TRANSPORTATION SCIENCE
Vol. 40, No. 4, November 2006, pp. 455–472
ISSN 0041-1655 | EISSN 1526-5447 | 06 | 4004 | 0455

DOI 10.1287/trsc.1050.0135

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

Stefan Ropke, David Pisinger
DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark
{sropke@diku.dk, pisinger@diku.dk}

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 locations 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 must also satisfy time window and capacity constraints.

This paper presents a heuristic for the problem based on an extension 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 subheuristics that 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 instances with up to 500 requests. 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 subheuristics instead of just one. We believe that the proposed heuristic is very robust and is able to adapt to various instance characteristics.

Key words: pickup and delivery problems with time windows; large neighborhood search; metaheuristics
History: Received: July 2004; revision received: August 2005; accepted: August 2005.

Introduction

In the considered variant of the pickup and delivery problem with time windows (PDPTW), we are given a number of *requests* and *vehicles*. A request consists of picking up goods at one location and delivering these goods to another location. 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 off. Furthermore, *service times* are associated with each pickup and delivery. The service times indicate how long it will take for the pickup or delivery to be performed. 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 at a location after the end of the time window of the location.

Each request is assigned a set of feasible vehicles. This can, for example, be used to model situations where some vehicles cannot enter a certain location because of the dimensions of the vehicle.

Each vehicle has a limited capacity, and it 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. Furthermore, each vehicle is assigned a start and end time. The start time indicates when the vehicle must leave its start location, and the end time denotes the latest allowable arrival at its end location. Note that the vehicle leaves its depot at the specified start time even though this may introduce a waiting time at the first location visited.

Our task is to construct valid routes for the vehicles. A route is valid if time windows and capacity constraints are obeyed along the route, each pickup is served before the corresponding delivery, corresponding pickup and deliveries are served on the same route, and the vehicle serves only 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. The time spent by a vehicle is defined as its arrival time at the end terminal minus its start time (which is given a priori), (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 to serve as many requests as possible. A mathematical model is presented in §1 to define the problem precisely.

The problem 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 contains the traveling salesman problem 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 survey recent work on the PDPTW. Although none of the references mentioned below consider exactly the same problem as ours, they all face the same core problem.

Nanry and Barnes (2000) 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. To test the heuristic, Nanry and Barnes create PDPTW instances from a set of standard vehicle routing problem with time windows (VRPTW) problems proposed by Solomon (1987). The heuristic is tested on instances with up to 50 requests. Li and Lim (2001) use a hybrid metaheuristic to solve the problem. The heuristic combines simulated annealing and tabu search. Their method is tested on the nine largest instances from Nanry and Barnes (2000), and they consider 56 new instances based on Solomon's VRPTW problems (1987). Lim, Lim, and Rodrigues (2002) apply "squeaky wheel" optimization and local search to the PDPTW. Their heuristic is tested on the set of problems proposed by Li and Lim (2001). Lau and Liang (2001) 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 (2003a) proposed a heuristic for the PDPTW based on large neighborhood search. The heuristic was tested on the problems proposed by Li and Lim (2001). The heuristic by Bent and Van Hentenryck is probably the most promising metaheuristic for the PDPTW proposed so far.

Gendreau et al. (1998) 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. The tabu search is parallelized, and the sequential and parallelized versions are compared.

Several column generation methods for PDPTW have been proposed. These methods include both exact and heuristic methods. Dumas et al. (1991) 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. (2003) consider a PDPTW with several extra real-life constraints, including 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, Pisinger, and Sig (2004) solve a PDPTW problem related to transportation of livestock. This introduces some extra constraints, such as precedence relations among the requests, meaning that some requests must be served before others 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. (2002).

The work presented in this paper is based on the master's thesis of Ropke (2002). In the papers by Pisinger and Ropke (2005) and Ropke and Pisinger (2006), 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 *multidepot vehicle routing problem*, and the *vehicle routing problem with backhauls*.

The rest of this paper is organized as follows: §1 defines the PDPTW problem formally; §2 describes the basic solution method in a general context; §3 describes how the solution method has been applied to PDPTW, and extensions to the method are presented; §4 contains the results of the computational tests. The computational test is focused on comparing the heuristic to existing metaheuristics and evaluating if the refinements presented in §3 improve the heuristic; §5 concludes the paper.

1. Mathematical Model

This section presents a mathematical model of the problem; it is based on the model proposed by

Desaulniers et al. (2002). The mathematical model serves as a formal description of the problem. As we solve the problem heuristically, we do not attempt to write the model on integer-linear form.

A problem instance of the pickup and delivery problem contains n requests and m vehicles. The problem is defined on a graph, $P = \{1, \ldots, n\}$ is the set of pickup nodes, $D = \{n+1, \ldots, 2n\}$ is the set of delivery nodes. Request i is represented by nodes i and $i+n$. K is the set of all vehicles, $|K| = m$. One vehicle might not be able to serve all requests; as an example, a request might require that the vehicle has a freezing compartment. K_i is the set of vehicles that are able to serve request i, and $P_k \subseteq P$ and $D_k \subseteq D$ are the set of pickups and deliveries, respectively, that can be served by vehicle k; thus, for all i and k: $k \in K_i \Leftrightarrow i \in P_k \wedge i \in D_k$. Requests where $K_i \neq K$ are called *special requests*. Define $N = P \cup D$ and $N_k = P_k \cup D_k$. Let $\tau_k = 2n + k$, $k \in K$, and let $\tau'_k = 2n + m + k$, $k \in K$ be the nodes that represent the start and end terminal, respectively, of vehicle k. The graph $G = (V, A)$ consists of the nodes $V = N \cup \{\tau_1, \ldots, \tau_m\} \cup \{\tau'_1, \ldots, \tau'_m\}$ and the arcs $A = V \times V$. For each vehicle, we have a subgraph $G_k = (V_k, A_k)$, where $V_k = N_k \cup \{\tau_k\} \cup \{\tau'_k\}$ and $A_k = V_k \times V_k$. For each edge $(i, j) \in A$ we assign a distance $d_{ij} \geq 0$ and a travel time $t_{ij} \geq 0$. It is assumed that distances and times are nonnegative; $d_{ij} \geq 0$, $t_{ij} \geq 0$ and that the times satisfy the triangle inequality; $t_{ij} \leq t_{il} + t_{lj}$ for all $i, j, l \in V$. For the sake of modeling, we also assume that $t_{i,n+i} + s_i > 0$; this makes elimination of subtours and the pickup-before-delivery constraint easy to model.

