Datalogisk Institut
KØBENHAVNS UNIVERSITET

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

Stefan Røpke, David Pisinger

Technical Report no. 2004/13
ISSN: 0107-8283

DIKU
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

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

Stefan Ropke, David Pisinger

8th October 2004

Abstract

The *pickup and delivery problem with time windows* is the problem of serving a number of transportation *requests* using a limited amount of vehicles. Each request involves moving a number of goods from a pickup location to a delivery location. Our task is to construct routes that visit all sites such that corresponding pickups and deliveries are placed on the same route and such that a pickup is performed before the corresponding delivery. The routes should furthermore satisfy a number of additional constraint such as time window and capacity constraints.

This paper presents a heuristic for the problem based on an extensions of the *Large Neighborhood Search* heuristic previously suggested for solving the vehicle routing problem with time windows. The proposed heuristic is composed of a number of competing sub-heuristics which are used with a frequency corresponding to their historic performance. This general framework is denoted *Adaptive Large Neighborhood Search*.

The heuristic is tested on more than 350 benchmark problems with up to 500 requests and it is able to improve the best known solutions from the literature for more than 50% of the problems.

The computational experiments indicate that it is advantageous to use several competing sub-heuristics instead of just one. Due to the many sub-heuristics the algorithm is very robust and can easily adapt to various instance characteristics.

Keywords: Pickup and Delivery Problem with Time Windows, Large Neighborhood Search, Simulated Annealing, Metaheuristics

Introduction

In the considered variant of the pickup and delivery problem with time windows (PDPTW), we are given n *requests* and m *vehicles*. A request $i \in \{1,\ldots,n\}$ consists of picking up l_i goods at one location and delivering these goods to another location. The two locations in a request will frequently be called *customers*. Two time windows are assigned to each request: A pickup time window that specifies when the goods can be picked up and a delivery time window that tells when the goods can be dropped of. Furthermore *service times* are associated with each pickup (s_i) and delivery (s'_i). The service time indicates how long time the pickup or delivery takes to perform. A vehicle is allowed to arrive at a location before the start of the time window of the location, but the vehicle must then wait until the start of the time window before initiating the operation. A vehicle may never arrive to a location after the end of the time window of the location.

Each request i has assigned a set of feasible vehicles F_i. A request i can only be served by a vehicle k if $k \in F_i$. This can for example be used to model situations where some vehicles cannot enter a certain location because of the dimensions of the vehicle. Requests where $F_i \neq \{1,\ldots,m\}$ are called *special requests*.

Each vehicle starts and ends its duty at given locations called *start* and *end terminals*. The start and end location do not need to be the same and two vehicles can have different start and end terminals. Each vehicle $k \in \{1,\ldots,m\}$ is able to carry a limited *capacity* C_k. Furthermore each vehicle has assigned a start and end time. The start time tells when the vehicle must leave its start location and the end time tells the latest allowable arrival at its end location. Note that the vehicle leaves its depot at the specified start time even though this might introduce waiting time at the first customer visited.

The problem is defined on a complete, directed graph $G = (V,E)$ where the node set of the graph $V = \{1,\ldots,2n+2m\}$ represent locations. We are given two $|V| \times |V|$ matrices $T = (t_{ij})$ and $D = (d_{ij})$ that specify

Datalogisk Institut
KØBENHAVNS UNIVERSITET

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

Stefan Røpke, David Pisinger

Technical Report no. 2004/13
ISSN: 0107-8283

DIKU
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

the travel time and distance associated with each edge $(i, j) \in E$. We assume that the travel times satisfy the triangle inequality and are non-negative. The only assumption about the distances is that they must be non-negative.

Our task is to construct valid routes for the vehicles. A route is valid if time windows and vehicle capacity are obeyed along the route, if each pickup is served before the corresponding delivery, if corresponding pickup and deliveries are served on the same route and if the vehicle only serves requests it is allowed to serve. The routes should be constructed such that they minimize the *cost* function to be described below.

As the number of vehicles is limited we might encounter situations where some requests cannot be assigned to a vehicle. These requests are placed in a virtual *request bank*. In a real world situation it is up to a human operator to decide what to do with such requests. The operator might for example decide to rent extra vehicles in order to serve the remaining requests.

The objective of the problem is to minimize a weighted sum consisting of the following three components: 1) The sum of the distance traveled by the vehicles. 2) The sum of the time spent by each vehicle, measured from the start time to of the vehicle to its arrival time at its end terminal. 3) The number of requests in the request bank. The three terms are weighted by the coefficients α, β and γ respectively. Normally a high value is assigned to γ in order to serve as many requests as possible.

The problem presented above was inspired from a real life vehicle routing problem related to transportation of raw materials and goods between production facilities of a major Danish food manufacturer. For confidentiality reasons, we are not able to present any data about the real life problem that motivated this research.

The problem is NP-hard as it for example contains the vehicle routing problem with time windows (VRPTW) [26] as a special case. The objective of this paper is to develop a method for finding good but not necessarily optimal solutions to the problem described above. The developed method should preferably be reasonably fast, robust and able to handle large problems. Thus it seems fair to turn to heuristic methods.

The next paragraphs surveys recent work on the PDPTW. Although none of the reference mentioned below consider exactly the same problem as ours, they all faces the same core problem.

Nanry and Barnes [13] are among the first to present a metaheuristic for the PDPTW. Their approach is based on a Reactive Tabu Search algorithm that combines several standard neighborhoods. In order to test the heuristic, Nanry and Barnes create PDPTW instances from a set of standard VRPTW problems proposed by Solomon [24]. The heuristic is tested on instances with up to 50 requests. Li and Lim [11] use a hybrid metaheuristic to solve the problem. The heuristic combines Simulated Annealing and Tabu search. Their method is tested on the 9 largest instances from Nanry and Barnes [13] and they consider 56 new test based on Solomon's VRPTW problems [24]. Lim, Lim and Rodrigues [12] apply "Squeaky wheel" optimization and local search to the PDPTW. Their heuristic is tested on the set of problems proposed by Li and Lim [11]. Lau and Liang [10] also apply Tabu search to PDPTW and they describe several construction heuristics for the problem. Special attention is given to how test problems can be constructed from VRPTW instances.

Recently Bent and Van Hentenryck [2] proposed a heuristic for the PDPTW based on Large Neighborhood Search. The heuristic was tested on the problems proposed by Li and Lim [11] and showed good performance. The heuristic by Bent and Van Hentenryck is probably the most promising metaheuristic for the PDPTW proposed so far.

Gendreau et al. [9] consider a dynamic version of the problem. An ejection chain neighborhood is proposed and steepest descent and Tabu search heuristics based on the ejection chain neighborhood are tested. Finally the Tabu search heuristic is tested in a parallelized version.

Several Column Generation methods for PDPTW have been proposed. These methods both include exact and heuristic methods. Dumas et al. [8] were the first to use column generation for solving PDPTW. They propose a branch and bound method that is able to handle problems with up to 55 requests.

Xu et al. [27] consider a PDPTW with several extra real-life constraints, for example multiple time windows, compatibility constraints and maximum driving time restrictions. The problem is solved using a column generation heuristic. The paper considers problem instances with up to 500 requests.

Sigurd et al. [22] solve a PDPTW problem related to transportation of livestock. This introduces some extra constraints, for example precedence relations among the requests, meaning that some requests should be served before others in order to avoid the spread of diseases. The problem is solved to optimality using column generation. The largest problems solved contain more than 200 requests.

A recent survey of pickup and delivery problem literature was made by Desaulniers et al. [7].

Algorithm 1 LNS heuristic

1 **Function** LNS($s \in \{solutions\}$, $q \in \mathbb{N}$)
2   solution s_{best}=s;
3   **repeat**
4     $s' = s$;
5   remove q requests from s'
6   reinsert removed requests into s';
7   **if** (f(s') < f(s_{best})) **then**
8       $s_{best} = s'$;
9   **if** accept(s', s) **then**
10       $s = s'$;
11   **until** stop-criterion met
12 **return** s_{best};

The work presented in this paper is based on the Masters Thesis of Ropke [17]. In the papers by Pisinger and Ropke [14], [18] it is shown how the heuristic presented in this paper can be extended to solve a variety of vehicle routing problems, for example the VRPTW, the *Multi Depot Vehicle Routing Problem* and the *Vehicle Routing Problem with Backhauls*.

The rest of this paper is organized as follows. Section 1 describes the basic solution method in a general context, Section 2 describes how the solution method has been applied to PDPTW and extensions to the method are presented. Section 3 contains the computational test. The computational test is focused on comparing the heuristic to existing metaheuristics and evaluating if the refinements presented in Section 2 improve the heuristic. Section 4 concludes the paper.

1 Solution method

Local search heuristics are often build on neighborhood moves that make small changes to the current solution, for example by moving a request from one route to another or exchanging two requests like it is done by for example Nanry and Barnes [13] and Li and Lim [11]. These kind of local search heuristics are able to investigate a huge number of solutions in short time, but a solution is only changed very little in each iteration. It is our belief, that such heuristics can have difficulties in moving from one promising area of the solution space to another, when faced with tightly constrained problems, even when embedded in metaheuristics.

One way of tackling this problem is by allowing the search to visit infeasible solutions by relaxing some constraints; see e.g. Cordeau et al. [5]. We take another approach — instead of using small "standard moves" we use very large moves that potentially can rearrange up to 30-40% of all requests in a single iteration. The price of doing this is that the computation time needed for performing and evaluating the moves becomes much larger compared to the smaller moves. The number of solutions evaluated by the proposed heuristic per time unit is only a fraction of the solutions that could be evaluated by a standard heuristic. Nevertheless very good performance is observed in the computational tests in Section 3.

The proposed heuristic is based on *Large Neighborhood Search (LNS)* introduced by Shaw [19]. The LNS heuristic has been applied to VRPTW with good results (see Shaw[19], [20] and Bent and Van Hentenryck [4]). Recently the heuristic has been applied to the PDPTW as well (Bent and Van Hentenryck [2]). The LNS heuristic itself is similar to the *ruin and recreate* heuristic proposed by Schrimpf et al. [21].

The pseudo-code for a minimizing LNS heuristic is shown in Algorithm 1. The pseudo-code assumes that an initial solution s already has been found, e.g. by a simple construction heuristic. The second parameter q determines the scope of the search.

Lines 5 and 6 in the algorithm are the interesting part of the heuristic. In line 5 a number of requests are removed from the current solution s' and in line 6 the requests are reinserted into the current solution again. The performance and robustness of the overall heuristic is very dependent on the choice of removal and insertion procedures. In the previously proposed LNS heuristics for VRPTW or PDPTW (see for example Shaw [19] or Bent and Van Hentenryck [2]) near-optimal methods were used for the reinsert operation. This was achieved using truncated branch and bound search. In this paper we take a different approach by using simple insertion heuristics for performing the insertions. Even though the insertion heuristics themselves usually deliver solutions of poor quality, the quality of the LNS heuristic is very good as the bad moves that are generated by the insertion heuristics lead to a fruitful diversification of the search process.

The rest of the code updates the so far best solution and determines if the new solution should be accepted. A simple accept criteria would be to accept all improving solutions. Such a criteria has been used in earlier LNS implementations (Shaw [19]). In this paper we use a simulated annealing accept criteria.

In line 11 we check if a stop criterion is met, in our implementation we stop when a certain number of iterations have been performed.

The parameter $q \in \{0, \cdots, n\}$ determines the size of the neighborhood. If q is equal to zero then no search at all will take place as no requests are removed. On the other hand if q is equal to n then the problem is resolved from scratch in each iteration. In general, one can say that the larger q is, the the easier it is to move around in the solution space, but when q gets larger each application of the insertion procedure is going to be slower. Furthermore if one uses a heuristic for inserting requests then choosing q too large might give bad results.

The LNS local search can be seen as an example of a very large scale neighborhood search as presented by Ahuja et al. in [1]. Ahuja et al. define very large scale neighborhoods as neighborhoods whose size grows exponentially as a function of the problem size, or neighborhoods that simply are too large to be searched explicitly in practice. The LNS local search fits into the last category as we have a large number of possibilities for choosing the requests to remove and a large number of possible insertions. One important difference between the proposed heuristic and most of the heuristics described in [1] is that the latter heuristics typically examine a huge number of solutions, albeit implicitly, while the LNS heuristic proposed in this paper only examines a relatively low number of solutions.

Instead of viewing the LNS process as a sequence of remove-insert operations it can also be viewed as a sequence of *fix-optimize* operations. In the fix operation a number of elements in the current solution are fixed. If for example the solution is represented as a vector of variables, the fix operation could fix a number of these variables at their current value. The optimize operation then re-optimizes the solution while respecting the fixation performed in the previous fix-operation. This way of viewing the heuristic might help us to apply the heuristic to problems where the remove-insert operations do not seem intuitive. One could use the term *fix-optimize* heuristic instead of LNS to avoid the confusion between the terms Large Neighborhood Search (LNS) which is a heuristic framework and very large scale neighborhoods (VLSN) which is a characterization of neighborhoods. We are going to use the term LNS in the rest of this paper though as it seems to be generally accepted in the scientific community. In Section 2.4 we introduce the term *Adaptive Large Neighborhood Search* (ALNS) to describe an algorithm using several large neighborhoods in an adaptive way.

2 LNS applied to PDPTW

This section describes how the LNS heuristic has been applied to the PDPTW. Compared to the LNS heuristic developed for the VRPTW and PDPTW by Shaw [19], [20] and Bent and Van Hentenryck [2], [4] the heuristic in this paper is different in several ways:

1. We are using several removal and insertion heuristics during the same search while the earlier LNS heuristics only used one method for removal and one method for insertions. The removal heuristics are described in Section 2.1 and the insertion heuristics are described in Section 2.2. The method for selecting which sub-heuristic to use is described in Section 2.3. The selection mechanism is guided by statistic information gathered during the search, as described in Section 2.4.

2. Simple and fast heuristics are used for the insertion of requests as opposed to the more complicated branch and bound methods proposed by Shaw [19], [20] and Bent and Van Hentenryck [2], [4].

3. The search is embedded in a simulated annealing metaheuristic where the earlier LNS heuristics used a simple descent approach. This is described in Section 2.5.

The present section also describes how the LNS heuristic can be used in a simple algorithm designed for minimizing the number of vehicles used to serve all requests.

Algorithm 2 Shaw Removal

```
1  Function ShawRemoval(s ∈ {solutions}, q ∈ ℕ, p ∈ ℝ+)
2    request : r = a randomly selected request from S;
3    set of requests : D = {r};
4    while |D| < q do
5      r = a randomly selected request from D;
6      Array : L = an array containing all request from s not in D;
7      sort L such that i < j ⇒ R(r,L[i]) < R(r,L[j]);
8      choose a random number y from the interval[0,1);
9      D = D ⋃ {L[y^p |L|]};
10   end while
11   remove the requests in D from s;
```

2.1 Request removal

This section describes three removal heuristics. All three heuristics take a solution and an integer q as input. The output of the heuristic is a solution where q requests have been removed. The heuristics *Shaw removal* and *Worst removal* furthermore have a parameter p that determines the degree of randomization in the heuristic.

