An Adaptive Large Neighbourhood Search for the Pickup and Delivery Problem with Time Windows

Bachelor Thesis Double Degree Programme Econometrics & Economics

Supervisor:
T.R. Visser MSc

Author:
S.A. Severijn 384431

Second Assessor:
Dr. R. Spliet

July 2, 2017

ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Abstract

In this research the Adaptive Large Neighbourhood Search (ALNS) of Røpke & Pisinger (2006a) is used to solve the Pickup and Delivery Problem with Time Windows (PDPTW), replicating the research of Røpke and Pisinger. ALNS uses a set of removal and insertion procedures to iteratively construct new solutions by deleting part of a high-quality old solution and rebuilding it. It is examined whether the ALNS implementation of this research can obtain similar results for problem instances of the pickup and delivery problem with time-windows constructed by Li & Lim (2003) and Røpke & Pisinger (2006a) and how the method might be improved. It was found that although the algorithm seemed to perform somewhat poorer than in the research of Røpke and Pisinger. A variant of the problem with time-dependent travel times is introduced, to examine whether this problem can also be solved efficiently by ALNS. Our finding is that time-dependent pickup and delivery problems with time windows can be solved successfully by ALNS, although the running times might increase substantially if the travel time distribution is complicated. Moreover, it was found that increases in the number of iterations might substantially increase the quality of the solutions found. A potential way to increase the performance of ALNS further, is using more sophisticated insertion procedures.

Contents

1 Introduction

Vehicle routing problems are heavily researched problems within the field of Operations Research, with high societal relevance. In vehicle routing problems, optimal delivery routes need to be designed to serve a set of customers, starting and ending at a depot, while adhering to some side constraints. Much literature exists on a wide range of solution techniques for many different variants of the Vehicle Routing Problem (VRP). One influential paper by Røpke & Pisinger (2006a), introduced a solution method which was later used to solve numerous problems within the class of vehicle routing problems.

The core of this research is the replication of the findings of Røpke and Pisinger. In their paper, they introduced Adaptive Large Neighbourhood Search (ALNS) and implemented this method for the Pickup and Delivery Problem with Time Windows (PDPTW). In this problem, a set of vehicles needs to pick up a number of goods at different locations and deliver those goods at another set of locations. There is a set of begin and end terminals and vehicles have limited capacity and may be restricted in which goods they can handle. It is allowed to leave some requests unrouted, although this imposes a penalty. The objective of the problem is to find a solution which minimizes the weighted sum of (i) the total distance travelled, (ii) total time spent by all vehicles, (iii) the number of unhandled requests.

The aim of this research is to examine how reliable and how suitable ALNS is for the PDPTW. Part of this, will be done by investigating how ALNS compares to the Large Neighbourhood Search (LNS) framework, which ALNS was built upon. Apart from replicating the results of Røpke and Pisinger, a variant of the PDPTW is considered, where travel times are dependent on the time of day. This models the reality where vehicles may experience congestion during peak hours more accurately. It will be examined whether the Adaptive Large Neighbourhood Search framework can be used successfully on the time-dependent PDPTW.

Little research has been done on Vehicle Routing Problems with time-dependent travel times. To our knowledge, the PDPTW with time-dependent travel times as described before, has only been discussed in the literature in a dynamic setting where the set of customers to serve is not fully available at the start of the time window, but not in the static setting as in this research. In contrast, the PDPTW has been discussed in numerous publications. Research on other time-dependent VRPs has also been relatively scarce.

Since in the years following the first publication of the ALNS procedure, the speed of computers has increased, less time may be needed to follow the same procedures. It is investigated how the saved time might be used to increase the performance of the metaheuristic, by investigating which aspects of the algorithm are best to spend extra time on.

This bachelor thesis is structured as follows: in §2 previous literature on vehicle routing problems, especially PDPTW problems, and the ALNS method will be discussed, in §3 a formal problem description for the PDPTW is given, in §4 the ALNS method will be described, in §5 the computational experiments will be elaborated upon, in §6 the results will be discussed and finally in §7 implications of this research be discussed.

2 Literature Review

The problem of interest, the PDPTW, is a special case of the wider class of vehicle routing problems. The vehicle routing problem was introduced as the truck dispatching problem by Dantzig & Ramser (1959). It is a generalization of the travelling salesman problem, a problem where the shortest tour of a set of locations needs to be found. There is no consensus in the field on the exact definition of the vehicle routing problem, but in essence, in a VRP a set of routes between different locations needs to be made and assigned to vehicles, with a limited capacity (Eksioglu et al., 2009). This has to be done while optimizing a certain objective function. In the original truck dispatching problem, there is one depot from which all routes need to depart and arrive and a set of geographical locations that have to be visited once by exactly one vehicle. There are travel costs associated with travelling from one location to another and these should be minimized.

There are numerous variants of this problem where one or more side constraints are added or generalizations are made. Eksioglu et al. (2009) made an overview of the scope of the class of vehicle routing

problems. In their taxonomy, they differentiate between physical characteristics, scenario characteristics and information characteristics. Physical characteristics include amongst others the number of depots, homogeneity or heterogeneity of vehicles, the type of distribution of the customers (concentrated or scattered) and the nature of the travel time (deterministic, time-dependent or stochastic).

Examples of scenario characteristics are the nature (deterministic or stochastic) of service times and demand quantities, and the nature of the time windows, e.g. hard time windows, which are time windows that have to be met, and soft time windows, time windows that impose a penalty in the objective function if violated. Information characteristics includes the certainty or uncertainty of information. Another characteristic is whether the information of the problem is static or evolves dynamically.

The objective function of a VRP is often a cost function, which may include the fixed costs of the number of vehicles needed and the variable costs proportional to the distance or time travelled. However, it may also be a service quality function, or a combination.

2.1 Solution techniques

For the varied set of vehicle routing problems, different solution techniques have been tried. Since the problems are NP-hard, it is often necessary to rely on heuristics, although solution techniques have been found that can solve reasonably large problem instances exactly. The most promising exact solution methods were based on branch-and-bound procedures, using, amongst others, Langrangian relaxation to obtain lower bounds (Toth & Vigo, 2002). The focus in this thesis will be on heuristic methods that can also obtain solutions for instances of larger sizes that cannot be solved exactly within a reasonable amount of time. Heuristics needed to solve these problems can be categorized as follows (Vidal et al., 2013):

- Constructive heuristics: these heuristics build routes from scratch in a greedy manner. Once a route is built, it is not changed anymore. The earliest heuristics in the field were often of this category. Moreover, constructive heuristics are still used as subheuristics in more recent algorithms. Raff (1983) have defined several subcategories of constructive heuristics:
  - Cluster first-route second procedures: here the locations to be visited are first clustered, often based on proximity. Afterwards routes are created in each cluster. As these subproblems are generally small, the second step might be done optimally.
  - Route first-cluster second procedures: in these types of procedures first one giant route is assembled consisting of all locations. Afterwards that route is split, in order to obtain a feasible solution. The assembling of the route can be modelled as a travelling salesman problem.
  - Savings or insertion procedures: in these types of procedures, routes are iteratively built by swapping or inserting locations which yield the largest saving or lowest cost increase, according to a cost function.
- Local improvement methods: these algorithms use neighbourhoods of a solution to move from one solution to a potentially better solution near the original solution. A neighbourhood consists of the set of solutions that can be obtained by performing a certain set of operations on the solution for which the neighbourhood is defined. Within this class, several approaches can be taken. Neighbourhoods may consist of exchanges of nodes within routes. In other heuristics, the ruin-and-recreate principal is applied, which means that a part of the solution is deleted and subsequently constructed towards a new solution.
- Metaheuristics: metaheuristics can be defined as 'solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space.' (Glover & Kochenberger, 2006) Two prominent metaheuristic categories for vehicle routing problems are:
  - Population-based methods: population-based algorithms generate new solutions from a set of existing algorithms, the population. The population updates gradually, such that the quality of the population increases over time.
  - Neighbourhood-centred methods: these methods iteratively search through the neighbourhood of a single solution at the time. An example of a metaheuristic approach is simulated annealing.

This approach mandates that the metaheuristic continues to the next solution with a certain probability, controlled by a variable named temperature. The temperature decreases over time, and, ceteris paribus, a lower temperature means a higher probability to continue to the next solution. Another important approach is tabu search. It is based on hill-climbing, that is, iteratively searching the neighbourhood of a solution towards a better solution. During the search, tabu conditions are imposed during the search to filter the neighbourhood such that the algorithm avoids visiting the same local optima multiple times. Moreover, worsening moves can be allowed, such that the search will not remain in a local optimum.

Although some heuristics may be applicable to multiple variants of the vehicle routing problem, several tackle specific variants (Pisinger & Røpke, 2007). For the problem category of interest, the PDPTW, most early research was focused on the Dial-and-Ride problem, which is a special case of the PDPTW, where user inconvenience is incorporated in the problem. The first research mainly designed insertion algorithms and cluster first-route second heuristics. The insertion algorithm of Jaw et al. (1986) is a prominent example, which was able to provide solutions for problem instances with more than thousand requests.

An exact algorithm for the PDPTW being able to handle moderately large problem instances was found by Dumas et al. (1991) using a dynamic programming approach. However, their algorithm was only suitable for some variants of the PDPTW and only if the capacity constraints are restrictive to the problem. Dial-and-ride problems were not solvable by their algorithm.

Towards the 21st century, the first metaheuristics for the PDPTW are described, which are generally easier to adapt to more problem variants. Gendreau et al. (1998) described a tabu search method for the case where transportation requests come in dynamically. In subsequent years, several other tabu search-based metaheuristics were described in the literature, of which the algorithm of Li & Lim (2003) is a notable example. Moreover, they created problem instances for the PDPTW, that are from that time onwards often used in the literature.

Bent & Van Hentenryck (2004) proposed another local search metaheuristic, Large Neighbourhood Search (LNS), which has shown to be able to match or improve upon a large share of found solutions of previously published heuristics.

2.2 Adaptive Large Neighbourhood Search

The algorithm to be replicated in this paper is described as the Adaptive Large Neighbourhood Search (ALNS) method. It is adaptive in the sense that during the execution of the algorithm it is determined which sub-methods perform best and are used often in the search. An extensive description of the method will be given in §4.

The ALNS metaheuristic is based upon the LNS approach, introduced by Shaw (1998). As the name suggest, large neighbourhood search is a neighbourhood-centred metaheuristic. It uses the ruin-and-recreate principle to search for new solutions. Bent & Van Hentenryck (2004) amended the method of Shaw for the PDPTW. In their paper, they introduced a two-stage method, where in the first stage they try to minimize the number of vehicles using simulated annealing and in the second stage try to minimize the total distance using the large neighbourhood search. Røpke & Pisinger (2006a) improve on LNS, by introducing a mechanism which dynamically chooses methods for insertion and deletion based on previous performance.

Although the ALNS algorithm was first designed for the PDPTW, it was later implemented for various other problems, mostly other vehicle routing problems. Shortly after the first description of the ALNS algorithm, Pisinger & Røpke (2007) generalized the heuristic to a more general Vehicle Routing Problem, which they named the rich pickup and delivery model. Several variants of the VRP were then transformed to this rich pickup and delivery problem, under which the Vehicle Routing Problem with Time Windows. Røpke & Pisinger (2006b) also implemented vehicle routing problems with backhauls by transforming the problems to the rich pickup and delivery problem.

Masson et al. (2013) used the same approach to solve a pickup and delivery problem with transfer points. Ribeiro & Laporte (2012) have implemented the ALNS metaheuristic in a VRP setting where the objective is to minimize the arrival times of customers. Azi et al. (2014) have used the metaheuristic

to implement a VRP-variant where vehicles may serve multiple trips, relevant in industries with perishable goods where routes need to be short. Demir et al. (2012) have used the ALNS method to the pollution-routing problem, in which routes and driving speed need to be optimized in order to minimize a combination of pollution and costs.

ALNS was implemented for the two-echelon vehicle routing problem, a VRP where goods have to pass through one of the intermediary facilities, by Hemmelmayr et al. (2012). A variation on this problem where vehicles are allowed to make multiple tours was implemented by Grangier et al. (2016).

The algorithm has also been implemented into other VRP contexts outside the scope of goods deliveries by numerous researchers. Salazar-Aguilar et al. (2011) have implemented the ALNS algorithm for a synchronised arc routing problem, which has applications for snow plowing. B. Li et al. (2016) have implemented the metaheuristic for the share-a-ride problem, where packages are delivered by taxi drivers if this does not cause much hinder to passengers. Kovacs et al. (2012) have used ALNS in a context where routes had to be made for maintenance tasks and technicians had to be teamed and scheduled.