Each node $i \in V$ has a service time s_i and a time window $[a_i, b_i]$. The service time represents the time needed for loading and unloading, and the time window indicates when the visit at the particular location must start; a visit to node i can only take place between time a_i and b_i. A vehicle is allowed to arrive at a location before the start of the time window, but it has to wait until the start of the time window before the visit can be performed. For each node $i \in N$, l_i is the amount of goods that must be loaded onto the vehicle at the particular node, $l_i \geq 0$ for $i \in P$, and $l_i = -l_{i-n}$ for $i \in D$. The capacity of vehicle $k \in K$ is denoted C_k.

Four types of decision variables are used in the mathematical model. x_{ijk}, $i, j \in V$, $k \in K$ is a binary variable that is one if the edge between node i and node j is used by vehicle k and zero otherwise. S_{ik}, $i \in V$, $k \in K$ is a nonnegative integer that indicates when vehicle k starts the service at location i, L_{ik}, $i \in V$, $k \in K$ is a nonnegative integer that is an upper bound on the amount of goods on vehicle k after servicing node i. S_{ik} and L_{ik} are only well defined when vehicle k actually visits node i. Finally z_i, $i \in P$ is a binary variable that indicates if request i is placed in the request bank. The variable is one if the request is placed in the request bank and zero otherwise.

A mathematical model is

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

Subject to:

$$\sum_{k \in K_i} \sum_{j \in N_k} x_{ijk} + z_i = 1 \quad \forall i \in P \quad (2)$$

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

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

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

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

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

$$a_i \leq S_{ik} \leq b_i \quad \forall k \in K, \forall i \in V_k \quad (8)$$

$$S_{ik} \leq S_{n+i,k} \quad \forall k \in K, \forall i \in P_k \quad (9)$$

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

$$L_{ik} \leq C_k \quad \forall k \in K, \forall i \in V_k \quad (11)$$

$$L_{\tau_k k} = L_{\tau'_k k} = 0 \quad \forall k \in K \quad (12)$$

$$x_{ijk} \in \{0, 1\} \quad \forall k \in K, \forall (i,j) \in A_k \quad (13)$$

$$z_i \in \{0, 1\} \quad \forall i \in P \quad (14)$$

$$S_{ik} \geq 0 \quad \forall k \in K, \forall i \in V_k \quad (15)$$

$$L_{ik} \geq 0 \quad \forall k \in K, \forall i \in V_k. \quad (16)$$

The objective function minimizes the weighted sum of the distance traveled, the sum of the time spent by each vehicle, and the number of requests not scheduled.

Constraint (2) ensures that each pickup location is visited or that the corresponding request is placed in the request bank. Constraint (3) ensures that the delivery location is visited if the pickup location is visited and that the visit is performed by the same vehicle. Constraints (4) and (5) ensure that a vehicle leaves every start terminal and a vehicle enters every end terminal. Together with constraint (6), this ensures that consecutive paths between τ_k and τ'_k are formed for each $k \in K$.

Constraints (7) and (8) ensure that S_{ik} is set correctly along the paths and that the time windows are obeyed. These constraints also make subtours impossible. Constraint (9) ensures that each pickup occurs before the corresponding delivery. Constraints (10), (11), and (12) ensure that the load variable is set correctly along the paths and that the capacity constraints of the vehicles are respected.

2. Solution Method

Local search heuristics are often built on neighborhood moves that make small changes to the current solution, such as moving a request from one route to another or exchanging two requests, as in Nanry and Barnes (2000) and Li and Lim (2001). These kinds of local search heuristics are able to investigate a huge number of solutions in a short time, but a solution is changed only 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, Laporte, and Mercier (2001). 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, as demonstrated in §4.

The proposed heuristic is based on *large neighborhood search* (*LNS*), introduced by Shaw (1997). The LNS heuristic has been applied to the VRPTW with good results (Shaw 1997, 1998; Bent and Van Hentenryck 2004). Recently the heuristic has been applied to the PDPTW as well (Bent and Van Hentenryck 2003a). The LNS heuristic itself is similar to the *ruin and recreate* heuristic proposed by Schrimpf et al. (2000).

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

Algorithm 1: LNS Heuristic.

```
1  Function LNS(s ∈ {solutions}, q ∈ 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;
```

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 (Shaw 1997; Bent and Van Hentenryck 2003a), near-optimal methods were used for the reinsert operation. This was achieved using a 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, because 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 criterion would be to accept all improving solutions. Such a criterion has been used in earlier LNS implementations (Shaw 1997). In this paper we use a simulated annealing accept criterion.

In Line 11 we check if a stop criterion is met. In our implementation we stop when a certain number of iterations has been performed.

The parameter $q \in \{0, \ldots, n\}$ determines the size of the neighborhood. If q is equal to zero, then no search at all will take place because 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 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. (2002). Ahuja et al. define very large-scale neighborhoods as neighborhoods whose sizes grow 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 Ahuja et al. (2002) is that the latter heuristics typically examine a huge number of solutions, albeit implicitly, while the LNS

heuristic proposed in this paper examines only 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 reoptimizes 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. In §3 we introduce the term *adaptive large neighborhood search* (ALNS) to describe an algorithm using several large neighborhoods in an adaptive way. A more general presentation of the ALNS framework can be found in a subsequent paper (Pisinger and Ropke 2005).

3. 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 (1997, 1998) and Bent and Van Hentenryck (2003a, 2004), 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 used only one method for removal and one method for insertions. The removal heuristics are described in §3.1 and the insertion heuristics are described in §3.2. The method for selecting which subheuristic to use is described in §3.3. The selection mechanism is guided by statistics gathered during the search, as described in §3.4. We are going to use the term *adaptive large neighborhood search* (*ALNS*) heuristic for an LNS heuristic that uses several competing removal and insertion heuristics and chooses between using statistics gathered during the search.

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 (1997, 1998) and Bent and Van Hentenryck (2003a, 2004).

3. The search is embedded in a simulated annealing metaheuristic where the earlier LNS heuristics used a simple descent approach. This is described in §3.5.

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. The vehicle minimization algorithm only works for homogeneous fleets without an upper bound on the number of vehicles available.

