algorithm performs well for all instance sizes. Thereby, the running time increases if the instance size grows. Furthermore, we found insights in the specific components of the algorithm. The inclusion of the PR component on the ALNS algorithm slightly improves the performance considering the obtained objective value. Thereby, the ALNS-PR algorithm is on average ahead of the ALNS algorithm by means of best solutions found during the search. Furthermore, a PR iteration results more often in finding improvements in the current solution. We also found that the PR component is longer able to maintain a relatively high chance of finding new best and improved solutions when more iterations are performed, while the ALNS component reduces its performance earlier on in the search. This indicates the value of an updating elite set, which is diversified and contains good quality solution. The method used to construct the elite set also showed to be an improvement on the method from Hof and Schneider [2019]. All in all a well-performing algorithm is developed, which can find competitive solutions in reasonable time. Thereby, insights are gathered in the performance of specific components of the algorithm, which can be used for future research.

8 Discussion & Future Research

Altough we have found several useful results, the methods discussed contain some limitations. First of all, the parameter tuning is performed on the instance with 45 customers. As can be seen in later results, the

ALNS component has more influence for smaller instances. The PR component only gets to perform a few iterations in this case. Therefore, the parameter tuning can be skewed towards the influence of the ALNS component. I.e. if the instance grows, the PR component occurs more often. In this case it might be useful to choose parameter such that we have a stronger diversifying ALNS component to balance the increased intensification of more PR iterations. Thus, the parameter tuning could be performed by considering all instance sizes.

Furthermore, the influence of the local search could be further examined. All five neighborhoods in the VND are assumed to be helpful. However, it might be the case that several neighborhoods can not find improvements most of the time. Then, the omission of neighborhoods might decrease running time, whilst not giving in on the objective value. Thereby, if iterations can be performed faster, one can choose to let the algorithm run for more iterations which can even improve the objective value. The same improvement may be obtained by considering a different order of neighborhoods. One could also consider the omission of removal and repair ALNS operators. For now, only the two removal operators which were suspected to have a negative impact based on operator scores where considered. However, omission of other ALNS operator could result in the same improvement as discussed for the local search.

Finally, the performance of a PR iteration versus an ALNS iteration is based on the number of times an improvement or new best solution is found. On the other hand, the quality of the improvement is not considered. This could lead to a deceived view on the performance of the ALNS and PR component. Furthermore, the running time of each component is not investigated. This could be a useful feature for future improvement of the algorithm.

As for further improvement of the algorithm, further research of several aspects can be considered. Since the PR component is observed to be able to transmit valuable characteristics of good solutions, future research in combining PR with other existing methods can be performed. An example worth considering is a combination between the GRASP repair operator and PR. If the ALNS is neglected, or is only allowed to perform a low number of iterations, new insights can be obtained. Another possibility would be to broaden the PR, e.g. one could examine the possibility of implementing a path relinking modification based on previous service times. One could also consider an adaptive PR component, by using multiple PR modifications in the same algorithm as future research.

References

George B Dantzig and John H Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

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

Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005a.

Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005b.

Niaz A Wassan and Gábor Nagy. Vehicle routing problem with deliveries and pickups: modelling issues and meta-heuristics solution approaches. *International Journal of Transportation*, 2(1):95–110, 2014.

Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

Saïd Salhi and Gábor Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society*, 50(10):1034–1042, 1999.

Jeng-Fung Chen and Tai-Hsi Wu. Vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Operational Research Society*, 57(5):579–587, 2006.

Nicola Bianchessi and Giovanni Righini. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers & Operations Research*, 34(2):578–594, 2007.

Gabor Nagy and Saı¨d Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162(1):126–141, 2005.

Joël Raucq, Kenneth Sörensen, and Dirk Cattrysse. Solving a real-life roll-on–roll-off waste collection problem with column generation. *Journal on Vehicle Routing Algorithms*, 2(1):41–54, 2019.

Juyoung Wy, Byung-In Kim, and Seongbae Kim. The rollon–rolloff waste collection vehicle routing problem with time windows. *European Journal of Operational Research*, 224(3):466–476, 2013.

Kristian Hauge, Jesper Larsen, Richard Martin Lusby, and Emil Krapper. A hybrid column generation approach for an industrial waste collection routing problem. *Computers & Industrial Engineering*, 71: 10–20, 2014.

Diego Cattaruzza, Nabil Absi, and Dominique Feillet. Vehicle routing problems with multiple trips. *4OR*, 14(3):223–259, 2016.