Adaptive Large Neighbourhood Search has also shown to be a valuable solving method for problems involving more decisions than routing and scheduling. Coelho et al. (2012) have developed an ALNS algorithm for the inventory routing problem, which is an extension of the vehicle routing problem where customers need to be supplied when their inventory runs out. Aksen et al. (2014) implemented the ALNS algorithm for a variant of the inventory routing problem where there is selection possible of customers. Adulyasak et al. (2012) have used ALNS to solve the production routing problem, an extension of the inventory routing problem where also the production schedule is incorporated.

In some cases, Adaptive Large Neighbourhood Search implementations have also been designed for applications outside the scope of Vehicle Routing Problems. Sørensen et al. (2012) have introduced the metaheuristic into the field of timetabling, specifically the case of high school timetabling. Related to this, Kristiansen & Stidsen (2012) have used ALNS to distribute students efficiently and evenly into classes. Other fields where the ALNS metaheuristic has been implemented include lot-sizing (Muller et al., 2012), scheduling (Muller, 2009) and determining the optimal location of a probability distribution in the field of statistics (Katterbauer et al., 2012).

2.3 Time-dependent Vehicle Routing Problem

A realistic, but relatively scarcely studied generalization of the vehicle routing problem is the time-dependent VRP, which is a physical characteristic using the classification of Eksioglu et al. (2009). In this variant, travel times for a specific trajectory differ depending on the departure time of a vehicle to model, for example, rush hours and off peak traffic times. Malandraki & Daskin (1992) give an early description of the time-dependent VRP, where the travel times are step-wise functions of the starting time. Allowing vehicles to wait at nodes before departure, the effective travel time functions are piecewise linear. Ichoua et al. (2003) introduce a more realistic model, where the travel speed is a stepwise functions of time, leading to piecewise linear travel time functions, which satisfy the First In, First Out (FIFO) principle: if a vehicle departures earlier, it also arrives earlier.

A large share of the literature on time-dependent vehicle routing problems generalize the problem to a dynamic setting, where some requests may only be known during the execution of the route. Examples are Haghani & Jung (2005), using a genetic, population-based algorithm and Pureza & Laporte (2008), discussing the PDPTW. Their problem description and solution method allows for any continuous travel time function adhering to the FIFO principle.

Solution methods for the time-dependent VRP have been mostly metaheuristics, for example the population-based ant colony method by Donati et al. (2008) and a neighbourhood-based search algorithm by Hashimoto et al. (2008). Hitherto, the large neighbourhood search or adaptive large neighbourhood search metaheuristic has been applied few times to any variant of the time-dependent vehicle routing problem. A notable exception is the research by Taş et al. (2014), which used both ALNS and Tabu Search to the VRP with stochastic and time-dependent travel times and soft time windows.

3 Problem description

The pickup and delivery problem with time windows (PDPTW) consists of a set of vehicles with limited capacity that ought to handle a subset of requests within predetermined time intervals. Each vehicle has a start terminal, from which it departs and an end terminal where it should arrive once its route has been completed. Each request consists of a pickup location and a delivery location. The objective of the problem is to minimize a weighted sum of (i) the total distance travelled, (ii) total time spent outside the terminals by the vehicles, (iii) the number of unhandled requests. Both the scenario where the travel times remain constant, as the scenario where the travel times are dependent on the time of the day are considered.

The problem consists of n delivery requests. Each request i is associated with pickup node $i \in \mathcal{P}$ and delivery node $i + n \in \mathcal{D}$. The set $\mathcal{P}$ consists of all the pickup nodes in the problem, the set $\mathcal{D}$ of all the delivery nodes. The set of vehicles in the problem is denoted by $\mathcal{K}$. There are m vehicles, $|\mathcal{K}| = m$, and each vehicle k is associated with a start terminal $\tau_k = 2n + k \in \mathcal{T}$ and an end terminal $\tau'_k = 2n+m+k \in \mathcal{T}'$. The set $\mathcal{T}$ denotes the start terminals in the problem and $\mathcal{T}'$ the set of end terminals.

Using above notation, the PDPTW can be described using an undirected graph $G = (\mathcal{V}, \mathcal{A})$. The set of vertices in the graph is $\mathcal{V} = \mathcal{P} \cup \mathcal{D} \cup \mathcal{T} \cup \mathcal{T}'$. The set of arcs in the graph is $\mathcal{A} = \mathcal{V} \times \mathcal{V}$, the Cartesian product of $\mathcal{V}$.

Each vehicle is able to serve a subset of the nodes in graph G: $\mathcal{P}_k \cup \mathcal{D}_k \subseteq \mathcal{P} \cup \mathcal{D}$. The associated subgraph $G_k = (\mathcal{V}_k, \mathcal{A}_k)$, is the set of nodes visitable by vehicle k, $\mathcal{V}_k = \mathcal{P}_k \cup \mathcal{D}_k \cup \tau_k \cup \tau'_k$ and the Cartesian product of $\mathcal{V}_k$: $\mathcal{A}_k = \mathcal{V}_k \times \mathcal{V}_k$, denoting the arcs vehicle k may travel.

Each arc (i, j) is associated with a distance d_{ij} and a travel time t_{ij}. It is assumed every distance and travel time d_{ij} and t_{ij} is nonnegative and that the triangle inequality holds for each triplet of arcs $t_{ij} + t_{jh} \geq t_{ih}$. This is necessary to avoid subtours. Each node j in the graph has a service time s_j associated to it and a time window $[a_j, b_j]$ in which the node may be visited. Each node also has an associated change in load l_j. For pickup nodes l_j is positive, for delivery nodes negative and for terminals l_j equals zero.

Each request i has a subset of vehicles $\mathcal{K}_i \subseteq \mathcal{K}$, that are able to handle request i. If a vehicle k can handle a request i, the pickup and delivery node of that request must be reachable for that vehicle: $k \in \mathcal{K}_i \iff i \in \mathcal{V}_k \wedge n + i \in \mathcal{V}_k$. Each vehicle k has a maximum capacity of c_k.

The decision variables for the problem are the binary variables x_{ijk}, indicating whether arc (i, j) is traversed by vehicle k, and the continuous variables S_{ik}, indicating the time when vehicle k visits location i. The latter is only well defined if vehicle k visits location i. There are two other types of variables used in the mathematical formulation following from these decision variables. L_{ik} are discrete variables denoting the load of vehicle k after visiting location i. As before, the variables are only defined if vehicle k visits location i. Lastly, the binary variable z_i denotes whether request i is unhandled. It assumes the value 1 if it is unhandled and 0 if it is handled.

Above notation is used to define the problem formally. The formulation largely follows the formulation and notation by Røpke & Pisinger (2006a). It is modified into a linear integer form and in such a manner that L_{ik} follows directly from x_{ijk} and S_{ik}.

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

Subject to:

$$\sum_{k \in \mathcal{K}_i} \sum_{j \in \mathcal{P}_k \cup \mathcal{D}_k} x_{ijk} + z_i = 1 \quad \forall i \in \mathcal{P}, \tag{2}$$

$$\sum_{j \in \mathcal{V}_k} x_{ijk} - \sum_{j \in \mathcal{V}_k} x_{j,n+i,k} = 0 \quad \forall k \in \mathcal{K}, i \in \mathcal{P}_k, \tag{3}$$

$$\sum_{i \in \mathcal{P}_k \cup \{\tau'_k\}} x_{\tau_k,i,k} = 1 \quad \forall k \in \mathcal{K}, \tag{4}$$

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

$$\sum_{i \in \mathcal{V}_k} x_{ijk} - \sum_{i \in \mathcal{V}_k} x_{jik} = 0 \quad \forall k \in \mathcal{K}, j \in \mathcal{P}_k \cup \mathcal{D}_k, \tag{6}$$

$$S_{ik} + s_i + t_{ij} - S_{jk} \leq \mathrm{M}_1(1 - x_{ijk}) \quad \forall k \in \mathcal{K}, (i,j) \in \mathcal{A}_k, \tag{7}$$

$$a_i \leq S_{ik} \quad \forall k \in \mathcal{K}, i \in \mathcal{V}_k, \tag{8}$$

$$S_{ik} \leq b_i \quad \forall k \in \mathcal{K}, i \in \mathcal{V}_k, \tag{9}$$

$$S_{ik} \leq S_{n+1,k} \quad \forall k \in \mathcal{K}, i \in \mathcal{P}_k, \tag{10}$$

$$L_{jk} - l_j - L_{ik} \geq \mathrm{M}_2^k(x_{ijk} - 1) \quad \forall k \in \mathcal{K}, (i,j) \in \mathcal{A}_k, \tag{11}$$

$$L_{jk} - l_j - L_{ik} \leq \mathrm{M}_2^k(1 - x_{ijk}) \quad \forall k \in \mathcal{K}, (i,j) \in \mathcal{A}_k, \tag{12}$$

$$L_{ik} \leq c_k \quad \forall k \in \mathcal{K}, i \in \mathcal{V}_k, \tag{13}$$

$$L_{\tau_k k} = 0 \quad \forall k \in \mathcal{K}, \tag{14}$$

$$L_{\tau_k k'} = 0 \quad \forall k \in \mathcal{K}, \tag{15}$$

$$x_{ijk} \in \mathbb{B} \quad \forall k \in \mathcal{K}, (i,j) \in \mathcal{A}_k, \tag{16}$$

$$z_i \in \mathbb{B} \quad \forall i \in \mathcal{P}, \tag{17}$$

$$L_{ik} \in \mathbb{N}_0 \quad \forall k \in \mathcal{K}, i \in \mathcal{V}_k, \tag{18}$$

$$S_{ik} \in \mathbb{R} \quad \forall k \in \mathcal{K}, i \in \mathcal{V}_k. \tag{19}$$

In the above formulation, the objective function (1) mandates that the weighted sum of the total travelled distance, the total time spent and the number of requests not accepted should be minimized. The coefficients α, β and γ denote the weights in the objective function for the distance, time travelled and unhandled request respectively.

Constraints (2) ensure that a request is put in the request bank if there is no vehicle leaving the corresponding pickup node. Constraints (3) denote that if a vehicle visits pickup node i, it should also visit the corresponding delivery node $n + i$.

Constraints (4) and (5) mandate that each vehicle should leave its begin terminal, respectively enter its end terminal. Constraints (6) impose that if a vehicle reaches a non-terminal node, it should also leave that node.

Constraints (7) ensure that for each vehicle k, the start of the service at location j is at the earliest the sum of the start of the service at location i, the service time at location i and the travel time from location i to j if location j is visited immediately after location i by vehicle k. M_1 denotes a large number, which can be set to the maximum of all the b_i.

Constraints (8) and (9) impose that the start of the service at all locations should be within their time windows. Constraints (10) require that if delivery i is done by vehicle k, it services the corresponding pickup node before the delivery node.

Constraints (11) and (12) ensure that the load of vehicle k after visiting location j equals the sum

of the load after visiting location i and the change in load at location j, if location j is visited immediately after location i by vehicle k. M_2^k is relevant in the case location j is visited by vehicle k at another point in time. For the equation to hold in these cases, M_2^k can be set to the capacity c_k of vehicle k.

Constraints (13) indicate that the load of each vehicle should always be smaller or equal than its capacity. Constraints (14) and (15) denote that each vehicle should start respectively end its tour empty.

Constraints (16) to (19) define the domain of the variables.

3.1 Time-dependent travel times

The extension of time-dependent travel times has small effects on the formulation previously described. The travel time from location i to location j with a departure time δ is denoted by $t_{ij}(\delta)$. An assumption on the travel time distribution is the First In, First Out principle, which can be formulated as follows:

$$\delta_1 \leq \delta_2 \iff \delta_1 + t_{ij}(\delta_1) \leq \delta_2 + t_{ij}(\delta_2) \quad \forall (i,j) \in \mathcal{A}, \delta_1, \delta_2 \geq 0. \tag{20}$$

The formal problem description is generalized to time-dependent travel times in Equation (7) by replacing t_{ij} by $t_{ij}(S_{ik} + s_i)$.

In the most general case, this generalization causes the triangle inequality to not hold anymore. However, if the travel time distributions of the problem are such that the travel times only depend on the starting time and the distance, the triangle inequality continues to hold, provided that the FIFO property holds for the travel time distributions. The former can be mathematically described as follows:

$$t_{ij}(\delta) = t(d_{ij}, \delta), \quad \forall (i,j) \in \mathcal{A}, \delta \geq 0. \tag{21}$$

Here $t(.)$ denotes the location-independent travel time distribution. This implies that each arcs has the same travel speed as the other arcs for each moment in time, while the travel speeds are allowed to vary over time, for the triangle inequality to hold.