3.1. 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. Furthermore, the heuristics *Shaw removal* and *worst removal* have a parameter p that determines the degree of randomization in the heuristic.

3.1.1. Shaw Removal Heuristic. This removal heuristic was proposed by Shaw (1997, 1998). In this section it is slightly modified to suit the PDPTW. The general idea is to remove requests that are somewhat similar, as we expect it to be reasonably easy to shuffle similar requests around and thereby create 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 because we might only be able to insert the requests at their original positions or in some bad positions. We define the similarity of two requests 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

$$\begin{aligned} R(i,j) = {} & \varphi(d_{A(i),A(j)} + d_{B(i),B(j)}) \\ & + \chi(|T_{A(i)} - T_{A(j)}| + |T_{B(i)} - T_{B(j)}|) \\ & + \psi|l_i - l_j| + \omega\left(1 - \frac{|K_i \cap K_j|}{\min\{|K_i|, |K_j|\}}\right). \end{aligned} \quad (17)$$

$A(i)$ and $B(i)$ denote the pickup and delivery locations of request i, and T_i indicates the time when location i is visited. d_{ij}, l_i, and K_i are defined in §1. Using the decision variable S_{ik} from §1, we can write T_i as $T_i = \sum_{k\in K}\sum_{j\in V_k} S_{ik}x_{ijk}$. 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 requests, and the term weighted by ω ensures that two requests 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 requests in the same way as described by Shaw (1997). The procedure for removing requests is shown in pseudocode 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 $p \geq 1$ introduces some randomness in the selection of the requests (a low value of p corresponds to much randomness).

