BACHELOR THESIS

ECONOMETRICS AND OPERATIONS RESEARCH

The Pickup and Delivery Problem with Time Windows: an Adaptive Large Neighborhood Search heuristic

---

Author:
Liana van der Hagen
409165

Supervisor:
T. R. Visser, MSc.

Second assessor:
Dr. R. Spliet

Erasmus School of Economics
ERASMUS UNIVERSITY ROTTERDAM

July 2, 2017

Abstract

The Pickup and Delivery Problem with Time Windows, concerns the transportation of goods between paired pickup and delivery locations. In this problem, which is a generalization of the classical Vehicle Routing Problem, pickup locations must be visited before the corresponding delivery locations by the same vehicle.
In this thesis we implement the adaptive large neighborhood search heuristic, as proposed by Ropke and Pisinger (2006). This heuristic is an extension of the large neighborhood search heuristic, in which the removal heuristics and insertion heuristics are picked at random with adaptive selection probabilities. These probabilities change during the search, based on previous performance.
We reproduce the results of Ropke and Pisinger (2006). In addition, we propose a new time related removal heuristic for the adaptive large neighborhood search heuristic and investigate an extension of the problem which includes a transfer point. At this transfer point the goods from the pickup location can be stored temporarily, after which a different vehicle can deliver these goods to its corresponding delivery location.
We found that the inclusion of the time related removal heuristic results in better solutions for part of the test instances. Moreover, the adaptive large neighborhood search heuristic for the extended problem with transfer point also gives better solutions for many of the test instances.

Contents

1 Introduction

In this thesis the Pickup and Delivery Problem with Time Windows (PDPTW) is studied. The PDPTW is a well-known generalization of the classical Vehicle Routing Problem. The problem consists of assigning transportation requests to vehicles and routing the vehicles. Each request consists of the pickup of goods at one customer location and the delivery of these goods at another location. These customer locations must be visited within a specified time window.

Typical examples of a PDPTW include service providers which are contracted to move goods from one location to another, where the locations, for example, correspond to warehouses, factories or stores. In the context of grocery stores, the suppliers of the supermarkets correspond to the pickup locations and the supermarkets themselves to the delivery locations. Research in home health care logistics concerns, among other things, the transportation of drugs from hospitals to patients' homes (Liu et al., 2013).

The PDPTW also has a lot of other practical applications for which additional constraints are necessary. An example of such an application is the transportation of elderly or handicapped people (Doerner & Salazar-González, 2014), which is often referred to as the Dial-a-Ride Problem. For transportation of persons, the service quality is very important. Specific type of transportation requirements can be included to ensure this quality. A service related constraint can, for example, set a maximum to the duration of the trip for each customer. The PDPTW can also be applied to many other applications, such as school bus routing and transportation of bulk products by ship from production to consumption harbours (Desaulniers et al., 2000).

Ropke and Pisinger (2006) propose an adaptive large neighborhood search (ALNS) heuristic for the PDPTW and have shown that this heuristic gives very good results on benchmark instances. The method uses fast and simple removal and insertion heuristics to modify the current solution. Removal heuristics remove transportation requests from their routes and insertion heuristics reinsert them into the solution. The adaptive aspect of the heuristic is that the probability of choosing a removal or insertion heuristic depends on the performance in previous iterations.

The aim of this research is to implement the proposed ALNS heuristic and replicate the results obtained by Ropke and Pisinger (2006). Moreover, we will try to obtain better solutions for the PDPTW by extending their approach. Ropke and Pisinger (2006) show that especially the removal heuristic has a large influence on the quality of the solution. Therefore, we include another heuristic for removing requests, that removes requests that are related in terms of time. As the time windows at the locations restrict the insertion possibilities of requests into routes, we believe that adding such a removal method could be beneficial.

Furthermore, we extend the PDPTW by introducing a transfer point. Using a transfer point, it is possible for a request to be split, such that two vehicles handle the same request. One vehicle picks up the load at the pickup location and delivers it at the transfer point. The other vehicle can pick this load up at a later time and deliver it at the corresponding delivery location. This can, for example, be beneficial when pickup and delivery locations are far apart from each other, but also when time windows are very restrictive.

The organization of this thesis is as follows. In Section 2, a literature review on the PDPTW is presented. A description of the problem is given in Section 3 and in Section 4 the ALNS heuristic is explained. The data for this research and results are discussed in Section 5. Finally, Section 6 contains a conclusion and a discussion.

2 Literature Review

The PDPTW and variants of this problem have been studied extensively in literature. Many solution methods are proposed and as the problem is NP-hard, the focus has been mostly on heuristics. A recent survey on the problem and some of its variants are given by Parragh, Doerner, and Hartl (2008).

Landrieu, Mati, and Binder (2001) develop a tabu search procedure to solve the problem with a single vehicle. Lau and Liang (2002) also use tabu search and test their method on random generated instances. In addition, the authors compare several construction heuristics for the problem. They propose the partitioned insertion heuristic, which combines the advantages of an insertion heuristic and a sweep heuristic, both modified for the PDPTW. Nanry and Barnes (2000) present a Reactive Tabu Search approach to solve the problem, which is an extension of tabu search. This search consists of dynamically alternating neighborhoods, an adaptive length of the tabu list and an escape mechanism.

Van der Bruggen, Lenstra, and Schuur (1993) develop a variable-depth exchange procedure for the problem with a single vehicle, in which the number of arcs to be exchanged is determined dynamically.

Perishable goods, such as catering foods, benefit from short transportation times. Farahani, Grunow, and Günther (2012) therefore propose a model that integrates the short-term production and distribution planning to optimize the time between the production and the delivery of goods. An iterated solution procedure is proposed in which large neighborhood search (LNS), which was first introduced by Shaw (1997), is used for the distribution planning part. The principle behind LNS is to improve the solution by destroying it, which involves the removal of requests from the solution. Then requests are inserted again, thus repairing the solution.

It has been shown that it is beneficial to split the total amount of goods for a request over different vehicles in the Pickup and Delivery Problem (Şahin et al., 2013). Nowak, Ergun, and White III (2008) prove that when the size of the request is slightly over 50% of the capacity of the vehicle, this benefit is the largest.

Bianchessi and Righini (2007) propose several heuristic algorithms for the problem with simultaneous pickup and delivery, where customers require the delivery of goods and the pickup of goods or waste.

Bent and Van Hentenryck (2006) use a two-stage approach for the PDPTW, where in the first stage the number of vehicles is minimized using Simulated Annealing. In the second stage the travel costs are minimized using LNS.

Ropke and Pisinger (2006) present an extension of this LNS, the adaptive large neighborhood search (ALNS) heuristic, and have shown that this heuristic gives very good results for the PDPTW. ALNS uses simple removal and insertion heuristics that are selected with adaptive selection probabilities. Using several heuristics and selecting them adaptively based on the current problem instance, makes this a robust method. In Pisinger and Ropke (2007), ALNS is tested on five other variants of the Vehicle Routing Problem, where the ALNS heuristic is able to improve many best known solutions. The authors found that a mixture of good and less good removal and insertion heuristics results in better solutions, than when only using good heuristics.

Cortés, Matamala, and Contardo (2010) introduce a strict formulation for the PDPTW with transfer points. The authors consider the transportation of people and therefore include an extra restriction on the maximum time each passenger can wait at the transfer point. They propose a branch-and-cut method to solve the problem. The extension of the PDPTW including a the transfer point that we consider in this thesis is similar to the problem in

Cortés et al. (2010). We focus, however, on the transportation of goods instead of people and investigate whether including the option of using a transfer point in the ALNS heuristic can be beneficial.

3 Problem Description

3.1 PDPTW

The PDPTW involves the design of optimal vehicle routes given a number of customer requests, $r_i = (i^+, i^-)$, consisting of a pickup node i^+ and a delivery node i^-. Each node j is assigned a time window $[a_j, b_j]$, which means that no service can be provided at the location before the start of the time window nor after the end of the time window. A vehicle is allowed to arrive at the customer location before the beginning of the time window, but has to wait to start service. Let $N = P \cup D$ be the set of all customer nodes, with P the set of pickup nodes and D the set of delivery nodes, where $|P| = n$. Let K be the set of all available vehicles, with $|K| = m$. Each request r_i is assigned a subset of vehicles K_{r_i}, which can serve this request. Analogous, P_k and D_k contain all pickup and delivery nodes, respectively, that can be served by vehicle k and $N_k = P_k \cup D_k$. Requests that cannot be served by every vehicle are called special requests. Each vehicle k has a limited capacity C_k and a given start terminal, τ_k, and end terminal, τ'_k. These terminals also have a specified time window, vehicles have to leave the start terminal at a_{τ_k} and must return to the end terminal before $b_{\tau'_k}$. The graph $G = (V, A)$ consists of all customer nodes, start terminals and end terminals and arcs $A = V \times V$. The graph $G_k = (V_k, A_k)$ is the subgraph for vehicle k, where V_k consists of the nodes that may be visited by vehicle k, that is $V_k = N_k \cup \{\tau_k\} \cup \{\tau'_k\}$ and $A_k = V_k \times V_k$. The distance and travel time from node i to node j are denoted by d_{ij} and t_{ij}, respectively. We assume that the travel times and distances satisfy the triangle inequality. Furthermore, s_i denotes the duration of service at node i and l_i the load that should be transported from node i if i is a pickup node, or the load that should be transported to that node if i is a delivery node. When constructing the routes, it should be taken into account that the pickup location and the delivery location are on the same route and the pickup location must be visited before the delivery location. The decision variable x_{ijk} is binary variable, equal to 1 if the edge between node i and node j is used by vehicle k. It might be that the set of available vehicles is not able to serve all requests. These requests are placed in a so-called request bank. In this thesis we do not allow for final solutions to have requests in the request bank, but in a transition stage of the heuristic the request bank may be used. The binary variable z_i is set to 1 if the request corresponding to pickup node i is placed in the request bank. Moreover, the decision variables S_{ik} and L_{ik} denote the start time of service by vehicle k at node i and an upperbound on the amount of goods that are in vehicle k after visiting node i, respectively. $T_i = \sum_{k \in K} \sum_{j \in V_k} S_{ik} x_{ijk}$ indicates the time at which service starts at node i.
A mathematical model of the problem, as given by Ropke and Pisinger (2006), is as follows.