In this research, the travel time distributions are based on one or more stepwise speed distributions, introduced by Ichoua et al. (2003), which all have their changes in travel speeds at the same points in time. If there is one speed distribution that holds for all the arcs $(i,j) \in \mathcal{A}$, the triangle inequality holds. If the stepwise distribution consists of P different segments, or steps, the travel time distribution constructed from it is piece-wise linear and contains at most $2(P-1)$ breakpoints. Each of the $P-1$ changes in travel speed are preceded with a time period where the change has no effect, since the travel finished before the change in travel speed occurred, followed by a time period where the change in travel speed occurs during the travel, which in its turn is followed by a period where the change in travel speed occurs before the travel. Therefore, each change in travel speed may induce two breakpoints in the travel time distribution.

If there are more stepwise speed distributions, the triangle inequality does not hold. Each arc can be divided in k different segments where each segment has its own travel speed distribution with P different steps in each travel speed distribution changing at the same time. This leads to a piece-wise linear travel time distribution with a maximum of $(2k)(P-1)$ breakpoints. As before, there are $P-1$ changes in travel speed for each distribution. The starting time of the whole arc for which the change in travel speeds starts taking effect differs for each of the k arc segments, since they may be preceded by other arc segments. Moreover, the point in time when the change in travel speed happens before the arc segment is traversed, also may differ for each arc segment. Therefore each arc segment may induce 2 breakpoints in the travel time distribution of an arc for each change in travel speed.

4 Methodology

The ALNS metaheuristic of Røpke & Pisinger (2006a) is used to solve the PDPTW in this research. The main idea of this metaheuristic method is that new solutions are created by rebuilding a relatively large part of the old solution, by means of removals and insertions of requests.

The algorithm is shown in Algorithm 1. It starts with an initial solution. In each iteration, procedures are chosen to do the removal and insertion in that iteration, based on previous performance of the removal and insertion procedures. Afterwards, a new solution is generated by deleting a portion of the request of the most recent accepted solution and reinserting the deleted requests to a feasible solution. A solution is accepted with a probability based on the difference in objective value with the last accepted solution. Based on the costs of the newly generated solution, scores of the procedures are increased, which in turn influence the weights of the procedures.

Algorithm 1 ALNS heuristic

Require: s is an initial solution, Φ is the number of iterations the heuristic will undertake, Ψ is the set of possible procedure combinations, w_{init} are the initial weights for choosing the procedures, θ is the number of iterations after which the weights w are updated

$s_{\text{best}} \leftarrow s$
iter $\leftarrow 0$
$w \leftarrow w_{\text{init}}$
while iter $< \Phi$ **do**
  $s' \leftarrow s$
  choose procedures $\psi \in \Psi$ based on w
  generate q
  delete q requests from s' using the procedures in ψ
  reinsert the q requests into s' using the procedures in ψ
  if $f(s') < f(s_{\text{best}})$ **then**
    $s_{\text{best}} \leftarrow s'$
  end if
  if accept s' **then**
    $s \leftarrow s'$
  end if
  update statistics
  if (iter modulo θ) $== 0$ **then**
    update w
  end if
  iter $\leftarrow$ iter $+ 1$
end while
return s_{best}

There is a range of removal and insertion procedures used in the ALNS method. ALNS is adaptive in the sense that the probability of using a specific removal and insertion procedure changes based on its previous performance in the same run of the ALNS method. After each iteration, statistics on the performance of the procedures are updated. After each predetermined interval of iterations, weights of the procedures are modified according to these statistics. The initial solution is obtained using the insertion procedures to build a new solution from scratch.

In the rest of this section, abovementioned steps of the ALNS metaheuristic for the pick-up and delivery problem with time windows as implemented in this research and in the research by Røpke & Pisinger (2006a) will be discussed in detail.

4.1 Removal procedures

All removal procedures choose q requests to remove out of the most recent accepted solution. The discrete number q is drawn randomly out of an uniform distribution: $q \sim \text{Unif}(4, \min(100, \xi n))$. The parameter ξ is the maximum fraction of requests that can be removed and n is the number of requests, as before. There are three different removal procedures, which will be discussed in the following subsections.

4.1.1 Shaw removal procedure

The Shaw removal procedure was initially proposed by Shaw (1997). The idea is that it is easier to shift around requests that are relatively alike, than requests that are not. Therefore, the procedure randomly chooses a request and removes another request with a probability based on the similarity to the chosen request. The relatedness is based on four components: distance, time, capacity and the set of vehicles that can handle the requests. The relatedness $R(i, j)$ between request i and request j is calculated using the following formula:

$$R(i,j) = \sigma_1\big(d_{A(i),A(j)} + d_{B(i),B(j)}\big) + \sigma_2\big(|T_{A(i)} - T_{A(j)}| + |T_{B(i)} - T_{B(j)}|\big) + \sigma_3|l_i - l_j| + \sigma_4\left(1 - \frac{|\mathcal{K}_i \cap \mathcal{K}_j|}{\min(|\mathcal{K}_i|, |\mathcal{K}_j|)}\right).$$

In this formula, $A(i)$ denotes the pickup location for request i, $B(i)$ denotes the delivery location for request i and T_i indicates the time node i is visited. The variables d_{ij} (distance) and l_i (load after a visit) and the sets $\mathcal{K}_i$ (set of vehicles that can handle request i) have been defined in §3. The σ-values are weights assigned to the different terms. In order to indeed have the correct weights between the relatedness components, the distances used in this procedure are standardized such that the largest distance in the problem has a value of 1. Similarly, the loads and visit times are standardized to the [0,1]-interval.

The pseudocode for the Shaw removal procedure is shown in Algorithm 2. In each iteration of the heuristic a request that will be deleted is chosen for which the relatedness values with other requests is calculated. Due to the randomness parameter p_S, generally larger than 1, requests that are relatively alike to the chosen request will be removed with a higher probability. This causes the set of requests to be removed to be often relatively homogeneous compared to the whole set of requests.

If a request was not handled in the input solution, the relatedness value is set to infinity. Since the order of the relatedness matters, all scheduled requests will have a higher probability to be selected than unscheduled requests. After the removal procedure, there are two types of unscheduled requests: (i) unscheduled requests that have been removed from the previous solution and should be reinserted and (ii) requests that were unscheduled in the previous solution and remain unscheduled in the next solution. A removal of a request that was not handled in the input solution, causes the request to transfer from the latter to the former category.

Algorithm 2 Shaw removal heuristic

Require: a solution s, the number of requests to be deleted q, a randomness parameter p_S
1: $r \leftarrow$ random request from s
2: $\mathcal{D} \leftarrow \{r\}$
3: **while** $|\mathcal{D}| < q$ **do**
4: $\quad r \leftarrow$ random request from $\mathcal{D}$
5: $\quad L \leftarrow$ array of requests in s and not in $\mathcal{D}$
6: $\quad y \leftarrow$ random number in [0,1)
7: $\quad x \leftarrow \lfloor y^{p_S}|L| \rfloor$
8: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup$ {request in L with x-th lowest relatedness value}
9: **end while**
10: remove all requests in $\mathcal{D}$ from their routes in s
11: **return** s

4.1.2 Random removal procedure

In the random removal procedure, each request is deleted with equal probability. This procedure can be seen as a special case of the Shaw removal procedure, with $p_S = 1$.

4.1.3 Worst removal procedure

The worst removal procedure uses the marginal costs of the request as the criterion for removal. The marginal costs of request r are defined as the difference in costs of the solution with request r and the solution where request r is not present. The latter excludes the penalty from unscheduled requests, γ, for that request r. Apart from this different criterion, the procedure is relatively similar to the Shaw removal heuristic, as can be seen in Algorithm 3. As before, the randomness parameter, here p_W, causes requests that are well suitable to be removed according to the criterion used, here requests with high marginal costs, to be removed with higher probability.

Algorithm 3 worst removal heuristic

Require: a solution s, the number of requests to be deleted q, a randomness parameter p_W

```
i ← 0
while i < q do
    L ← array of requests in s and not in 𝒟
    y ← random number in [0,1)
    x ← ⌊y^{p_W} |L|⌋
    remove request in L with x-th lowest marginal cost from its route in s
    i ← i + 1
end while
return s
```

4.2 Insertion procedures

For all insertion procedures, the decision which request to insert first is based on the marginal costs of insertion. The set of procedures consist of a greedy and a number of regret procedures. For the simplest procedure, the greedy insertion, the marginal costs of the cheapest feasible insertion positions are calculated for each route, for each unscheduled request. The insertion is done for the request and the route where the insertion can be done in the cheapest manner. Let the marginal costs of the cheapest insertion of request r into route i be denoted as $g(r, i)$. Then, the greedy procedure chooses request r to be inserted in route i by the formula:

$$\arg\min_{r,i} g(r, i) \tag{22}$$

The greedy procedure has drawbacks, since it prefers easy insertable request before requests that are more difficult to insert and thus more expensive. The regret procedure circumvents this problem, by choosing to insert the the request with the largest regret value: the difference in cost between the cheapest place of insertion in the cheapest route and the cheapest place of insertion in the second cheapest route. This regret procedure chooses to insert the request with the biggest regret value into the cheapest route. Let the variable y_{rj} denote the j-th cheapest route to insert request r in. Then the regret procedure chooses to insert request r in route y_{r1} by the formula

$$\arg\max_{r} g(r, y_{r2}) - g(r, y_{r1}) \tag{23}$$

This procedure can also be generalized: instead of looking solemnly at the difference between the best and second best route, the general regret-k procedure looks at the sum of the differences between the best and the second best until the k-best route:

$$\arg\max_{r} \sum_{j=2}^{k} g(r, y_{rj}) - g(r, y_{r1}) \tag{24}$$

It may occur that some requests cannot be inserted into some routes. In this case, $g(.)$ is set to positive infinity. This causes the regret-k algorithm to insert requests that do not have k routes where the request can be inserted feasibly, to be inserted first. If there are multiple request that cannot be inserted in at least k routes, the request with the lowest amount of insertable routes is inserted first.

For the greedy as well as the regret-k algorithm, the chosen request is inserted into the route leading to the least additional costs. In the case these costs are higher than γ, the penalty for unscheduled

requests, the request is not inserted. The same applies if there is not a route available where the request can be inserted in.

To add a component of randomness into the insertion procedures, two variants of the insertion procedures exist: one where the actual costs of insertion are used and one where the costs of insertion are incremented with a noise term. This noise term is a random number in the interval $[-\eta \max_{(i,j)\in\mathcal{A}}\{d_{ij}\}, \eta \max_{(i,j)\in\mathcal{A}}\{d_{ij}\}]$. Here η denotes a randomness parameter, which is multiplied with the maximum arc distance, using the notation as in §3. The sum of the costs of insertion and the noise term cannot be smaller than zero.

All the insertion procedures build routes in parallel. That is, requests may be added to any route at any time. This in contrast to sequential procedures that build a solution route by route.

4.3 Adaptiveness

The adaptiveness of the adaptive large neighbourhood search metaheuristic comes from different weights used for choosing removal and insertion procedures, that change during the execution of the algorithm. These weights depend on statistics of the performance of the different procedures described. Before each iteration, the removal procedure, the insertion procedure and the decision whether to use noise in the objective, from here on called the noise procedure, are randomly determined based on weights that change during the process of the ALNS. The weights are adjusted after each segment, based on the statistics collected in that segment. A segment is a group of iterations of fixed and constant size.

For each procedure, scores are kept. After each iteration the solution resulting of that iteration is assigned a score value. That value is added to the scores of the procedures used in that iteration: one of the three removal procedures, one of the insertion procedures, and the noise procedure. Each solution is scored according to the score parameters ρ_1, ρ_2 and ρ_3 as follows:

- ρ_1: if the solution is better than the previously found best solution,
- ρ_2: if the solution has not been found before and is better than the last previously accepted solution,
- ρ_3: if the solution has not been found before, is worse than the last previously accepted solution, but has been accepted.

If none of these cases apply, the solution gets a score of zero. Only new solutions are awarded scores in order to reward diversification. Hashtables are used to check whether a certain solution has found before.