2.1.1 Shaw removal heuristic

This removal heuristic was proposed by Shaw in [19, 20]. In this section it is slightly modified to suit the PDPTW. The general idea is to remove requests that somehow are similar, as we expect it to be reasonably easy to shuffle similar requests around and thereby creating new, perhaps better solutions. If we choose to remove requests that are very different from each other then we might not gain anything when reinserting the requests as we might only be able to insert the request at their original positions or in some bad positions. We define the similarity of two request i and j using a *relatedness measure* $R(i,j)$, the lower $R(i,j)$ is, the more related are the two requests.

The relatedness measure used in this paper consists of four terms: a distance term, a time term, a capacity term and a term that considers the vehicles that can be used to serve the two requests; these terms are weighted using the weights φ, χ, ψ and ω respectively. The relatedness measure is given by:

$$\begin{aligned} R(i,j) \;\; = \;\; & \varphi\left(d_{A(i),A(j)} + d_{B(i),B(j)}\right) + \chi\left(\left|T_{A(i)} - T_{A(j)}\right| + \left|T_{B(i)} - T_{B(j)}\right|\right) \\ & + \psi\left|l_i - l_j\right| + \omega\left(1 - \frac{\left|F_i \bigcap F_j\right|}{\min\left\{\left|F_i\right|, \left|F_j\right|\right\}}\right) \end{aligned} \tag{1}$$

$A(i)$ and $B(i)$ denote the pickup and delivery locations of request i and T_x indicates the time when location x is visited, d_{ij}, l_i and F_i are defined in the Introduction. The term weighted by φ measures distance, the term weighted by χ measures temporal connectedness, the term weighted by ψ compares capacity demand of the request and the term weighted by ω ensures that two request get a high relatedness measure if only a few or no vehicles are able to serve both requests. It is assumed that d_{ij}, T_x and l_i are normalized such that $0 \le R(i,j) \le 2(\varphi+\chi)+\psi+\omega$. This is done by scaling d_{ij}, T_x and l_i such that they only take on values from $[0,1]$. Notice that we cannot calculate $R(i,j)$ if request i or j is placed in the request bank.

The relatedness is used to remove customers in the same way as described by Shaw [19]. The procedure for removing customers is shown in pseudo code in Algorithm 2. The procedure initially chooses a random request to remove and in the subsequent iterations it chooses requests that are similar to the already removed requests. A determinism parameter $p \ge 1$ introduces some randomness in the selection of the requests (a low value of p corresponds to much randomness).

Notice that the sorting in line 7 can be avoided in an actual implementation of the algorithm as it is sufficient to use a linear time selection algorithm [6] in line 9.

Algorithm 3 Worst Removal

```
Function WorstRemoval(s ∈ {solutions}, q ∈ ℕ, p ∈ ℝ+)
  while q > 0 do
    Array : L = All planned requests i, sorted by descending cost(i,s);
    choose a random number y in the interval[0,1);
    request : r = L[y^p |L|];
    remove r from solution s;
    q = q − 1;
  end while
```

2.1.2 Random removal

The random removal algorithm simply selects q requests at random and removes them from the solution. The random removal heuristic can be seen as a special case of the Shaw removal heuristic with $p = 1$. We have implemented a separate random removal heuristic though, as it obviously can be implemented to run faster than the Shaw removal heuristic.

2.1.3 Worst removal

Given a request i served by some vehicle in a solution s we define the *cost* of the request as $cost(i,s) = f(s) - f_{-i}(s)$ where $f_{-i}(s)$ is the cost of the solution without request i (the request is not moved to the request bank, but removed completely). It seems reasonable to try to remove requests with high cost and inserting them at another place in the solution to obtain a better solution value, therefore we propose a removal heuristic that removes requests with high cost $cost(i,s)$.

The worst removal heuristic is shown in pseudo-code in Algorithm 3, it reuses some of the ideas from Section 2.1.1.

Notice that the removal is randomized, with the degree of randomization controlled by the parameter p like in Section 2.1.1. This is done to avoid situations where the same requests are removed over and over again.

One can say that the Shaw removal heuristic and the worst removal heuristic belong to two different classes of removal heuristics. The Shaw heuristic is biased towards selecting requests that "easily" can be exchanged, while the worst-removal selects the requests that appear to be placed in the wrong position in the solution.

2.2 Inserting requests

Insertion heuristics for vehicle routing problems are typically divided into two categories; *sequential* and *parallel* insertion heuristics. The difference between the two classes is that sequential heuristics build one route at a time while parallel heuristics construct several routes at the same time. Parallel and sequential insertion heuristics are discussed in further detail in [15]. The heuristics presented in this paper are all parallel. The reader should observe that the insertion heuristic proposed here will be used in a setting where they are given a number of partial routes and a number of requests to insert — they seldom build the solution from scratch.

2.2.1 Basic greedy heuristic

The basic greedy heuristic is a simple construction heuristic. It performs at most n iterations as it inserts one request in each iteration. Let $\Delta f_{i,k}$ denote the change in objective value incurred by inserting request i into route k at the position that increases the objective value the least. If we cannot insert request i in route k then we set $\Delta f_{i,k} = \infty$. We then define c_i as $c_i = \min_{k\in R}\{\Delta f_{i,k}\}$ where R is the set of routes. In other words, c_i is the "cost" of inserting request i at its best position overall. We denote this position by *the minimum cost position*. Finally we choose the request i that minimizes

$$\min_{i \in U} \; c_i \tag{2}$$

where U is the set of unplanned requests and inserts it at its minimum cost position. This process continues until all requests have been inserted or no more requests can be inserted.

Observe that we in each iteration only change one route (the one we inserted into), and we do not have to recalculate insertion cost in all the other routes. This property is used in the concrete implementation to speed up the insertion heuristics.

An obvious problem with this heuristic is that it often postpones the placement of "hard" requests (requests which are expensive to insert, that is requests with large c_i) to the last iterations where we do not have many opportunities for inserting the requests as many of the routes are "full". The heuristic presented in the next section tries to circumvent this problem.

2.2.2 Regret heuristics

The *regret* heuristic tries to improve upon the basic greedy heuristic by incorporating a kind of look ahead information when selecting the request to insert. Let $x_{ik} \in \{1,...,m\}$ be a variable that indicates the route for which request i has the k'th lowest insertion cost, that is $\Delta f_{i,x_{ik}} \leq \Delta f_{i,x_{ik'}}$ for $k \leq k'$. Using this notation we can express c_i from Section 2.2.1 as $c_i = \Delta f_{i,x_{i1}}$. In the regret heuristic we define a *regret value* c_i^* as $c_i^* = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}$. In other words, the regret value is the difference in the cost of inserting the request in its best route and its second best route. In each iteration the regret heuristic chooses to insert the request i that maximizes

$$\max_{i \in U} \quad c_i^*$$

The request is inserted at its minimum cost position; ties are broken by selecting the insertion with lowest cost. Informally speaking, we choose the insertion that we will regret most if it is not done now.

The heuristic can be extended in a natural way to define a class of regret heuristics: The *regret-k* heuristic is the construction heuristic that in each construction step chooses to insert the request i that maximizes:

$$\max_{i \in U} \quad \left\{ \sum_{j=1}^{k} \left(\Delta f_{i,x_{ij}} - \Delta f_{i,x_{i1}} \right) \right\} \tag{3}$$

Ties are broken by selecting the request with best insertion cost. The request is inserted at its minimum cost position. The regret heuristic presented at the start of this section is a regret-2 heuristic and the basic insertion heuristic from Section 2.2.1 is a regret-1 heuristic because of the tie-breaking rules. Informally speaking, heuristics with $k > 2$ investigate the cost of inserting a request on the k best routes and chooses to insert the request whose cost difference between inserting it into the best route and the $k-1$ best routes is largest. Compared to a regret-2 heuristic, regret heuristics with large values of k discover earlier that the possibilities for inserting a requests becomes limited.

Regret heuristics have been used by Potvin and Rousseau [15] for the VRPTW. The heuristic in their paper can be categorized as a regret-k heuristic with $k = m$ as all routes are considered in an expression similar to (3). The authors do not use the change in the objective value for evaluating the cost of an insertion, but use a special cost function. Regret heuristics can also be used for combinatorial optimization problems outside the vehicle routing domain, an example of an application to the Generalized Assignment Problem was described by Trick [25].

Like in the previous section we use the fact that we only change one route in each iteration to speed up the regret heuristic.

2.3 Choosing a removal and an insertion heuristic

In Section 2.1 we defined three removal heuristics (`shaw`, `random` and `worst` removal), and in Section 2.2 we defined a class of insertion heuristics (`basic insertion`, `regret-2`, `regret-3`, etc.). One could select one removal and one insertion heuristic and use these throughout the search, but in this paper we propose to use all heuristics. The reason for doing this is that for example the regret-2 heuristic may be well suited for one type of instance while the regret-4 heuristic may be the best suited heuristic for another type of instance. We believe that alternating between the different removal and insertion heuristics give us a more robust heuristic overall.

In order to select the heuristic to use, we assign weights to the different heuristics and use a *roulette wheel selection principle*. If we have k heuristics with weights w_i, $i \in \{1,2,\cdots,k\}$ we select heuristic j with probability

$$\frac{w_j}{\sum_{i=1}^{k} w_i} \tag{4}$$

Notice that the insertion heuristic is selected independently of the removal heuristic (and vice versa). It is possible to set these weights by hand but it can be quite involved process if many removal and insertion heuristics are used. Instead an adaptive weights adjusting algorithm is proposed in Section 2.4.

2.4 Adaptive weights adjustment

This section describes how the weights w_j introduced in Section 2.3 can be automatically adjusted using statistic information from earlier iterations. The basic idea is to keep track of a score for each heuristic. The score measures how well the heuristic has performed recently, a high score corresponds to a successful heuristic. The entire search is divided into a number of *segments*. A segment is just a number of iterations of the LNS heuristic; here we define a segment as 100 iterations. The score of all heuristics is set to zero at the start of each segment. The score of a heuristic is increased by either σ_1, σ_2 or σ_3 in the following situations:

Parameter	Description
σ_1	The last remove-insert operation resulted in a new global best solution.
σ_2	The last remove-insert operation resulted in a solution that has not been accepted before. The cost of the new solution is better than the cost of current solution.
σ_3	The last remove-insert operation resulted in a solution that has not been accepted before. The cost of the new solution is worse than the cost of current solution, but the solution was accepted.

The case for σ_1 should be pretty clear: If a heuristic is able to find a new overall best solution, then it has done well. Similarly if a heuristic has been able to find a solution that has not been visited before and it is accepted by the accept criteria in the LNS search then the heuristic has been successful as it has brought the search forward. It seems sensible to distinguish between the two situations corresponding to parameter σ_2 and σ_3 because we prefer heuristics that can improve the solution, but we are also interested in heuristics that can diversify the search, these are rewarded by σ_3. It is important to note that we only reward unvisited solutions, this is to encourage heuristics that are able to explore new parts of the solution space. We keep track of visited solutions by assigning a hash key to each solution and storing the key in a hash table.

In each iteration we apply two heuristics: a removal heuristic and an insertion heuristic. The scores for both heuristics are updated by the same amount as we can not tell whether it was the removal or the insertion that was the reason for the "success".

At the end of each segment we calculate new weights using the recorded scores. Let w_{ij} be the weight of heuristic i used in segment j as the weight is used in equation (4). In the first segment we weight all heuristics equally. After we have finished segment j we calculate the weight for all heuristics i to be used in segment $j+1$ like this:

$$w_{i,j+1} = w_{ij}(1-r) + r\frac{S_i}{a_i}$$

S_i is the score of heuristic i obtained during the last segment and a_i is the number of times we have attempted to use heuristic i during the last segment. The *reaction factor* r controls how quickly the weight adjustment algorithm reacts to changes in the effectiveness of the heuristics. If r is zero then we do not use the scores at all and stick to the initial weights. If r is set to one then we let the score obtained in the last segment decide the weight.

Figure 1 shows an example of how the weights of the three removal heuristics progress over time for a certain problem instance. The plots are decreasing because of the simulated annealing acceptance criteria to be described in the next section; towards the end of the search we only accept good moves and therefore it is harder for the heuristic to get high scores.

We are going to use the term *Adaptive Large Neighborhood Search* (ALNS) heuristic for a LNS heuristic that uses several competing removal and insertion heuristics and chooses between them using the mechanism described in this and the previous section.

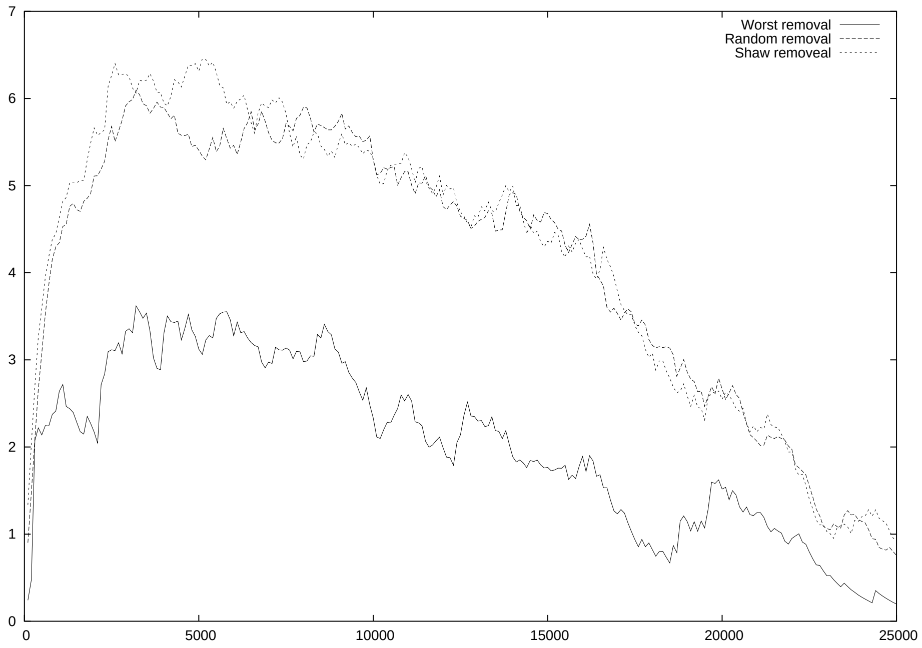

Figure 1: The figure shows an example of how the weights for the three removal heuristics progressed during one application of the heuristic. The iteration number is shown along the *x*-axis and the weight is shown along the *y*-axis. The graph illustrates that for the particular problem, the *random* removal and the *Shaw* removal heuristics perform virtually equally well, while the *worst* heuristic performs worst. Consequently the *worst* heuristic is not used as often as the two other heuristics.

2.5 Acceptance and stopping criteria

As described in Section 1 a simple acceptance criteria would be to only accept solutions that are better than the current solution. This would give us a descent heuristic like the one proposed by Shaw [19]. Such a heuristic has a tendency to get trapped in local minima so it seems sensible to accept solutions that are worse than the current solution from time to time. To do this, we use the acceptance criteria from simulated annealing. That is we accept a solution s' given the current solution s with probability $e^{-\frac{f(s')-f(s)}{T}}$ where $T > 0$ is the *temperature*.