Mathematical model

$$\min \quad \alpha \sum_{k \in K} \sum_{(i,j) \in A} d_{i,j} x_{ijk} + \beta \sum_{k \in K} (S_{\tau'_k,k} - a_{\tau_k}) + \gamma \sum_{i \in P} z_i \tag{1}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{j \in N_k} x_{ijk} + z_i = 1 \qquad \forall i \in P \tag{2}$$

$$\sum_{j \in V_k} x_{ijk} - \sum_{j \in V_k} x_{i,n+i,k} = 0, \qquad \forall k \in K,\ \forall i \in P_k \tag{3}$$

$$\sum_{j \in P_k \cup \tau'_k} x_{\tau_k,j,k} = 1 \qquad \forall k \in K \tag{4}$$

$$\sum_{i \in D_k \cup \tau_k} x_{i,\tau'_k,k} = 1 \qquad \forall k \in K \tag{5}$$

$$\sum_{i \in V_k} x_{ijk} - \sum_{i \in V_k} x_{jik} = 0 \qquad \forall k \in K,\ \forall j \in N_k \tag{6}$$

$$x_{ijk} = 1 \Rightarrow S_{ik} + s_i + t_{ij} \leq S_{jk} \qquad \forall k \in K,\ \forall (i,j) \in A_k \tag{7}$$

$$a_i \leq S_{ik} \leq b_i \qquad \forall k \in K,\ \forall i \in V_k \tag{8}$$

$$S_{ik} \leq S_{n+i,k} \qquad \forall k \in K,\ \forall i \in P_k \tag{9}$$

$$x_{ijk} = 1 \Rightarrow L_{ik} + l_j \leq L_{jk} \qquad \forall k \in K,\ \forall (i,j) \in A_k \tag{10}$$

$$L_{ik} \leq C_k \qquad \forall k \in K,\ \forall i \in V_k \tag{11}$$

$$L_{\tau_k k} = L_{\tau'_k k} = 0 \qquad \forall k \in K \tag{12}$$

$$x_{ijk} \in \mathbb{B} \qquad \forall k \in K,\ \forall (i,j) \in A_k \tag{13}$$

$$z_i \in \mathbb{B} \qquad \forall i \in P \tag{14}$$

$$S_{ik} \geq 0 \qquad \forall k \in K,\ \forall i \in V_k \tag{15}$$

$$L_{ik} \geq 0 \qquad \forall k \in K,\ \forall i \in V_k. \tag{16}$$

The objective function (1) minimizes a weighted sum of the total distance travelled, the time spent by all vehicles and the number of requests that are placed in the request bank. Constraints (2) specify that either a pickup node is in one route or the corresponding request is placed in the request bank. Constraints (3) ensure that the pickup location and the delivery location are visited by the same vehicle and check that the delivery node is visited if and only if the pickup node is visited. Constraints (4),(5) and (6) specify that each vehicle has to leave its start terminal, arrive at its end terminal and when visiting a customer node it must arrive at that node and leave that node, respectively. Constraints (7) and (8) ensure that service cannot start at a location before the vehicle is able to arrive at that location, it cannot start before the beginning of the time window or after the end of the time window at that location, respectively. Constraints (9) are included to make sure that each delivery location is visited later in time than its corresponding pickup location. Constraints (10),(11) and (12) ensure that the loads on the vehicles are set correctly and satisfy the capacity constraints. Finally, constraints (13), (14), (15) and (16) specify the domain of the variables.

3.2 PDPTW with transfer point

We extend the PDPTW by allowing goods to be transferred from one vehicle to another. The goods from the pickup location can be delivered at a so-called transfer point. Thereafter, another vehicle can transport the goods from this transfer point to the corresponding delivery location of the request. We include one transfer point with a specified location and a specified service time to the problem. We assume that there is unlimited storage space and no time window at this point. Another assumption we make is that an unlimited number of vehicles may be loaded and unloaded at each point in time at this transfer point. In the PDPTW with transfer point, each request $r_i = (i^+, i^-)$ can be served regularly or be split into two requests $r_i^1 = (i^+, t_i^-)$ and $r_i^2 = (t_i^+, i^-)$, where $t_i^- \in T^-$ denotes the transfer point as delivery node and $t_i^+ \in T^+$ denotes the transfer point as pickup node for request r_i. The sets T^- and T^+ consist of the transfer nodes for all requests. Cortés et al. (2010)

also consider the transfer point to consist of two nodes, to capture the difference between loading and unloading at the transfer point. The authors, however, consider a fixed loading and unloading time at the transfer point, whereas we model this as two separate service durations for both transfer nodes, $s_{t_i^-}$ and $s_{t_i^+}$. The two sub-requests r_i^1 and r_i^2 can be considered as regular requests that are not split and therefore the constraints are similar to those in the problem without transfer point. However, the requests r_i^1 and r_i^2 should have suitable start times of service, where t^+ should be served after t^-. When a request r_i is split, thus the transfer point is used, the following must hold.

$$S_{t_i^-,k_1} + s_{t_i^-} \leq S_{t_i^+,k_2}. \tag{17}$$

Here $k_1 \in K$ does not necessarily have to be different from $k_2 \in K$. Node t_i^- must first be serviced by vehicle k_1, before service can start at node t_i^+ by vehicle k_2.

4 Methodology

4.1 Adaptive large neighborhood search

In this section the main characteristics of the ALNS approach are discussed. LNS iterations are performed, using in every iteration another removal and insertion method, based on previous performance. Algorithm 1 shows the pseudocode of the ALNS heuristic. Here R denotes the set of removal heuristics and I the set of insertion heuristics.

Algorithm 1 ALNS

```
1: Input: initial solution s_start
2: s_best ← s_start
3: s ← s_start
4: initialize weights w
5: while stopping criteria not met do
6:     i ← 0
7:     set scores to 0
8:     while i < segment size do
9:         determine q
10:        select x ∈ R and y ∈ I according to w
11:        s' ← y(x(s))
12:        if accept(s,s') then
13:            s ← s'
14:        if f(s') ≤ f(s_best) then
15:            s_best ← s'
16:        update scores
17:        update temperature T
18:        i ← i + 1
19:    update w
20: return s_best
```

In each iteration of the ALNS heuristic the number of requests to be removed is determined randomly. This number, q, is in the interval $4 \leq q \leq min(100, \xi n)$, where n is the number of requests and ξ is a predetermined parameter. The removal and insertion heuristics are randomly determined with probabilities based on their weights, w. Then q requests are

removed by the chosen removal heuristic x and inserted back into the solution by insertion heuristic y, which is denoted by $y(x(s))$. This results in a solution that can either be accepted or not, based on acceptance criteria from Simulated Annealing. Simulated Annealing uses a temperature T which determines the probability of accepting a non-improving solution and this temperature is updated after each iteration. We also keep track of the best known solutions obtained so far in the heuristic. The weights w determine the probability of selecting each of the removal and insertion heuristics. These weights are not updated after each iteration, but after each segment, consisting of a predefined number of iterations. After each iteration, the scores that partly determine the weights are updated. The heuristic stops after having performed N iterations.
In section 4.1.1 we discuss how an initial solution is obtained. In Section 4.1.2 and 4.1.3 the removal and insertion heuristics are discussed, respectively. The acceptance criterion, the principle for selecting the removal and insertion methods and the additional noise term that is added to the objective function are discussed in Section 4.1.4, Section 4.1.5 and Section 4.1.5, respectively. We also discuss a two-stage method, in which the primary objective is the minimization of vehicles, in Section 4.1.7. Finally in Section 4.2 and 4.3 we discuss the extensions to the original ALNS heuristic.

4.1.1 Initial solution

Before starting the ALNS heuristic, we first require an initial solution for the problem. We use the following simple insertion heuristic to construct this first solution. The costs of all feasible insertion positions are evaluated for both the pickup and delivery locations for all routes. We insert the request with the lowest insertion cost, that is, the smallest change in objective value when inserting the request in its best position. This is repeated until no request can be inserted into the routes. Remaining requests are placed in the request bank. To check if the performance of the ALNS heuristic depends on the initial solution, we also change this construction heuristic a bit. When calculating the insertion cost of a request in a vehicle, we add an extra term if this vehicle is empty: $4\, max_{i,j \in V} d_{ij}$. We do so, to avoid the case of too many vehicles in the solution. The purpose of this modified insertion heuristic, which we name penalized insertion heuristic, is to investigate the influence of an initial solution on the final solution obtained by the ALNS heuristic.