Cagri Koc and Ismail Karaoglan. A branch and cut algorithm for the vehicle routing problem with multiple use of vehicles. In *41st International Conference on Computers & Industrial Engineering*, pages 554–559, 2011.

Juan Carlos Rivera, H Murat Afsar, and Christian Prins. Multistart evolutionary local search for a disaster relief problem. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 129–141. Springer, 2013.

Enrico Angelelli and Renata Mansini. The vehicle routing problem with time windows and simultaneous pick-up and delivery. In *Quantitative Approaches to Distribution Logistics and Supply Chain Management*, pages 249–267. Springer, 2002.

Hsiao-Fan Wang and Ying-Yen Chen. A genetic algorithm for the simultaneous delivery and pickup problems with time window. *Computers & Industrial Engineering*, 62(1):84–95, 2012.

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

Chao Wang, Dong Mu, Fu Zhao, and John W Sutherland. A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering*, 83:111–122, 2015.

Yong Shi, Yanjie Zhou, Toufik Boudouh, and Olivier Grunder. A lexicographic-based two-stage algorithm for vehicle routing problem with simultaneous pickup–delivery and time window. *Engineering Applications of Artificial Intelligence*, 95:103901, 2020.

Farah Belmecheri, Christian Prins, Farouk Yalaoui, and Lionel Amodeo. An ant colony optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *IFAC Proceedings Volumes*, 42(4):1550–1555, 2009.

Farah Belmecheri, Christian Prins, Farouk Yalaoui, and Lionel Amodeo. Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *Journal of Intelligent Manufacturing*, 24(4):775–789, 2013.

Julian Hof and Michael Schneider. An adaptive large neighborhood search with path relinking for a class of vehicle-routing problems with simultaneous pickup and delivery. *Networks*, 74(3):207–250, 2019.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, 2014.

Vera C Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12):3215–3228, 2012.

Fred Glover. Tabu search and adaptive memory programming—advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Springer, 1997.

Sin C Ho and Michel Gendreau. Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12 (1-2):55–72, 2006.

Hideki Hashimoto and Mutsunori Yagiura. A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 254–265. Springer, 2008.

Alireza Rahimi-Vahed, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. A path relinking algorithm for a multi-depot periodic vehicle routing problem. *Journal of Heuristics*, 19(3):497–524, 2013.

I Or. *Traveling Salesman-Type Combinational Problems and their Relation to the Logistics if Regional Blood Banking*. PhD thesis, Ph. D. Dissertation, Northwestern University, 1976.

Appendix A Parameter Tuning

Table 10: Performance measures of parameter tuning 1: χ^{grasp}

χ^{grasp}	UB (*SD*)	Time	Iterations
0.2	28815.20 (*410.98*)	22.68s	153.52
0.3	28788.63 (*389.67*)	20.36s	161.06
0.4	28856.77 (*425.95*)	20.99s	151.28

Table 11: Performance measures of parameter tuning 2: ζ and ι

	ζ \ ι	3	5	7
UB (*SD*)	0.15	29042.90 (*463.55*)	28865.43 (*411.50*)	28918.68 (*430.51*)
	0.25	28932.93 (*441.64*)	28860.54 (*366.98*)	28824.80 (*298.62*)
	0.35	28939.08 (*442.62*)	28738.56 (*325.53*)	28726.71 (*301.09*)
Time	0.15	15.02s	18.28s	17.84s
	0.25	15.97s	21.75s	21.63s
	0.35	19.44s	26.56s	26.66s
Iterations	0.15	150.42	152.64	142.26
	0.25	149.30	158.10	147.46
	0.35	163.10	156.08	157.48

Table 12: Performance measures of parameter tuning 3: ψ

ψ	UB (*SD*)	Time	Iterations
5	28713.45 (*339.01*)	24.02s	147.58
20	28775.14 (*383.09*)	25.25s	153.58
50	28641.58 (*295.64*)	23.45s	149.94

Table 13: Performance measures of parameter tuning 4: σ^{imp}, σ^{acc} and σ^{best}