After each segment of iterations, the scores are used to adjust the weights by the following formula:

$$w_{i,j+1} = w_{ij}(1-r) + r\frac{\pi_i}{\Theta_i} \tag{25}$$

In the above equation, w_{ij} is the score of removal, insertion or noise procedure i in segment j. The parameter r is the reaction factor, which is a value in the interval $[0, 1]$. The score of procedure i is denoted by π_i. After the last iteration of each segment has been executed, the scores π_i are reset to 0. Lastly, Θ_i denotes the number of times procedure i is used. After each segment, these counts are reset as well. The weights w_{ij} are used to determine the probability a certain procedure is chosen as follows:

$$p_{ij} = \frac{w_{ij}}{\sum_{h\in\mathcal{I}} w_{hj}}. \tag{26}$$

The value p_{ij} denotes the probability of choosing procedure i in segment j. The set $\mathcal{I}$ denotes the set of procedures, procedure i is in: the set of all insertion procedures, the set of all removal procedures, or the set of the noise procedures.

4.4 Acceptance criterion

A solution is accepted to become the new center from which new solutions in the neighbourhood are sought, based on simulated annealing. The idea of simulated annealing is that the further in the search for a solution, the fewer times a new non-improving solution is accepted, ceteris paribus. Each new solution s' is accepted with probability $e^{-(f(s')-f(s))/T}$, given the most recent accepted solution s. In this equation, $f(s)$ denotes the total costs of solution s and T denotes the temperature. Note that if s'

is better than s, it is always accepted. If s' contains some unscheduled requests, it can in contrast to the algorithm of Røpke and Pisinger be accepted, although this in practice seldom occurs, since the penalties for having unscheduled requests, γ, is for many problem instances high, such that the probability to accept becomes virtually zero. It is relevant for instances with low values of γ.

After each iteration, the temperature T is multiplied with the cooling rate c, where $0 < c < 1$, leading to a decreasing temperature in the course of the ALNS metaheuristic. The start temperature is determined indirectly, such that a solution ω% worse than the initial solution is accepted with a probability of 50%. The value ω is a parameter. If the initial solution determining the start temperature contains some unscheduled requests, these costs γ are excluded, since these often will be fairly large and may lead to undesirably high starting temperatures.

Algorithm 4 Vehicle Minimization Stage

Require: s is an initial solution, Φ_V is the maximum number of iterations the heuristic will undertake, ϕ is the maximum number of iterations allowed without progress, Ψ is the set of possible procedure combinations, w_{init} are the initial weights for choosing the procedures, θ is the number of iterations after which the weights w are updated

$s_{\text{best}} \leftarrow s$
$w \leftarrow w_{\text{init}}$
iter $\leftarrow 0$
iter2 $\leftarrow 0$
while iter $< \Phi_V$ **and** iter2 $< \phi$ **do**
  $s' \leftarrow s$
  choose procedures $\psi \in \Psi$ based on w
  generate q
  delete q requests from s' using the procedures in ψ
  reinsert the q requests into s' using the procedures in ψ
  if s' contains no unscheduled requests **then**
    $s_{\text{best}} \leftarrow s'$
    remove last route from s'
    $s \leftarrow s'$
    iter2 $\leftarrow 0$
  else
    if accept s' **then**
      if s' contains fewer unscheduled requests than s **or** s' contains fewer than 5 unscheduled requests **then**
        iter2 $\leftarrow 0$
      else
        iter2 $\leftarrow$ iter2 + 1
      end if
      $s \leftarrow s'$
    else
      iter2 $\leftarrow$ iter2 + 1
    end if
  end if
  update statistics
  if (iter modulo θ) == 0 **then**
    update weights
  end if
  iter $\leftarrow$ iter + 1
end while
return s_{best}

4.5 Initial solution

The initial solution from which new solutions are constructed is found by using the previously described insertion procedures. First, a specific insertion procedure and noise procedure are chosen, using the initial weights w_{init} for the ALNS procedure. These procedures are used to build a new solution from scratch. The procedures are not scored for building the initial solution.

4.6 Vehicle minimization

For many applications, the objective function of the PDPTW has a hierarchal structure, where the objective is first to minimize the number of vehicles and given this number of vehicles to minimize the total distance travelled. Moreover, all requests should be scheduled. Adaptive Large Neighbourhood Search can be applied in modified form to also handle these problems, following the two-stage approach introduced by Bent & Van Hentenryck (2004). The first stage is the vehicle minimization stage, for which the implementation for the ALNS for PDPTW, also used by Røpke & Pisinger (2006a) is shown in Algorithm 4. In the second stage, the usual procedure is followed, with the feasible solution with the lowest number of vehicles used as the initial solution.

In the first stage, the usual removals and reinsertions are undertaken. However, if a solution is found where all requests are routed, that solution is saved and the last route and the related vehicle are deleted from the solution. The requests of that route are not immediately scheduled, leading to an infeasible solution. In subsequent iterations, it is tried to build a new feasible solution. This continues until the maximum number of iterations, Φ_V, has been reached, or if there has been a long period of θ iterations without improvement.

The vehicle minimization procedure only functions properly if the vehicles are homogeneous. That is, all vehicles can handle all requests. Moreover, γ should be set to a high number, such that the algorithm will try to avoid not scheduling requests and the acceptance criterion will not accept infeasible solutions.

Also in the vehicle minimization procedure, simulated annealing is used to accept or reject solutions. For this purpose, ω_V is used to calculate the start temperature and c_V is used as the cooling rate. In the second stage, a new start temperature is calculated and used in the subsequent iterations.

4.7 Time-dependent travel times

The Adaptive Large Neighbourhood Search metaheuristic can also be applied to the situation of time-dependent travel times, as long as the FIFO principle holds. However, this will lead to longer running times. For both the situations where travel times are dependent as the situation where travel times are independent, the insertion procedures need to examine every possible insertion position and check whether this leads to feasible and minimum cost solutions. If there n nodes in the route, the pickup node can be placed at $n-1$ different positions, after all nodes in the route, except for the end terminal. The delivery node can be placed at all positions after the pickup node. Lastly, for the nodes after the delivery node it needs to be checked whether the visit time is still within the time windows. This means that the worst case complexity for checking the optimal position of a request in a route is $O(n^2)$, which also means $O(n^2)$ travel times have to be calculated per route per request to be inserted.

For time-independent PDPTWs, the distances and travel times were pre-calculated and saved. Obtaining the travel time thus requires $O(1)$ time. For time-dependent PDPTWs, the distances were pre-calculated and saved, but the travel times were calculated from those distances during the iterations. The complexity of those calculations depends linearly on the number of arc segments with separate travel speed distributions, k, since these are calculated one by one. The complexity of calculating the travel time for each arc segment, depends on the number of breakpoints occurring during the traversing of the arc segment, which is at most the number of breakpoints in the arc segment $2(P-1)$, although this would be zero or one in most cases. Introducing time-dependent travel times thus increases the complexity of insertion from $O(n^2)$ to $O(2k(P-1)n^2)$ and may increase computation time substantially.

5 Experiments

In order to evaluate the effectiveness of ALNS for the PDPTW and the time-dependent PDPTW, several problem instances previously used in the literature where solved using the Adaptive Large Neighbourhood Search metaheuristic.

5.1 Parameter setting

For all experiments, the parameter setting proposed by Røpke & Pisinger (2006a) was used. This setting consists of the following parameters:

- $\sigma_1 = 9$, the weight for the distance term in the relatedness measure,
- $\sigma_2 = 3$, the weight for the time term in the relatedness measure,
- $\sigma_3 = 2$, the weight for the load term in the relatedness measure,
- $\sigma_4 = 5$, the weight for the overlapping set of vehicles in the relatedness measure,
- $p_S = 6$, the randomness parameter for the Shaw removal procedure,
- $p_W = 3$, the randomness parameter for the worst removal procedure,
- $\omega = 0.05$, the parameter determining the start temperature,
- $\omega_V = 0.35$, the parameter determining the start temperature during the vehicle minimization stage,
- $c = 0.99975$, the cooling rate of the temperature,
- $c_V = 0.9999$, the cooling rate of the temperature during the vehicle minimization rate,
- $\rho_1 = 33$, the score for finding a new best solution,
- $\rho_2 = 9$, the score for finding a new solution better than the last accepted solution,
- $\rho_3 = 13$, the score for finding a new accepted solution,
- $r = 0.1$, the reaction parameter for updating scores,
- $\eta = 0.025$, the noise parameter,
- $\xi = 0.4$, the maximum fraction of requests to be removed in the removal procedures,
- $\Phi = 25000$, the number of iterations, excluding the vehicle minimization stage,
- $\Phi_V = 25000$, the maximum number of iterations during the vehicle minimization stage,
- $\phi = 2000$, the maximum number of iterations without progress during the vehicle minimization stage.
- $\theta = 100$, the segment size after which the procedure weights are updated

The implementation of the ALNS method of this research includes the regret-2, regret-3 and regret-4 insertion procedures as well as the regret-m procedure, where m is the number of routes. Regret-k procedures were only used if $m \leq k$. The three removal procedures discussed before, the Shaw removal procedure, the random removal procedure and the worst removal procedure, were used. Both noise procedures, using noise and not using in the objective function for insertions were used. This is the same configuration as used by Røpke and Pisinger.

The starting weights w_{i0} of all the procedures i where set to 100. Only the proportion of the weights is of importance. However, choosing low initial weights may in some tightly constrained problems where few feasible solutions are found, lead to the weights decreasing too quickly towards zero. Choosing high initial weights overcomes this problem.

5.2 Data sets

For the experiments we used two different sets of problem instances. The first are the PDPTW benchmarks constructed by Li & Lim (2003). Their benchmarks consist of 354 problem instances. Their problems are slightly more constrained than the problem described before, since each instance only contains one depot from which all vehicles depart and return. Moreover, all vehicles are able to handle all requests and every request needs to be handled. The objective for these problems is to minimize the number of vehicles and given this minimal amount of vehicles minimize the distance travelled, the hierarchal objective function.

There are 354 instances generated by Li and Lim, which are generated from the benchmarks for the Vehicle Routing Problem with Time Windows by Solomon (1987). The problems have approximate sizes of 100, 200, 400, 600, 800 and 1000 nodes to visit. There are six different categories of problems: problems where the geographical data is randomly generated (R), problems where the customers are clustered (C) and problems where there is a mix of the former two categories (RC). For these three categories, there are problems with long time horizons and short time horizons, leading to long respectively short routes. Each category for each problem size has approximately 10 problem instances. Each instance has a code, for example 400R1-1, where the first numbers denote the problem size, the letter the distribution of customers, the third number equals one if the problem has a short time horizon and two if it has a long time horizon and the number after the hyphen denotes which instance of the category, with approximate size 10, it is.

The second data set are problem instances generated by Røpke & Pisinger (2006a) and are instances of the problem as described in §3, with constant travel speeds. In the objective function $f(.)$ both α and β are set to 1, while γ is set to 100,000, such that it will always be beneficial to schedule all requests if possible. The data set consists of one instance of each possible combination of the following categories, totalling 48 problem instances:

- Geographical distribution: (i) clustered, (ii) semi-clustered, (iii) random,
- Request type: (i) all requests can be served by all vehicles, (ii) half of the requests can only be served by a subset of the vehicles,
- Terminals: (i) the begin terminals of each route are at the same location as the end terminals, (ii) the begin terminals and end terminals are at different locations,
- Problem size: (i) 100, (ii) 200, (iii) 500 and (iv) 1000 nodes to be served.

The second set of problem instances was also modified to the time-dependent PDPTW. For this purpose, the travel times are calculated using a speed distribution. In the original instances, the speed is 1. Following Ichoua et al. (2003), a stepwise travel speed function is used, which causes the travel time functions to be piecewise linear.

One of the distributions of Figliozzi (2012) is used. In this distribution, the time horizon is divided into five periods of equal length that have the following travel speeds: [2.00, 1.00, 1.50, 1.00, 2.00]. This can be interpreted as a day where in the very early morning little traffic is present, so the travel speed is high. Afterwards, there is congestion due to the morning rush hour, leading to a relatively low travel speed. In the middle of the day, there is somewhat less traffic and the travel speed is thus somewhat higher than during the rush hours, but lower than at the extremes of the day. The fourth time period can be interpreted as the evening rush hour and the fifth time period as the relatively quiet late evening. The average speed throughout the day is 1.50 and thus higher than the travel speed of 1 of the original problem. It is decided to let the lowest travel speed be equal to original travel speed, since lower speeds during some parts of the day may lead to infeasible problem instances.

Another scenario is where the center of the area of the problem instance has a different, slower speed distribution than the rest of the area. This scenario is from here on called the city center scenario, since it can be interpreted as a city where the center is always relatively congested, causing the speeds outside rush hours to increase relatively little. For the city center the following stepwise travel speed distribution is used: [1.50, 1.00, 1.25, 1.00, 1.50]. The average travel speed in the city center is 1.25, compared to 1.50 outside the city center.