The temperature starts out at T_{start} and is decreased every iteration using the expression $T = T \cdot c$, where $0 < c < 1$ is the *cooling rate* . A good choice of T_{start} is dependent on the problem instance at hand, so instead of specifying T_{start} as a parameter we calculate T_{start} by inspecting our initial solution: First we calculate the cost z' of this solution using a modified objective function. In the modified objective function, γ (cost of having requests in the request bank) is set to zero. The start temperature is now set such that a solution that is w percent worse than the current solution is accepted with probability 0.5. The reason for setting γ to zero is that this parameter typically is large and could cause us to set the starting temperature to a too large number if the initial solution had some requests in the request bank. Now w is a parameter that has to be set. We denote this parameter the *start temperature control parameter*.

The algorithm stops when a specified number of LNS iterations have passed.

2.6 Applying noise to the objective function

As the proposed insertion heuristics are quite myopic, it could be a good idea to try to add some diversification to the heuristics such that they not always make the move that seems best locally. This is achieved by adding a noise term to the objective function. Every time we calculate the cost C of an insertion of a request into a route, we also calculate a random number *noise* in the interval $[-maxN, maxN]$ and calculate the modified insertion costs $C' = max\{0, C + noise\}$. At each iteration we decide if we should use C or C' to determine the insertions to perform. This decision is taken by the adaptive mechanism described earlier by keeping track of how often the noise applied insertions and the "clean" insertions are successful.

In order to make the amount of noise related to the properties of the problem instance we calculate $maxN = \eta \cdot \max_{i,j \in V} \{d_{ij}\}$, where η is a parameter that controls the amount of noise. We have chosen to let $maxN$ be dependent on the distances d_{ij} as the distances are an important part of the objective in all of the problems we consider in this paper.

It might seem superfluous to add noise to the insertion heuristics as the heuristics are used in a simulated annealing framework that already contains randomization. We believe that the noise applications are important as our neighborhood is searched by means of the insertion heuristics and not randomly sampled. Without the noise applications we do not get the full benefit of the simulated annealing metaheuristic. This conjecture is supported by the computational experiments reported in table 3.

2.7 Minimizing the number of vehicles used

In the vehicle routing literature one often tries to minimize the number of vehicles used to serve all customers. The heuristic proposed so far is not able to cope with such an objective, but by using a simple two stage algorithm that minimizes the number of vehicles in the first stage and then minimizes a secondary objective (typically traveled distance) in the second stage we can handle such problems. A two-stage method was also used by Bent and Van Hentenryck [4], [2], but while they used two different neighborhoods and metaheuristics for the two stages we use the same heuristic in both stages.

The vehicle minimization stage works as follows: First an initial feasible solution is created using a sequential insertion method that constructs one route at a time until all customers have been planned. The number of vehicles used in this solution is the initial estimate on the number of vehicles necessary. Next step is to remove one route from our feasible solution. The requests on the removed route are placed in the request bank. The resulting problem is solved by our LNS heuristic. When the heuristic is run, a high value is assigned to γ such that requests are moved out of the request bank if possible. If the heuristic is able to find a solution that serves all requests, a new candidate for the minimum number of vehicles have been found. When such a solution has been found, the LNS heuristic is immediately stopped, one more route is removed from the solution and the process is reiterated. If the LNS heuristic terminates without finding a solution where all requests are served, then the algorithm steps back to the last solution encountered in which all requests were served. This solution is used as starting solution in the second stage of the algorithm which simply consists of applying the normal LNS heuristic.

In order to keep the running time of the vehicle minimization stage down, this stage is only allowed to spend Φ LNS iterations all together such that if the first application of the LNS heuristic for example spends a iterations to find a solution where all requests are planned, then the vehicle minimization stage is only allowed to perform $\Phi - a$ LNS iterations to minimize the number of vehicles further. Another way to keep the running time limited is to stop the LNS heuristic when it seems unlikely that a solution exists in which all requests are planned. In practice this is implemented by stopping the LNS heuristic if 5 or more requests are unplanned and no improvement in the number of unplanned requests has been found in the last τ LNS iterations. In the computational experiments Φ was set to 25000 and τ was set to 2000.

3 Computational experiments

In this section we describe our computational experiments. We first introduce a set of tuning instance in Section 3.1. In Section 3.2 we evaluate the performance of the proposed construction heuristics on the tuning instances. In Section 3.3 we describe how the parameters of the ALNS heuristic were tuned, and in Section 3.4 we present the results obtained by the ALNS heuristic and simpler LNS heuristics.

3.1 Tuning instances

First a set of representative tuning instances is identified. The tuning instances should have a fairly limited size as we want to perform numerous experiments on the tuning problems and they should somehow be related to the problems our heuristic is targeted at. In the case at hand we want to solve some standard benchmark instances and a new set of randomly generated instances.

Our tuning set consists of 16 instances. The first four instances are LR1_2_1, LR202, LRC1_2_3, and LRC204 from Li and Lim's benchmark problems [11], containing between 50 and 100 requests. The maximum allowable vehicles were set to one more than reported by Li and Lim to make it easier for the heuristic to find solutions with no requests in the request bank. The last 12 instances are new randomly generated instances.

These instances contain both single depot and multi depot problems and problems with requests that only can be served by a subset of the vehicle fleet. All randomly generated problems contain 50 requests. The random instances were generated with the problem generator described in [16].

3.2 Evaluation of construction heuristics

First we examine how the simple construction heuristics from Section 2.2 perform on the tuning problems, to see how well they work without the LNS framework. The construction heuristics regret-1, regret-2, regret-3, regret-4 and regret-m have been implemented. Table 1 shows the result of the test. As the construction heuristics are deterministic, the results were produced by applying the heuristics to each of the 16 test problems once.

	Basic greedy	Regret-2	Regret-3	Regret-4	Regret-m
Avg. gap (%)	40.7	30.3	26.3	26.0	27.7
Fails	3	3	3	2	0
Time (s)	0.02	0.02	0.02	0.02	0.03

Table 1: Performance of construction heuristics. Each column in the table corresponds to one of the construction heuristics. These simple heuristics were not always able to construct a solution where all requests are served, hence for each heuristic we report the number of times this happened in the *fails* row. The *Avg. gap* row shows the average relative difference between the found solution and the best known solution. Only solutions where all requests are served are included in the calculations of the average relative difference. The last row shows the average time (in seconds) needed for applying the heuristic to one problem, running on a 1.5 GHz Pentium 4.

The results show that the proposed construction heuristics are very fast, but also very imprecise. Basic greedy is the worst heuristic, while all the regret heuristics are comparable with respect to the solution quality. Regret-m stands out though, as it is able to serve all requests in all problems. It would probably be possible to improve the results shown in Table 1 by introducing seed customers as proposed by e.g. Solomon [24]. We are not going to report on such experiments in this paper though. It might be surprising that these very imprecise heuristics can be used as the foundation of a much more precise local search heuristic, but as we are going to see in the following, this is indeed possible.

3.3 Parameter tuning

This part of the paper serves two purposes. First it describes how the parameters used for producing the results in Section 3.4 were found. Next, it tries to unveil which part of the heuristic that contribute most to the solution quality.

3.3.1 Parameters

This section determines the parameters that need to be tuned. We first review the removal parameters. Shaw removal is controlled by five parameters: φ, χ, ψ, ω and p, while the worst removal is controlled by one parameter p_{worst}. Random removal has no parameters. The insertion heuristics are parameter free when we have chosen the regret degree.

In order to control the acceptance criteria we use two parameters, w and c. The weight adjustment algorithm is controlled by four parameters, σ_1, σ_2, σ_3 and r. Finally we have to determine a noise rate η and a parameter ξ that controls how many request we remove in each iteration. In each iteration, we chose a random number τ that satisfies $4 \leq \tau \leq \min(100, \xi n)$ and remove τ requests.

We stop the search after 25000 LNS iterations as this resulted in a fair tradeoff between time and quality.

3.3.2 LNS parameter tuning

Despite the large number of parameters used in the LNS heuristic, it turns out that it is relatively easy to find a set of parameters that work well for a large range of problems. We use the following strategy for tuning the parameters: First a fair parameter setting is produced by an ad-hoc trial-an-error phase, this parameter setting was found while developing the heuristic. This parameter setting is improved in the second phase by allowing one parameter to take a number of values while the rest of the parameters are kept fixed. For each parameter setting we apply the heuristic on our set of test problems five times, and the setting that shows the best average behavior (in terms of average deviation from the best known solutions) is chosen. We now move on to the next parameter, using the values found so far and the values from the initial tuning for the parameters that have not been considered yet. This process continues until all parameters have been tuned. One might go over the parameters once again using the new set of parameters as starting point to further optimize the parameters, but we stopped after one pass.

One of the experiments performed during the parameter tuning sought to determine the value of the parameter ξ that controls how many requests we remove and insert in each iteration. This parameter should intuitively have a significant impact on the results our heuristic is able to produce. We tested the heuristic with ξ ranging from 0.05 to 0.5 with a step size of 0.05. Table 2 shows the influence of ξ. When ξ is too low the heuristic is not able to move very far in each iteration, and it has a higher chance of being trapped in one suboptimal area of the search space. On the other hand, if ξ is large then we can easily move around in the search space, but we are probably stretching the capabilities of our insertion heuristics. The insertion heuristics work fairly well when they must insert a limited number of requests into a partial solution, but they cannot build a good solution from scratch as seen in Section 3.2. The results in Table 2 shows that $\xi = 0.4$ is a good choice. One should notice that the heuristic gets slower when ξ increases because the removals and insertions take longer time when more requests are involved, thus the comparison in Table 2 is not completely fair.

ξ	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
Avg. gap (%)	1.75	1.65	1.21	0.97	0.81	0.71	0.81	0.49	0.57	0.57

Table 2: Parameter ξ vs. solution quality. The first row shows the value of the parameter ξ that was tested and the second row shows the gap between the average solution obtained and the best solutions produced in the experiment.

The complete parameter tuning resulted in the following parameter vector (φ, χ, ψ, ω, p, p_{worst}, w, c, σ_1, σ_2,σ_3, r, η, ξ) = (9, 3, 2, 5, 6, 3, 0.05, 0.99975, 33, 9, 13, 0.1, 0.025, 0.4). Our experiments also indicated that it was possible to improve the performance of the vehicle minimization algorithm by setting (w, c) = (0.35, 0.9999) while searching for solutions that serve all requests. This corresponds to a higher start temperature and a slower cooling rate. This indicates that more diversification is needed when trying to minimize the number of vehicles compared to the situation where one just minimizes the traveled distance.

3.3.3 LNS configurations

This section evaluates how the different removal and insertion heuristics behave when used in a LNS heuristic. In most of the test cases a simple LNS heuristic was used that only involved one removal heuristic and one insertion heuristic. Table 3 shows a summary of this experiment.

The first six experiments aim at determining the influence of the removal heuristic. We see that Shaw removal performs best, the worst removal heuristic is second, and the random removal heuristic gives the worst performance. This is reassuring as it shows that the two slightly more complicated removal heuristics actually are better than the simplest removal heuristic. These results also illustrate that the removal heuristic can have a rather large impact on the solution quality obtained, thus experimenting with other removal heuristics would be interesting and could prove beneficial.

The next eight experiments show the performance of the insertion heuristics. Here we have chosen Shaw removal as removal heuristic because it performed best in the previous experiments. In these experiments we see that all insertion heuristics perform quite well, and they are quite hard to distinguish from each other. Regret-3 and Regret-4 coupled with noise addition are slightly better than the rest though. An observation that applies to all experiments is that application of noise seems to help the heuristic. It is interesting to note that the basic insertion heuristic nearly performs as well as the regret heuristics when used in a LNS framework. This is surprising seen in the light of Table 1 where the basic insertion heuristic performed particularly bad. This observation may indicate that the LNS method is relatively robust with respect to the insertion method used.

The last row of the table shows the performance of ALNS. As one can see, it is on par with the two best simple approaches, but not better, which at first may be a little disappointing. The result show though, that the adaptive mechanism is able to find a sensible set of weights, and it is our hypothesis that the ALNS heuristic

	Conf.	Shaw	Rand	Worst	Reg-1	Reg-2	Reg-3	Reg-4	Reg-*m*	Noise	Avg. gap (%)
LNS	1			•		•					2.7
	2			•		•				•	2.6
	3		•			•					5.4
	4		•			•				•	3.2
	5	•				•					2.0
	6	•				•				•	1.6
	7	•			•						2.2
	8	•			•					•	1.6
	9	•					•				1.8
	10	•					•			•	**1.3**
	11	•						•			2.0
	12	•						•		•	**1.3**
	13	•							•		1.8
	14	•							•	•	1.7
ALNS	15	•	•	•	•	•	•	•	•	•	**1.3**

Table 3: Simple LNS heuristics compared to the full adaptive LNS with dynamic weight adjustment. The first column shows if the configuration should be considered as a LNS or a ALNS heuristic. The second column is the configuration number, columns three to five indicate which removal heuristics that were used. Columns six to ten indicate which insertion heuristics that were used. Column eleven states if noise were added to the objective function during insertion of customers (in this case noise was added to the objective function in 50% of the insertions for the simple configurations 1-14 while in configuration 15 the number of noise-insertions was controlled by the adaptive method). Column twelve shows the average performance of the different heuristics. As an example, in configuration four we used random removal together with the regret-2 insertion heuristic and we applied noise to the objective value. This resulted in a set of solutions whose objective values on average were 3.2% above the best solutions found during the whole experiment.

is more robust than the simpler LNS heuristics. That is, the simple configuration may fail to produce good solutions on other types of problems while the ALNS heuristic continues to perform well. One of the purposes of the experiments in Section 3.4 is to confirm or disprove this hypothesis.

3.4 Results

This section provides computational experiments performed to test the performance of the heuristic. There are three major objectives for this section:

1. To compare the ALNS heuristic to a simple LNS heuristic that only contains one removal and one insertion heuristic.
2. To determine if certain problem properties influence the (A)LNS heuristics ability to find good solutions.
3. To compare the ALNS heuristic with state-of-the-art PDPTW heuristics from the literature.

As the model considered in this paper is quite complicated, it is hard to find any benchmark instances that consider exactly the same model and objective function. The benchmark instances that come closest to the model considered in this paper are the instances constructed by Nanry and Barnes [13] and the instances constructed by Li and Lim [11]. Both data sets are single depot pickup and delivery problems with time windows, constructed from VRPTW problems. We are only reporting results on the data set proposed by Li and Lim, as the Nanry and Barnes instances are easy to solve due to their small size.

In order to clarify if ALNS heuristic is worthwhile compared to a simpler LNS heuristic we are going to show results for both the ALNS heuristic and the best simple LNS heuristic from Table 3. Configuration 12 was chosen as representative for the simple LNS heuristics as it performed slightly better than configuration 10. In the following we refer to the full and simple LNS heuristic as ALNS and LNS respectively.

All experiments were performed on a 1.5 GHz Pentium IV PC with 256 MB internal memory, running Linux. The implemented algorithm measures travel times and distances using double precision floating point numbers. The parameter setting found in Section 3.3.2 was used in all experiments unless otherwise stated.