4.1.2 Removal heuristics

We consider three removal heuristics: the Shaw removal heuristic, the worst removal heuristic and a random removal heuristic. All three heuristics destroy the solution, by removing q requests.

Shaw removal heuristic In the Shaw removal heuristic q requests that are related in terms of distance, start time of service, size of the load and vehicles that are able to serve these requests, have a larger probability of being removed from the solution s. This relatedness between two requests r_i and r_j is measured by the relatedness measure $R(r_i, r_j)$, given as follows.

$$R(r_i, r_j) = \phi \left(d_{i^+,j^+} + d_{i^-,j^-} \right) + \chi \left(|T_{i^+} - T_{j^+}| + |T_{i^-} - T_{j^-}| \right) + \psi \left(l_i - l_j \right) + \omega \left(1 - \frac{|K_{r_i} \cap K_{r_j}|}{min\{|K_{r_i}|, |K_{r_j}|\}} \right), \tag{18}$$

where the same notation is used as in Section 3.1. The terms are weighted by predefined parameters ϕ, χ, ψ and ω. The terms d_{ij}, T_i and l_i are scaled by dividing them by the

maximum value over all edges for d_{ij}, or nodes for T_i and l_i. In this way the measure is normalized, such that $0 \leq R(r_i, r_j) \leq 2(\phi + \chi) + \psi + \omega$. The smaller $R(r_i, r_j)$ is, the more related the requests r_i and r_j are.
The idea of removing requests that are similar is that there could be more possibilities when reinserting the requests into the solution other than the place from where the requests where removed. A request can more likely be inserted in routes where related requests were previously in.
Pseudocode for the Shaw removal heuristic is shown in Algorithm 2. The routed requests are sorted in array L according to the relatedness measure. The parameter $p \geq 1$ introduces some randomness in the selection of requests to be removed. A larger value of p corresponds to a larger probability that requests in the beginning of L, thus more related, are selected.

Algorithm 2 Shaw removal heuristic

1: **Input:** solution s, parameter q, parameter p
2: $Z \leftarrow$ random routed request from s
3: **while** $|Z| < q$ **do**
4: select request $x \in Z$ uniform randomly
5: Array: $L \leftarrow$ requests from s not in Z
6: sort L according to relatedness to x:
7: $i < j \iff R(L[i], x) < R(L[j], x)$
8: $y \leftarrow$ random number in interval $[0, 1)$
9: $Z \leftarrow Z \cup L\left[\lfloor y^p |L| \rfloor\right]$
10: remove requests in Z from their routes in s

Random removal heuristic The random removal heuristic selects in each iteration uniform randomly a routed request to be removed. The only parameter used in this procedure is q, the number of requests to be removed.

Worst removal In the worst removal heuristic, q requests for which removing leads to the smallest objective values are removed. The idea behind this removal heuristic is that for requests that are inserted in positions for which removing results in much lower objective values, there must exist better insertion positions. The heuristic is similar to the Shaw removal heuristic, as it includes a parameter p_{worst} that introduces some randomness in the selection of the requests. However, the sorting of array L, containing the routed requests, is based on the change in objective value when removing a request $L[i]$, denoted by $c(L[i])$.

Algorithm 3 Worst removal heuristic

1: **Input:** solution s, parameter q, parameter p_{worst}
2: $Z \leftarrow \emptyset$
3: **while** $|Z| < q$ **do**
4: Array: $L \leftarrow$ requests from s not in Z
5: sort L:
6: $i < j \iff c(L[i]) > c(L[j])$
7: $y \leftarrow$ random number in interval $[0, 1)$
8: $Z \leftarrow Z \cup L\left[\lfloor y^{p_{worst}} |L| \rfloor\right]$
9: remove requests in Z from s

4.1.3 Insertion heuristics

Insertion is based on the following heuristics: the basic greedy heuristic and a set of regret heuristics. These insertion heuristics insert the unrouted requests back into the solution.

Basic greedy heuristic In each iteration of the basic greedy heuristic, the request is inserted that leads to the lowest objective value when inserting it at its best position. That is, for each request the insertion possibilities inside all vehicles are evaluated and the best insertion place for both pickup and delivery node are determined. Then, request r_{i^*} is inserted, for which

$$r_{i^*} = \arg\min_{r_i \in U} \Delta f^1_{r_i}, \tag{19}$$

where U is the set of unplanned requests and $\Delta f^1_{r_i}$ denotes the change in objective value when inserting request r_i in the best position.

$$\Delta f^1_{r_i} = \min_{k \in K} \{\Delta f^1_{r_i,k}\}, \tag{20}$$

where $\Delta f^1_{r_i,k}$ is the change in objective value when inserting request r_i in vehicle k at the best position. When no insertion in vehicle k is possible, we set $\Delta f^1_{r_i,k} = \infty$.

Regret-k heuristic The regret-2 heuristic inserts the request that leads to the largest difference in insertion cost when not inserting it at the best position, but at its second best position. The insertion cost is the change in objective value when inserting the unrouted request into the solution. Anticipating more successive insertions occurs in the regret-3, regret-4 and regret-m heuristic, where m is the number of vehicles. Request r^*_i is inserted, for which

$$r^*_i = \arg\max_{r_i \in U} \left(\sum_{j=2}^{k} \Delta f^j_{r_i} - \Delta f^1_{r_i} \right), \tag{21}$$

where $\Delta f^j_{r_i}$ denotes the change in objective value when inserting request r_i in the j-th best position. The degree of regret is denoted by k. In contrast with the basic greedy heuristic, the regret-k heuristic anticipates on successive insertions, and therefore avoids leaving bad insertions for later.

4.1.4 Acceptance criteria

After the removal and insertion operators are applied, the ALNS heuristic determines whether or not the obtained solution is accepted. To avoid getting trapped in a local minimum, we do not only accept solutions that are better than the current solution, but sometimes we also accept worse solutions. We use acceptance criteria from Simulated Annealing, where we accept solution s' with probability $e^{-(f(s')-f(s))/T}$, where s is the current solution, $f(s)$ is the objective value of solution s and $T > 0$ is the temperature. In every iteration, T is updated as follows: $T = Tc$, where c is the cooling rate. T is initialized with T_{start}, depending on the initial solution as follows. The start temperature is chosen such that a solution that is $w\%$ worse than the initial solution is accepted with probability 0.5. To determine T_{start}, we do not include the costs of the requests in the request bank. These costs could have a very large influence on T_{start}, as we set the γ parameter to a large number in order to have a final solution with no requests in the request bank. As the heuristic only needs a few iterations to obtain a solution with an empty request bank, the costs of requests in the request bank could cause a too large start temperature in the remaining iterations.

4.1.5 Selection principle

We divide the ALNS heuristic in segments of 100 iterations. In every segment, new weights for the removal and insertion heuristics are calculated. These weights determine the probability of selecting that heuristic. The probability of choosing heuristic j with weight $w_{j,h}$ in segment h is

$$\frac{w_{j,h}}{\sum_{i \in G} w_{i,h}}, \tag{22}$$

where $G = \{1, .., g\}$, with g the number of removal or insertion heuristics included in the search. The weight for the next segment $h+1$ is based on the weight of the previous segment h, the number of times attempted to use the heuristic in this segment θ_j, a specific score obtained in this segment π_j, and a parameter r:

$$w_{j,h+1} = w_{j,h}(1 - r) + r\frac{\pi_j}{\theta_j}. \tag{23}$$

At the start of each segment in the ALNS heuristic, the score for each of the heuristics is initialized at zero. After each iteration t in that segment, the scores of both the used removal and insertion heuristic are updated by the same number (σ_1, σ_2 or σ_3), depending on the solution found in this iteration as follows:

$$\pi_j^{t+1} = \pi_j^t + \begin{cases} \sigma_1 & \text{if new global best solution} \\ \sigma_2 & \text{if better than current solution, not visited before} \\ \sigma_3 & \text{if worse than current solution, accepted now but not visited before.} \end{cases} \tag{24}$$

The heuristic only increments the scores when a solution is obtained that we did not already visit before. To keep track of the visited solutions, a unique hash code representing the solution is generated and stored in a list. We do not increment the scores for already visited in order to diversify the search.