If the geographical extremes of the problem instance is denoted by [N,W,S,E], denoting respectively the most northern, western, southern and eastern location of the problem. The city center is a rectangle encompassed by the lines:

$$y = S + \frac{3}{4}(N - S)$$
$$y = S + \frac{1}{4}(N - S)$$
$$x = W + \frac{1}{4}(E - W)$$
$$x = W + \frac{3}{4}(E - W)$$

Since the travel times do not only depend on the distance and the departure time, Equation (21) does not hold anymore and therefore the triangle inequality is not valid anymore. It is assumed that arcs between two locations are always the shortest routes between the two locations, although faster routes may be possible in some cases if the city center is avoided. Therefore, it is in this scenario possible that insertions may speed up the route.

In case an arc (i, j) partially falls inside the city center and partially outside the city center, the arc is divided in k segments, where each segment corresponds to the area it is in, city center or outside the city center. The value k can be at most three, which is the case if an arc traverses the city center, but both the start and end node of the arc are outside the city center. The travel times of the segments are calculated sequentially, using for each segment the speed distribution of the area it falls in.

5.3 Configurations

To examine different aspects of the ALNS algorithm, several experimental computations were performed using different configurations. The instances of Li and Lim were run in five different manners. In the first set of runs, the ALNS algorithm was applied without the vehicle minimization stage, from here on called 1-stage ALNS, in order to examine the strength of the algorithm for optimizing the objective value it was originally designed for. The only objective is to optimize the distance travelled for these instances.

Afterwards, the ALNS metaheuristic was compared to the LNS metaheuristic, which is the ALNS metaheuristic using only one insertion procedure, one removal procedure and either never using noise in the objective function for the insertion procedures or always using noise. The procedures used are those found to be the best LNS configuration by Røpke and Pisinger: the Shaw removal procedure, the regret-4 insertion procedure and using noise in the objective function for insertions.

Both LNS and ALNS were run on the Li & Lim instances with the vehicle minimization stage, from here on called 2-stage LNS resp. 2-stage ALNS. Comparisons are made between LNS and ALNS, as well as between ALNS with and without the vehicle minimization stage.

Lastly, to test the sensitivity of ALNS to the number of iterations, two-stage ALNS was run with the number of iterations in the vehicle minimization stage, Φ_V, and the number of iterations in the second stage, Φ set to 100,000. Intermediate results after 10,000, 25,000 and 50,000 iterations were saved for both stages. The Li and Lim instances of approximate size of 100, 200 and 400 nodes were solved ten times for each of the three procedures mentioned above. Due to the relatively long running times of the larger instances, these were not run. Furthermore, the configurations with 100,000 iterations were run five times. Moreover, given the results, ALNS was modified to test whether allowing infeasible solutions to be accepted improves the solution.

The instances of Røpke and Pisinger were run in five different manners. Since vehicle minimization is neither an objective, nor is every instance suitable for the vehicle minimization procedure, due to heterogeneous vehicle fleets, the vehicle minimization procedure was not applied for the these instances. Both ALNS and LNS were applied to the normal scenario, which were also performed by Røpke and Pisinger. Secondly, the instances were solved using the speed distribution of Figliozzi by ALNS. These instances are compared to the scenario where the travel speed is constant, but equal to 1.50, the average of the Figliozzi speed distribution. Lastly, the instances were solved for the city center scenario. This was done for $\Phi = 25,000$ iterations and $\Phi = 100,000$ iterations, saving the intermediate results after 10,000,

25,000 and 50,000 iterations.

ALNS respectively LNS were ran ten times for each problem instance for each of the experiments on the instances of Røpke and Pisinger, except for the configuration with 100,000 iterations, when ALNS was run five times for each instance.

The ALNS calculations on the Li and Lim instances were performed using a 3.7 GHz desktop computer with an Intel Core i3-6100 processor and 4 GB random access memory running on Windows 10. The computations for the Røpke and Pisinger instances were done on a 3 GHz laptop with an Intel Core i7-4720HQ processor with 16 GB random access memory running Windows 10.

The heuristic was implemented in Java 8. For travel time and distance calculations double precision was used, since this is the dominant convention for the problem instances of Li and Lim. The final objective values were rounded to two decimal places.

6 Results

6.1 Li & Lim instances

Table 1 shows the results of 1-stage LNS and 1-stage ALNS for the Li & Lim instances. In this and further tables, the average gap is calculated using the formula:

$$Average\ gap = \frac{1}{\sum_{j \in \mathcal{J}} |\mathcal{S}_j|} \sum_{j \in \mathcal{J}} \sum_{s \in \mathcal{S}_j} \frac{f(s) - f(s_j^*)}{f(s_j^*)} \tag{27}$$

In this equation, $\mathcal{J}$ is the set of problems, $\mathcal{S}_j$ is the set of solutions found by the algorithm of interest for problem j, s_j^* is the best solution for problem j, which may or may not part of $\mathcal{S}$. The function $f(.)$ is the objective function, which is the weighted sum of the minimal distance travelled, the total time spent by vehicles outside the terminals and unscheduled requests. Note that this also applies to results using the vehicle minimization criterion. Therefore, solutions that are worse than the best solution, might still receive negative gap values.

Table 1: Comparison ALNS and LNS without vehicle minimization

			Best Sol.		Avg. Gap (%)		Avg. Time (s)	
	Nodes	Problems	ALNS	LNS	ALNS	LNS	ALNS	LNS
Results	100	56	56	36	0.31	1.12	9	8
	200	60	58	7	1.20	3.20	34	32
	400	60	60	0	2.63	2.88	185	150
Røpke & Pisinger	100	56	52	50	0.19	0.50	49	55
	200	60	49	15	0.72	1.41	305	314
	400	60	52	6	2.36	4.29	585	752

Table 1 shows the results of this research and that of the research by Røpke & Pisinger (2006a) for the Li & Lim instances without the vehicle minimization stage. The column *Nodes* shows the problem size, the column *Problems* the number of problem instances. For the results of this experiment, the *Best solution* columns show the amount of times the ALNS resp. LNS algorithm have found the same or better solution than the other algorithm. The *Average gap* columns show the average gap to best solution found by either of the two algorithms for the ALNS resp. LNS. For the results of Røpke and Pisinger, the best solution may also be the best solution found by Li and Lim, if better than found by both ALNS and LNS. The *Average time* columns show the average running time for the ALNS resp. LNS metaheuristic.

In Table 1, it can be seen that the Adaptive Large Neighbourhood Search outperforms its non-adaptive variant for all instance sizes solved in this research, as was found by Røpke and Pisinger. The results of Table 1 obtained by this research are not directly comparable with the results of Røpke and Pisinger, since they only provide figures comparing their results with the results obtained by Li & Lim (2003). However, the solutions of Li and Lim for all instances of more than 100 nodes cannot be obtained, since

their web site is off line.

It can be noted that the average gap to the best solution appears to be slightly larger in this research than in the research of Røpke and Pisinger in most cases. Moreover, for the instances of 100 nodes, the ALNS algorithm found 50 solutions better or equal than both Li and Lim and the LNS algorithm, compared to 52 by Røpke and Pisinger. The LNS algorithm found 36 results equal or better than Li and Lim and the ALNS algorithm, whereas this figure was 50 for Røpke and Pisinger. It can thus be deduced that especially the LNS metaheuristic performs relatively poorly, both compared to ALNS, as is expected, but also to the LNS metaheuristic in Røpke & Pisinger (2006a). Moreover, both heuristics are somewhat less robust.

Interestingly enough, the average running time of the LNS metaheuristic is slightly faster than the ALNS metaheuristic, whereas this is opposite for the heuristics implemented by Røpke and Pisinger. However, the absolute running times are three- to tenfold shorter.

For a specific problem, problem RC400-10, the tenth instance with random and clustered customers and 400 requests of Li & Lim, the average running times over all the iterations of a single run of the different procedures are shown in Table 2. The regret procedures seem slightly faster than the greedy procedure. Moreover, the worst removal appears to be slightly slower than the Shaw removal procedure. Note that these figures must be interpreted with caution. For example, the relatively high running time of the greedy procedure, might be due to running time of the insertion it mandates, which are included in the running time of the procedures.

Table 2: Running times

Procedure	Run Time (ms)
Shaw	4.4
Random	0.4
Worst	5.4
Greedy	8.9
Regret-2	7.5
Regret-3	7.3
Regret-4	6.7
Regret-m	8.1

The results in the table provide an explanation for the phenomenon that LNS is faster than ALNS, whereas this is not the case in this research. Here, the greedy procedure seems slowest of the insertion procedures, whereas Røpke and Pisinger noted that the regret method was somewhat slower than the greedy method.

Even though 1-stage ALNS does not try to minimize the number of vehicles used, it often appears optimal to do so. To examine whether ALNS performs comparably without vehicle minimization stage to ALNS with the vehicle minimization stage, the best solutions according to the hierarchal objective function were saved during the iterations of 1-stage ALNS and compared with the solutions found by 2-stage ALNS. The results are shown in Table 3.

From that table it can be deduced that mainly for small, easy to solve problem instances, 1-stage ALNS performs reasonably well in minimizing the number of vehicles. Moreover, 1-stage ALNS often performs better in the instances with clustered customers, compared to other geographical distributions. However, for a large majority of the instances 2-stage ALNS performs better than 1-stage ALNS. In running time, 1-ALNS is often slightly faster than 2-ALNS. However, for the runs of 400 customers, 2-ALNS, unexpectedly, was often faster than 1-ALNS. Since the difference in running time between the two algorithms is substantial and the two runs happened relatively far apart in time, this raises the suspicion that the reason may be attributed to the device and not to the algorithm.

Table 3: Comparison 1-ALNS and 2-ALNS

			Best Sol.		Least Veh.		Avg. Time (s)	
Nodes	Dist.	Problems	1-ALNS	2-ALNS	1-ALNS	2-ALNS	1-ALNS	2-ALNS
100	C	16	13	20	15	20	7	7
100	R	23	20	20	20	20	12	15
100	RC	17	13	20	14	20	9	11
200	C	20	15	18	17	20	20	21
200	R	20	2	20	4	20	44	64
200	RC	20	1	20	2	20	42	53
400	C	20	8	16	10	20	99	79
400	R	20	1	19	1	20	314	180
400	RC	20	0	20	1	20	143	187

Table 3 compares the solutions of 1-ALNS with those of 2-ALNS for the Li and Lim instances. The column *Nodes* shows the problem sizes, the column *Dist.* the geographical distribution of the customers. Column *Problems* shows the number of problems of the characteristics in the two aforementioned columns. The *Best Sol.* columns show the number of times the best solution found by 1-ALNS resp. 2-ALNS was better than or equal to the best solution found by the other algorithm, according to the hierarchical objective function. The *Least Veh.* columns show the number of times the best solution found by 1-ALNS resp. 2-ALNS had the same number of vehicles as the best solution found by the two algorithms. The *Avg. Time* columns show the average running time of 1-ALNS resp. 2-ALNS. The objective values of the best 1-ALNS and 2-ALNS solutions can be found in the appendix.

Table 4 compares the best found solutions of 2-stage ALNS and 2-stage LNS. When examining the performance of 2-stage ALNS and 2-stage LNS, the picture is similar to that of 1-stage ALNS and 1-stage LNS. It is apparent that also for two stages ALNS performs better than LNS, as can be seen in Table 4. Both the number of vehicles and the total distance from all solutions combined is better for the ALNS algorithm than for LNS. However, ALNS in this experiment seemed to perform somewhat poorer than in the research of Røpke and Pisinger, especially for the larger instances. For the instances of 100 nodes, both ALNS implementations lead to the same solutions, which are equal to the best found solutions in the literature.

Table 4: Comparison totals of best 2-stage ALNS and 2-stage LNS solutions

	Nodes	Vehicles	Distance	Avg. Time (s)	Avg. Gap (%)
ALNS	100	402	58,060	11.5	0.48
	200	616	183,244	44.9	4.09
	400	1206	462,608	149	7.74
LNS	100	402	58,092	9.4	2.70
	200	639	199,046	41.2	8.79
	400	1305	541,663	136	7.91
ALNS Røpke & Pisinger	100	402	58,060	66	
	200	606	180,931	264	
	400	1158	422,201	881	

Table 4 shows the results for the Li & Lim instances including the vehicle minimization stage. The first horizontal segment exhibits figures for the 2-stage ALNS metaheuristic, the second segment for 2-stage LNS and the third segment show the results obtained by the 2-stage ALNS metaheuristic of Røpke and Pisinger. The *Nodes* column shows the problem size, the *Vehicles* column the total amount of vehicles that need to be used according to the best solutions found by the respective algorithms for all instances of the size corresponding to the respective row. The *Distance* column shows the total distance of all instances for the given problem size according to the best solutions found. The *Average Time* column show the average time used to calculate the best solutions. The *Average Gap* column shows the average gap to the best solution found by the algorithm of the relevant row segment. These figures were not reported by Røpke and Pisinger. The objective values of the best 2-stage ALNS results can be found in the appendix.