3.4.1 Comparing ALNS and LNS using the Li & Lim instances

This section compares the ALNS and LNS heuristics using the benchmark instances proposed by Li and Lim [11]. The data set contains 354 problems with between 100 and 1000 customers. The data set can be downloaded from [23].

In this section we use the distance driven as our objective even though vehicle minimization is the standard primary objective for these instances. The reason for this decision is that distance minimization makes comparison of the heuristics easier and distance minimization is the original objective of the proposed heuristic. The number of vehicle available for serving the customers are set to the minimum values reported by Li and Lim in [11] and on their web page which unfortunately no longer is online.

The heuristics were applied 10 times to each instance with 400 or less customers and 5 times to each instance with more than 400 customers. The experiments are summarized in Table 4.

		Best known solutions		Avg. gap (%)		Average time (s)		Fails	
#customers	#problems	ALNS	LNS	ALNS	LNS	ALNS	LNS	ALNS	LNS
100	56	52	50	0.19	0.50	49	55	0	0
200	60	49	15	0.72	1.41	305	314	0	0
400	60	52	6	2.36	4.29	585	752	0	0
600	60	54	5	0.93	3.20	1069	1470	0	0
800	60	46	5	1.73	3.27	2025	3051	0	2
1000	58	47	4	2.26	4.22	2916	5252	0	1

Table 4: Summary of results obtained on Li and Lim instances. The first column gives the problem size; the next column indicates the number of problems in the data set of the particular size. The rest of the table consists of four major columns, each divided into two sub columns, one for the ALNS and one for LNS. The column *Best known solutions* indicates for how many problems the best known solution was identified. The best known solution is either the solution reported by Li and Lim or the best solution identified by the (A)LNS heuristics depending on which is best. The next column indicates how far the average solution is from best known solution, this number is averaged over all problems of a particular size. The next column shows how long time the heuristic on average spends to solve a problem. The last column shows the number of times the heuristic failed to find a solution where all request are served by the given number of vehicles in all the attempts to solve a particular problem.

The results show that the ALNS heuristic on all four terms performs better than the LNS heuristic. One also notices that the ALNS heuristic becomes even more attractive as the problem size increases. It may seem odd that the LNS heuristic spends more time compared to the ALNS heuristic when they both perform the same number of LNS iterations. The reason for this behavior is that the Shaw removal heuristic used by the LNS heuristic is more time consuming compared to the two other removal heuristics.

3.4.2 New instances

This section provides results on some new problem instances that contain features of the model that were not used in the standard benchmark problems considered in Section 3.4.1. These features are: multiple depots, routes with different start and end terminals and *special* requests that only can be served by a certain subset of the vehicles. Furthermore three types of geographical distributions of customers are considered: problems with customers distributed uniformly in the plane, problems with customers located in 10 clusters and problems with 50% of the customers located in 10 clusters and 50% of the customers distributed uniformly. These three types of customer distributions were inspired by Solomon's VRPTW benchmark problems [24], and the problems are similar to the R, the C and the RC Solomon problems respectively. We consider problems with 50, 100, 250 and 500 requests, all problems are multi depot problems. For each problem size we generated 12 problems as we tried every combination of the three problem features shown below:

- Route type: 1) A route starts and ends at the same location. 2) A route starts and ends at different sites.
- Request type: 1) All requests are normal requests. 2) 50% of the requests are *special* requests. The special requests can only be served by a subset of the vehicles. In the test problems each special request could only be served by between 30 to 60 percent of the vehicles.

- Customer distributions: 1) Uniform, 2) Clustered, 3) Semi-clustered.

The way the instances were generated is described in further detail in [16], the instances and the instance generator can be downloaded from www.diku.dk/~sropke. The heuristics were tested by applying them to each of the 48 problems 10 times. Table 5 shows a summary of the results found, the complete results can be found in [16].

We observe the same tendencies as in Table 4; ALNS is still superior to LNS, but one notices that the gap in solution quality between the two methods are smaller for this set of instances while the difference in running time is larger compared to the results on the Li and Lim instances. One also notices that it seems harder to solve small instances of this problem class compared to the Li and Lim instances.

		Best known solutions		Avg. gap (%)		Average time (s)	
#requests	#problems	ALNS	LNS	ALNS	LNS	ALNS	LNS
50	12	8	5	1.44	1.86	23	34
100	12	11	1	1.54	2.18	83	142
250	12	7	5	1.39	1.62	577	1274
500	12	9	3	1.18	1.32	3805	8146
Sum:	48	35	14	5.55	6.98	4488	9596

Table 5: Summary of results obtained on new instances. The captions of the table should be interpreted as in Table 4. The last row sums each column. Notice that the size of the problems in this table is given as number of requests and not the number of customers.

Table 6 summarizes how the problem features influence the average solution quality. These results show that the clustered problems are the hardest to solve, while the uniformly distributed instances are the easiest. The results also indicate that special requests make the problem slightly harder to solve, probably due to the fact that the special requests make the problem more constrained. The route type experiments compare the situation where routes start and end at the same place (the typical situation considered in the literature) to the situation were each route start and end at different places. Here we expect the last case to be the easiest to solve, as we by having different start and end positions for our routes gain information about the area the route most likely should cover. The results in Table 6 confirm these expectations.

In addition to investigate the question of how the model features influence the average solution quality obtained by the heuristics we also want to know if the presence of some features could make LNS behave better than ALNS. For the considered features the answer is negative.

Feature	ALNS	LNS
Distribution: Uniform	1.04%	1.50%
Distribution: Clustered	1.89%	2.09%
Distribution: Semi-clustered	1.23%	1.64%
Normal Requests	1.24%	1.47%
Special Requests	1.54%	2.02%
Start of route = end of route	1.59%	2.04%
Start of route $\neq$ end of route	1.19%	1.45%

Table 6: Summary of the influence of certain problem features on the heuristic solutions. The two columns correspond to the two heuristic configurations. Each row shows the average solution quality for each feature. The average solution quality is defined as the average of the average gap for all instances with a specific feature. To be more precise, the solution quality is calculated using the formula: $q(h) = \frac{1}{|F|}\sum_{i \in F}\left(\frac{1}{10}\sum_{j=1}^{10}\frac{c(i,j,h)-c'(i)}{c'(i)}\right)$ where F is the set of instances with a specific feature, $c'(i)$ is the cost of the best known solution to instance i and $c(i,j,h)$ is the cost obtained in the jth experiment on instance i using heuristic h.

3.5 Comparison to existing heuristics

This section compares the ALNS heuristics to existing heuristics for the PDPTW. The comparison is performed using the benchmark instances proposed by Li and Lim [11] that also were used in Section 3.4.1. When PDPTW problems have been solved in the literature, the primary objective has been to minimize the number of vehicles used while the secondary objective has been to minimize the traveled distance. For this purpose we use the vehicle minimization algorithm described in Section 2.7. The ALNS heuristic was applied 10 times to each instance with 200 or less customers and 5 times to each instance with more than 200 customers. The experiments are summarized in Tables 7, 8 and 9. It should be noted that it was necessary to decrease the w parameter and increase the c parameter when the 1000 customers instances were solved in order to get reasonable solution quality. Apart from that, the same parameter setting has been used for all instances.

In the literature, four heuristics have been applied to the benchmark problems: The heuristic by Li and Lim [11], the heuristic by Bent and Van Hentenryck [2] and two commercial heuristics: A heuristic developed by SINTEF and a heuristic developed by TetraSoft A/S. Detailed results for the two last heuristics are not available but some results obtained using these heuristics can be found on a web page maintained by SINTEF [23]. The heuristic that has obtained the best overall solution quality so far is probably the one by Bent and Van Hentenryck [2] (shortened BH heuristic in the following), therefore the ALNS heuristic is compared to this heuristic in Table 7. The complete results from BH heuristic can be found in [3]. The results given for the BH heuristics are the best obtained among 10 experiments (though for the 100 customer instances only 5 experiments were performed). The *Avg. TTB* column shows the average time needed for the BH heuristic to obtain its best solution. For the ALNS heuristic we only list the time used in total as this heuristic, due to its simulated annealing component, usually finds its best solution towards the end of the search. The BH heuristic was tested on a 1.2 GHz Athlon processor and the running times of the two heuristics should therefore be comparable (we believe that the Athlon processor is at most 20% slower than our computer). The results show that the ALNS heuristic overall dominates the BH heuristic, especially as the problem sizes increases. It is also clear that the ALNS heuristic is able to improve considerably on the previously best known solutions and that the vehicle minimization algorithm works very well despite its simplicity. The last two columns in Table 7 summarize the best results obtained using several experiments with different parameter settings, which show that the results obtained by ALNS actually can be improved even further.

Table 8 compares the results obtained by ALNS with the best known solutions from the literature. It can be seen that ALNS improves more than half of the solutions and achieves a solution that is at least as good as the previously best known solution for 80% of the problems.

The two afore-mentioned tables only dealt with the best solutions found by the ALNS heuristic, Table 9 shows the average solution quality obtained by the heuristic. These numbers can be compared to those in Table 7.

Overall, one can conclude that the ALNS heuristic should be considered as a state of the art heuristic for the PDPTW. The cost of the best solutions identified during the experiments are listed in Tables 10 to 15.

3.6 Computational tests conclusion

In Section 3.4 we stated three objectives for our computational experiments. The tests fulfilled these objectives as we saw that: 1) The adaptive LNS heuristic that combines several removal and construction heuristics displays superior performance compared to the simple LNS heuristic that only uses one insertion heuristic and one removal heuristic. 2) Certain problem characteristics influence the performance of the LNS heuristic but we did not find that any characteristics could make the LNS heuristic perform better than the ALNS heuristic. 3) The LNS heuristic indeed is able to find good quality solutions in a reasonable amount of time, and the heuristic outperforms previously proposed heuristics.

The experiments also illustrate the importance of testing heuristics on large set of problem instances as the difference between LNS and ALNS only really becomes apparent when we consider large instances. Note that the problems that need to be solved in the real world often have dimensions comparable to or greater than the biggest problems solved in this paper.

Finally the computational experiments performed in Section 3.3.3 indicated that a simple LNS heuristic seems to be more sensitive to the choices of removal heuristic compared to the choices of insertion heuristics. It would be interesting to see if this holds in general for other problems as well.

	Best known 2003		BH best				ALNS best of 10 or 5			ALNS best	
#customers	#veh.	Dist	#veh.	Dist	Avg. TTB	Avg. time	#veh.	Dist	Avg. time	#veh.	Dist
100	402	58060	402	58062	68	3900	402	58060	66	402	56060
200	615	178380	614	180358	772	3900	606	180931	264	606	180419
400	1183	421215	1188	423636	2581	6000	1158	422201	881	1157	420396
600	1699	873850	1718	879940	3376	6000	1679	863442	2221	1664	860898
800	2213	1492200	2245	1480767	5878	8100	2208	1432078	3918	2181	1423063
1000	2698	2195755	2759	2225190	6174	8100	2652	2137034	5370	2646	2122922

Table 7: This table compares the ALNS heuristic to existing heuristics using the Li and Lim benchmark instances. Each row in the table corresponds to a set of problems with the same number of customers. Each of these problem sets contains between 56 and 60 instances (see Table 8). The first column indicates the number of customers in each problem; the next two columns give the total number of vehicles used and the total distance traveled in the previously best known solutions as listed on the SINTEF web page [23] in the summer of 2003. The next four columns show information about the solutions obtained by Bent and Van Hentenryck's heuristic. The two columns *Avg. TTB* and *Avg. time* show the average time needed to reach the best solution and the average time spent on each instance respectively. Both columns report the time needed to perform one experiment on one instance. The next three columns report the solutions obtained in the experiment with the ALNS heuristic where the heuristic where applied either 5 or 10 times to each problem. The last two columns report the best solutions obtained in several experiments with our ALNS heuristic and with various parameter settings. Note that Bent and Van Hentenryck in some cases have found slightly better results than reported on the SINTEF web page in 2003. This is the reason why the number of vehicles used by the BH heuristic for the 200 customer problems is smaller than in the best known solutions.

		ALNS best of 10 or 5		ALNS best	
#customers	#problems	<PB	≤PB	<PB	≤PB
100	56	0	54	0	55
200	60	22	42	27	57
400	60	40	47	41	55
600	60	41	45	51	57
800	60	37	42	48	53
1000	58	50	54	51	55

Table 8: Comparison of the LNS heuristic to the previously best known solutions. The table is grouped by problems size. The first column shows the problem size, the next column shows the number of problems with that size. The next two columns give additional information about the experiment where the ALNS heuristic were applied 5 or 10 times to each instance. The columns $<PB$ reports how many times the best solution found by the ALNS heuristic was strictly better than the previously best known solution. The column $\leq PB$ shows how many times the best solution found by ALNS was at least as good as the previously best known solution. The last two columns show information about the best solutions obtained during experimentation with different parameter settings.

#customers	Avg. #veh.	Avg. Dist
100	403	58249
200	608	181707
400	1168	425817
600	1686	867930
800	2223	1432321
1000	2677	2129032

Table 9: The ALNS heuristic was applied 10 times to each problem with 200 or less customers and 5 times to each problem with more than 200 customers. The best solutions reported in Table 7 and 8 were of course not obtained in all experiments. This table shows the average number of vehicles and average distance traveled obtained. These number can be compared to the figures in Table 7

	R1		R2		C1		C2		RC1		RC2	
1	**19**	**1650.80**	**4**	**1253.23**	**10**	**828.94**	**3**	**591.56**	**14**	**1708.80**	**4**	**1406.94**
2	**17**	**1487.57**	**3**	**1197.67**	**10**	**828.94**	**3**	**591.56**	**12**	**1558.07**	**3**	**1374.27**
3	**13**	**1292.68**	**3**	**949.40**	**9**	**1035.35**	3	591.17	**11**	**1258.74**	**3**	**1089.07**
4	**9**	**1013.39**	**2**	**849.05**	**9**	**860.01**	**3**	**590.60**	**10**	**1128.40**	**3**	**818.66**
5	**14**	**1377.11**	**3**	**1054.02**	**10**	**828.94**	**3**	**588.88**	**13**	**1637.62**	**4**	**1302.20**
6	**12**	**1252.62**	**3**	**931.63**	**10**	**828.94**	**3**	**588.49**	**11**	**1424.73**	**3**	**1159.03**
7	**10**	**1111.31**	**2**	**903.06**	**10**	**828.94**	**3**	**588.29**	**11**	**1230.14**	**3**	**1062.05**
8	**9**	**968.97**	**2**	**734.85**	**10**	**826.44**	**3**	**588.32**	**10**	**1147.43**	**3**	**852.76**
9	**11**	**1208.96**	**3**	**930.59**	**9**	**1000.60**						
10	**10**	**1159.35**	**3**	**964.22**								
11	**10**	**1108.90**	**2**	**911.52**								
12	**9**	**1003.77**										

Table 10: Best results, 100 customers. The Li and Lim benchmark instances are divided into six sets: R1, R2, C1, C2, RC1 and RC2. Each of the major columns corresponds to one of these sets, the column at the left give the problem number. For each problem instance we report the number of vehicles and the distance traveled in the best solution obtained during experimentation. Bold numbers indicates best known solutions.