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;
```

Notice that the sorting in Line 7 can be avoided in an actual implementation of the algorithm because it is sufficient to use a linear time selection algorithm (Cormen et al. 2001) in Line 9.

3.1.2. 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 however, because it obviously can be implemented to run faster than the Shaw removal heuristic.

3.1.3. 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 insert them at another place in the solution to obtain a better solution value; therefore, we propose a removal heuristic that removes requests with high $cost(i, s)$.

The worst removal heuristic is shown in pseudocode in Algorithm 3. It reuses some of the ideas from §3.1.1.

Notice that the removal is randomized, with the degree of randomization controlled by the parameter p as in §3.1.1. This is done to avoid situations where the same requests are removed over and over again.

Algorithm 3: Worst Removal.

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

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 toward selecting requests that can "easily" be exchanged, while the worst-removal selects the requests that appear to be placed in the wrong position in the solution.

3.2. 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 Potvin and Rousseau (1993). 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.

3.2.1. 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 $c_i = \min_{k \in K}\{\Delta f_{i,k}\}$. 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

$$\min_{i \in U} c_i \tag{18}$$

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

Observe that in each iteration we only change one route (the one we inserted into), and we do not have to recalculate insertion costs 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 that 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 because many of the routes are "full." The heuristic presented in the next section tries to circumvent this problem.

3.2.2. Regret Heuristics. The *regret* heuristic tries to improve on the basic greedy heuristic by incorporating a kind of look-ahead information when selecting the request to insert. Let $x_{ik} \in \{1, \ldots, m\}$ be a variable that indicates the route for which request i has the kth 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 §3.2.1 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

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

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:

$$\max_{i \in U} \left\{ \sum_{j=1}^{k} (\Delta f_{i,x_{ij}} - \Delta f_{i,x_{i1}}) \right\}. \tag{19}$$

If some requests cannot be inserted in at least $m - k + 1$ routes, then the request that can be inserted in the fewest number of routes (but still can be inserted in at least one route) is inserted. 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 §3.2.1 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 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 request become limited.

Regret heuristics have been used by Potvin and Rousseau (1993) for the VRPTW. The heuristic in their paper can be categorized as a regret-k heuristic with $k = m$, because all routes are considered in an expression similar to (19). 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 described by Martello and Toth (1981). As in the previous section, we use the fact that we only change one route in each iteration to speed up the regret heuristic.

3.3. Choosing a Removal and an Insertion Heuristic

In §3.1 we defined three removal heuristics (Shaw, random, and worst removal), and in §3.2 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 heuristic best suited for another type of instance. We believe that alternating between the different removal and insertion heuristics gives us a more robust heuristic overall.

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, $i \in \{1, 2, \ldots, k\}$, we select heuristic j with probability

$$\frac{w_j}{\sum_{i=1}^{k} w_i}. \tag{20}$$

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 a quite involved process if many removal and insertion heuristics are used. Instead, an adaptive weight-adjusting algorithm is proposed in §3.4.

3.4. Adaptive Weight Adjustment

This section describes how the weights w_j introduced in §3.3 can be automatically adjusted using statistics from earlier iterations. The basic idea is to keep track of a score for each heuristic, which 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 a number of iterations of the ALNS heuristic; here we define a segment as 100 iterations. The score of all

Table 1 Score Adjustment Parameters

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.

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 situations shown in Table 1.

The case for σ_1 is 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 ALNS 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 parameters 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, and 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 cannot 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 used in formula (20). 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 $j+1$ as follows:

$$w_{i,\,j+1} = w_{ij}(1-r) + r\frac{\pi_i}{\theta_i}.$$

π_i is the score of heuristic i obtained during the last segment, and θ_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.

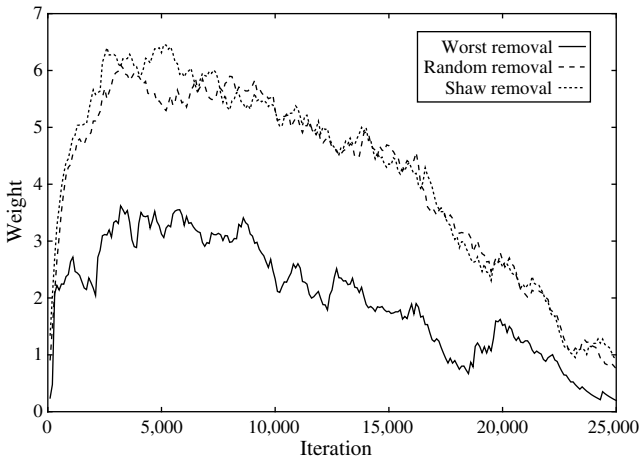

Figure 1 The Figure Shows an Example of How the Weights for the Three Removal Heuristics Progressed During One Application of the Heuristic

Notes. 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.

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. Toward the end of the search we only accept good moves, and therefore it is harder for the heuristic to get high scores.

3.5. Acceptance and Stopping Criteria

As described in §2, a simple acceptance criterion would be to accept only solutions that are better than the current solution. This would give us a descent heuristic like the one proposed by Shaw (1997). However, such a heuristic has a tendency to get trapped in a local minimum, so it seems sensible to on occasion accept solutions that are worse than the current solution. 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 $e^{-(f(s')-f(s))/T}$ 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% 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.

3.6. Applying Noise to the Objective Function

As the proposed insertion heuristics are quite myopic, we believe that it is worthwhile to randomize the insertion heuristics such that they do 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 $[-max\,N, max\,N]$ 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.

To make the amount of noise related to the properties of the problem instance, we calculate $max\,N = \eta \cdot \max_{i,j \in V}\{d_{ij}\}$, where η is a parameter that controls the amount of noise. We have chosen to let $max\,N$ 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; however, 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 4.

3.7. Minimizing the Number of Vehicles Used

Minimization of the number of vehicles used to serve all requests is often considered as first priority in the vehicle routing literature. 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. The vehicle minimization algorithm only works for problems with a homogeneous fleet. We also assume that the number of vehicles available is unlimited, such that constructing an initial feasible solution always can be done. A two-stage method was also used by Bent and Van Hentenryck (2003a, 2004), 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 requests have been planned. The number of vehicles used in this solution is the initial estimate on the number of vehicles necessary. The 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 has 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 a starting solution in the second stage of the algorithm, which simply consists of applying the normal LNS heuristic.