4.1.6 Noise term in the objective function

With the same selection principle that we use for the selection of the heuristics, we choose whether or not we add a noise term to the insertion costs in the insertion heuristics, based on earlier performance. We modify the insertion cost $c(r_i)$ of request r_i to randomize the insertion a bit such that insertion does not always take place when it is locally the best. The term that is added is a random number in the interval $[-X, X]$, where X is a parameter times the maximum distance between two nodes in the graph.

$$X = \eta \, max_{i,j \in V} d_{ij} \tag{25}$$

As this noise term can also be negative, we round negative insertion costs to zero. The modified insertion cost $c'(r_i)$ for request r_i is then as follows:

$$c'(r_i) = \max\{0, c(r_i) + X\}. \tag{26}$$

4.1.7 Vehicle minimization

In the ALNS heuristic discussed so far, we minimize the objective (1) as stated in Section 3.1. However, the minimization of the number of vehicles used to serve all requests is also often considered to be the primary objective. We therefore discuss a two-stage method, proposed by Ropke and Pisinger (2006), in which the vehicle minimization is the primary

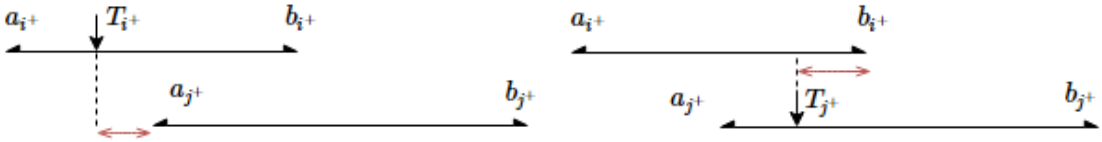
objective and the secondary objective is the objective (1) as in Section 3.1. This two-stage method can only be used for a homogeneous fleet.
First an initial solution is created, assuming that the number of vehicles is unlimited. For this we use a sequential insertion heuristic, which constructs one route at a time. This heuristic inserts requests with the smallest insertion costs in their best positions in the route, until no more insertions in that route are feasible. We keep constructing routes until all requests are routed.
In the first stage of this algorithm, we perform at most Φ iterations in total. One route is selected uniform randomly to be removed from the solution. The requests in this route are placed in the request bank. We then perform ALNS iterations, and stop the heuristic immediately when a solution is found with no requests in the request bank. We obtained a solution in which all requests can be served with one vehicle less. The number of remaining iterations is then reduced by the number of iterations used to obtain this solution and thus to empty the request bank. The procedure starts again, removing another route from the solution and performing again ALNS iterations. We stop this stage if the Φ iterations are performed or if the request bank contains more than 5 requests and the size of the request bank has not reduced in the past τ iterations. If the first stage is terminated and the solution contains requests in the request bank, we step back to the previous solution in which all requests were routed.
The second stage of the algorithm consists of performing the original ALNS heuristic, starting with the solution found in the first stage.

4.2 Extended ALNS: Time related removal heuristic

As an extension to the framework of Ropke and Pisinger (2006), we add a new removal method to the ALNS heuristic. In the PDPTW each location has a specified time window and the start time of service at the pickup location has a large influence on the possibilities for inserting the delivery node in the route. Thus, the time windows are usually very restrictive in the possibilities of inserting requests. We therefore add a removal method that focusses on removing requests that are similar in terms of time. The heuristic we add to the ALNS heuristic is an adaptation of the Shaw removal heuristic. The same procedure is used as in Algorithm 2 in Section 4.1.2, but with another relatedness measure:

$$\begin{aligned} R(r_i, r_j) = &-min\{b_{j^+} - T_{i^+}, T_{i^+} - a_{j^+}\} - min\{b_{i^+} - T_{j^+}, T_{j^+} - a_{i^+}\} \\ &- min\{b_{j^-} - T_{i^-}, T_{i^-} - a_{j^-}\} - min\{b_{i^-} - T_{j^-}, T_{j^-} - a_{i^-}\}. \end{aligned} \tag{27}$$

This measure consists of 4 terms comparing the start of service times of both the pickup and the delivery location of a request with the time window of the other request. The relatedness measure $R(r_i, r_j)$ indicates how much slack there is between the start of service times and time windows. The first two terms compare the time windows and start of service times of the pickup locations. Figure 1 illustrates what these terms measure. The first and second term of $R(r_i, r_j)$ are indicated by a red arrow. The terms for the delivery locations are similar to those for the pickup locations.



(a) $-min\{b_{j^+} - T_{i^+}, T_{i^+} - a_{j^+}\}$ (b) $-min\{b_{i^+} - T_{j^+}, T_{j^+} - a_{i^+}\}$

Figure 1: Illustration relatedness measure for the pickup locations

The first term compares the start of service time at the pickup location of request r_i to the time window of the pickup location of request r_j (Figure 1a). The service time at the pickup location of request r_i lies outside the time window of the pickup location of request r_j. In this case the term measures the distance in time from the closest boundary of the time window of request r_j. A large term indicates that is not likely that j^+ can be inserted at the old position of i^+. Thus, removing two requests with a large relatedness measure can result in insertion of the requests at the same place as where they were previously inserted. Vice versa, the second term compares the start of service time at the pickup location of request r_j to the time window of i^+. Now the service time of j^+ lies inside the time window of pickup location i^+ of request r_i (Figure 1b). The term is negative and smaller when it is further away from the closest boundary of the interval. In this case it is more likely that i^+ can be placed at the position of j^+. Thus, the smaller the sum of four terms, the more related the requests are in terms of time. By taking into account the time windows we keep in mind that inserting the other request into the route will almost inevitably change the start of service times, due to difference in locations.

4.3 ALNS with transfer possibility

We adapt the proposed ALNS heuristic for the PDPTW with transfer point. The greedy insertion heuristic is adapted in such way that not only full requests can be placed into a route, but requests can be divided into two sub-requests: from pickup location to transfer point and from transfer point to delivery location. The number of insertion possibilities increases when allowing for requests to split, using a transfer point.
Consider a request $r_i = (i^+, i^-)$. Its possible transfer nodes are represented by t_i^- and t_i^+, where the first is to denote the transfer node as a delivery node and the second as a pickup node. The heuristic inserts the request for which insertion leads to lowest objective value, but in contrast to the basic greedy insertion heuristic, the lowest objective value can either be from inserting r_i or from inserting ${r_i}^1$ and ${r_i}^2$. Here ${r_i}^1 = (i^+, t_i^-)$ and ${r_i}^2 = (t_i^+, i^-)$ are the split requests. For each request we thus check the best position for insertion of the original request r_i and the best position for inserting the sub-requests ${r_i}^1$ and ${r_i}^2$. We insert the request, either using a transfer node or not, that leads to the lowest value of $min\left\{c(r_i), c({r_i}^1) + c({r_i}^2)\right\}$. Here $c(r_i)$ is the insertion cost of request r_i, the change in objective value when inserting r_i. For every insertion possibility of ${r_i}^1$, all possibilities of insertion of ${r_i}^2$ are considered and the insertions that lead to the lowest joint costs are compared to the costs of inserting the original full request r_i. Figure 2 illustrates how the two sub-requests could be placed into a solution.

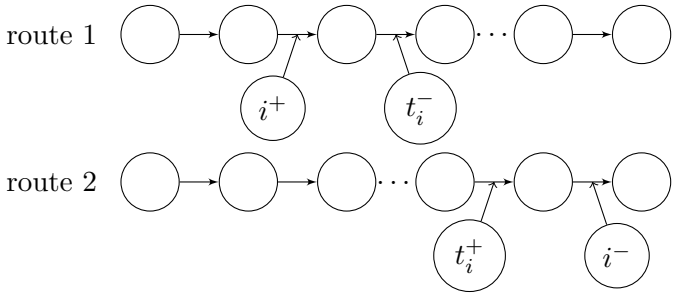
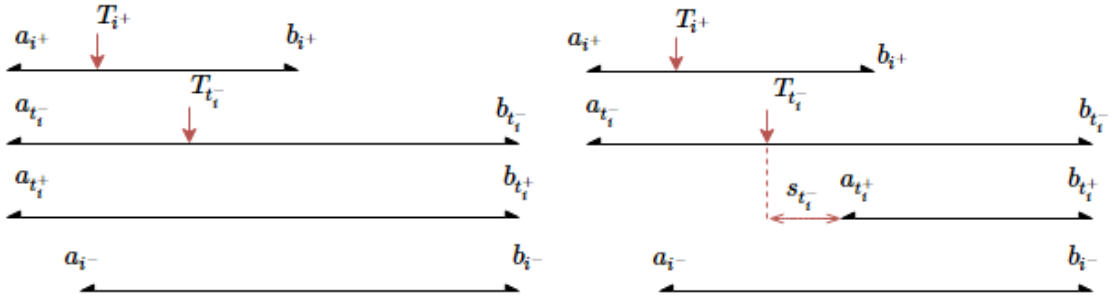


Figure 2: Inserting request into two routes

As goods can only be picked up at the transfer point after they have been delivered there, we need to model a temporal dependency between t_i^- and t_i^+. We do so, by assigning time windows to these nodes. Service at t_i^+ cannot start before service at t_i^- is completed. This is illustrated in Figure 3. The time windows of t_i^- and t_i^+ are initialized as follows: $[a_{t_i^+}, b_{t_i^+}] = [a_{i^+}, b_{i^-}]$ and $[a_{t_i^-}, b_{t_i^-}] = [a_{i^+}, b_{i^-}]$. For each insertion place of the first sub-request, ${r_i}^1$, the time window of the transfer point t_i^+ is adjusted to account for temporal synchronization at the transfer point: $[a_{t_i^+}, b_{t_i^+}] = [T_{t_i^-} + s_{t_i^-}, b_{i^-}]$. The load can only be picked up at the transfer point after it has been delivered at the transfer point and service has been provided.