Although not reported in their research, the average gap values seem high. A possible explanation may be that the algorithm spends too much time with the same solutions and therefore does not explore the

solution space enough. This hypothesis was based on the fact that depending on the problem instance, ALNS found in some runs as little as 20 different feasible solutions.

To test the above hypothesis, the 2-stage ALNS metaheuristic was amended slightly such that solutions with unscheduled requests would have a realistic chance to be accepted. For accepting a solution, the costs of an unscheduled request was not set to γ, but to the extra distance travelled when scheduling an additional route to handle solemnly that request, multiplied by a parameter ζ. This parameter was set to 0, 0.5 and 1. A small subset of the Li and Lim instances, the 200C and 200R instances, were each solved five times using this modified method. Table 5 shows the totals of the best solutions found by each configuration. The procedural change appears to not have the desirable effect. The total number of vehicles over the the set of instances is for the three experimental settings substantially higher and slightly decreasing if the penalty for having unscheduled request increases.

Table 5: Comparison totals modified 2-stage ALNS and 2-stage ALNS

ζ	Avg. Gap	Total Vehicles	Total Distance
0.0	2.35	472	127,695
0.5	2.90	467	123,361
1.0	2.44	463	123,413
	2.37	418	112,003

Table 5 shows summaries of the results found by the the modified ALNS average gap. The rows correspond to the configuration with the parameter ζ, as shown in the first column. The *Avg. Gap* column corresponds to the average gap to the best solution found by the algorithm corresponding to the relevant row. The *Total Vehicle* and *Total Distance* columns show the total number of vehicles and distance of the best solutions found by the algorithm of the corresponding row. The last row shows the results for the first five runs normal 2-stage ALNS, to ensure a fair comparison.

Moreover, there appears to be no strong correlation between the number of feasible solutions found and the gap to the best solution for a set of ALNS runs. For each Li and Lim problem instance of 400 nodes solved by 2-stage ALNS, the correlation between the gap to the best solution found by 2-stage ALNS and the number of solutions found where all requests were scheduled during the ALNS procedure was calculated. The average of these sixty figures was 0.084, indicating that the quantity of feasible solutions has minor or no effect on the strength of the best solution found.

Table 6 shows results for the first stage of 2-stage ALNS, when varying the maximum number of iterations of that stage Φ_V and the maximum number of iterations without progress ϕ. The results are intermediate results of runs with $\Phi_V = \phi = 100,000$, therefore a higher number of runs can only lead to improvements, whereas for separate runs this need not be the case, due to the stochastic nature of ALNS.

From the table, it can be deduced that the best trade-off between the number of iterations is dependent on the problem size. Whereas the smallest problem instances, of 100 nodes, can find strong solutions with $\Phi_V = 10,000$ and $\phi = 2000$, the solutions can be increased substantially with a larger number of iterations for instances of 400 nodes. Moreover, it appears that the criterion to stop searching for a better solution when there was no progress for ϕ iterations, also stops in situations where there is a reasonable chance to decrease the number of vehicles further, considering the improvements when ϕ is increased.

Table 6: Comparison variations in iterations vehicle minimization stage

		100 Nodes		200 Nodes		400 Nodes	
Φ_V	ϕ	Best Veh.	Total Veh.	Best Veh.	Total Veh.	Best Veh.	Total Veh.
10,000	2000	402	2021	618	3137	1229	6276
	5000	402	2019	618	3135	1228	6266
	10,000	402	2019	618	3135	1227	6265
25,000	2000	402	2019	616	3123	1215	6202
	5000	402	2019	615	3120	1213	6177
	10,000	402	2019	615	3119	1212	6173
	25,000	402	2019	615	3119	1211	6171
50,000	2000	402	2019	616	3121	1210	6181
	5000	402	2019	614	3117	1207	6149
	10,000	402	2019	614	3115	1204	6135
	50,000	402	2019	614	3115	1203	6126
100,000	2000	402	2019	616	3121	1210	6179
	5000	402	2019	614	3117	1206	6144
	10,000	402	2019	614	3115	1201	6124
	100,000	402	2017	613	3107	1200	6104

Table 6 shows the number of vehicles the vehicle minimization stage resulted in for varying numbers of Φ_V and ϕ. The *Best Veh.* columns show the total amount of vehicles using the best solution found by the configuration of the corresponding row, for the instances of 100, 200 and 400 nodes respectively. The *Total Veh.* columns show the total number of vehicles of all the five runs found for all instances of the relevant size.

It is also striking that there is a lot of variation in the number of vehicles needed to schedule all requests, since the total number of vehicles of all the runs in for all configurations exceeds the best number of vehicles multiplied with the number of runs, five. ALNS thus appears to need multiple attempts, to deliver strong results.

Table 7 shows the results for varying values of the number of iterations Φ of the second stage of 2-stage ALNS, using the solutions found after 100,000 iterations in the first stage. As before, the results are intermediate results of the highest number of iterations.

Table 7: Comparison variations in iterations second stage

Nodes	Φ	Distance	Avg. Gap (%)
100	10,000	58,164	0.36
	25,000	58,080	0.36
	50,000	58,079	0.29
	100,000	58,079	0.18
200	10,000	187,770	2.76
	25,000	185,508	2.17
	50,000	184,304	1.91
	100,000	183,110	1.37
400	10,000	477,896	2.98
	25,000	465,716	3.19
	50,000	456,151	3.39
	100,000	447,517	3.15

Table 7 shows the results of the second stage for varying numbers of Φ. The *Distance* column shows the total distance using the best solution found by the configuration of the corresponding row, for the instances of 100, 200 and 400 nodes respectively. The *Avg. Gap* columns show the average gap of the results found by the configuration to the best solution found by that configuration. The objective values of the best solutions after 100,000 iterations can be found in the appendix.

As in the first stage, the number of iterations is of more importance, the larger the problem size, when

comparing the total distances of the best solutions found for different values of Φ. Moreover, the average gap seems increasing in the problem size, and decreasing for small problem instances, but not for the instances of 400 nodes. A possible explanation is that for small problem instances, the algorithm is close to the optimal solution for a relatively low amount of iterations, such that a higher number of iterations avoids finding weak solutions, but only increases the best solution found marginally, leading to lower average gap values. For larger problem instances, larger improvements of the best solution found are still possible, such that both the best and weak solutions improve, compared to lower amounts of iterations, leaving the average gap more or less constant.

It is interesting to compare the total distances found in Table 7 with those in Table 4. For the small instances, the totals of ALNS with 25,000 iterations but 10 runs are somewhat stronger than the the totals of ALNS with 50,000 iterations and 5 runs, whereas for the instances of 400 nodes, this is not the case. It may thus be beneficial to run small instances frequently, with a relatively low amount of iterations and larger instances fewer times but longer per run. However, the difference in the length of the first stage might have also played a role.

6.2 Instances Røpke & Pisinger

Table 8 shows the results of 1-stage ALNS and 1-stage LNS on the instance of Røpke and Pisinger. Some observations of the previous subsection do not hold anymore. First of all, the average gap to the best solution is decreasing for the problem size, instead of increasing for ALNS. For LNS there is not a clear straight-line pattern. The reason may be that the large instances of Røpke and Pisinger are not tightly constrained problem instances and the best solution often contain a few routes that serve no requests. Therefore, it is likely to be easier to find good solutions for these instances.

Table 8: Comparison ALNS and LNS instances of Røpke & Pisinger

			Best sol.		Avg. gap (%)		Avg. time (s)	
	Nodes	Problems	ALNS	LNS	ALNS	LNS	ALNS	LNS
Results	100	12	12	0	8.13	13.83	3.2	3.6
	200	12	12	0	5.16	16.12	11.7	16.0
	500	12	12	0	4.35	26.00	105	165
	1000	12	12	0	1.13	1.08	302	439
Røpke & Pisinger	100	12	8	5	1.44	1.86	23	34
	200	12	11	1	1.54	2.18	83	142
	500	12	7	5	1.39	1.62	577	1274
	1000	12	9	3	1.18	1.32	3805	8146

Table 8 shows the results of ALNS and LNS on the problem instances generated by Røpke and Pisinger. It has the same structure as Table 1. In the first segment of rows, the results obtained in this research are displayed, in the second segment of rows, the results of Røpke and Pisinger are shown.

Another observation is that the difference between LNS and ALNS is larger in this research than in the research by Røpke and Pisinger, which also holds for the Li & Lim instances. Moreover, ALNS is substantially faster than LNS, in contrast to the previous results where it was slightly slower. This may again be caused by the difference in nature of the problem instances. In their paper, Røpke and Pisinger also find that the running time of ALNS compared to LNS is relatively better for their problem instances than for the instances of Li and Lim.

In Table 9, the solutions of the time-dependent instances of Røpke and Pisinger are compared with the solutions of time-independent instances with a travel speed of 1.5, the average of the time-dependent PDPTW without a city center, and the standard scenario. These results show that for the four different scenarios the average gap of the solutions to the best solution is comparable. ALNS thus seems equally robust for the time-dependent PDPTW as for the time-independent PDPTW. Furthermore, the costs of the time independent scenario with a speed of 1.5 are slightly lower than for the time-dependent scenarios, which implies the algorithm is not able to benefit from the time-dependent travel times, by scheduling more in high-speed hours and less in low-speed hours. It should be noted however, that none of the instances were tightly constrained for the scenarios with time-dependence, since the average speed was

well above the original speed. Compared to the original scenarios, time-dependence has lead to a cost decrease. For more tightly constrained problems the results might have been different.

Table 9: Comparison time-dependent and time-independent PTPTW

	Nodes	Avg. Gap (%)	Best Costs	Avg. time (s)
Time-dependent	100	7.45	820,052	4.7
	200	5.54	1,606,328	17.6
	500	4.41	3,761,918	149
	1000	1.15	7,498,478	376
Time-dependent with center	100	6.56	828,463	7.5
	200	4.84	1,636,732	31.3
	500	4.33	3,802,222	275
	1000	1.09	7,574,892	611
Time Independent Speed 1.5	100	6.75	805,531	3.3
	200	5.08	1,584,436	13.9
	500	4.69	3,712,798	131
	1000	1.20	7,361,431	344
Time Independent Speed 1.0	100	8.13	878,138	3.2
	200	5.16	1,743,124	11.7
	500	4.35	4,029,069	105
	1000	1.13	7,908,350	302

Table 9 shows the results for the time-dependent PDPTW in the first segment of rows, the time-dependent PDPTW with a city center in the middle segment of rows, and the time-independent PDPTW in the last segment of rows obtained by ALNS. In *Nodes* column, the problem size is stated, the *Avg. Gap* column displays the average gap to the best solution found by that algorithm. The *Best Costs* column displays the total costs of the best solution for all problem instances of the relevant size. The last column shows the average running time for the instances of the relevant size. The objective values of the best solutions for all four scenarios can be found in the appendix.

Lastly, the running time for ALNS of the time-dependent PDPTW is slightly higher than the running time of the time-independent PDPTW. However, the running time of the time-dependent PDPTW with a city center is twice the size of the running time of the time-independent PDPTW. This reflects the extra computations needed to calculate the travel time.

Table 10 shows the final and intermediate results for the time-dependent instances of Røpke and Pisinger with city center where the number of iterations equals $\Phi = 100,000$. In contrast with the instances of Li& Lim, the solutions for the small instances can be improved substantially by increasing the number of iterations. The the total costs decrease is more or less similar for the four different problem sizes, percentage wise. This may mean that the time-dependent PDPTW may need more iterations to find high-quality solutions, although it may also be due to the nature of the instances of Røpke and Pisinger.

Another notable aspect is that the average gap to the best solution found is increasing in the number of iterations, where there appeared a small decreasing trend for the instances of Li & Lim. However, the same explanation may still apply, since for the more difficult problems, the gap did not seem to decrease. Since the best solution still increases substantially when increasing the number of iterations, the gap may also increase subsequentially. For the time-dependent instances of Røpke and Pisinger, the average quality of the solutions improved slower than the quality of the best solutions.