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 five 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 25,000 and τ was set to 2,000.

3.8. Discussion

Using several removal and insertion heuristics during the search may be seen as using local search with several neighborhoods. To the best of our knowledge, this idea has not been used in the LNS literature before. The related Variable Neighborhood Search (VNS) was proposed by Mladenović and Hansen (1997). VNS is a metaheuristic framework using a parameterized family of neighborhoods. The metaheuristic has received quite a lot of attention in the recent years and has provided impressive results for many problems. Where ALNS makes use of several

unrelated neighborhoods, VNS typically is based on a single neighborhood that is searched with variable depth.

Several metaheuristics can be used at the top level of ALNS to help the heuristic escape a local minimum. We have chosen to use simulated annealing because the ALNS heuristic already contains the random sampling element. For a further discussion of metaheuristic frameworks used in connection with ALNS, see the subsequent paper (Pisinger and Ropke 2005).

The request bank is an entity that makes sense for many real-life applications. In the problems considered in §4 we do not accept solutions with unscheduled requests, but the request bank allows us to visit infeasible solutions in a transition stage, improving the overall search. The request bank is particularly important when minimizing the number of vehicles.

4. Computational Experiments

In this section we describe our computational experiments. We first introduce a set of tuning instances in §4.1. In §4.2 we evaluate the performance of the proposed construction heuristics on the tuning instances. In §4.3 we describe how the parameters of the ALNS heuristic were tuned, and in §4.4 we present the results obtained by the ALNS heuristic and a simpler LNS heuristic.

4.1. Tuning Instances

First, a set of representative tuning instances is identified. The tuning instances must have a fairly limited size because we want to perform numerous experiments on the tuning problems, and they should somehow be related to the problems at which our heuristic is targeted. 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 (2001), containing between 50 and 100 requests. The number of available vehicles was set to one more than that 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 randomly generated instances. These instances contain both single-depot and multidepot problems and problems with requests that only can be served by a subset of the vehicle fleet. All randomly generated problems contain 50 requests.

4.2. Evaluation of Construction Heuristics

First, we examine how the simple construction heuristics from §3.2 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 2 shows the results 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.

Table 2 Performance of Construction Heuristics

	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

Notes. 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 IV.

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, however, because it is able to serve all requests in all problems. It would probably be possible to improve the results shown in Table 2 by introducing seed requests as proposed by Solomon (1987). However, we are not going to report on such experiments in this paper. 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 sections, this is indeed possible.

4.3. Parameter Tuning

This part of the paper serves two purposes. First, it describes how the parameters used for producing the results in §4.4 were found. Next, it tries to unveil which part of the heuristic contributes most to the solution quality.

4.3.1. 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.

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 control how many requests we

remove in each iteration. In each iteration, we chose a random number ρ that satisfies $4 \leq \rho \leq \min(100, \xi n)$, and remove ρ requests. We stop the search after 25,000 LNS iterations as this resulted in a fair trade-off between time and quality.

4.3.2. 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 works 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-and-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. Although it would be possible to process the parameters once again using the new set of parameters as a starting point to further optimize the parameters, we stopped after one pass.

One of the experiments performed during the parameter tuning sought to determine the value of the parameter ξ, which 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 3 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 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 §4.2. The results in Table 3 show that $\xi = 0.4$ is a good choice. One must notice that the heuristic gets slower when ξ increases because the removals and insertions take longer when more requests are involved; thus, the comparison in Table 3 is not completely fair.

Table 3 Parameter ξ vs. Solution Quality

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

Notes. The first row shows the values of the parameter ξ that were 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 $(\varphi, \chi, \psi, \omega, p, p_{worst}, w, c, \sigma_1, \sigma_2, \sigma_3, r, \eta, \xi) = (9, 3, 2, 5, 6, 3, 0.05, 0.99975, 33, 9, 13, 0.1, 0.025, 0.4)$. 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.

To tune the parameters we start from an initial guess, and then tune one parameter at a time. When all parameters are tuned, the process is repeated. In this way the calibration order plays a minor role. Although the parameter tuning is quite time consuming, it could easily be automated. In our subsequent papers (Ropke and Pisinger 2006; Pisinger and Ropke 2005), where 11 variants of the vehicle routing problem are solved using the heuristic proposed in this paper, we only retuned a few parameters and obtained very convincing results, so it seems that a complete tuning of the parameters only needs to be done once.

4.3.3. LNS Configurations. This section evaluates how the different removal and insertion heuristics behave when used in an 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 4 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 because 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 the Shaw removal as the 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,

Table 4 Simple LNS Heuristics Compared to the Full Adaptive LNS with Dynamic Weight Adjustment

	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**

Notes. The first column shows if the configuration must be considered as an LNS or an ALNS heuristic. The second column is the configuration number, Columns 3 to 5 indicate which removal heuristics were used. Columns 6 to 10 indicate which insertion heuristics were used. Column 11 states if noise was added to the objective function during insertion of requests (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 12 shows the average performance of the different heuristics. As an example, in Configuration 4 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.

however. 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 performs nearly as well as the regret heuristics when used in an LNS framework. This is surprising seen in the light of Table 2, where the basic insertion heuristic performed particularly badly. 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 seem disappointing. The results show, however, that the adaptive mechanism is able to find a sensible set of weights, and it is our hypothesis that the ALNS heuristic 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 §4.4 is to confirm or disprove this hypothesis.

4.4. Results

This section provides computational experiments conducted 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.