	R1		R2		C1		C2		RC1		RC2	
1	**20**	**4819.12**	**5**	**4073.10**	**20**	**2704.57**	**6**	**1931.44**	**19**	**3606.06**	**6**	**3605.40**
2	**17**	**4621.21**	**4**	**3796.00**	**19**	**2764.56**	**6**	**1881.40**	**15**	**3674.80**	**5**	**3327.18**
3	**15**	**3612.64**	**4**	**3098.36**	**17**	**3128.61**	**6**	**1844.33**	13	3178.17	**4**	**2938.28**
4	**10**	**3037.38**	**3**	**2486.14**	**17**	**2693.41**	**6**	**1767.12**	**10**	**2631.82**	**3**	**2887.97**
5	**16**	**4760.18**	**4**	**3438.39**	**20**	**2702.05**	**6**	**1891.21**	**16**	**3715.81**	**5**	**2776.93**
6	14	4178.24	**4**	**3201.54**	**20**	**2701.04**	**6**	**1857.78**	**17**	**3368.66**	**5**	**2707.96**
7	**12**	**3550.61**	**3**	**3135.05**	**20**	**2701.04**	**6**	**1850.13**	**14**	**3668.39**	4	3056.09
8	**9**	**2784.53**	**2**	**2555.40**	**20**	**2689.83**	**6**	**1824.34**	**13**	**3174.55**	**4**	**2399.95**
9	**14**	**4354.66**	**3**	**3930.49**	**18**	**2724.24**	**6**	**1854.21**	**13**	**3226.72**	**4**	**2208.49**
10	**11**	**3714.16**	**3**	**3344.08**	**17**	**2943.49**	**6**	**1817.45**	**12**	**2951.29**	**3**	**2550.56**

Table 11: Best results, 200 customers.

	R1		R2		C1		C2		RC1		RC2	
1	**40**	**10639.75**	8	9758.46	**40**	**7152.06**	**12**	**4116.33**	**36**	**9127.15**	**12**	**7471.01**
2	**31**	**10015.85**	**7**	**9496.64**	**38**	**8012.43**	**12**	**4144.29**	**31**	**8346.06**	**11**	**6303.36**
3	**23**	**8840.46**	**6**	**8116.53**	**33**	**8308.94**	**12**	**4431.75**	25	7387.40	**9**	**5438.20**
4	**16**	**6744.33**	**4**	**6649.78**	30	6878.00	12	4038.00	**19**	**5838.58**	**5**	**5322.43**
5	**29**	**10599.54**	**7**	**8574.84**	**40**	**7150.00**	**12**	**4030.63**	**33**	**8773.75**	**11**	**6120.13**
6	25	9525.45	**6**	**7995.06**	**40**	**7154.02**	**12**	**3900.29**	**31**	**8177.90**	**9**	**6479.56**
7	**19**	**8200.37**	**5**	**6928.61**	**40**	**7149.43**	**12**	**3962.51**	**29**	**7992.08**	**8**	**6361.26**
8	**14**	**5946.44**	**4**	**5447.40**	**39**	**7111.16**	**12**	**3844.45**	**27**	**7613.43**	**7**	**5928.93**
9	**24**	**9886.14**	**6**	**8043.20**	**36**	**7452.21**	**12**	**4188.93**	**26**	**8013.48**	**7**	**5303.53**
10	**21**	**8016.62**	**5**	**7904.77**	**35**	**7387.13**	**12**	**3828.44**	**24**	**7065.73**	**6**	**5760.78**

Table 12: Best results, 400 customers.

	R1		R2		C1		C2		RC1		RC2	
1	59	22838.65	**11**	**21945.30**	60	14095.64	**19**	**7977.98**	**53**	**17924.88**	**16**	**14817.72**
2	**45**	**20246.18**	**10**	**19666.59**	58	14379.53	**18**	**10277.23**	**44**	**16302.54**	**14**	**12758.77**
3	**37**	**18073.14**	**8**	**15609.96**	**50**	**14683.43**	**17**	**8728.30**	**36**	**14060.31**	**10**	**12812.67**
4	**28**	**13269.71**	**6**	**10819.45**	**47**	**13648.03**	**17**	**8041.97**	**25**	**10950.52**	**7**	**10574.87**
5	**38**	**22562.81**	**9**	**19567.41**	**60**	**14086.30**	**19**	**8047.37**	**47**	**16742.55**	**14**	**13009.52**
6	**32**	**20641.02**	**8**	**17262.96**	**60**	**14090.79**	**19**	**8094.11**	**44**	**16894.37**	**13**	**12643.98**
7	**25**	**17162.90**	**6**	**15812.42**	**60**	**14083.76**	**19**	**7998.18**	**39**	**15394.87**	**11**	**12007.65**
8	**19**	**11957.59**	**5**	**10950.90**	**59**	**14554.27**	**18**	**7579.93**	**36**	**15154.79**	**10**	**12163.43**
9	**32**	**21423.05**	**8**	**18799.36**	**54**	**14706.12**	**18**	**9501.00**	**35**	**15134.24**	**9**	**13768.01**
10	**27**	**18723.13**	**7**	**17034.63**	**53**	**14879.30**	**17**	**8019.94**	**31**	**13925.51**	**8**	**12016.94**

Table 13: Best results, 600 customers.

	R1		R2		C1		C2		RC1		RC2	
1	**80**	**39315.92**	**15**	**33816.90**	**80**	**25184.38**	**24**	**11687.06**	**67**	**32268.95**	**20**	**23289.40**
2	**59**	**34370.37**	**12**	**32575.97**	78	26062.17	**24**	**14358.92**	57	28395.39	**18**	**21786.62**
3	**44**	**29718.09**	**10**	**25310.53**	**65**	**25918.45**	**24**	**13198.29**	50	24354.36	**16**	**16586.31**
4	**25**	**21197.65**	**7**	**19506.42**	**60**	**22970.88**	**23**	**13376.82**	**35**	**18241.91**	**12**	**14122.05**
5	**50**	**39046.06**	**12**	**32634.29**	**80**	**25211.22**	25	12329.80	61	30995.48	**18**	**20292.92**
6	**42**	**33659.50**	**10**	**27870.80**	**80**	**25164.25**	**24**	**12702.87**	58	28568.61	**16**	**21088.57**
7	**32**	**27294.19**	**8**	**25077.85**	**80**	**25158.38**	**25**	**11855.86**	54	28164.41	**15**	**19695.96**
8	**21**	**19570.21**	**5**	**19256.79**	**78**	**25348.45**	**24**	**11482.88**	**49**	**26150.65**	**13**	**19009.33**
9	**42**	**36126.69**	**10**	**30791.77**	**73**	**25541.94**	**24**	**11629.61**	**47**	**24930.70**	**12**	**19003.68**
10	**32**	**30200.86**	**9**	**28265.24**	**71**	**25712.12**	**24**	**11578.58**	**42**	**24271.52**	**10**	**19766.78**

Table 14: Best results, 800 customers.

	R1		R2		C1		C2		RC1		RC2	
1	**100**	**56903.88**	**19**	**45422.58**	**100**	**42488.66**	**30**	**16879.24**	85	48702.83	**22**	**35073.70**
2	**80**	**49652.10**	**15**	**47824.44**	**95**	**43870.19**	**31**	**18980.98**	**73**	**45135.70**	**21**	**30932.74**
3	**54**	**42124.44**	**11**	**39894.32**	**82**	**42631.11**	**30**	**17772.49**	**55**	**35475.72**	**16**	**28403.51**
4	**28**	**32133.36**	**8**	**28314.95**	**74**	**39443.00**	**29**	**18089.93**	**40**	**27747.04**	**12**	**23083.20**
5	**61**	**59135.86**	**14**	**53209.98**	100	42477.41	**31**	**17137.53**	**76**	**49816.18**	**18**	**34713.96**
6	**50**	**48637.63**	**12**	**43792.11**	**101**	**42838.39**	**31**	**17198.01**	**69**	**44469.08**	**17**	**31485.26**
7	**37**	**38936.54**	**9**	**36728.20**	**100**	**42854.99**	**31**	**19117.67**	**64**	**41413.16**	**17**	**29639.63**
8	**26**	**29452.32**	**7**	**26278.09**	**98**	**42951.56**	**30**	**17018.63**	**60**	**40590.17**	-	-
9	**50**	**52223.15**	**13**	**48447.49**	**92**	**42391.98**	**31**	**17565.95**	**57**	**39587.85**	-	-
10	**40**	**46218.35**	**11**	**44155.66**	**90**	**42435.16**	**29**	**17425.55**	52	36195.00	**12**	**29402.90**

Table 15: Best results, 1000 customers. Two entries are missing as the corresponding problem instances no longer exist.

4 Conclusion

This paper presented an extension to the large neighborhood search and the ruin and recreate heuristic called adaptive LNS. The heuristic was tested on the pickup and delivery problem with time windows achieving good results in a reasonable amount of time. The idea of combining several sub heuristics in the same search proved to be successful.

As the proposed model is quite general it seems interesting to examine if the model and heuristic can be used to solve other vehicle routing problems. We are currently working on this topic and the results are very promising as the heuristic has been able to discover new best solutions to standard benchmarks for vehicle routing problems with time windows and multi-depot vehicle routing problems and other vehicle routing problems [14], [18].

It would also be interesting to apply the ideas presented in this paper to other combinatorial optimization problems. The adaptive LNS framework is easily applicable to most problems, taking advantage of the numerous robust and fast construction heuristic designed during the last decades for various optimization problems.

5 Acknowledgments

The authors wish to thank Jakob Birkedal Nielsen for helpful discussions during the development of the heuristic presented in this paper and Emilie Danna for valuable criticism of the paper.

6 Appendix

Table 16 to 21 shows detailed information about the solutions found during the experiment described in Section 3.5.

References

[1] R.K. Ahuja, O. Ergun, J.B. Orlin, A.P. Punnen, *A survey of Very Large Scale Neighborhood Search Techniques*, Discrete Applied Mathematics **123** (2002), 75-102.

[2] R. Bent, P. Van Hentenryck, *A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows*, Proceedings of the International Conference on Constraint Programming (CP-2003), Lecture Notes in Computer Science **2833**, 123-137.

[3] R. Bent, P. Van Hentenryck, *A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows*, Appendix, available at http://www.cs.brown.edu/people/rbent/pickup-appendix.ps.

[4] R. Bent, P. Van Hentenryck, *A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows*, To appear in Transportation Science.

[5] J.-F. Cordeau, G. Laporte, A. Mercier, *A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows*, Journal of the Operational Research Society **52** (2001), 928-936.

[6] T.H. Cormen, C.E., Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms, Second Edition*, MIT Press, 2001.

[7] G. Desaulniers, J. Desrosiers, A. Erdmann, M.M. Solomon, F. Soumis, *The VRP with Pickup and Delivery*, in P. Toth and D. Vigo (eds.): The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications **9** (2002), SIAM, Philadelphia, 225-242.

[8] Y. Dumas, J. Desrosiers, F. Soumis, *The pickup and delivery problem with time windows*, European Journal of Operational Research **54** (1991), 7-22.

[9] M. Gendreau, F. Guertin, J.-Y. Potvin, R. Séguin, *Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-ups and Deliveries*, Tech. Rep. CRT-98-10, Centre de Recherche sur les Transport, Universite de Montreal.

[10] H. C. Lau, Z. Liang, *Pickup and Delivery with Time Windows : Algorithms and Test Case Generation*, The 13th IEEE Conference on Tools with Artificial Intelligence, ICTAI-2001, Dallas, USA, 333-340.

[11] H. Li, A. Lim, *A Metaheuristic for the Pickup and Delivery Problem with Time Windows*, The 13th IEEE Conference on Tools with Artificial Intelligence, ICTAI-2001, Dallas, USA, 160-170.

[12] H. Lim, A. Lim, B. Rodrigues, *Solving the Pick up and Delivery Problem with Time Windows using "Squeaky Wheel" Optimization with Local Search*, Technical Report 2002, Singapore Management University, Paper No. 7-2002

[13] W.P. Nanry, J.W. Barnes, *Solving the pickup and delivery problem with time windows using reactive tabu search*, Transportation Research Part B **34** (2000), 107-121.

[14] D. Pisinger, S. Ropke, *A general heuristic for vehicle routing problems*, working paper, Department of computer science, University of Copenhagen 2004.

[15] J.-Y. Potvin, J.-M. Rousseau, *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, European Journal of Operational Research **66** (1993), 331-340.

[16] S. Ropke, *A Problem Generator for Pickup and Delivery Problems with Time Windows*, Technical Report, Department of Computer Science, University of Copenhagen 2004.

[17] S. Ropke, *Heuristics for the Multi-Vehicle Pickup and Delivery Problem with Time Windows*, Masters Thesis 01-11-8, Department of Computer Science, Univerity of Copenhagen, 2002.

[18] S. Ropke, D. Pisinger, *A Unified Heuristic for a Large Class of Vehicle Routing Problems with Backhauls*, to appear in European Journal of Operational Research.

[19] P. Shaw, *A new local search algorithm providing high quality solutions to vehicle routing problems*, Technical report, Department of Computer Science, University of Strathclyde, Scotland, 1997.

[20] P. Shaw, *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, Proceedings CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), 1998.

[21] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck, *Record Breaking Optimization Results Using the Ruin and Recreate Principle*, Journal of Computational Physics **159** (2000), 139-171.

[22] M. Sigurd , D. Pisinger, M. Sig, *The Pickup and Delivery Problem With Time Windows and Precedences*, Transportation Science **38** (2004), 197-209.

[23] SINTEF, *Vehicle Routing and Travelling Salesperson Problems*, Web page: http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html

[24] M.M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research **35** (1987), 254-265.

[25] M.A Trick, *A Linear Relaxation Heuristic for the Generalized Assignment Problem*, Naval Research Logistics **39** (1992), 137-152.

[26] P. Toth, D. Vigo, *An Overview of Vehicle Routing Problems*, In P. Toth and D. Vigo (eds.): The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications **9** (2002), SIAM, Philadelphia, 1-26.

[27] H. Xu, Z.-L. Chen, S. Rajagopal, S. Arunapuram, *Solving a Practical Pickup and Delivery Problem*, Transportation Science **37** (2003), 347-364.