	σ^{best}	σ^{imp} \ σ^{acc}	1	3	5
UB (*SD*)	6	3	28715.31 (*379.35*)	28625.25 (*297.49*)	28680.30 (*306.34*)
		6	28709.61 (*338.72*)	28643.45 (*326.34*)	28617.31 (*263.02*)
		9	28712.56 (*339.77*)	28627.19 (*308.91*)	28655.82 (*294.92*)
	9	3	28673.73 (*312.67*)	28620.35 (*315.18*)	28597.83 (*294.35*)
		6	28605.47 (*288.54*)	28576.53 (*243.01*)	28660.42 (*340.32*)
		9	28678.65 (*311.66*)	28791.58 (*374.42*)	28666.04 (*279.83*)
	15	3	28698.69 (*332.62*)	28625.37 (*320.83*)	28671.41 (*301.06*)
		6	28736.54 (*352.22*)	28725.38 (*390.05*)	28654.62 (*287.44*)
		9	28718.83 (*346.36*)	28648.65 (*274.50*)	28697.90 (*327.21*)
Time	6	3	26.37s	25.80s	26.13s
		6	23.74s	24.25s	28.18s
		9	24.47s	24.52s	24.65s
	9	3	23.63s	23.42s	24.29s
		6	24.86s	24.12s	27.78s
		9	26.40s	25.50s	27.50s
	15	3	23.65s	24.54s	25.28s
		6	24.88s	23.12s	25.47s
		9	24.90s	27.44s	22.29s
Iterations	6	3	154.02	162.40	165.44
		6	148.48	151.38	176.76
		9	157.06	154.50	158.32
	9	3	151.66	148.46	153.72
		6	159.96	152.06	175.55
		9	165.86	161.58	175.14
	15	3	149.72	156.96	159.38
		6	156.22	147.88	161.24
		9	161.16	173.12	143.18

Table 14: Performance measures of parameter tuning 5: α and γ

	α \ γ	10	20	30
UB (*SD*)	0.1	28660.84 (*319.52*)	28785.81 (*354.49*)	28637.69 (*325.93*)
	0.2	28654.57 (*303.12*)	28582.82 (*239.97*)	28636.70 (*288.03*)
	0.3	28625.51 (*296.59*)	28701.95 (*283.07*)	28677.82 (*340.66*)
Time	0.1	27.34s	29.56s	31.92s
	0.2	29.34s	24.12s	25.81s
	0.3	24.51s	25.74s	26.83s
Iterations	0.1	159.18	169.38	166.48
	0.2	164.22	152.06	153.14
	0.3	153.76	154.44	157.48

Table 15: Performance measures of parameter tuning 6: ρ

ρ	UB (*SD*)	Time	Iterations
0.5	28662.89 (*342.93*)	30.51s	158.26
0.7	28703.28 (*326.07*)	29.78s	160.16
0.9	28688.63 (*294.84*)	27.91s	165.08

Table 16: Performance measures of parameter tuning 7: λ

λ	UB (*SD*)	Time	Iterations
5	28607.64 (*269.44*)	25.42s	154.92
10	28595.34 (*292.39*)	24.93s	151.36
15	28648.35 (*309.88*)	26.84s	161.14

Table 17: Performance measures of parameter tuning 8: τ

τ	UB (*SD*)	Time	Iterations
3600	28678.18 (*288.29*)	23.41s	145.84
14400	28652.32 (*335.88*)	24.03s	159.52
28800	28634.18 (*342.77*)	20.69s	140.64

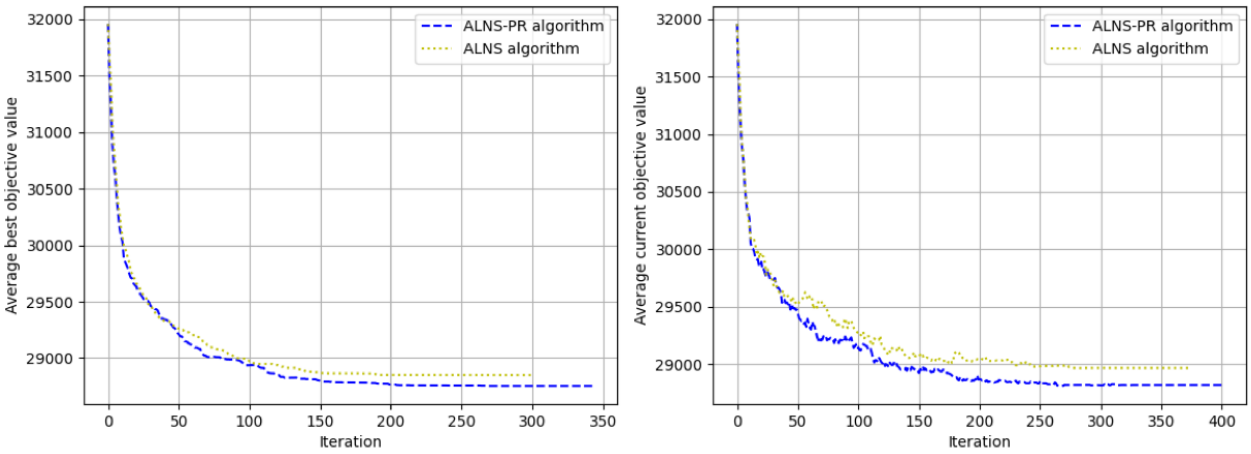
Table 18: Performance measures of parameter tuning 9: δ