To clarify if the 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 4. Configuration 12 was chosen as representative for the simple LNS heuristics as it performed slightly better than Configuration 10. In the following sections 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 §4.3.2 was used in all experiments unless otherwise stated.

4.4.1. Data Sets. 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 (2000) and the instances constructed by Li and Lim (2001). Both data sets are single-depot pickup and delivery problems with time windows,

Table 5 Summary of Results Obtained on Li and Lim Instances (2001)

Number of locations	Number of problems	Best known solutions		Avg. gap (%)		Average time (s)		Fails	
		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	1,069	1,470	0	0
800	60	46	5	1.73	3.27	2,025	3,051	0	2
1,000	58	47	4	2.26	4.22	2,916	5,252	0	1

Notes. 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 subcolumns, 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 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 requests are served by the given number of vehicles in all the attempts to solve a particular problem.

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 size.

The problem considered by Li and Lim was simpler than the one considered in this paper because: (1) it did not contain multiple depots; (2) all requests must be served; (3) all vehicles were assumed to be able to serve all requests. When solving the Li and Lim instances using the ALNS heuristic, we set α to one and β to zero in our objective function. In §4.5 we minimize the number of vehicles as first priority while in §4.4.2 we only minimize the distance driven.

In order to test all aspects of the model proposed in this paper, we also introduce some new, randomly generated instances. These instances are described in §4.4.3.

4.4.2. Comparing ALNS and LNS Using the Li and Lim Instances. This section compares the ALNS and LNS heuristics using the benchmark instances proposed by Li and Lim (2001). The data set contains 354 instances with between 100 and 1,000 locations. The data set can be downloaded from SINTEF.

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 vehicles available for serving the requests is set to the minimum values reported by Li and Lim in (2001) and on their Web page, which unfortunately is no longer online.

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

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 other two removal heuristics.

4.4.3. New Instances. This section provides results on randomly generated PDPTW instances that contain features of the model that were not used in the Li and Lim (2001) benchmark problems considered in §4.4.2. These features are: multiple depots, routes with different start and end terminals, and *special* requests that can only be served by a certain subset of the vehicles. When solving these instances we set $\alpha = \beta = 1$ in the objective function so that distance and time are weighted equally in the objective function. We do not perform vehicle minimization because the vehicles are inhomogeneous.

Three types of geographical distributions of requests are considered: problems with locations distributed uniformly in the plane, problems with locations distributed in 10 clusters, and problems with 50% of the locations put in 10 clusters and 50% of the locations distributed uniformly. These three types of problems were inspired by Solomon's VRPTW benchmark problems (1987), 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 multidepot problems. For each problem size we generated

Table 6 Summary of Results Obtained on New Instances

Number of requests	Number of problems	Best known solutions		Avg. gap (%)		Average time (s)	
		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	1,274
500	12	9	3	1.18	1.32	3,805	8,146
Sum:	48	35	14	5.55	6.98	4,488	9,596

Notes. The captions of the table should be interpreted as in Table 5. 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 locations.

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 locations.
- 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% of the vehicles.
- Geographical distributions: (1) uniform, (2) clustered, and (3) semiclustered.

The instances can be downloaded from www.diku.dk/∼sropke. The heuristics were tested by applying them to each of the 48 problems 10 times. Table 6 shows a summary of the results found. In the table we list the number of problems for which the two heuristics find the best known solution. The best known solution is simply the best solution found throughout this experiment.

We observe the same tendencies as in Table 5; 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 (2001) instances. One also notices that it seems harder to solve small instances of this problem class compared to the Li and Lim instances.

Table 8 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. The route-type experiments compare the situation where routes start and end at the same location (the typical situation considered in the literature) to the situation where each route starts and ends at different locations. Here we expect the last case to be the easier 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 8 confirm these expectations.

In addition to investigating 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.

4.5. 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 (2001) that also were used in §4.4.2. When PDPTW problems have been solved in the literature,

Table 7 Comparison Between ALNS Heuristic and Existing Heuristics

Number of locations	Best known 2003		BH best				ALNS best of 10 or 5			ALNS best	
	Number of vehicles	Dist.	Number of vehicles	Dist.	Avg. TTB	Avg. time	Number of vehicles	Dist.	Avg. time	Number of vehicles	Dist.
100	402	58,060	402	58,062	68	3,900	402	58,060	66	402	56,060
200	615	178,380	614	180,358	772	3,900	606	180,931	264	606	180,419
400	1,183	421,215	1,188	423,636	2,581	6,000	1,158	422,201	881	1,157	420,396
600	1,699	873,850	1,718	879,940	3,376	6,000	1,679	863,442	2,221	1,664	860,898
800	2,213	1,492,200	2,245	1,480,767	5,878	8,100	2,208	1,432,078	3,918	2,181	1,423,063
1,000	2,698	2,195,755	2,759	2,225,190	6,174	8,100	2,652	2,137,034	5,370	2,646	2,122,922

Notes. Each row in the table corresponds to a set of problems with the same number of locations. Each of these problem sets contain between 56 and 60 instances (see Table 9). The first column indicates the number of locations 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 in the summer of 2003. The next four columns show information about the solutions obtained by Bent and Van Hentenryck's heuristic (2003a). 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 was 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 locations problems is smaller than in the best known solutions.

Table 8 Summary of the Influence of Certain Problem Features on the Heuristic Solutions

Feature	ALNS (%)	LNS (%)
Distribution: uniform	1.04	1.50
Distribution: clustered	1.89	2.09
Distribution: semiclustered	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 ≠ end of route	1.19	1.45

Notes. 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: $q(h) = (1/|F|) \sum_{i \in F} ((1/10) \sum_{j=1}^{10} (c(i, j, h) - c'(i))/c'(i))$ 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.