	Best known			FULL					LNS best known	
	veh.	cost	References	avg. sol.	avg. #veh.	best sol.	best #veh.	avg. time (s)	veh.	cost
LR101	**19**	**1650.8**	LL	1650.80	19.0	**1650.80**	**19**	40	**19**	**1650.80**
LR102	**17**	**1487.57**	LL	1487.57	17.0	**1487.57**	**17**	47	**17**	**1487.57**
LR103	**13**	**1292.68**	LL	1292.68	13.0	**1292.68**	**13**	45	**13**	**1292.68**
LR104	**9**	**1013.39**	LL	1013.39	9.0	**1013.39**	**9**	26	**9**	**1013.39**
LR105	**14**	**1377.11**	LL	1377.11	14.0	**1377.11**	**14**	40	**14**	**1377.11**
LR106	**12**	**1252.62**	LL	1252.62	12.0	**1252.62**	**12**	41	**12**	**1252.62**
LR107	**10**	**1111.31**	LL	1111.31	10.0	**1111.31**	**10**	44	**10**	**1111.31**
LR108	**9**	**968.97**	LL	968.97	9.0	**968.97**	**9**	25	**9**	**968.97**
LR109	**11**	**1208.96**	SAM	1208.96	11.0	**1208.96**	**11**	41	**11**	**1208.96**
LR110	**10**	**1159.35**	LL	1159.35	10.0	**1159.35**	**10**	35	**10**	**1159.35**
LR111	**10**	**1108.9**	LL	1108.90	10.0	**1108.90**	**10**	44	**10**	**1108.90**
LR112	**9**	**1003.77**	LL	1003.77	9.0	**1003.77**	**9**	27	**9**	**1003.77**
LC101	**10**	**828.94**	LL	828.94	10.0	**828.94**	**10**	43	**10**	**828.94**
LC102	**10**	**828.94**	LL	828.94	10.0	**828.94**	**10**	44	**10**	**828.94**
LC103	**9**	**1035.35**	BH	1037.77	9.0	**1035.35**	**9**	49	**9**	**1035.35**
LC104	**9**	**860.01**	SAM	860.15	9.0	**860.01**	**9**	63	**9**	**860.01**
LC105	**10**	**828.94**	LL	828.94	10.0	**828.94**	**10**	41	**10**	**828.94**
LC106	**10**	**828.94**	LL	828.94	10.0	**828.94**	**10**	42	**10**	**828.94**
LC107	**10**	**828.94**	LL	828.94	10.0	**828.94**	**10**	43	**10**	**828.94**
LC108	**10**	**826.44**	LL	826.44	10.0	**826.44**	**10**	46	**10**	**826.44**
LC109	**9**	**1000.6**	BH	1000.60	9.0	**1000.60**	**9**	35	**9**	**1000.60**
LRC101	**14**	**1708.8**	LL	1708.80	14.0	**1708.80**	**14**	38	**14**	**1708.80**
LRC102	**12**	**1558.07**	SAM	1558.07	12.0	**1558.07**	**12**	41	**12**	**1558.07**
LRC103	**11**	**1258.74**	LL	1258.74	11.0	**1258.74**	**11**	43	**11**	**1258.74**
LRC104	**10**	**1128.4**	LL	1128.40	10.0	**1128.40**	**10**	52	**10**	**1128.40**
LRC105	**13**	**1637.62**	LL	1637.62	13.0	**1637.62**	**13**	42	**13**	**1637.62**
LRC106	**11**	**1424.73**	SAM	1424.73	11.0	**1424.73**	**11**	42	**11**	**1424.73**
LRC107	11	1230.15	LL	1230.14	11.0	**1230.14**	**11**	43	**11**	**1230.14**
LRC108	**10**	**1147.43**	SAM	1147.43	10.0	**1147.43**	**10**	25	**10**	**1147.43**
LR201	**4**	**1253.23**	SAM	1253.23	4.0	**1253.23**	**4**	69	**4**	**1253.23**
LR202	**3**	**1197.67**	LL	1197.67	3.0	**1197.67**	**3**	60	**3**	**1197.67**
LR203	**3**	**949.4**	LL	949.40	3.0	**949.40**	**3**	98	**3**	**949.40**
LR204	**2**	**849.05**	LL	849.05	2.0	**849.05**	**2**	181	**2**	**849.05**
LR205	**3**	**1054.02**	LL	1054.02	3.0	**1054.02**	**3**	58	**3**	**1054.02**
LR206	**3**	**931.63**	LL	931.63	3.0	**931.63**	**3**	86	**3**	**931.63**
LR207	**2**	**903.06**	LL	903.06	2.0	**903.06**	**2**	187	**2**	**903.06**
LR208	**2**	**734.85**	LL	734.85	2.0	**734.85**	**2**	285	**2**	**734.85**
LR209	**3**	**930.59**	SAM	930.59	3.0	**930.59**	**3**	73	**3**	**930.59**
LR210	**3**	**964.22**	LL	964.22	3.0	**964.22**	**3**	77	**3**	**964.22**
LR211	**2**	**911.52**	SAM	906.69	2.2	**911.52**	**2**	126	**2**	**911.52**
LC201	**3**	**591.56**	LL	591.56	3.0	**591.56**	**3**	36	**3**	**591.56**
LC202	**3**	**591.56**	LL	591.56	3.0	**591.56**	**3**	59	**3**	**591.56**
LC203	**3**	**585.56**	LL	591.17	3.0	591.17	3	81	3	591.17
LC204	**3**	**590.6**	SAM	590.60	3.0	**590.60**	**3**	141	**3**	**590.60**
LC205	**3**	**588.88**	LL	588.88	3.0	**588.88**	**3**	48	**3**	**588.88**
LC206	**3**	**588.49**	LL	588.49	3.0	**588.49**	**3**	60	**3**	**588.49**
LC207	**3**	**588.29**	LL	588.29	3.0	**588.29**	**3**	62	**3**	**588.29**
LC208	**3**	**588.32**	LL	588.32	3.0	**588.32**	**3**	69	**3**	**588.32**
LRC201	**4**	**1406.94**	SAM	1406.94	4.0	**1406.94**	**4**	38	**4**	**1406.94**
LRC202	**3**	**1374.27**	LL	1387.74	3.8	1374.79	3	82	**3**	**1374.27**
LRC203	**3**	**1089.07**	SAM	1089.07	3.0	**1089.07**	**3**	69	**3**	**1089.07**
LRC204	**3**	**818.66**	SAM	818.66	3.0	**818.66**	**3**	173	**3**	**818.66**
LRC205	**4**	**1302.2**	LL	1302.20	4.0	**1302.20**	**4**	75	**4**	**1302.20**
LRC206	**3**	**1159.03**	SAM	1337.75	3.0	**1159.03**	**3**	48	**3**	**1159.03**
LRC207	**3**	**1062.05**	SAM	1062.05	3.0	**1062.05**	**3**	66	**3**	**1062.05**
LRC208	**3**	**852.76**	LL	852.76	3.0	**852.76**	**3**	88	**3**	**852.76**
Tot.	402	58054		58249.42	403.00	58060.03	402	3680	402	58059.50
Avg.								66		
< PB						1			1	
<= PB						54			55	
#B		55				54			55	

Table 16: Results on 100-customer problems solved with vehicle minimization as primary objective. The first column contains the name of the problem, columns two to four show information about the previously best known solutions. Columns two and three give the number of vehicles in the solution and the total traveled distance. Column four refers to the method that first found the solution (LL: Li and Lim [11], BH: Bent and Van Hentenryck [2], SAM: SINTEF heuristic, TS: TetraSoft A/S heuristic). The next five columns show information about the solutions obtained by the ALNS LNS heuristic. The first two of these columns show the average distance traveled and the average number of vehicles (averaged over the 10 experiments performed). The two next column display the the best solution obtained in the 10 experiments. The column *avg. time* displays the average time needed to perform one experiment in seconds. The two last columns show the best results obtained during experimentation with various parameter settings. The last 5 columns provide some summary information. The *Tot.* and *Avg.* rows respectively sums and averages entries in the columns. The *<PB* row indicates how many solutions that are better than the previously best known solution and the *<=PB* row indicates how many solution that are at least as good as the previously best known solution. *#B* reports the number of overall best known solutions that were obtained. Best known solutions are marked with bold font.

	Best known			FULL					LNS best known	
	veh.	cost	References	avg. sol.	avg. #veh.	best sol.	best #veh.	avg. time (s)	veh.	cost
LR1_2_1	**20**	**4819.12**	LL	4819.12	20.0	**4819.12**	**20**	137	**20**	**4819.12**
LR1_2_2	17	4666.09	BH	4625.99	17.0	**4621.21**	**17**	149	**17**	**4621.21**
LR1_2_3	**15**	**3612.64**	TS	3626.13	15.0	**3612.64**	**15**	173	**15**	**3612.64**
LR1_2_4	10	3146.06	BH	3088.07	10.0	3058.12	10	228	**10**	**3037.38**
LR1_2_5	**16**	**4760.18**	BH	4852.41	16.0	**4760.18**	**16**	136	**16**	**4760.18**
LR1_2_6	**14**	**4175.16**	BH	4261.23	14.0	4184.80	14	164	14	4178.24
LR1_2_7	12	3851.36	BH	3580.94	12.0	3551.47	12	173	**12**	**3550.61**
LR1_2_8	9	2871.67	BH	2823.91	9.0	**2784.53**	**9**	226	**9**	**2784.53**
LR1_2_9	14	4411.54	BH	4438.36	14.0	**4354.66**	**14**	144	**14**	**4354.66**
LR1_2_10	11	3744.95	BH	3787.23	11.0	3741.29	11	146	**11**	**3714.16**
LRC1_2_1	**19**	**3606.06**	SAM	3606.06	19.0	**3606.06**	**19**	136	**19**	**3606.06**
LRC1_2_2	15	3681.36	BH	3684.82	15.0	**3674.80**	**15**	143	**15**	**3674.80**
LRC1_2_3	**13**	**3161.75**	BH	3211.85	13.0	3178.17	13	183	13	3178.17
LRC1_2_4	10	2655.27	BH	2660.26	10.0	2641.67	10	284	**10**	**2631.82**
LRC1_2_5	**16**	**3715.81**	BH	3718.57	16.0	3716.72	16	141	**16**	**3715.81**
LRC1_2_6	**17**	**3368.66**	SAM	3372.68	17.0	3368.74	17	141	**17**	**3368.66**
LRC1_2_7	15	3417.16	BH	3525.21	14.7	**3668.39**	**14**	140	**14**	**3668.39**
LRC1_2_8	14	3087.62	BH	3220.69	13.2	**3174.55**	**13**	144	**13**	**3174.55**
LRC1_2_9	14	3129.65	BH	3259.40	13.1	**3226.72**	**13**	140	**13**	**3226.72**
LRC1_2_10	13	2833.85	BH	2968.69	12.1	2967.70	12	156	**12**	**2951.29**
LC1_2_1	**20**	**2704.57**	LL	2704.57	20.0	**2704.57**	**20**	146	**20**	**2704.57**
LC1_2_2	**19**	**2764.56**	LL	2764.56	19.0	**2764.56**	**19**	141	**19**	**2764.56**
LC1_2_3	17	3134.08	BH	3142.99	17.0	3136.42	17	155	**17**	**3128.61**
LC1_2_4	17	2698.73	TS	2711.42	17.0	2704.41	17	209	**17**	**2693.41**
LC1_2_5	**20**	**2702.05**	LL	2702.05	20.0	**2702.05**	**20**	137	**20**	**2702.05**
LC1_2_6	**20**	**2701.04**	LL	2701.04	20.0	**2701.04**	**20**	133	**20**	**2701.04**
LC1_2_7	**20**	**2701.04**	LL	2701.04	20.0	**2701.04**	**20**	139	**20**	**2701.04**
LC1_2_8	**20**	**2689.83**	LL	2689.83	20.0	**2689.83**	**20**	145	**20**	**2689.83**
LC1_2_9	**18**	**2724.24**	LL	2724.24	18.0	**2724.24**	**18**	157	**18**	**2724.24**
LC1_2_10	18	2741.56	LL	2967.24	17.0	**2943.49**	**17**	104	**17**	**2943.49**
LR2_2_1	**5**	**4073.1**	SAM	4110.08	5.0	**4073.10**	**5**	230	**5**	**4073.10**
LR2_2_2	**4**	**3796**	SAM	4194.32	4.0	4113.64	4	249	**4**	**3796.00**
LR2_2_3	**4**	**3098.36**	SAM	3209.80	4.0	**3098.36**	**4**	696	**4**	**3098.36**
LR2_2_4	3	2487.65	TS	2495.48	3.0	2491.87	3	1191	**3**	**2486.14**
LR2_2_5	**4**	**3438.39**	SAM	3440.71	4.0	3439.40	4	207	**4**	**3438.39**
LR2_2_6	**4**	**3201.54**	LL	3204.44	4.0	3201.86	4	499	**4**	**3201.54**
LR2_2_7	3	3190.75	LL	3216.40	3.0	**3135.05**	**3**	521	**3**	**3135.05**
LR2_2_8	3	2187.01	TS	2613.39	2.0	2559.70	2	1114	**2**	**2555.40**
LR2_2_9	4	3198.44	SAM	3272.31	3.9	**3930.49**	**3**	425	**3**	**3930.49**
LR2_2_10	3	3377.45	SAM	3387.47	3.0	3360.74	3	342	**3**	**3344.08**
LRC2_2_1	6	3690.1	BH	3722.20	6.0	3622.11	6	117	**6**	**3605.40**
LRC2_2_2	6	2666.01	BH	3403.75	5.0	**3327.18**	**5**	201	**5**	**3327.18**
LRC2_2_3	4	3141.28	SAM	3138.84	4.0	2965.88	4	323	**4**	**2938.28**
LRC2_2_4	4	2190.88	TS	3006.86	3.0	2891.10	3	993	**3**	**2887.97**
LRC2_2_5	**5**	**2776.93**	BH	2786.49	5.0	2782.83	5	302	**5**	**2776.93**
LRC2_2_6	**5**	**2707.96**	SAM	2713.57	5.0	2710.14	5	302	**5**	**2707.96**
LRC2_2_7	**4**	**3050.03**	BH	3140.57	4.0	3056.09	4	217	4	3056.09
LRC2_2_8	4	2401.84	BH	2409.16	4.0	2404.09	4	286	**4**	**2399.95**
LRC2_2_9	4	2209.54	SAM	2214.37	4.0	2210.88	4	410	**4**	**2208.49**
LRC2_2_10	3	2699.55	BH	2558.03	3.1	2551.67	3	467	**3**	**2550.56**
LC2_2_1	**6**	**1931.44**	SAM	1931.44	6.0	**1931.44**	**6**	100	**6**	**1931.44**
LC2_2_2	**6**	**1881.4**	LL	1881.40	6.0	**1881.40**	**6**	157	**6**	**1881.40**
LC2_2_3	**6**	**1844.33**	SAM	1845.57	6.0	1844.66	6	234	**6**	**1844.33**
LC2_2_4	**6**	**1767.12**	LL	1772.02	6.0	1768.22	6	427	**6**	**1767.12**
LC2_2_5	**6**	**1891.21**	LL	1891.21	6.0	**1891.21**	**6**	121	**6**	**1891.21**
LC2_2_6	**6**	**1857.78**	SAM	1857.93	6.0	**1857.78**	**6**	150	**6**	**1857.78**
LC2_2_7	**6**	**1850.13**	SAM	1850.60	6.0	**1850.13**	**6**	151	**6**	**1850.13**
LC2_2_8	**6**	**1824.34**	LL	1825.88	6.0	1824.73	6	193	**6**	**1824.34**
LC2_2_9	**6**	**1854.21**	SAM	1854.43	6.0	**1854.21**	**6**	193	**6**	**1854.21**
LC2_2_10	**6**	**1817.45**	LL	1818.04	6.0	**1817.45**	**6**	245	**6**	**1817.45**
Tot.	615	178380		181707.35	608.10	180930.62	606	15815	606	180418.58
Avg.								264		
< PB						22				27
<= PB						42				57
#B		33				31				57