Table 10: Varying iterations for time-dependent scenario with city center

Nodes	Iterations	Costs	Avg. Gap (%)
100	10,000	921,949	2.42
	20,000	836,710	5.02
	50,000	799,226	6.40
	100,000	770,649	6.48
200	10,000	1,740,574	2.24
	20,000	1,641,792	4.23
	50,000	1,570,993	5.06
	100,000	1,518,107	6.03
500	10,000	4,012,349	2.00
	20,000	3,834,376	3.10
	50,000	3,729,335	3.73
	100,000	3,677,792	3.76
1000	10,000	7,602,403	1.20
	20,000	7,479,431	1.09
	50,000	7,398,252	1.59
	100,000	7,226,459	2.18

Table 10 displays the total costs of all the best solutions found after the number of iterations of the horizontal segment in the *Best Costs* column. The *Avg. Gap* column shows the average gap of the solutions found after the number of iterations of the horizontal segment,to the best solution found after the same number of iterations. The objective values of the best solutions after 100,000 iterations can be found in the appendix.

7 Conclusion

Adaptive Large Neighbourhood Search is applied to numerous combinatorial optimization problems, due to the promising results compared to other metaheuristics. One of the aims of this research was to examine whether ALNS indeed is able to provide high quality solutions. It can be concluded that although the metaheuristic performed relatively well, compared to the original solutions of Li & Lim (2003) and compared to a similar method with a single procedure, LNS, it performed somewhat weaker than the ALNS implementation of Røpke and Pisinger, especially for larger problem instances. Although the exact reason for this difference is difficult to discover, the low number of feasible solutions found in some problem instances may be problematic. Intuitively, this causes too few opportunities to find optimal solutions and few different neighbourhoods are scanned for better solutions. Lowering the threshold for accepting solutions to be able to visit more neighbourhoods has shown to be unlikely to be beneficial. A method to improve the performance of ALNS might therefore be to increase the strength of the insertion procedures, such that more high-quality feasible solutions are found. This could be achieved by additionally using a more sophisticated insertion procedure, for example the procedures of Shaw (1997) and Bent & Van Hentenryck (2006).

A second observation is that the vehicle minimization stage of 2-stage ALNS is often crucial to find good solutions. Since for the Li and Lim instances, the number of vehicles found was often higher than the optimum and minimizing the number of vehicles is the most important criterion, the performance of ALNS could potentially be increased by allocating a larger share of the number of iterations to the first stage at the cost of the second stage.

Since the running times decreased due to improved technology, the trade-off between speed and quality may be changed. If more time were allocated to solving instances, the two previous recommendations, (i) using better, but likely more computationally expensive, insertion procedures and (ii) increasing the number of runs in the vehicle minimization stage, could be among the first aspects to examine. Increasing ϕ, the maximum number of iterations without improvement in the vehicle minimization stage, might also substantially increase performance, especially for difficult to solve instances.

When examining the results, it can be concluded that although ALNS often finds high quality solutions for PDPTW problems, the robustness of the metaheuristic is not always equally high and dependent on

the problem size and possibly also on the instances, such as the tightness of the constraints. A recommendation for further research is to examine more closely which instance characteristic cause ALNS to be relatively unstable. The variability of the solutions of ALNS imposes a trade-off between the number of runs and number of iterations within a run, which is dependent on the size of the instances and the nature of the instances. For the larger instances of Røpke and Pisinger, it is recommended to have a relatively high number of iterations and low number of runs, compared to the instances of Li & Lim. A recommendation for practical applications is to experimentally determine whether running ALNS multiple times increases solutions substantially and in that manner determine a suitable trade-off.

Adaptive Large Neighbourhood Search seems suitable for being applied to problem instances with time-dependence, although it is hard to comment on the quality of the solutions, due to a lack of benchmark problem instances for time-dependent PDPTW. The robustness of ALNS for these instances is comparable to the robustness of time-independent PDPTW, however a higher number of iterations might be needed to achieve high-quality solutions. The running time of ALNS is highly sensitive to the complexity of calculating the travel times. Therefore, using more complex travel time distributions than piecewise linear or implementing a high number of different speed zones may increase the running time heavily. However, using relatively simple travel time distributions might be a valuable addition for practical applications on the favourable side of the trade-off between realism and computational speed. Further research should be done to compare the quality of the solutions of ALNS on time-dependent PDPTW, by amongst others creating benchmarks for the problem.

References

Adulyasak, Y., Cordeau, J.-F., & Jans, R. (2012). Optimization-based adaptive large neighborhood search for the production routing problem. *Transportation Science*, *48*(1), 20–45.

Aksen, D., Kaya, O., Salman, F. S., & Tüncel, Ö. (2014). An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research*, *239*(2), 413–426.

Azi, N., Gendreau, M., & Potvin, J.-Y. (2014). An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, *41*, 167–173.

Bent, R., & Van Hentenryck, P. (2004). A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, *38*(4), 515–530.

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.

Coelho, L. C., Cordeau, J.-F., & Laporte, G. (2012). The inventory-routing problem with transshipment. *Computers & Operations Research*, *39*(11), 2537–2548.

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

Demir, E., Bektaş, T., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, *223*(2), 346–359.

Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E., & Gambardella, L. M. (2008). Time dependent vehicle routing problem with a multi ant colony system. *European journal of operational research*, *185*(3), 1174–1191.

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

Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, *57*(4), 1472–1483.

Figliozzi, M. A. (2012). The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, *48*(3), 616–636.

Gendreau, M., Guertin, F., Potvin, J.-Y., & Séguin, R. (1998). Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Tech. Rep. CRT-98-10, Centre de recherche sur les transports, Universite Montreal, Canada*.

Glover, F. W., & Kochenberger, G. A. (2006). *Handbook of metaheuristics* (Vol. 57). Springer Science & Business Media.

Grangier, P., Gendreau, M., Lehuédé, F., & Rousseau, L.-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, *254*(1), 80–91.

Haghani, A., & Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. *Computers & operations research*, *32*(11), 2959–2986.

Hashimoto, H., Yagiura, M., & Ibaraki, T. (2008). An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, *5*(2), 434–456.

Hemmelmayr, V. C., Cordeau, J.-F., & Crainic, T. G. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, *39*(12), 3215–3228.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European journal of operational research*, *144*(2), 379–396.

Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., & Wilson, N. H. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, *20*(3), 243–257.

Katterbauer, K., Oguz, C., & Salman, S. (2012). Hybrid adaptive large neighborhood search for the optimal statistic median problem. *Computers & Operations Research*, *39*(11), 2679–2687.

Kovacs, A. A., Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 1–22.

Kristiansen, S., & Stidsen, T. R. (2012). Adaptive large neighborhood search for student sectioning at danish high schools. In *Proceedings of the ninth international conference on the practice and theory of automated timetabling (patat 2012)*.

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

Li, B., Krushinsky, D., Van Woensel, T., & Reijers, H. A. (2016). An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research*, *66*, 170–180.

Malandraki, C., & Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation science*, *26*(3), 185–200.

Masson, R., Lehuédé, F., & Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, *47*(3), 344–355.

Muller, L. F. (2009). An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In *Mic 2009: The viii metaheuristics international conference.*

Muller, L. F., Spoorendonk, S., & Pisinger, D. (2012). A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research*, *218*(3), 614–623.

Pisinger, D., & Røpke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, *34*(8), 2403–2435.

Pureza, V., & Laporte, G. (2008). Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. *Infor*, *46*(3), 165.

Raff, S. (1983). Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, *10*(2), 6369117149195–67115147193211.

Ribeiro, G. M., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & operations research*, *39*(3), 728–735.

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

Røpke, S., & Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, *171*(3), 750–775.

Salazar-Aguilar, M. A., Langevin, A., & Laporte, G. (2011). An adaptive large neighborhood search heuristic for a snow plowing problem with synchronized routes. In *Network optimization* (pp. 406–411). Springer.

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

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming* (pp. 417–431).

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

Sørensen, M., Kristiansen, S., & Stidsen, T. R. (2012). International timetabling competition 2011: An adaptive large neighborhood search algorithm. In *9th international conference on the practice and theory of automated timetabling (patat 2012)* (pp. 489–492).

Taş, D., Dellaert, N., van Woensel, T., & de Kok, T. (2014). The time-dependent vehicle routing problem with soft time windows and stochastic travel times. *Transportation Research Part C: Emerging Technologies*, *48*, 66–83.

Toth, P., & Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, *123*(1), 487–512.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, *231*(1), 1–21.

8 Appendix

The following tables provide the best solutions found for the Li & Lim instances, the instances of Røpke and Pisinger and the time-dependent versions. Table 11 provides the best solutions for the different scenarios: the regular setting, the setting with a speed of 1.5, the time-dependent scenario without a city center, the time-dependent scenario with a city center and the scenario with a city center with $\Phi = 100,000$.

The other tables provide the best found solutions for 1-stage ALNS, 2-stage ALNS and 2-stage ALNS with $\Phi_V = \Phi = \phi = 100,000$.

Table 11: Solutions Røpke instances

Instance	Speed 1.0	Speed 1.5	Time-Dep.	City Center	Time-Dep Long
100A	70,626.96	66,579.84	67,441.51	68569.37	60,761.64
100B	79,471.15	74,705.95	74,705.95	74930.74	68,985.71
100C	77,135.62	71,715.31	72,177.62	70795.36	69,581.8
100D	89,894.21	81,781.02	82,631.47	82762.17	79,290.74
100E	58,339.45	53,799.85	58,511.47	58937.52	54,739.75
100F	63,331.88	62,239.83	58,553.02	59485.67	52,192.34
100G	67,673.36	64,181.67	66,545.15	67644.50	62,745.33
100H	75,293.23	68,795.09	69,601.49	70227.50	64,902.37
100I	68,582.73	59,089.08	60,988.26	63940.30	59,332.53
100J	75,672.07	67,390.64	68,779.90	70001.75	67,173.87
100K	73,843.11	63,380.35	63,437.74	66194.43	62,488.9
100L	78,274.57	718,72.95	76,678.08	74973.51	68,453.86
200A	160,318.13	137,084.97	142,934.43	144,715.08	133,863.98
200B	144,213.81	131,276.01	133,991.96	144,717.73	127,018.27
200C	153,024.89	144,602.41	144,632.85	148,460.59	138,075.01
200D	163,239.42	150,929.47	152,513.82	149,263.07	145,828.03
200E	125,490.88	111,038.98	113,646.44	118,473.60	108,030.91
200F	134,679.25	130,692.60	133,502.43	128,359.75	114,349.32
200G	138,313.05	127,677.69	127,237.38	129,349.01	119,037.58
200H	138,486.25	125,708.93	117,224.03	129,342.42	113,101.04
200I	143,228.98	122,045.91	132,814.16	134,819.63	122,956.79
200J	148,215.59	133,013.67	134,380.61	138,032.69	129,034.0
200K	137,429.29	124,402.29	127,423.25	126,342.23	127,390.05
200L	156,484.72	145,962.63	146,027.03	144,856.48	139,421.65
500A	364,042.80	321,151.36	324,274.65	342,745.85	315,271.46
500B	338,098.37	317,137.42	317,137.42	320,996.44	315,624.59
500C	350,729.93	325,974.46	330,445.38	322,721.79	321,941.07
500D	375,076.78	346,723.00	354,112.76	353,348.80	348,909.2
500E	256,229.68	246,718.50	248,283.42	246,603.62	229,368.31
500F	320,190.45	288,768.27	299,324.12	297,781.03	286,883.85
500G	297,581.91	289,837.64	284,774.47	287,662.91	272,245.1
500H	306,348.30	290,103.21	285,561.60	306,535.95	288,081.2
500I	354,200.52	306,030.27	317,723.46	317,505.90	312,432.72
500J	358,170.29	317,555.44	324,100.50	327,707.62	320,874.68
500K	340,862.24	321,119.82	333,982.27	335,541.93	325,522.38
500L	367,537.72	341,679.23	342,197.85	343,070.33	340,637.9
1000A	679,881.34	624,851.46	649,543.14	650,289.36	611,868.59
1000B	699,519.74	630,541.26	651,908.47	665,876.33	632,445.34
1000C	699,470.19	649,349.17	665,312.94	671,929.89	640,195.74
1000D	737,182.91	684,514.11	702,616.43	709,351.04	666,153.32
1000E	538,231.53	510,476.69	514,780.74	522,147.09	487,533.19
1000F	538,643.55	508,335.69	510,358.70	516,767.07	489,148.95
1000G	604,860.62	568,809.61	575,902.95	573,739.30	563,114.29
1000H	633,859.94	595,803.34	603,011.87	615,230.83	576,942.64
1000I	664,037.75	615,801.13	621,463.50	632,385.84	601,199.97
1000J	694,509.92	645,407.87	656,925.10	662,147.83	644,769.31
1000K	687,095.20	640,430.62	650,918.21	657,444.02	632,351.63
1000L	731,057.43	687,109.86	695,735.80	697,583.29	680,735.90