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 §3.7. The ALNS heuristic was applied 10 times to each instance with 200 or fewer locations and 5 times to each instance with more than 200 locations. The experiments are summarized in Tables 7, 9, and 10. It should be noted that it was necessary to decrease the w parameter and increase the c parameter when the instances with 1,000 locations were solved 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 (2001), the heuristic by Bent and Van Hentenryck (2003a), and two commercial heuristics: a heuristic developed by SINTEF and a heuristic developed by TetraSoft A/S. Detailed results for the last two heuristics are not available, but some results obtained using these heuristics can be found on a Web page maintained by SINTEF (http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html). The heuristic that has obtained the best overall solution quality so far is probably the one by Bent and Van Hentenryck (2003a) (shortened to BH heuristic in the following), therefore the ALNS heuristic is compared to this heuristic in Table 7. The complete results from the BH heuristic can be found in Bent and Van Hentenryck (2003b). The results given for the BH heuristic are the best obtained among 10 experiments (though for the 100 location instances only five 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 because this heuristic—because of its simulated annealing component—usually finds its best solution toward 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 overall the ALNS heuristic dominates the BH heuristic, especially as the problem sizes increase. 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 9 compares the results obtained by ALNS with the best known solutions from the literature. It

Table 9 Comparison with Previously Best Known Solutions

Number of locations	Number of problems	ALNS best of 10 or 5		ALNS best	
		<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
1,000	58	50	54	51	55

Notes. The table is grouped by problem size. The first column shows the problem size; the next column shows the number of problems of that size. The next two columns give additional information about the experiment where the ALNS heuristic was applied 5 or 10 times to each instance. The columns <PB report how many times the best solution found by the ALNS heuristic was strictly better than the previously best known solution. The columns ≤PB show 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.

Table 10 Average Performance of the ALNS Heuristic

Number of locations	Avg. number of vehicles	Avg. dist.
100	403	58,249
200	608	181,707
400	1,168	425,817
600	1,686	867,930
800	2,223	1,432,321
1,000	2,677	2,129,032

Notes. The best solutions reported in Tables 7 and 9 were of course not obtained in all experiments. This table shows the average number of vehicles and average distance traveled obtained. These numbers can be compared to the figures in Table 7.

Table 11 Best Results, 100 Locations

	R1		R2		C1		C2		RC1		RC2	
1	**19**	**1,650.80**	**4**	**1,253.23**	**10**	**828.94**	**3**	**591.56**	**14**	**1,708.80**	**4**	**1,406.94**
2	**17**	**1,487.57**	**3**	**1,197.67**	**10**	**828.94**	**3**	**591.56**	**12**	**1,558.07**	**3**	**1,374.27**
3	**13**	**1,292.68**	**3**	**949.40**	**9**	**1,035.35**	3	591.17	**11**	**1,258.74**	**3**	**1,089.07**
4	**9**	**1,013.39**	**2**	**849.05**	**9**	**860.01**	**3**	**590.60**	**10**	**1,128.40**	**3**	**818.66**
5	**14**	**1,377.11**	**3**	**1,054.02**	**10**	**828.94**	**3**	**588.88**	**13**	**1,637.62**	**4**	**1,302.20**
6	**12**	**1,252.62**	**3**	**931.63**	**10**	**828.94**	**3**	**588.49**	**11**	**1,424.73**	**3**	**1,159.03**
7	**10**	**1,111.31**	**2**	**903.06**	**10**	**828.94**	**3**	**588.29**	**11**	**1,230.14**	**3**	**1,062.05**
8	**9**	**968.97**	**2**	**734.85**	**10**	**826.44**	**3**	**588.32**	**10**	**1,147.43**	**3**	**852.76**
9	**11**	**1,208.96**	**3**	**930.59**	**9**	**1,000.60**						
10	**10**	**1,159.35**	**3**	**964.22**								
11	**10**	**1,108.90**	**2**	**911.52**								
12	**9**	**1,003.77**										

Notes. The Li and Lim (2001) 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 gives 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 indicate best known solutions.

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 aforementioned tables dealt only with the best solutions found by the ALNS heuristic. Table 10 shows the average solution quality obtained by the heuristic. These numbers can be compared to those in Table 7. It is worth noticing that the average solution sometimes has a lower distance than the "best of 10 or 5" solution in Table 7; this is the case in the last row. This is possible because the heuristic finds solutions that use more than the minimum number of vehicles, and this usually makes solutions with shorter distances possible.

Overall, one can conclude that the ALNS heuristic must 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 11 to 16.

4.6. Computational Tests Conclusion

In §4.4 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 is indeed 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 sets of problem instances, because 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 §4.3.3 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.

Table 12 Best Results, 200 Locations

	R1		R2		C1		C2		RC1		RC2	
1	**20**	**4,819.12**	**5**	**4,073.10**	**20**	**2,704.57**	**6**	**1,931.44**	**19**	**3,606.06**	**6**	**3,605.40**
2	**17**	**4,621.21**	**4**	**3,796.00**	**19**	**2,764.56**	**6**	**1,881.40**	**15**	**3,674.80**	**5**	**3,327.18**
3	**15**	**3,612.64**	**4**	**3,098.36**	**17**	**3,128.61**	**6**	**1,844.33**	13	3,178.17	**4**	**2,938.28**
4	**10**	**3,037.38**	**3**	**2,486.14**	**17**	**2,693.41**	**6**	**1,767.12**	**10**	**2,631.82**	**3**	**2,887.97**
5	**16**	**4,760.18**	**4**	**3,438.39**	**20**	**2,702.05**	**6**	**1,891.21**	**16**	**3,715.81**	**5**	**2,776.93**
6	14	4,178.24	**4**	**3,201.54**	**20**	**2,701.04**	**6**	**1,857.78**	**17**	**3,368.66**	**5**	**2,707.96**
7	**12**	**3,550.61**	**3**	**3,135.05**	**20**	**2,701.04**	**6**	**1,850.13**	**14**	**3,668.39**	4	3,056.09
8	**9**	**2,784.53**	**2**	**2,555.40**	**20**	**2,689.83**	**6**	**1,824.34**	**13**	**3,174.55**	**4**	**2,399.95**
9	**14**	**4,354.66**	**3**	**3,930.49**	**18**	**2,724.24**	**6**	**1,854.21**	**13**	**3,226.72**	**4**	**2,208.49**
10	**11**	**3,714.16**	**3**	**3,344.08**	**17**	**2,943.49**	**6**	**1,817.45**	**12**	**2,951.29**	**3**	**2,550.56**

Table 13 Best Results, 400 Locations

	R1		R2		C1		C2		RC1		RC2	
1	**40**	**10,639.75**	8	9,758.46	**40**	**7,152.06**	**12**	**4,116.33**	**36**	**9,127.15**	**12**	**7,471.01**
2	**31**	**10,015.85**	**7**	**9,496.64**	**38**	**8,012.43**	**12**	**4,144.29**	**31**	**8,346.06**	**11**	**6,303.36**
3	**23**	**8,840.46**	**6**	**8,116.53**	**33**	**8,308.94**	**12**	**4,431.75**	25	7,387.40	**9**	**5,438.20**
4	**16**	**6,744.33**	**4**	**6,649.78**	30	6,878.00	12	4,038.00	**19**	**5,838.58**	**5**	**5,322.43**
5	**29**	**10,599.54**	**7**	**8,574.84**	**40**	**7,150.00**	**12**	**4,030.63**	**33**	**8,773.75**	**11**	**6,120.13**
6	25	9,525.45	**6**	**7,995.06**	**40**	**7,154.02**	**12**	**3,900.29**	**31**	**8,177.90**	**9**	**6,479.56**
7	**19**	**8,200.37**	**5**	**6,928.61**	**40**	**7,149.43**	**12**	**3,962.51**	**29**	**7,992.08**	**8**	**6,361.26**
8	**14**	**5,946.44**	**4**	**5,447.40**	**39**	**7,111.16**	**12**	**3,844.45**	**27**	**7,613.43**	**7**	**5,928.93**
9	**24**	**9,886.14**	**6**	**8,043.20**	**36**	**7,452.21**	**12**	**4,188.93**	**26**	**8,013.48**	**7**	**5,303.53**
10	**21**	**8,016.62**	**5**	**7,904.77**	**35**	**7,387.13**	**12**	**3,828.44**	**24**	**7,065.73**	**6**	**5,760.78**

Table 14 Best Results, 600 Locations

	R1		R2		C1		C2		RC1		RC2	
1	59	22,838.65	**11**	**21,945.30**	60	14,095.64	**19**	**7,977.98**	**53**	**17,924.88**	**16**	**14,817.72**
2	**45**	**20,246.18**	**10**	**19,666.59**	58	14,379.53	**18**	**10,277.23**	**44**	**16,302.54**	**14**	**12,758.77**
3	**37**	**18,073.14**	**8**	**15,609.96**	**50**	**14,683.43**	**17**	**8,728.30**	**36**	**14,060.31**	**10**	**12,812.67**
4	**28**	**13,269.71**	**6**	**10,819.45**	**47**	**13,648.03**	**17**	**8,041.97**	**25**	**10,950.52**	**7**	**10,574.87**
5	**38**	**22,562.81**	**9**	**19,567.41**	**60**	**14,086.30**	**19**	**8,047.37**	**47**	**16,742.55**	**14**	**13,009.52**
6	**32**	**20,641.02**	**8**	**17,262.96**	**60**	**14,090.79**	**19**	**8,094.11**	**44**	**16,894.37**	**13**	**12,643.98**
7	**25**	**17,162.90**	**6**	**15,812.42**	**60**	**14,083.76**	**19**	**7,998.18**	**39**	**15,394.87**	**11**	**12,007.65**