Table 17: Results on 200-customer problems

	Best known			FULL					LNS best known	
	veh.	cost	References	avg. sol.	avg. #veh.	best sol.	best #veh.	avg. time (s)	veh.	cost
LR1_4_1	**40**	**10639.75**	TS	10652.59	40.0	**10639.75**	**40**	351	**40**	**10639.75**
LR1_4_2	31	10533.33	SAM	10125.79	31.0	**10015.85**	**31**	554	**31**	**10015.85**
LR1_4_3	24	8831.1	SAM	8846.24	23.3	8908.01	23	613	**23**	**8840.46**
LR1_4_4	17	5551.47	LL	6974.01	16.0	6814.84	16	575	**16**	**6744.33**
LR1_4_5	30	10233.59	TS	10606.32	29.1	**10599.54**	**29**	457	**29**	**10599.54**
LR1_4_6	**25**	**9456.68**	BH	9686.93	25.0	9573.68	25	554	25	9525.45
LR1_4_7	21	8012.3	SAM	8170.00	19.7	**8200.37**	**19**	610	**19**	**8200.37**
LR1_4_8	15	6320.03	SAM	6093.04	14.1	6044.40	14	568	**14**	**5946.44**
LR1_4_9	25	10313.6	SAM	9908.16	24.7	**9886.14**	**24**	480	**24**	**9886.14**
LR1_4_10	22	8249.87	SAM	8233.16	21.0	8145.03	21	516	**21**	**8016.62**
LR2_4_1	**8**	**9726.88**	BH	10243.45	8.0	9786.02	8	467	8	9758.46
LR2_4_2	8	7971.09	SAM	9995.30	7.0	9717.03	7	761	**7**	**9496.64**
LR2_4_3	6	9794.4	SAM	8586.52	6.0	**8116.53**	**6**	1451	**6**	**8116.53**
LR2_4_4	5	5116.24	LL	6948.40	4.0	6695.51	4	3409	**4**	**6649.78**
LR2_4_5	7	9314.23	SAM	8893.25	7.0	8642.63	7	1096	**7**	**8574.84**
LR2_4_6	6	9439.98	SAM	8156.35	6.0	8089.75	6	1236	**6**	**7995.06**
LR2_4_7	5	7935.54	SAM	7126.64	5.0	**6928.61**	**5**	2019	**5**	**6928.61**
LR2_4_8	4	6043.41	LL	5591.83	4.0	**5447.40**	**4**	4603	**4**	**5447.40**
LR2_4_9	6	8552.29	SAM	8613.50	6.0	8135.86	6	780	**6**	**8043.20**
LR2_4_10	6	7449.9	TS	8008.78	5.2	**7904.77**	**5**	1385	**5**	**7904.77**
LC1_4_1	**40**	**7152.06**	SAM	7152.06	40.0	**7152.06**	**40**	585	**40**	**7152.06**
LC1_4_2	39	7326.93	BH	7395.61	38.9	**8012.43**	**38**	597	**38**	**8012.43**
LC1_4_3	35	7896.36	SAM	8538.36	33.1	**8308.94**	**33**	628	**33**	**8308.94**
LC1_4_4	**30**	**6451.68**	LL	7013.38	30.7	7021.92	30	558	30	6878.00
LC1_4_5	**40**	**7150**	SAM	7150.00	40.0	**7150.00**	**40**	508	**40**	**7150.00**
LC1_4_6	**40**	**7154.02**	LL	7154.02	40.0	**7154.02**	**40**	520	**40**	**7154.02**
LC1_4_7	**40**	**7149.43**	SAM	7149.43	40.0	**7149.43**	**40**	529	**40**	**7149.43**
LC1_4_8	**39**	**7111.16**	LL	7111.86	39.0	**7111.16**	**39**	542	**39**	**7111.16**
LC1_4_9	36	7539.92	SAM	7471.34	36.1	7458.43	36	462	**36**	**7452.21**
LC1_4_10	36	7181.05	TS	7278.25	35.8	7474.07	35	501	**35**	**7387.13**
LC2_4_1	**12**	**4116.33**	LL	4116.33	12.0	**4116.33**	**12**	319	**12**	**4116.33**
LC2_4_2	**12**	**4144.29**	SAM	4145.71	12.0	4144.49	12	455	**12**	**4144.29**
LC2_4_3	12	4624.76	SAM	4533.47	12.0	4483.34	12	681	**12**	**4431.75**
LC2_4_4	**12**	**3743.95**	LL	4123.21	12.0	4081.93	12	1169	12	4038.00
LC2_4_5	**12**	**4030.63**	TS	4030.97	12.0	4030.64	12	366	**12**	**4030.63**
LC2_4_6	**12**	**3900.29**	SAM	3905.41	12.0	3902.25	12	475	**12**	**3900.29**
LC2_4_7	**12**	**3962.51**	BH	3976.03	12.0	3969.69	12	481	**12**	**3962.51**
LC2_4_8	**12**	**3844.45**	SAM	3879.38	12.0	3867.31	12	549	**12**	**3844.45**
LC2_4_9	12	4198.61	SAM	4229.42	12.0	4209.49	12	604	**12**	**4188.93**
LC2_4_10	**12**	**3828.44**	BH	3846.45	12.0	3839.11	12	811	**12**	**3828.44**
LRC1_4_1	37	8944.58	TS	9059.11	36.5	**9127.15**	**36**	498	**36**	**9127.15**
LRC1_4_2	31	8642.74	SAM	8189.18	32.0	8404.51	31	550	**31**	**8346.06**
LRC1_4_3	**25**	**7307.09**	BH	7413.29	25.7	7429.00	25	644	25	7387.40
LRC1_4_4	19	5944.14	TS	5918.81	19.0	5901.86	19	909	**19**	**5838.58**
LRC1_4_5	34	9133.11	SAM	8760.38	34.0	8715.74	34	487	**33**	**8773.75**
LRC1_4_6	31	8817.39	SAM	8236.27	31.2	8198.96	31	475	**31**	**8177.90**
LRC1_4_7	30	7869.45	BH	7969.23	29.8	**7992.08**	**29**	500	**29**	**7992.08**
LRC1_4_8	28	7887.67	SAM	7625.79	27.9	**7613.43**	**27**	510	**27**	**7613.43**
LRC1_4_9	27	8215.25	SAM	7942.38	26.8	**8013.48**	**26**	494	**26**	**8013.48**
LRC1_4_10	24	7404.91	SAM	7190.05	24.0	7103.78	24	503	**24**	**7065.73**
LRC2_4_1	13	6655.52	SAM	7750.57	12.0	**7471.01**	**12**	553	**12**	**7471.01**
LRC2_4_2	11	7467.34	SAM	6385.15	11.0	6332.52	11	1102	**11**	**6303.36**
LRC2_4_3	9	5480.25	TS	5485.05	9.0	5459.06	9	2126	**9**	**5438.20**
LRC2_4_4	6	4279.05	LL	5446.01	5.0	5405.16	5	4032	**5**	**5322.43**
LRC2_4_5	**11**	**6120.13**	BH	6147.77	11.0	6140.07	11	827	**11**	**6120.13**
LRC2_4_6	10	6002.63	SAM	6540.83	9.1	**6479.56**	**9**	757	**9**	**6479.56**
LRC2_4_7	9	5737.02	SAM	6497.14	8.0	**6361.26**	**8**	707	**8**	**6361.26**
LRC2_4_8	8	5364.31	SAM	6004.71	7.1	5968.27	7	834	**7**	**5928.93**
LRC2_4_9	7	6892.23	SAM	5469.65	7.0	5394.73	7	1275	**7**	**5303.53**
LRC2_4_10	7	5057.81	TS	6124.51	6.0	**5760.78**	**6**	1243	**6**	**5760.78**
Tot.	1183	421215		425816.87	1167.80	422201.17	1158	52850	1157	420395.99
Avg.								881		
< PB						40				41
<= PB						47				55
#B		19				25				55

Table 18: Results on 400-customer problems

	Best known			FULL					LNS best known	
	veh.	cost	References	avg. sol.	avg. #veh.	best sol.	best #veh.	avg. time (s)	veh.	cost
LR1_6_1	**59**	**22838.3**	BVH	23070.74	59.0	22975.40	59	1443	59	22838.65
LR1_6_2	45	20985.7	BVH	20714.68	45.0	20614.87	45	1438	**45**	**20246.18**
LR1_6_3	37	18685.9	BVH	18619.94	37.0	18548.01	37	1620	**37**	**18073.14**
LR1_6_4	28	13945.59	TS	13677.43	28.0	13604.92	28	2119	**28**	**13269.71**
LR1_6_5	39	22985.63	SAM	21983.13	39.0	**22562.81**	**38**	1105	**38**	**22562.81**
LR1_6_6	33	21427.75	SAM	20373.88	33.0	20060.42	33	1299	**32**	**20641.02**
LR1_6_7	27	17070.51	SAM	16615.48	26.6	16746.97	26	1476	**25**	**17162.90**
LR1_6_8	20	12669.88	SAM	12412.57	19.0	12302.45	19	1916	**19**	**11957.59**
LR1_6_9	34	21273.3	BVH	20917.36	33.2	20765.52	33	1059	**32**	**21423.05**
LR1_6_10	28	19337.5	SAM	18400.79	28.0	18233.75	28	989	**27**	**18723.13**
LR2_6_1	12	18840.8	BVH	22245.55	11.0	22049.96	11	1245	**11**	**21945.30**
LR2_6_2	11	17452.75	TS	20038.78	10.0	**19666.59**	**10**	2089	**10**	**19666.59**
LR2_6_3	9	17598.73	SAM	16161.38	8.0	15897.51	8	3729	**8**	**15609.96**
LR2_6_4	7	11771.45	TS	11627.85	6.0	10916.25	6	12849	**6**	**10819.45**
LR2_6_5	10	19347.2	SAM	20529.74	9.0	20079.56	9	1300	**9**	**19567.41**
LR2_6_6	9	19889.05	SAM	18788.90	8.0	17599.80	8	2238	**8**	**17262.96**
LR2_6_7	7	16262	BVH	16052.41	6.0	15877.37	6	6915	**6**	**15812.42**
LR2_6_8	6	11652.95	TS	11175.02	5.0	11026.09	5	10329	**5**	**10950.90**
LR2_6_9	9	18853.4	BVH	19465.02	8.0	19180.31	8	2123	**8**	**18799.36**
LR2_6_10	7	18449.18	SAM	17599.63	7.0	17261.53	7	1928	**7**	**17034.63**
LC1_6_1	**60**	**14095.6**	LL	14095.64	60.0	14095.64	60	1453	60	14095.64
LC1_6_2	**58**	**14379.5**	BVH	14383.04	58.0	14380.37	58	1440	58	14379.53
LC1_6_3	51	14569.3	BVH	14676.36	50.8	15028.86	50	1153	**50**	**14683.43**
LC1_6_4	48	13567.51	LL	13806.44	49.0	13750.06	49	1066	**47**	**13648.03**
LC1_6_5	**60**	**14086.3**	LL	14086.30	60.0	**14086.30**	**60**	1201	**60**	**14086.30**
LC1_6_6	**60**	**14090.79**	SAM	14090.79	60.0	**14090.79**	**60**	1198	**60**	**14090.79**
LC1_6_7	**60**	**14083.76**	SAM	14083.76	60.0	**14083.76**	**60**	1203	**60**	**14083.76**
LC1_6_8	**59**	**14554.27**	SAM	14557.89	59.0	14554.81	59	1263	**59**	**14554.27**
LC1_6_9	55	14626.25	TS	14676.34	56.0	14596.57	56	1261	**54**	**14706.12**
LC1_6_10	54	14627.2	TS	14918.57	55.6	14711.59	55	1329	**53**	**14879.30**
LC2_6_1	**19**	**7977.98**	SAM	7977.98	19.0	**7977.98**	**19**	1137	**19**	**7977.98**
LC2_6_2	19	8253.67	SAM	10612.70	18.0	10384.03	18	1277	**18**	**10277.23**
LC2_6_3	18	7436.5	BVH	7781.67	17.8	9007.34	17	2033	**17**	**8728.30**
LC2_6_4	18	8200.89	TS	8279.98	17.2	8281.94	17	2303	**17**	**8041.97**
LC2_6_5	**19**	**8047.37**	BVH	8068.59	19.0	8061.74	19	1268	**19**	**8047.37**
LC2_6_6	19	8169.95	TS	8149.37	19.0	8129.87	19	1016	**19**	**8094.11**
LC2_6_7	19	8038.56	BVH	8108.38	19.0	8086.65	19	1133	**19**	**7998.18**
LC2_6_8	18	7808.16	SAM	7632.38	18.0	7616.85	18	1067	**18**	**7579.93**
LC2_6_9	19	8134.25	SAM	8173.11	19.0	8160.19	19	1225	**18**	**9501.00**
LC2_6_10	18	7555.35	TS	7529.02	18.0	7511.89	18	1775	**17**	**8019.94**
LRC1_6_1	53	17930	BVH	18017.12	53.0	17965.79	53	1342	**53**	**17924.88**
LRC1_6_2	45	16040.3	BVH	16090.72	44.8	**16302.54**	**44**	1389	**44**	**16302.54**
LRC1_6_3	36	14407.6	BVH	14395.28	36.0	14310.59	36	1725	**36**	**14060.31**
LRC1_6_4	25	11308.6	BVH	11260.62	25.0	11097.51	25	2496	**25**	**10950.52**
LRC1_6_5	47	16803.9	BVH	16837.12	47.8	16831.90	47	1256	**47**	**16742.55**
LRC1_6_6	44	18205.25	SAM	17059.61	45.0	16994.01	45	1175	**44**	**16894.37**
LRC1_6_7	39	16407.68	SAM	15582.48	39.6	15565.62	39	1135	**39**	**15394.87**
LRC1_6_8	36	15352.6	BVH	15346.86	36.0	15174.29	36	1099	**36**	**15154.79**
LRC1_6_9	36	15751.84	SAM	15092.82	36.2	15000.49	36	1141	**35**	**15134.24**
LRC1_6_10	31	14304.37	SAM	14036.50	32.0	13940.77	32	1058	**31**	**13925.51**
LRC2_6_1	17	13111.6	BVH	14989.05	16.0	14844.71	16	1194	**16**	**14817.72**
LRC2_6_2	15	11463	BVH	12856.00	14.0	12801.40	14	2106	**14**	**12758.77**
LRC2_6_3	11	15167.3	BVH	12413.60	10.6	**12812.67**	**10**	4830	**10**	**12812.67**
LRC2_6_4	8	12512.5	BVH	10461.14	7.4	**10574.87**	**7**	13452	**7**	**10574.87**
LRC2_6_5	14	15576.76	SAM	13287.40	14.0	13216.21	14	1827	**14**	**13009.52**
LRC2_6_6	13	12655.11	SAM	12717.44	13.0	12709.04	13	1826	**13**	**12643.98**
LRC2_6_7	11	13996.73	SAM	12109.64	11.0	12070.35	11	1397	**11**	**12007.65**
LRC2_6_8	11	14572.07	SAM	12681.15	10.0	12565.94	10	2341	**10**	**12163.43**
LRC2_6_9	10	12262.51	TS	14236.58	9.0	13966.61	9	2094	**9**	**13768.01**
LRC2_6_10	9	12379.46	TS	12300.10	8.0	12129.35	8	2340	**8**	**12016.94**
Tot.	1699	873850		867929.80	1686.60	863441.95	1679	133234	1664	860898.44
Avg.								2221		
< PB						41			51	
<= PB						45			57	
#B		9				9			57	