Table 12: Solutions Li & Lim instances 100 nodes

	1-ALNS		2-ALNS		Long 2-ALNS	
Instance	Veh.	Costs	Veh.	Costs	Veh.	Costs
100C1-1	10	828.94	10	828.94	10	828.94
100C1-2	10	828.94	10	828.94	10	828.94
100C1-3	10	827.86	9	1035.35	9	1035.35
100C1-4	9	873.56	9	860.01	9	860.01
100C1-5	10	828.94	10	828.94	10	828.94
100C1-6	10	828.94	10	828.94	10	828.94
100C1-7	10	828.94	10	828.94	10	828.94
100C1-8	10	826.44	10	826.44	10	826.44
100C1-9	10	827.82	9	1000.60	9	1019.83
100C2-1	3	591.56	3	591.56	3	591.56
100C2-2	3	591.56	3	591.56	3	591.56
100C2-3	3	591.17	3	591.17	3	591.17
100C2-4	3	590.60	3	590.60	3	590.60
100C2-5	3	588.88	3	588.88	3	588.88
100C2-6	3	588.49	3	588.49	3	588.49
100C2-7	3	588.29	3	588.29	3	588.29
100C2-8	3	588.32	3	588.32	3	588.32
100R1-1	19	1650.80	19	1650.80	19	1650.80
100R1-2	17	1487.57	17	1487.57	17	1487.57
100R1-3	13	1292.68	13	1292.68	13	1292.68
100R1-4	9	1013.39	9	1013.39	9	1013.39
100R1-5	14	1377.11	14	1377.11	14	1377.11
100R1-6	12	1252.62	12	1252.62	12	1252.62
100R1-7	10	1111.31	10	1111.31	10	1111.31
100R1-8	9	968.97	9	968.97	9	968.97
100R1-9	11	1208.96	11	1208.96	11	1208.96
100R1-10	11	1165.83	10	1159.35	10	1159.35
100R1-11	10	1108.90	10	1108.90	10	1108.90
100R1-12	10	1027.12	9	1003.77	9	1003.77
100R2-1	4	1253.23	4	1253.23	4	1253.23
100R2-2	4	1239.95	3	1197.67	3	1197.67
100R2-3	3	949.40	3	949.40	3	949.40
100R2-4	2	849.05	2	849.05	2	849.05
100R2-5	3	1054.02	3	1054.02	3	1054.02
100R2-6	3	931.63	3	931.63	3	931.63
100R2-7	3	930.63	2	903.06	2	903.06
100R2-8	2	734.85	2	734.85	2	734.85
100R2-9	3	930.59	3	930.59	3	930.59
100R2-10	3	964.22	3	964.22	3	964.22
100R2-11	3	884.29	2	911.52	2	911.52
100RC1-1	14	1708.80	14	1708.80	14	1708.80
100RC1-2	12	1558.07	12	1558.07	12	1558.07
100RC1-3	11	1258.74	11	1258.74	11	1258.74
100RC1-4	10	1128.40	10	1128.40	10	1128.40
100RC1-5	13	1637.62	13	1637.62	13	1637.62
100RC1-6	11	1424.73	11	1424.73	11	1424.73
100RC1-7	11	1230.14	11	1230.14	11	1230.14
100RC1-8	10	1147.43	10	1147.43	10	1147.43
100RC2-1	4	1406.94	4	1406.94	4	1406.94
100RC2-2	4	1390.56	3	1374.27	3	1374.27
100RC2-3	3	1149.73	3	1089.07	3	1089.07
100RC2-4	3	818.66	3	818.66	3	818.66
100RC2-5	4	1302.20	4	1302.20	4	1302.20
100RC2-6	3	1159.03	3	1159.03	3	1159.03
100RC2-7	3	1062.05	3	1062.05	3	1062.05
100RC2-8	3	852.76	3	852.76	3	852.76

Table 13: Solutions Li & Lim instances 200 nodes

	1-ALNS		2-ALNS		Long ALNS	
Instance	Veh.	Costs	Veh.	Costs	Veh.	Costs
200C1-1	20	2704.57	20	2704.57	20	2704.57
200C1-2	19	2764.56	19	2764.56	19	2764.56
200C1-3	18	2772.18	17	3160.13	17	3158.24
200C1-4	17	2873.00	17	2728.63	17	2695.48
200C1-5	20	2702.05	20	2702.05	20	2702.05
200C1-6	20	2701.04	20	2701.04	20	2701.04
200C1-7	20	2701.04	20	2701.04	20	2701.04
200C1-8	20	2689.83	20	2689.83	20	2689.83
200C1-9	18	2724.24	18	2724.24	18	2752.83
200C1-10	18	2741.56	18	2859.33	17	2988.79
200C2-1	6	1931.44	6	1931.44	6	1931.44
200C2-2	7	1917.82	6	1881.40	6	1881.40
200C2-3	6	1848.37	6	1844.70	6	1844.33
200C2-4	7	1812.57	6	1773.35	6	1788.05
200C2-5	6	1891.21	6	1891.21	6	1891.21
200C2-6	6	1857.78	6	1857.78	6	1857.78
200C2-7	6	1850.13	6	1850.13	6	1850.13
200C2-8	6	1828.40	6	1877.88	6	1873.07
200C2-9	6	1854.21	6	1854.21	6	1854.21
200C2-10	6	1817.45	6	1817.45	6	1817.45
200R1-1	20	4819.12	20	4819.12	20	4819.12
200R1-2	19	4093.05	17	4777.49	17	4811.05
200R1-3	17	3563.99	15	3695.61	15	3655.42
200R1-4	13	2842.71	11	3306.30	10	3479.11
200R1-5	18	4226.61	17	4450.00	17	4396.81
200R1-6	17	3892.95	14	4362.36	14	4298.13
200R1-7	15	3346.80	12	4160.10	12	4333.32
200R1-8	11	2819.56	9	3255.31	9	2991.87
200R1-9	17	4138.91	15	4565.11	14	4804.52
200R1-10	15	3492.77	12	3807.78	12	3719.97
200R2-1	6	4187.00	5	4073.10	5	4073.10
200R2-2	5	3827.21	4	3796.00	4	3796.00
200R2-3	5	3156.27	4	3098.36	4	3098.36
200R2-4	4	2090.25	3	2500.75	3	2487.69
200R2-5	4	3463.20	4	3439.61	4	3439.60
200R2-6	4	3201.54	4	3201.54	4	3201.54
200R2-7	4	2620.31	3	3379.67	3	3280.30
200R2-8	4	1851.65	2	2653.28	2	2625.01
200R2-9	4	3340.11	4	3198.44	4	3198.44
200R2-10	4	2820.56	3	3518.42	3	3526.34
200RC1-1	19	3606.06	19	3606.06	19	3606.06
200RC1-2	18	3342.42	15	3784.71	15	3869.67
200RC1-3	15	3239.02	13	3305.19	13	3311.77
200RC1-4	12	2647.47	10	2879.65	10	2778.86
200RC1-5	17	3751.40	16	3752.13	16	3722.48
200RC1-6	17	3431.37	17	3384.26	17	3368.66
200RC1-7	16	3780.14	15	3594.96	15	3577.51
200RC1-8	15	3184.04	14	3328.27	14	3194.62
200RC1-9	15	3176.03	14	3452.42	14	3304.33
200RC1-10	14	2864.11	13	2933.44	13	2916.54
200RC2-1	7	2997.06	6	3699.26	6	3802.26
200RC2-2	6	2737.46	5	3194.82	5	3170.95
200RC2-3	6	2576.88	4	3023.62	4	2913.32
200RC2-4	5	2228.04	3	2915.10	3	3133.62
200RC2-5	6	2836.53	5	2777.64	5	2776.93
200RC2-6	5	2707.96	5	2707.96	5	2707.96
200RC2-7	5	2653.00	4	3222.62	4	3153.16
200RC2-8	5	2440.25	4	2402.61	4	2404.81
200RC2-9	4	2213.82	4	2208.97	4	2208.49
200RC2-10	4	2053.95	3	2696.88	3	2704.60

Table 14: Solutions Li & Lim instances 400 nodes

	1-ALNS		2-ALNS		Long ALNS	
Instance	Veh.	Costs	Veh.	Costs	Veh.	Costs
400C1-1	40	7152.06	40	7152.06	40	7152.06
400C1-2	40	7158.12	39	7733.65	39	7425.67
400C1-3	38	7416.79	34	9028.60	34	8870.55
400C1-4	33	7565.70	31	7580.75	31	7600.03
400C1-5	40	7150.00	40	7150.00	40	7150.00
400C1-6	40	7154.02	40	7161.38	40	7154.02
400C1-7	40	7149.43	40	7149.43	40	7149.43
400C1-8	40	7168.24	39	7421.63	39	7200.22
400C1-9	38	7748.93	38	8403.79	37	7963.70
400C1-10	37	8013.14	37	7807.29	36	7636.83
400C2-1	12	4116.33	12	4116.33	12	4116.33
400C2-2	13	4185.55	12	4204.05	12	4144.29
400C2-3	13	4210.04	13	4336.52	12	4707.87
400C2-4	13	4640.29	12	4861.46	12	4150.82
400C2-5	14	4278.41	13	4289.61	12	4030.63
400C2-6	13	3999.13	12	4002.70	12	3900.37
400C2-7	13	4353.09	13	4766.85	13	4032.19
400C2-8	12	3932.55	12	4091.51	12	3922.80
400C2-9	15	4597.90	13	5422.18	13	4484.52
400C2-10	13	3961.06	12	4030.77	12	4410.70
400R1-1	40	10,667.19	40	10,692.42	40	10,841.09
400R1-2	35	9678.68	32	10,906.81	32	10,668.97
400R1-3	30	8768.58	26	10,152.58	26	9076.90
400R1-4	22	7157.08	18	7995.45	17	7545.95
400R1-5	36	9781.85	31	11,536.7	31	11,316.31
400R1-6	32	10,569.74	27	10,420.38	27	9894.15
400R1-7	27	7979.94	22	9025.51	22	8666.66
400R1-8	20	7190.89	16	7382.87	16	7403.02
400R1-9	32	9827.41	28	10,885.50	27	10,157.94
400R1-10	26	8977.82	23	9192.11	22	9105.88
400R2-1	11	14,014.71	8	10,125.32	8	9763.13
400R2-2	9	8460.59	7	10,545.08	7	10,781.86
400R2-3	9	7017.65	6	10,289.48	6	9709.61
400R2-4	7	8841.78	4	7329.24	4	7844.36
400R2-5	9	12,872.78	7	9700.22	7	9833.15
400R2-6	9	6879.98	6	10,296.23	6	9451.36
400R2-7	7	10,243.93	5	7524.90	5	7510.99
400R2-8	6	7049.43	4	6900.07	4	5960.61
400R2-9	9	7098.10	6	9439.40	6	8877.99
400R2-10	8	9087.62	6	9744.25	6	9108.33
400RC1-1	37	9516.99	37	9087.48	37	9026.06
400RC1-2	35	8211.65	33	8210.66	33	8246.39
400RC1-3	30	8030.87	26	8752.01	26	8105.57
400RC1-4	22	6610.60	19	6541.76	19	6433.17
400RC1-5	37	9131.36	34	9342.58	34	9445.36
400RC1-6	35	8596.61	32	8796.54	32	8501.46
400RC1-7	34	8465.97	31	8532.92	31	8292.45
400RC1-8	32	8376.53	29	8611.99	29	8064.02
400RC1-9	32	8404.00	28	8978.38	29	8262.55
400RC1-10	28	7847.56	26	8678.26	26	7672.22
400RC2-1	14	6761.66	12	8104.64	12	8019.35
400RC2-2	13	6422.71	11	6631.5	11	6320.11
400RC2-3	11	6542.36	9	5640.26	9	5531.56
400RC2-4	8	3863.48	5	5599.23	5	5645.64
400RC2-5	13	6500.93	11	6206.89	11	6121.15
400RC2-6	12	10,179.96	10	7035.10	10	6199.30
400RC2-7	11	5933.95	8	7707.52	8	6655.21
400RC2-8	9	5729.47	7	6768.02	7	6726.78
400RC2-9	9	5008.95	7	6204.35	7	5846.56
400RC2-10	8	5047.7	7	6382.78	7	7680.89