8	**19**	**11,957.59**	**5**	**10,950.90**	**59**	**14,554.27**	**18**	**7,579.93**	**36**	**15,154.79**	**10**	**12,163.43**
9	**32**	**21,423.05**	**8**	**18,799.36**	**54**	**14,706.12**	**18**	**9,501.00**	**35**	**15,134.24**	**9**	**13,768.01**
10	**27**	**18,723.13**	**7**	**17,034.63**	**53**	**14,879.30**	**17**	**8,019.94**	**31**	**13,925.51**	**8**	**12,016.94**

Table 15 Best Results, 800 Locations

	R1		R2		C1		C2		RC1		RC2	
1	**80**	**39,315.92**	**15**	**33,816.90**	**80**	**25,184.38**	**24**	**11,687.06**	**67**	**32,268.95**	**20**	**23,289.40**
2	**59**	**34,370.37**	**12**	**32,575.97**	78	26,062.17	**24**	**14,358.92**	57	28,395.39	**18**	**21,786.62**
3	**44**	**29,718.09**	**10**	**25,310.53**	**65**	**25,918.45**	**24**	**13,198.29**	50	24,354.36	**16**	**16,586.31**
4	**25**	**21,197.65**	**7**	**19,506.42**	**60**	**22,970.88**	**23**	**13,376.82**	**35**	**18,241.91**	**12**	**14,122.05**
5	**50**	**39,046.06**	**12**	**32,634.29**	**80**	**25,211.22**	25	12,329.80	61	30,995.48	**18**	**20,292.92**
6	**42**	**33,659.50**	**10**	**27,870.80**	**80**	**25,164.25**	**24**	**12,702.87**	58	28,568.61	**16**	**21,088.57**
7	**32**	**27,294.19**	**8**	**25,077.85**	**80**	**25,158.38**	**25**	**11,855.86**	54	28,164.41	**15**	**19,695.96**
8	**21**	**19,570.21**	**5**	**19,256.79**	**78**	**25,348.45**	**24**	**11,482.88**	**49**	**26,150.65**	**13**	**19,009.33**
9	**42**	**36,126.69**	**10**	**30,791.77**	**73**	**25,541.94**	**24**	**11,629.61**	**47**	**24,930.70**	**12**	**19,003.68**
10	**32**	**30,200.86**	**9**	**28,265.24**	**71**	**25,712.12**	**24**	**11,578.58**	**42**	**24,271.52**	**10**	**19,766.78**

Table 16 Best Results, 1,000 Locations

	R1		R2		C1		C2		RC1		RC2	
1	**100**	**56,903.88**	**19**	**45,422.58**	**100**	**42,488.66**	**30**	**16,879.24**	85	48,702.83	**22**	**35,073.70**
2	**80**	**49,652.10**	**15**	**47,824.44**	**95**	**43,870.19**	**31**	**18,980.98**	**73**	**45,135.70**	**21**	**30,932.74**
3	**54**	**42,124.44**	**11**	**39,894.32**	**82**	**42,631.11**	**30**	**17,772.49**	**55**	**35,475.72**	**16**	**28,403.51**
4	**28**	**32,133.36**	**8**	**28,314.95**	**74**	**39,443.00**	**29**	**18,089.93**	**40**	**27,747.04**	**12**	**23,083.20**
5	**61**	**59,135.86**	**14**	**53,209.98**	100	42,477.41	**31**	**17,137.53**	**76**	**49,816.18**	**18**	**34,713.96**
6	**50**	**48,637.63**	**12**	**43,792.11**	**101**	**42,838.39**	**31**	**17,198.01**	**69**	**44,469.08**	**17**	**31,485.26**
7	**37**	**38,936.54**	**9**	**36,728.20**	**100**	**42,854.99**	**31**	**19,117.67**	**64**	**41,413.16**	**17**	**29,639.63**
8	**26**	**29,452.32**	**7**	**26,278.09**	**98**	**42,951.56**	**30**	**17,018.63**	**60**	**40,590.17**	—	—
9	**50**	**52,223.15**	**13**	**48,447.49**	**92**	**42,391.98**	**31**	**17,565.95**	**57**	**39,587.85**	—	—
10	**40**	**46,218.35**	**11**	**44,155.66**	**90**	**42,435.16**	**29**	**17,425.55**	52	36,195.00	**12**	**29,402.90**

Note. Two entries are missing as the corresponding problem instances no longer exist.

5. Conclusions

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 subheuristics in the same search proved to be successful.

Because the proposed model is quite general, it would be 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 because the heuristic has been able to discover new best solutions to standard benchmarks for vehicle routing problems with time windows, multidepot vehicle routing problems, and other vehicle routing problems (Pisinger and Ropke 2005; Ropke and Pisinger 2006).

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 heuristics designed during the last decades for various optimization problems.

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. Furthermore, they would like to thank the three anonymous referees for valuable comments and corrections.

References

Ahuja, R. K., O. Ergun, J. B. Orlin, A. P. Punnen. 2002. A survey of very large scale neighborhood search techniques. *Discrete Appl. Math.* **123** 75–102.

Bent, R., P. Van Hentenryck. 2003a. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Lecture Notes in Computer Science*, Vol. 2833. *Proc. Internat. Conf. Constraint Programming* (*CP*-2003), 123–137.

Bent, R., P. Van Hentenryck. 2003b. A two-stage hybrid algorithm for pickup and delivery vehicle routing problem with time windows. Appendix available at http://www.cs.brown.edu/people/rbent/pickup-appendix.ps.

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

Cordeau, J.-F., G. Laporte, A. Mercier. 2001. A unified tabu search heuristic for vehicle routing problems with time windows. *J. Oper. Res. Soc.* **52** 928–936.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, C. Stein. 2001. *Introduction to Algorithms*, 2nd ed. MIT Press, Boston, MA.

Desaulniers, G., J. Desrosiers, A. Erdmann, M. M. Solomon, F. Soumis. 2002. The VRP with pickup and delivery. P. Toth, D. Vigo, eds. *The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications*, Vol. 9. SIAM, Philadelphia, PA, 225–242.

Dumas, Y., J. Desrosiers, F. Soumis. 1991. The pickup and delivery problem with time windows. *Eur. J. Oper. Res.* **54** 7–22.

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

Lau, H. C., Z. Liang. 2001. Pickup and delivery with time windows: Algorithms and test case generation. *ICTAI-2001, 13th IEEE Conf. Tools Artificial Intelligence*, Dallas, TX, 333–340.

Li, H., A. Lim. 2001. A metaheuristic for the pickup and delivery problem with time windows. *ICTAI-2001, 13th IEEE Conf. Tools Artificial Intelligence*, Dallas, TX, 160–170.

Lim, H., A. Lim, B. Rodrigues. 2002. Solving the pick up and delivery problem with time windows using "squeaky wheel" optimization with local search. Technical Report 2002, Paper 7-2002, Singapore Management University, Singapore.

Martello, S., P. Toth. 1981. An algorithm for the generalized assignment problem. J. P. Brans, ed. *Operational Research* '81. North-Holland, New York.

Mladenović, N., P. Hansen. 1997. Variable neighborhood search. *Comput. Oper. Res.* **24**(11) 1097–1100.

Nanry, W. P., J. W. Barnes. 2000. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Res. Part B* **34** 107–121.

Pisinger, D., S. Ropke. 2005. A general heuristic for vehicle routing problems. *Comput. Oper. Res.* Forthcoming. http://www.sciencedirect.com

Potvin, J.-Y., J.-M. Rousseau. 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *Eur. J. Oper. Res.* **66** 331–340.

Ropke, S. 2002. Heuristics for the multi-vehicle pickup and delivery problem with time windows. Masters Thesis 01-11-8, Department of Computer Science, University of Copenhagen, Denmark.

Ropke, S., D. Pisinger. 2004. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Technical report, Department of Computer Science, University of Copenhagen, Denmark.

Ropke, S., D. Pisinger. 2006. A unified heuristic for a large class of vehicle routing problems with backhauls. *Eur. J. Oper. Res.* **171** 750–775.

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

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. *Proc. CP-98* (*Fourth Internat. Conf. Principles Practice Constraint Programming*).

Schrimpf, G., J. Schneider, H. Stamm-Wilbrandt, G. Dueck. 2000. Record breaking optimization results using the ruin and recreate principle. *J. Comput. Phys.* **159** 139–171.

Sigurd, M., D. Pisinger, M. Sig. 2004. The pickup and delivery problem with time windows and precedences. *Transportation Sci.* **38** 197–209.

SINTEF. Vehicle routing and travelling salesperson problems. http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* **35** 254–265.

Toth, P., D. Vigo. 2002. An overview of vehicle routing problems. P. Toth, D. Vigo, eds. *The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications*, Vol. 9. SIAM, Philadelphia, PA, 1–26.

Trick, M. A. 1992. A linear relaxation heuristic for the generalized assignment problem. *Naval Res. Logist.* **39** 137–152.

Xu, H., Z.-L. Chen, S. Rajagopal, S. Arunapuram. 2003. Solving a practical pickup and delivery problem. *Transportation Sci.* **37** 347–364.