(a) Initial time windows

(b) Time windows after inserting ${r_i}^1$

Figure 3: Time windows when using a transfer point

We still include the original basic greedy insertion heuristic which does not use a transfer point in the ALNS heuristic, as the new insertion heuristic including transfer point increases run time.

We also have to adapt our removal heuristics to handle these sub-requests with transfer point properly. If the request is split into two sub-requests and one of the sub-requests is removed from the solution by a removal heuristic, we also remove the other sub-request. In the insertion heuristics we evaluate the insertion of the original request, which can be split again into two sub-requests. We choose to evaluate the insertion of the original request, as this leads to more insertion possibilities due to the restriction of temporal dependency between the two sub-requests.

5 Experimental results

5.1 The benchmark data sets

We test the heuristics on the 116 instances from Li and Lim (2003), containing around 100 and 200 locations. These instance can be downloaded from SINTEF (2008). We also use the 24 instances that were randomly generated by Ropke and Pisinger (2006), containing 50 and 100 requests. The instances from Li and Lim (2003) are divided into 6 sets: R1, R2, C1, C2, RC1 and RC2. Here the R, C and RC indicate that the set contains instances that are uniformly distributed in space, clustered or a mix of both, respectively. The 1 indicates that the set contains instances with a small scheduling horizon, thus few customers per route, whereas 2 indicates the opposite. Inside each set the instances differ in width of time windows.
The difference between the instances generated by Li and Lim (2003) and Ropke and Pisinger (2006) is that the latter contains multiple terminals and vehicles with different start and end terminals. In addition, some of the instances from Ropke and Pisinger (2006) contain special requests, requests that can only be served by some of the vehicles. The instances each contain a different combination of route type, request type and geographical distributions. As route type we consider vehicles with the same start and end terminal and vehicles with different start and end terminal. Each instance can either contain normal requests or a combination of normal and special requests. The geographical distributions are either uniform over space, clustered or a mix of both.

5.2 ALNS experiments

We run three versions of the heuristic separately: the original ALNS heuristic proposed by Ropke and Pisinger (2006), the ALNS heuristic with the extra time related removal heuristic and the ALNS heuristic applied to the problem with a transfer point. In all runs we set $N = 25,000$. We use one transfer point and for each instance it is located at the centre of the customer locations. The service time at a transfer node, which can be either unloading or loading time, is set to $\frac{1}{3}$ of the average service time of all customer nodes, since we assume that loading and unloading at the transfer point can be more efficient. This results in a total handling time of $\frac{2}{3}$ of the average service time of all customer nodes at the transfer point.
We run the different versions of the ALNS heuristic 5 times to each of the instances of 100 locations from Li and Lim (2003) and of 50 and 100 requests from Ropke and Pisinger (2006). We also apply the original ALNS heuristic 5 times on the instances with 200 locations of Li and Lim (2003). The original ALNS heuristic with the penalized insertion heuristic and the original ALNS heuristic with the vehicle minimization procedure are applied to the instances from Li and Lim (2003) with 100 locations, where we set $\Phi = 25,000$ and $\tau = 2,000$. For the instances with 50 and 100 requests from Ropke and Pisinger (2006) we also compare the results obtained by ALNS with those obtained by using LNS. As Ropke and Pisinger (2006) show that the LNS configurations with the Shaw removal heuristic, the regret-4 heuristic and noise in the objective function leads to the best results, we will also use this configuration.
The methods are coded in Java. All experiments were performed on a 2.8 GHz Intel Core 2 Duo PC with 4 GB internal memory, running Windows 8.1. The travel times and distances are calculated with double floating point precision.

5.3 Parameter settings

For both sets of instances we have a different objective. For the instances from from Li and Lim (2003) the objective is to minimize the total travelled distance, setting $\alpha = 1$ and $\beta = 0$ in the objective (1). To ensure that the final solution contains all requests, γ is set to $100,000$. The objective for the instances generated by Ropke and Pisinger (2006) is to additionally minimize the total time spent by the vehicles, setting $\alpha = 1$, $\beta = 1$ and $\gamma = 100,000$.

In our implementation we use the same parameter values as Ropke and Pisinger (2006), except for the values of ξ and r, which control how many requests are removed in each iteration and how the weights are adjusted, respectively. The value of ξ can have a large impact on the solution quality, but also on the running time of each iteration. We tune r as we believe it might be beneficial to have a greater reaction factor. A low reaction factor keeps the weight at about the same level during the search, while a higher reaction factor can make the weights converge faster. Our tuning set consists of 8 instances: LR1_2_1, LR202, LRC1_2_3, and LRC204 from Li and Lim (2003) and 50A, 50B, 50C, 50D from Ropke and Pisinger (2006). These instance are also in the tuning set of Ropke and Pisinger (2006), but in addition to these instances the authors include 8 more instances containing 50 requests. We first set all parameters equal to those given by Ropke and Pisinger (2006) and vary ξ between 0.3 and 0.5 with a step size of 0.05. We run each of the instances 5 times for each value of ξ. We choose the ξ that leads to the smallest average deviation (av. gap) from the best obtained solution during this tuning experiment. The results of this tuning can be found in Table 1. The smallest average gap is obtained when setting $\xi = 0.50$. Having this new parameter setting, we vary r between 0.1 and 0.5 with a step size of 0.1. The results can be found in Table 2, where it is shown that $r = 0.2$ leads to the smallest average gap. The final parameter setting is given by the following vector: $(\phi, \chi, \psi, \omega, p, p_{worst}, w, c, \sigma_1, \sigma_2, \sigma_3, r, \eta, \xi) = (9, 3, 2, 5, 6, 3, 0.05, 0.99975, 33, 9, 13, 0.2, 0.025, 0.5)$. Ropke and Pisinger (2006) found that for the vehicle minimization stage different values for w and c give better results: $(w, c) = (0.35, 0.9999)$. Hence, we also use these parameters in this first stage of the two-stage vehicle minimization method.

We obtain a different parameter vector than the one obtained by Ropke and Pisinger (2006). They found $\xi = 0.4$ and $r = 0.1$. This could be due to the fact that the authors used a larger tuning set. However, the instances we used have already quite diverse characteristics. Another plausible reason is the difference in implementation details. We therefore choose to use our own parameter vector.

Table 1: Results of tuning ξ

ξ	0.30	0.35	0.40	0.45	**0.50**
Av. gap (%)	4.98	4.52	3.96	4.40	3.67

Notes. The first row contains the value for parameter ξ. The second row contains the average deviation from the best obtained solution in the tuning experiment. In bold the best parameter value is shown.

Table 2: Results of tuning r

r	0.1	**0.2**	0.3	0.4	0.5
Av. gap(%)	4.42	4.16	4.40	4.19	5.02

Notes. The first row contains the value for parameter r. The second row contains the average deviation from the best obtained solution in the tuning experiment. In bold the best parameter value is shown.

5.4 Results original ALNS

In this section we discuss the results of our implementation of the ALNS heuristic as proposed by Ropke and Pisinger (2006). In Table 3 and Table 4 we report the number of vehicles and the objective values for the instances from Li and Lim (2003) with 100 and 200 locations, respectively. In addition, we report the average run time for each set of instances. The instances with a larger scheduling horizon (R2, C2 and RC2) have a larger run time than those with a smaller scheduling horizon. The average run time for all instances is larger than the run time of Ropke and Pisinger (2006). However, the average run time of 200 locations relative to the average run time of 100 locations is almost the same.
The objective values and number of vehicles in Table 3 and Table 4 deviate from the solutions obtained by Ropke and Pisinger (2006). This can be due to several factors. Firstly, Ropke and Pisinger (2006) run each of the instances 10 times instead of 5 times. Furthermore, we chose to minimize the total distance travelled and not to minimize the number of vehicles for these instances. We therefore obtain for some instances solutions with lower objective values than the best known solutions, but with more vehicles. For the instance LC109, for example, Ropke and Pisinger (2006) obtained a solution with 9 vehicles and a objective value of 1000.60, whereas our solution consists of 11 vehicles, but a lower objective value: 937.45. Furthermore, we used other parameters for ξ and r, this could also cause different results.

Table 3: Best results Li & Lim 100

	R1		R2		C1		C2		RC1		RC2	
	#	f	#	f	#	f	#	f	#	f	#	f
1	22	1806.16	7	1344.69	15	1245.25	6	829.46	18	1915.98	7	1603.32
2	20	1652.72	6	1406.48	15	1258.71	6	834.29	17	1910.24	6	1591.65
3	15	1416.69	6	1170.10	12	1012.18	6	811.57	12	1343.55	5	1278.44
4	13	1266.45	4	1164.16	11	889.56	5	774.70	12	1342.57	6	1042.24
5	18	1610.99	6	1239.28	14	1193.72	6	809.34	17	1933.22	7	1457.40
6	15	1447.31	5	1164.37	14	1154.90	6	789.41	16	1756.66	6	1436.63
7	15	1391.80	5	1133.80	15	1224.27	6	817.07	13	1451.61	6	1538.76
8	13	1280.33	4	1101.14	13	1128.36	6	813.95	13	1398.21	5	1130.27
9	15	1475.73	5	1143.68	11	937.45						
10	15	1348.67	5	1131.96								
11	13	1279.89	4	1108.70								
12	12	1223.63										
CPU(s)		136		362		152		261		130		253

Notes. Results of the ALNS heuristic of the Li and Lim (2003) instances with 100 locations. These instances are divided into 6 sets, containing up to 12 problem instances. The major columns correspond to these sets and the left column indicates the number of the problem. For each instance the number of vehicles and the best objective value in the experiment are reported in the columns denoted by # and f respectively. In the last row, denoted by CPU(s), the average run time for each set of instances is reported in seconds.