δ	UB (*SD*)	Time	Iterations
0.0025	28651.37 (*283.55*)	30.39s	149.18
0.01	28615.02 (*290.76*)	24.12s	146.52
0.05	28677.18 (*317.31*)	37.45s	178.18

Table 19: Performance measures of parameter tuning 10: ξ, ξ^{reset} and n^{sa}

	ξ	$\xi^{reset} \backslash n^{sa}$	25	100	125
UB (*SD*)	50	$\frac{1}{3}\xi$	28720.32 (*377.75*)	28814.84 (*357.09*)	28774.31 (*328.91*)
		$\frac{2}{3}\xi$	28873.74 (*439.87*)	28809.19 (*324.19*)	28723.34 (*342.78*)
		None	28817.47 (*384.14*)	28750.10 (*287.75*)	28740.80 (*373.25*)
	75	$\frac{1}{3}\xi$	28685.24 (*314.18*)	28704.34 (*348.08*)	28714.95 (*315.47*)
		$\frac{2}{3}\xi$	28702.08 (*336.86*)	28632.04 (*346.95*)	28704.59 (*394.31*)
		None	28731.77 (*348.79*)	28704.12 (*308.02*)	28649.29 (*378.55*)
	125	$\frac{1}{3}\xi$	28592.65 (*337.86*)	28598.63 (*265.51*)	28553.24 (*235.47*)
		$\frac{2}{3}\xi$	28577.32 (*232.22*)	28545.31 (*227.11*)	28652.27 (*336.02*)
		None	28622.78 (*328.12*)	28628.53 (*286.62*)	28604.03 (*256.58*)
Time	50	$\frac{1}{3}\xi$	19.62s	18.11s	15.53s
		$\frac{2}{3}\xi$	14.23s	16.74s	16.04s
		None	13.26s	15.58s	15.09s
	75	$\frac{1}{3}\xi$	21.39s	22.21s	22.88s
		$\frac{2}{3}\xi$	20.28s	21.70s	29.11s
		None	18.11s	23.01s	23.94s
	125	$\frac{1}{3}\xi$	30.22s	33.52s	35.08s
		$\frac{2}{3}\xi$	33.37s	37.64s	33.37s
		None	27.22s	32.41s	38.86s
Iterations	50	$\frac{1}{3}\xi$	100.50	112.46	104.38
		$\frac{2}{3}\xi$	99.02	111.22	106.62
		None	90.86	97.62	96.18
	75	$\frac{1}{3}\xi$	151.06	149.78	150.92
		$\frac{2}{3}\xi$	141.14	141.28	156.30
		None	124.64	150.96	154.54
	125	$\frac{1}{3}\xi$	217.84	229.92	241.88
		$\frac{2}{3}\xi$	235.32	254.72	226.82
		None	194.24	219.18	263.26