Table 19: Results on 600-customer problems

	Best known			FULL					LNS best known	
	veh.	cost	References	avg. sol.	avg. #veh.	best sol.	best #veh.	avg. time (s)	veh.	cost
LR181	80	39374.4	LL	39847.80	80.0	39719.88	80	2867	**80**	**39315.92**
LR182	59	36122.5	BVH	35197.46	59.0	34746.99	59	2719	**59**	**34370.37**
LR183	45	31763	BVH	30506.10	44.0	30301.99	44	2984	**44**	**29718.09**
LR184	26	23454.57	SAM	21738.05	25.6	21900.66	25	3458	**25**	**21197.65**
LR185	52	39743.88	SAM	37834.13	52.4	37856.78	52	2051	**50**	**39046.06**
LR186	42	35011.85	SAM	33815.72	42.6	34315.99	42	2250	**42**	**33659.50**
LR187	34	28551.92	SAM	27347.55	32.8	28327.14	32	2720	**32**	**27294.19**
LR188	24	21891.97	SAM	20182.46	21.2	20256.27	21	2982	**21**	**19570.21**
LR189	44	36550.5	SAM	35693.92	43.0	35531.29	43	1890	**42**	**36126.69**
LR1810	34	31443.25	SAM	29741.89	33.6	29587.53	33	1891	**32**	**30200.86**
LR281	16	29961.22	SAM	34422.50	15.0	34124.11	15	2009	**15**	**33816.90**
LR282	13	37565.81	SAM	30839.74	12.8	33326.43	12	4507	**12**	**32575.97**
LR283	11	30046.47	SAM	26211.39	10.0	25446.52	10	8134	**10**	**25310.53**
LR284	8	24925.57	SAM	20085.04	7.0	**19506.42**	**7**	24419	**7**	**19506.42**
LR285	12	34256.18	SAM	34919.19	12.0	33961.98	12	2515	**12**	**32634.29**
LR286	10	30688.6	SAM	29070.99	10.0	28629.45	10	5827	**10**	**27870.80**
LR287	9	28524.9	BVH	25809.90	8.0	**25077.85**	**8**	7397	**8**	**25077.85**
LR288	7	19878.42	TS	18168.34	6.0	17800.02	6	29265	**5**	**19256.79**
LR289	11	34700.25	SAM	30325.20	10.8	31891.23	10	3025	**10**	**30791.77**
LR2810	10	31906.16	SAM	29604.30	9.0	28941.03	9	3425	**9**	**28265.24**
LC181	**80**	**25184.38**	SAM	25184.38	80.0	**25184.38**	**80**	2663	**80**	**25184.38**
LC182	**78**	**26056.2**	BVH	26186.79	78.0	26131.65	78	2712	78	26062.17
LC183	66	26700.6	BVH	26135.96	66.8	26308.88	66	2591	**65**	**25918.45**
LC184	61	23427.2	BVH	23880.34	62.4	23786.46	62	1892	**60**	**22970.88**
LC185	**80**	**25211.22**	SAM	25211.22	80.0	**25211.22**	**80**	2207	**80**	**25211.22**
LC186	**80**	**25164.25**	SAM	25164.25	80.0	**25164.25**	**80**	2210	**80**	**25164.25**
LC187	**80**	**25158.38**	SAM	25158.38	80.0	**25158.38**	**80**	2249	**80**	**25158.38**
LC188	78	25427.1	BVH	25262.20	79.0	25255.06	79	2187	**78**	**25348.45**
LC189	74	25536	BVH	26352.66	75.4	26363.13	74	2488	**73**	**25541.94**
LC1810	72	26364.93	TS	26896.75	75.0	26522.79	74	2394	**71**	**25712.12**
LC281	**24**	**11687.06**	SAM	11687.06	24.0	**11687.06**	**24**	1030	**24**	**11687.06**
LC282	25	12575	BVH	12634.54	25.0	12614.42	25	2462	**24**	**14358.92**
LC283	25	12500.5	BVH	13687.38	24.0	13551.68	24	2010	**24**	**13198.29**
LC284	24	13438.1	TS	12662.06	24.0	12593.32	24	3046	**23**	**13376.82**
LC285	**25**	**12298.9**	BVH	12357.15	25.0	12350.55	25	1237	25	12329.80
LC286	25	12064.8	BVH	12112.84	25.0	12090.57	25	1713	**24**	**12702.87**
LC287	25	11899.18	TS	11895.72	25.0	11878.10	25	1360	**25**	**11855.86**
LC288	24	11724.46	TS	11649.71	24.0	11592.23	24	1520	**24**	**11482.88**
LC289	24	11700.86	TS	11685.81	24.0	11673.27	24	1862	**24**	**11629.61**
LC2810	24	12139.06	TS	11693.40	24.0	11615.76	24	1874	**24**	**11578.58**
LRC181	67	32587.9	BVH	32275.83	67.6	**32268.95**	**67**	2206	**67**	**32268.95**
LRC182	**56**	**28843.1**	BVH	28306.81	58.4	28180.05	58	2515	57	28395.39
LRC183	**49**	**24933.9**	BVH	24672.74	51.0	24628.67	51	3207	50	24354.36
LRC184	35	18768.4	BVH	18696.22	35.0	18666.34	35	4276	**35**	**18241.91**
LRC185	**60**	**32578.04**	SAM	31439.49	63.2	31121.74	63	2218	61	30995.48
LRC186	**56**	**29971.97**	SAM	29037.55	59.8	28934.95	59	2135	58	28568.61
LRC187	**53**	**29948.45**	SAM	28696.11	55.8	28543.20	55	1944	54	28164.41
LRC188	49	28160.88	SAM	26889.40	50.8	26971.48	50	2105	**49**	**26150.65**
LRC189	47	26668.91	SAM	25538.12	48.6	25578.39	48	2016	**47**	**24930.70**
LRC1810	43	25787.27	SAM	24424.49	44.2	24156.12	44	2004	**42**	**24271.52**
LRC281	21	21486.1	LL	21905.03	20.8	23476.51	20	2217	**20**	**23289.40**
LRC282	19	19127.96	SAM	20056.42	19.2	19930.17	19	3522	**18**	**21786.62**
LRC283	17	18842.56	TS	16423.77	16.4	16846.85	16	6751	**16**	**16586.31**
LRC284	13	17693.9	BVH	14406.39	12.0	**14122.05**	**12**	19037	**12**	**14122.05**
LRC285	18	21626.63	TS	20541.12	18.0	20474.88	18	2725	**18**	**20292.92**
LRC286	16	25106.28	SAM	21271.46	16.0	21209.60	16	2792	**16**	**21088.57**
LRC287	15	23808.4	SAM	20402.90	15.0	19764.32	15	3187	**15**	**19695.96**
LRC288	13	24260	SAM	19670.06	13.0	19423.27	13	3722	**13**	**19009.33**
LRC289	13	19514	BVH	19548.71	12.0	19267.46	12	3702	**12**	**19003.68**
LRC2810	12	19865.4	BVH	19257.95	10.8	20530.09	10	4736	**10**	**19766.78**
Tot.	2213	1492200		1432320.80	2223.00	1432077.81	2208	235063	2181	1423062.65
Avg.								3918		
< PB						37				48
<= PB						42				53
#B		12				9				53

Table 20: Results on 800-customer problems

	Best known			FULL, Both IA = 0.01					LNS best known	
	veh.	cost	References	avg. sol.	avg. #veh.	best sol.	best #veh.	avg. time (s)	veh.	cost
LR1101	100	57977	BVH	57172.54	100.0	57016.58	100	4576	**100**	**56903.88**
LR1102	80	52361.61	SAM	49937.45	80.0	49765.70	80	4495	**80**	**49652.10**
LR1103	54	44890.55	SAM	42886.53	54.0	42681.33	54	4473	**54**	**42124.44**
LR1104	31	32336.04	SAM	31450.33	29.0	**32133.36**	**28**	4522	**28**	**32133.36**
LR1105	64	58260.68	SAM	58138.72	61.6	**59135.86**	**61**	3474	**61**	**59135.86**
LR1106	51	49697.85	SAM	47333.63	51.8	**48637.63**	**50**	3673	**50**	**48637.63**
LR1107	39	39861.97	SAM	38315.35	38.2	**38936.54**	**37**	3598	**37**	**38936.54**
LR1108	29	31515.87	SAM	29674.35	26.4	**29452.32**	**26**	4892	**26**	**29452.32**
LR1109	52	52282.36	SAM	51412.70	51.0	**52223.15**	**50**	3126	**50**	**52223.15**
LR11010	42	45710.21	SAM	45873.80	41.0	**46218.35**	**40**	2841	**40**	**46218.35**
LR2101	19	45835.55	SAM	47201.18	19.0	45493.36	19	3158	**19**	**45422.58**
LR2102	16	48817.75	SAM	51094.71	15.4	50925.97	15	5324	**15**	**47824.44**
LR2103	13	43094.14	SAM	38654.94	12.0	37778.15	12	12055	**11**	**39894.32**
LR2104	10	32993.09	SAM	28821.03	8.6	29783.60	8	26496	**8**	**28314.95**
LR2105	15	56010.62	SAM	53453.03	14.8	55497.90	14	4244	**14**	**53209.98**
LR2106	13	48225.07	SAM	46388.49	12.4	46145.75	12	6565	**12**	**43792.11**
LR2107	11	38336.76	SAM	36506.87	9.6	38322.91	9	14455	**9**	**36728.20**
LR2108	8	32493.7	SAM	27137.04	7.0	26631.41	7	26592	**7**	**26278.09**
LR2109	14	55587.14	SAM	52093.74	13.0	50990.04	13	3844	**13**	**48447.49**
LR21010	12	47678.69	SAM	44815.46	11.6	46117.94	11	5945	**11**	**44155.66**
LC1101	**100**	**42488.66**	SAM	42488.66	100.0	**42488.66**	**100**	4025	**100**	**42488.66**
LC1102	96	43437.2	BVH	43417.56	95.8	**43870.19**	**95**	4008	**95**	**43870.19**
LC1103	85	42483.61	SAM	42589.34	82.6	**42631.11**	**82**	4123	**82**	**42631.11**
LC1104	76	39613.83	SAM	38950.40	74.8	**39443.00**	**74**	3617	**74**	**39443.00**
LC1105	**100**	**42477.4**	SAM	42477.41	100.0	42477.41	100	3603	100	42477.41
LC1106	**101**	**42838.39**	SAM	42838.39	101.0	**42838.39**	**101**	3714	**101**	**42838.39**
LC1107	**100**	**42854.99**	TS	42855.17	100.0	**42854.99**	**100**	3556	**100**	**42854.99**
LC1108	99	42711.7	BVH	42964.24	98.0	42954.34	98	3637	**98**	**42951.56**
LC1109	93	42899.1	BVH	42614.87	92.2	**42391.98**	**92**	3508	**92**	**42391.98**
LC11010	91	42243.4	TS	42715.95	90.2	**42435.16**	**90**	3582	**90**	**42435.16**
LC2101	**30**	**16879.24**	TS	16879.24	30.0	**16879.24**	**30**	1502	**30**	**16879.24**
LC2102	32	17598.6	BVH	19210.16	31.4	19116.33	31	2171	**31**	**18980.98**
LC2103	30	19198.95	SAM	17503.99	30.8	17940.74	30	3651	**30**	**17772.49**
LC2104	30	17726	LL	19076.31	30.2	18418.52	30	4120	**29**	**18089.93**
LC2105	31	17466.42	TS	17149.07	31.0	**17137.53**	**31**	2561	**31**	**17137.53**
LC2106	31	17352.7	TS	18276.39	31.0	17217.15	31	2012	**31**	**17198.01**
LC2107	32	18131.36	TS	19306.15	32.0	17721.20	32	2796	**31**	**19117.67**
LC2108	30	17974.2	SAM	17266.57	30.0	17035.24	30	2745	**30**	**17018.63**
LC2109	31	17769.6	BVH	17825.02	31.2	17667.44	31	2809	**31**	**17565.95**
LC21010	30	18249.85	SAM	18342.21	30.2	17266.19	30	3297	**29**	**17425.55**
LRC1101	**84**	**49315.3**	BVH	48997.27	85.4	48934.66	85	3638	85	48702.83
LRC1102	73	45679.5	BVH	45351.71	73.0	45272.96	73	3966	**73**	**45135.70**
LRC1103	55	36570.5	BVH	35393.15	55.4	**35475.72**	**55**	4397	**55**	**35475.72**
LRC1104	41	28979.2	BVH	28013.33	40.2	27930.03	40	6042	**40**	**27747.04**
LRC1105	76	51455.4	BVH	50012.71	76.2	**49816.18**	**76**	3372	**76**	**49816.18**
LRC1106	69	47014.55	SAM	44308.41	70.2	**44469.08**	**69**	3132	**69**	**44469.08**
LRC1107	65	43321.51	SAM	41395.55	65.2	**41413.16**	**64**	3047	**64**	**41413.16**
LRC1108	60	42968.34	SAM	40946.68	61.0	**40590.17**	**60**	3017	**60**	**40590.17**
LRC1109	57	42549.12	SAM	39708.07	58.0	**39587.85**	**57**	2837	**57**	**39587.85**
LRC11010	**51**	**38274.02**	SAM	36184.43	52.2	36195.00	52	2930	52	36195.00
LRC2101	23	36894.98	SAM	32969.29	23.2	**35073.70**	**22**	2864	**22**	**35073.70**
LRC2102	22	28019.7	LL	29945.79	22.2	31054.84	21	4749	**21**	**30932.74**
LRC2103	19	30226.39	SAM	27201.83	17.8	28662.28	17	9528	**16**	**28403.51**
LRC2104	14	25836.7	BVH	22976.06	12.8	23611.31	12	28075	**12**	**23083.20**
LRC2105	19	39344.9	SAM	31946.46	18.8	**34713.96**	**18**	3945	**18**	**34713.96**
LRC2106	18	29947.9	SAM	30362.74	18.0	29577.50	18	2356	**17**	**31485.26**
LRC2107	18	31633.3	BVH	29915.31	17.2	29822.82	17	4432	**17**	**29639.63**
LRC21010	13	31361.45	SAM	30293.97	12.2	30160.05	12	5729	**12**	**29402.90**
Tot.	2698	2195755		2129031.74	2677.80	2137033.93	2652	311441	2646	2122921.51
Avg.								5370		
< PB						50			51	
<= PB						54			55	
#B		7				25			55	

Table 21: Results on 1000-customer problems