Table 4: Best results Li & Lim 200

	R1		R2		C1		C2		RC1		RC2	
	#	f	#	f	#	f	#	f	#	f	#	f
1	21	4955.39	9	4255.11	29	4166.95	11	2586.19	25	4372.05	10	3199.92
2	21	4346.40	9	4260.28	25	3783.39	12	2503.26	22	3686.40	11	3104.03
3	20	3815.58	9	3797.96	22	3497.49	10	2345.80	20	3678.33	10	3093.21
4	16	3239.33	7	3355.73	19	3044.51	9	2444.65	16	3268.60	7	2917.85
5	20	4608.73	8	4114.26	29	4010.36	12	2484.62	22	4338.89	9	3378.00
6	20	4172.73	9	4065.89	29	4004.83	12	2550.97	22	3928.92	9	3274.58
7	18	3637.09	8	3495.12	28	3899.45	12	2611.44	20	3806.99	9	3028.32
8	15	3103.52	6	2783.57	25	3644.65	12	2499.40	19	3694.14	8	3000.25
9	19	4396.47	7	4051.56	26	3791.34	12	2512.85	18	3587.17	8	3020.38
10	18	3778.74	7	3586.90	23	3486.97	11	2531.43	19	3486.86	7	2760.70
CPU(s)		1181		2193		987		1748		1025		1644

Notes. Results of the ALNS heuristic of the Li and Lim (2003) instances with 200 locations. Each set contains 10 instances. The columns and rows should be interpreted as in Table 3.

The results obtained with the penalized insertion method for the initial solution and the results obtained using the two-stage vehicle minimization method are summarized in Table 5. The objective values and the number of vehicles for each instance separately can be found in Table 12 and Table 13 in Appendix A. In Table 5 the total number of vehicles and the total distance are reported, based on the best solution obtained for each instance. The average run time per instance is also reported. The results of Table 3 are also summarized to be able to make a comparison.

The number of vehicles obtained by the ALNS heuristic using the different initial solution, in which additional vehicles were penalized, is only one less than when using the original insertion heuristic. On the other hand, the total travelled distance increased 278. The

desired effect of solutions with less vehicles, does not occur with this heuristic. Nevertheless, this heuristic changed the solutions of many instances, indicating that an initial solution can have a lot of influence on the final solution obtained by ALNS.
The two-stage method, with the first priority being the minimization of the number of vehicles, results in a solution with 72 less vehicles. The total travelled distance is 0.3% higher with this two-stage method, but depending on the objectives in the problem, this relative small increase in total distance could be outweighed by the decrease in the number of vehicles of 12.5%
As expected, the original ALNS heuristic with the two different initial solutions have a similar run time. Surprisingly, the vehicle minimization two-stage method has a smaller run time on average, even though this method performs more iterations in total. The sets with instances with a smaller scheduling horizon (R1, C1 and RC1) have a larger run time than with the original ALNS, whereas the sets containing instances with a larger scheduling horizon (R2, C2 and RC2) have a smaller run time compared to the run times of the original ALNS. The sets R2, C2 and RC2 also have the largest percentage change in the number of vehicles used in the final solution. It could, thus, be that less insertion possibilities must be evaluated as there are less vehicles and therefore run time decreases. We believe, however, that this decrease in run time could be due to the difference in implementation of both the heuristics. In the original ALNS heuristic without vehicle minimization, we consider a fixed number of available vehicles as heterogeneous vehicles and therefore must evaluate insertion in all of these vehicles, even if the fleet is homogeneous. In the implementation of the two-stage vehicle minimization algorithm, we only consider the vehicles that are determined in the first stage of the method.

Table 5: Summary results Li & Lim 100

	Vehicles	Distance	Av. time (s)
Original ALNS	576	70964	217
Penalized insertion	575	71242	214
Vehicle minimization	504	71181	191

Notes. Summary of the results for the instances of Li and Lim (2003) obtained with the original ALNS heuristic (Table 3), the original ALNS heuristic with penalized insertion method for the initial solution and the vehicle minimization two-stage method. The total number of vehicles and the total travelled distance are reported. We also report the average run time.

The results for the instances from Ropke and Pisinger (2006) are shown in Table 6. We report the results relative to the best solution in these experiments and compared them to the solutions obtained using LNS. The objective values and number of vehicles can be found in Table 14 in Appendix A.
It can be seen that LNS performs worse in terms of solution quality than ALNS for both the instances with 50 and with 100 requests. In contrast to Ropke and Pisinger (2006), our implementation of the ALNS heuristic performs better for the instances with 100 requests, whereas the authors found a smaller average gap for the instances with 50 requests. The run times for both methods increase a lot from 50 to 100 requests. This increase is larger for the LNS heuristic.

Table 6: Results Ropke & Pisinger 50 and 100 requests

	Av. gap (%)		Av. time (s)	
	ALNS	LNS	ALNS	LNS
50 requests	1.88	3.23	67	60
100 requests	1.51	3.13	441	503

Notes. Results obtained with ALNS and LNS. The major columns correspond to the average gap from the best solution obtained from ALNS or LNS and the average run time in seconds.

5.5 Results ALNS with time related removal heuristic

In Table 7 and Table 8 the results for the heuristic including the extra time related removal heuristic are shown for the instances from Li and Lim (2003) and Ropke and Pisinger (2006), respectively. A solution in bold means that it is better than the solution obtained using the original ALNS heuristic. The other values are either the same or worse than the solutions in Table 3.
Most benefit for the instances from Li and Lim (2003) is in the sets C1 and R1. These sets contain instances with a small scheduling horizon. There is, however, no clear pattern on the characteristics of the instances for which this extension leads to better results.
The run times for this extension are similar to those of the original ALNS heuristic, which is not surprising as this extra heuristic is an adaptation of the Shaw removal heuristic.

Table 7: Best results Li & Lim 100, ALNS with time related removal

	R1		R2		C1		C2		RC1		RC2	
	#	f	#	f	#	f	#	f	#	f	#	f
1	22	1806.16	7	1345.04	16	1280.91	6	829.46	18	1915.98	7	1629.92
2	20	1652.72	6	**1386.29**	15	**1256.45**	6	**821.60**	17	**1896.77**	7	1619.15
3	16	1441.94	6	1208.23	11	**996.96**	6	814.26	12	1425.49	6	1292.76
4	12	**1255.20**	5	1195.41	10	**841.14**	5	782.40	12	**1315.18**	6	1067.08
5	18	1614.51	6	1260.98	14	**1191.62**	6	811.06	17	**1925.77**	6	**1440.86**
6	15	1479.84	6	1206.72	14	1180.27	6	795.85	16	1794.09	7	1457.70
7	14	**1385.18**	5	1231.86	14	**1190.46**	6	**815.26**	14	1502.16	6	**1524.92**
8	13	**1275.35**	5	1135.29	13	**1070.10**	6	**808.84**	13	1441.16	5	**1111.50**
9	16	**1468.46**	6	**1138.77**	11	952.19						
10	14	**1311.52**	7	1237.23								
11	13	1303.79	4	**1060.80**								
12	12	**1213.72**										
CPU(s)		134		313		145		230		125		224

Notes. In this table the results are shown of the ALNS heuristic including the time related removal heuristic of the Li and Lim (2003) instances with 100 locations. The columns and rows should be interpreted as in Table 3. In bold are the objective values that are lower than obtained using ALNS without including the time related removal heuristic.

Table 8: Best results Ropke & Pisinger 50 and 100 requests, ALNS with time related removal

	50 requests		100 requests	
	#	f	#	f
A	13	76236.90	26	159446.85
B	14	86252.97	25	142301.04
C	13	**75377.71**	27	142313.83
D	14	**85636.46**	28	157054.90
E	15	69563.36	27	**136898.12**
F	14	**68680.07**	27	138385.70
G	14	67587.62	25	128971.82
H	14	72957.49	28	**136412.34**
I	13	68362.58	27	148481.26
J	15	85279.23	28	**153154.60**
K	13	**72899.72**	25	**129244.56**
L	14	**79329.38**	26	**145048.81**
CPU(s)		65		378

Notes. Results of the ALNS heuristic with time related removal heuristic of the Ropke and Pisinger (2006) instances with 50 and 100 requests. The columns and rows should be interpreted as in Table 3. In bold are the objective values that are lower than obtained using ALNS without including the time related removal heuristic.