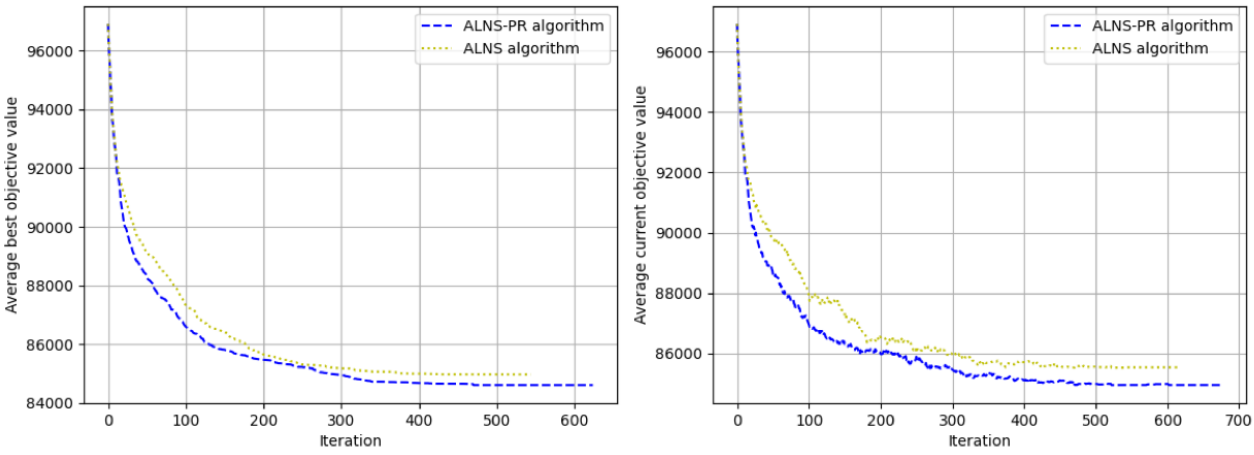
Appendix B Algorithm results



(a) Average best solution value

(b) Average current solution value

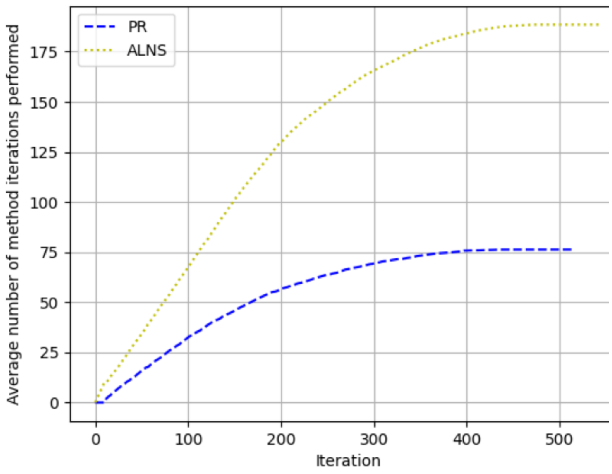
Figure 6: Average best and current solution found per iteration with the ALNS and ALNS-PR algorithm and 45 customers



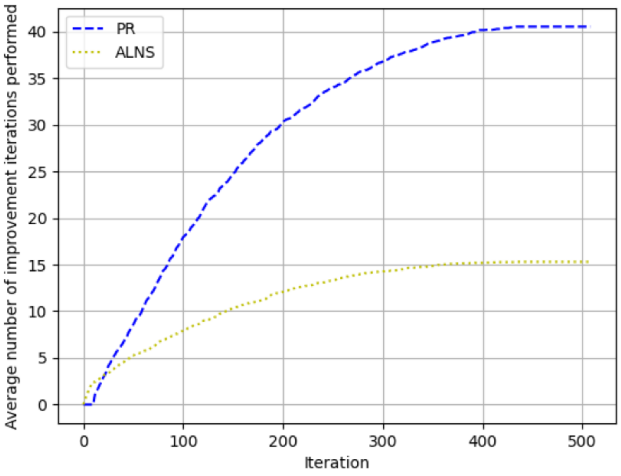
(a) Average best solution value

(b) Average current solution value

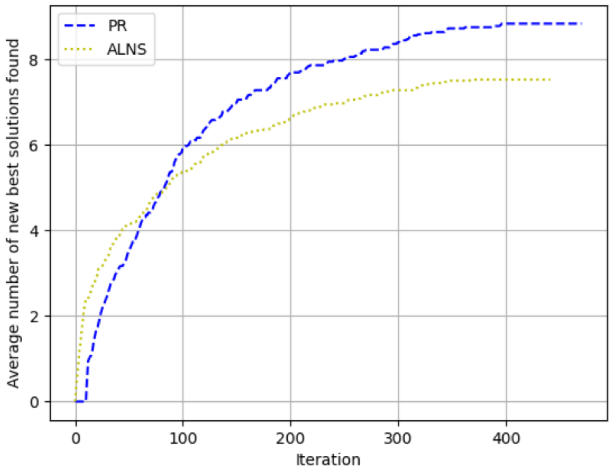
Figure 7: Average best and current solution found per iteration with the ALNS and ALNS-PR algorithm and 200 customers



(a) The number of iterations performed by ALNS and PR



(b) The number of iterations resulting in an improvement



(c) The number of iterations resulting in a new best solution

Figure 8: Total number of iterations performed, iterations resulting in an improvement and iterations resulting in a new best solution by ALNS or PR for each stage in the ALNS-PR algorithm with 100 customers