5.6 Results ALNS with transfer point

In Table 9 and Table 10 the results for the problem including a transfer point are shown. An objective value in bold indicates that it is better than that of the solution obtained using the original ALNS heuristic. For the instances from Li and Lim (2003), in slightly less than 50% of the cases a better solution is obtained. For the instances generated by Ropke and Pisinger (2006) especially the instances containing 50 requests benefit from the possibility of using a transfer point. The objective values that are not bold indicate either a worse solution or the same objective value as for the original ALNS heuristic.
As expected, the run time increased for all instances, where the largest percentage change occurred in the set RC2. This increase in run time is, not surprisingly, strongly related to the number of times that the greedy insertion heuristic with transfer point is used. This heuristic investigates much more insertion possibilities than the basic greedy insertion heuristic.

Table 9: Best results Li & Lim 100, ALNS with transfer point

	R1		R2		C1		C2		RC1		RC2	
	#	f	#	f	#	f	#	f	#	f	#	f
1	22	1806.16	7	**1338.26**	16	1257.95	6	829.46	17	**1863.46**	7	**1596.63**
2	20	1652.72	6	**1401.28**	15	**1252.61**	6	**824.80**	18	**1900.73**	7	**1582.37**
3	16	**1414.03**	6	1172.07	12	**994.02**	6	827.04	13	1483.72	7	1332.31
4	13	**1241.27**	5	1169.67	11	891.47	5	**773.03**	12	**1285.00**	5	**1007.11**
5	18	1622.50	5	**1233.95**	14	1242.07	6	**801.22**	17	**1921.98**	6	1484.86
6	15	1459.28	6	1221.88	14	**1145.89**	6	791.36	15	**1700.47**	6	1451.41
7	15	1410.40	5	1229.68	14	**1199.79**	6	**805.95**	14	1490.41	6	**1514.17**
8	12	**1244.92**	5	**1093.34**	13	**1124.76**	6	822.40	13	1417.93	6	1173.77
9	16	1530.65	6	1159.61	11	985.40						
10	15	**1341.23**	7	1214.78								
11	13	1287.88	4	**1052.71**								
12	13	1239.82										
CPU(s)		204		3544		404		2252		187		2597

Notes. Results of the ALNS heuristic including a transfer point of the Li and Lim (2003) instances with 100 locations. The columns and rows should be interpreted as in Table 3. In bold are the objective values that are lower than obtained using ALNS without including a transfer point.

Table 10: Best results Ropke & Pisinger 50 and 100 requests, ALNS with transfer point

	50 requests		100 requests	
	#	f	#	f
A	13	77024.74	27	160341.86
B	14	85930.44	26	142990.21
C	14	**75428.88**	26	142022.89
D	14	**85638.67**	29	160088.30
E	15	**69277.09**	27	**136079.70**
F	14	**68129.17**	27	137809.43
G	14	66657.12	25	129381.47
H	14	73236.17	28	138422.96
I	13	68247.95	27	**146750.35**
J	14	**83333.95**	29	155663.56
K	14	74823.60	25	131904.08
L	14	**78596.63**	26	**143136.59**
CPU(s)		104		503

Notes. Results of the ALNS heuristic with transfer point of the Ropke and Pisinger (2006) instances with 50 and 100 requests. The columns and rows should be interpreted as in Table 3. In bold are the objective values that are lower than obtained using ALNS without including a transfer point.

Of the 80 instances this adapted heuristic was tested on, only for 4 instances the transfer point is used in the final solution. For most of the other instances a transfer point is only used in the transition phase of the heuristic, but not in the final solution. For the instance LRC101, for example, the greedy insertion heuristic with transfer point is used in 75.9% of the iterations. In 8.5% of the iterations in which this heuristic was used, at least one request was split into two sub-requests.

Table 11 shows the number of requests that are using the transfer point for these 4 instances. Striking is that only for 1 instance this solution including a transfer point has a better objective value than obtained by the original ALNS heuristic. For the LRC205 instance, the transfer point is used for 11 requests, which is over 20% of all requests. The objective value obtained for this instance is however 1.9% higher than when using the original ALNS, but one less vehicle is used. The greedy insertion heuristic with transfer point is used in 80.8% of the iterations, in which on average 8 requests are split. We believe that, due to the many insertion possibilities, it is possible that the ALNS heuristic needs to perform more iterations to give better results for this instance.

Table 11: Instances using transfer point in final solution

	Number of requests	Better solution
LR207	2	false
LR210	5	false
LRC201	2	true
LRC205	11	false

Notes. This table shows the instances for which a transfer point is used in the final solution. The number of requests for which this transfer point is used is reported. The last column indicates if the best solution obtained by the ALNS heuristic using a transfer point is better than the best solution in the original ALNS heuristic.

The solutions of the instance LRC201 are demonstrated in Figure 4. Figure 4a shows the solution obtained with the original ALNS heuristic and Figure 4b shows the solution for the problem with transfer point. Both solutions consist of seven routes, starting and ending at the terminal, denoted by the black dot. The arcs from the terminal to the first customer and from the last customer to the terminal are left out in all routes. In this way the routes are more clearly visible. For demonstration purposes, the routes in which the split requests are in the solution with transfer point, are accentuated. In addition, we highlight the original pickup and delivery locations of these requests $r_i = (i^+, i^-)$ and $r_j = (j^+, j^-)$. We also accentuate their routes in the solution of the problem without transfer point. The transfer point is indicated by a purple square.
Some interesting observations can be made from this figure. The route in which i^+ is in the solution without transfer point, is similar to the route that visits this transfer point. This could be because i^- is close to the transfer point and therefore there are small changes in the best route, which makes the solution with transfer point better. Moreover, the two delivery nodes i^- and j^- are visited by the same vehicle. Hence, the transfer point is only visited once to pick up the goods that have to be delivered at i^- and j^-. The route in Figure 4a, where r_j is in, visits locations that are very dispersed. Because the request is split in the solution which uses a transfer point, this vehicle is able to serve less dispersed locations and j^+ and j^- can be visited separately by vehicles that cover less distant locations.

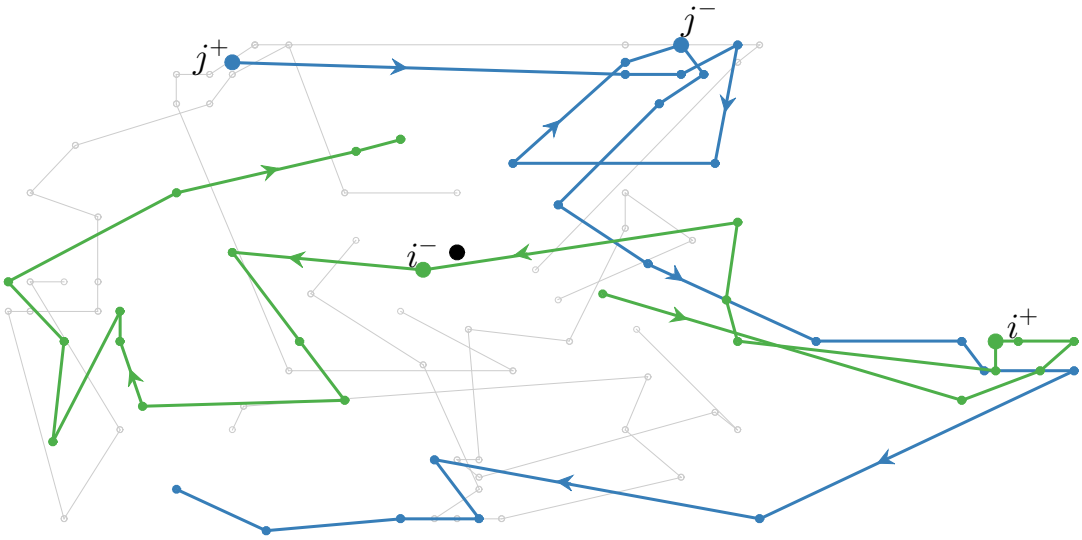


Figure 4 (a) PDPTW

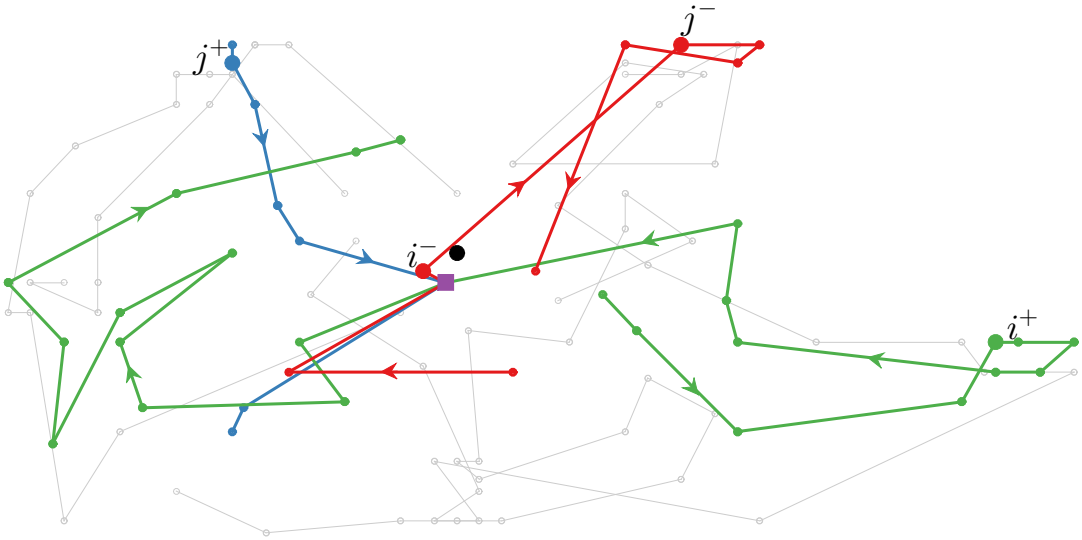


Figure 4 (b) PDPTW with transfer point

Figure 4: Solutions of LRC201

6 Conclusion

This thesis focuses on the ALNS heuristic as proposed by Ropke and Pisinger (2006). In the first part of this thesis the aim was to reproduce the results obtained by Ropke and Pisinger (2006). We implemented the ALNS heuristic and tested it on benchmark instances for the Pickup and Delivery Problem with Time Windows, in which we minimized the total travelled distance and for some instances the total time spent by the vehicles.
The results obtained by this heuristic deviate from those of Ropke and Pisinger (2006). Thus, some changes could be made to improve the results. To obtain parameter settings that are suitable for our implementation, all parameters could be tuned, instead of only two parameters. Furthermore, the heuristic could be run more times to each of the test instances to obtain better solutions. Depending on the objective in the specific application, the number of vehicles could be minimized by using the two-step approach proposed by Ropke and Pisinger (2006). We showed that this method resulted in much fewer vehicles and only slightly larger travelled distances in total for the instances this method was tested on, than when the only objective is to minimize the travelled distance. This two-stage approach does however require the assumption of a homogeneous fleet, which was not the case in some benchmark instances. Moreover, as the initial solution can have an impact on the quality of the solution obtained in the ALNS heuristic, more research could be done in finding good initial solutions.
In the second part of this thesis we tried to improve the solutions obtained by the ALNS heuristic as proposed by Ropke and Pisinger (2006). We included an extra removal heuristic in the ALNS heuristic, which removes requests that are related in terms of time. With this adapted heuristic we obtained for some instances better results than with the original ALNS. Especially for instances with a smaller scheduling horizon and therefore less transportation requests per route.
We also extended the problem by the introduction of a transfer point, so two different vehicles can handle the same request. We therefore adapted the ALNS heuristic. The additional possibilities for the insertion of requests increased run times, but the adapted heuristic was able to improve many solutions compared to the original ALNS heuristic. We saw that mostly in the transition phase of the heuristic a transfer point was used, rather than in the final solution. More extensive analysis could be dedicated to the inclusion of a transfer point in the heuristic, even if this point is not allowed to be actually used in the final solution.
More research could be done on both these extensions of the original ALNS heuristic. The impact on the solution quality of a removal heuristic can be large, so it is definitely worth investigating and tuning good removal heuristics. For the problem with a transfer point, research could be done on the benefits of removing only part of the request (that includes a transfer node) in the removal heuristics, instead of removing all sub-requests. In addition, it is interesting to investigate if better solutions can be obtained when using more than one transfer point. This comes, however, at a price: even larger run times are expected when additional transfer points are introduced. Furthermore, in future research these extensions could be applied to larger test instances.

References

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

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

Cortés, C. E., Matamala, M., & Contardo, C. (2010). The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, *200*(3), 711–724.

Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M. M., & Soumis, F. (2000). *The vrp with pickup and delivery*. Montréal: Groupe d'études et de recherche en analyse des décisions.

Doerner, K. F., & Salazar-González, J.-J. (2014). Pickup-and-delivery problems for people transportation. *Vehicle Routing: Problems, Methods, and Applications*, *18*, 193–198.

Farahani, P., Grunow, M., & Günther, H.-O. (2012). Integrated production and distribution planning for perishable food products. *Flexible Services and Manufacturing Journal*, *24*(1), 28–51.

Landrieu, A., Mati, Y., & Binder, Z. (2001). A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. *Journal of Intelligent Manufacturing*, *12*(5), 497–508.

Lau, H. C., & Liang, Z. (2002). Pickup and delivery with time windows: Algorithms and test case generation. *International Journal on Artificial Intelligence Tools*, *11*(03), 455–472.

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

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

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

Nowak, M., Ergun, Ö., & White III, C. C. (2008). Pickup and delivery with split loads. *Transportation Science*, *42*(1), 32–43.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, *58*(2), 81–117.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, *34*(8), 2403–2435.

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

Şahin, M., Çavuşlar, G., Öncan, T., Şahin, G., & Aksu, D. T. (2013). An efficient heuristic for the multi-vehicle one-to-one pickup and delivery problem with split loads. *Transportation Research Part C: Emerging Technologies*, *27*, 169–188.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *Technical report, APES Group, Department of Computer Science, University of Strathclyde, Glasgow, Scotland*.

SINTEF. (2008). *Li & lim benchmark.* https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/. (Accessed: 2017-05-05)

Van der Bruggen, L., Lenstra, J. K., & Schuur, P. (1993). Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, *27*(3), 298–311.

A Appendix

Table 12: Best results Li & Lim 100 with penalized insertion method

	R1		R2		C1		C2		RC1		RC2	
	#	f	#	f	#	f	#	f	#	f	#	f
1	22	1806.16	6	1312.54	16	1257.95	6	829.46	18	1907.10	6	1557.59
2	20	1652.72	7	1426.13	15	1261.14	6	823.25	17	1897.80	6	1613.95
3	15	1416.69	6	1228.89	11	999.88	6	827.23	12	1343.55	6	1340.25
4	12	1254.77	4	1184.55	11	878.57	5	821.70	12	1342.57	5	1032.92
5	18	1622.50	6	1293.17	14	1191.46	6	793.77	17	1925.07	7	1476.42
6	15	1437.73	4	1199.49	14	1154.90	6	776.24	16	1763.19	7	1456.63
7	15	1391.80	4	1180.44	14	1198.30	6	824.50	14	1488.18	6	1445.29
8	13	1280.33	4	1106.50	13	1130.38	6	817.24	13	1398.21	6	1173.05
9	14	1480.84	6	1139.05	11	951.67						
10	15	1348.67	6	1159.95								
11	13	1294.10	4	1099.08								
12	12	1223.63										
CPU(s)		143		355		143		258		128		249

Notes. Results of the ALNS heuristic of the Li and Lim (2003) instances with 100 locations, using the penalized insertion method to construct the initial solution. The columns and rows should be interpreted as in Table 3.

Table 13: Best results Li & Lim 100 with vehicle minimization algorithm

	R1		R2		C1		C2		RC1		RC2	
	#	f	#	f	#	f	#	f	#	f	#	f
1	22	1806.16	5	1367.05	14	1211.99	5	889.51	16	1863.62	5	1599.14
2	20	1652.71	5	1382.64	13	1225.94	4	874.61	17	1900.93	4	1694.01
3	14	1386.64	3	1144.59	10	1076.95	4	842.34	12	1430.43	4	1401.14
4	12	1253.90	3	1088.25	10	923.66	4	788.33	12	1271.84	3	1013.59
5	17	1588.72	4	1359.42	13	1286.49	4	893.89	16	1860.17	5	1440.48
6	15	1447.24	4	1209.41	12	1096.23	4	832.44	15	1659.62	4	1419.94
7	13	1383.57	3	1293.47	13	1213.78	4	879.95	13	1440.37	5	1400.19
8	12	1280.95	3	1001.65	11	932.54	4	826.41	12	1421.10	4	1179.64
9	15	1492.85	4	1126.37	10	871.62						
10	13	1344.95	3	1199.30								
11	13	1269.50	3	1181.63								
12	12	1256.87										
CPU(s)		167		252		153		221		150		195

Notes. Results of the two-stage vehicle minimization method of the Li and Lim (2003) instances with 100 locations. The columns and rows should be interpreted as in Table 3.

Table 14: Best results Ropke 50 & 100 requests

	ALNS 50 requests		ALNS 100 requests		LNS 50 requests		LNS 100 requests	
	#	f	#	f	#	f	#	f
A	13	76150.56	26	157418.82	13	76825.03	26	160220.87
B	14	84394.66	25	142029.56	14	85507.39	26	144599.70
C	13	75611.96	26	140765.28	13	75743.75	27	142202.36
D	15	87706.31	28	156155.94	15	87249.37	28	157211.51
E	14	69354.13	28	137373.52	14	69771.91	28	139389.04
F	14	68939.01	27	137304.72	14	71148.87	27	140330.53
G	14	66508.54	25	127799.53	14	67712.45	25	131622.48
H	14	71829.14	28	136902.85	14	72908.76	28	139412.88
I	13	68058.81	27	147501.05	13	69002.75	27	149576.13
J	14	83527.31	28	153286.94	15	86189.21	28	157168.46
K	13	73307.26	24	131247.05	13	74443.04	25	132261.40
L	15	79688.26	26	145615.13	14	77806.12	26	145623.70
CPU(s)		67		441		60		503

Notes. Results of the ALNS heuristic and LNS heuristic of the Ropke and Pisinger (2006) instances with 50 and 100 requests. The left column indicates the type of problem. For each instance the number of vehicles and the best solution in the experiment are reported in the columns denoted by # and f respectively. In the last row, denoted by CPU(s), the average run time is reported in seconds.