A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows

Russell Bent and Pascal Van Hentenryck

Brown University, Box 1910 Providence, RI 02912, USA
{rbent,pvh}@cs.brown.edu

Abstract. This paper presents a two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows and multiple vehicles (PDPTW). The first stage uses a simple simulated annealing algorithm to decrease the number of routes, while the second stage uses LNS to decrease total travel cost. Experimental results show the effectiveness of the algorithm which has produced many new best solutions on problems with 100, 200, and 600 customers. In particular, it has improved 47% and 86% of the best solutions on the 200 and 600-customer benchmarks, sometimes by as much as 3 vehicles. These results further confirm the benefits of two-stage approaches in vehicle routing. They also answer positively the open issue in the original LNS paper, which advocated the use of LNS for the PDPTW and argue for the robustness of LNS with respect to side-constraints.

1 Introduction

Multiple vehicle routing problems with time windows (VRPTW) have received considerable attention in the last decades. These problems are often approached by meta-heuritics, since problems with as few as 100 customers are currently beyond the scope of state-of-the-art systematic search algorithms. Recent work on the VRPTW has produced significant improvements in solution quality and execution time, often by combining several approaches or heuristics. Comparatively, little research was devoted to pickup and delivery problems with multiple vehicles and time windows (PDPTW) until recently (e.g., [14–16, 21]). Customers in the PDPTW are divided into pickup and delivery pairs. Given such a pair (p, d), a routing must service customers p and d with the same vehicle and must schedule the pickup customer p before the delivery customer d. In standard benchmarks [16], the goal is to minimize the number of used vehicles and, in case of ties, the total travel cost.

The difficulty in pickup and delivery problems, which partly explains why it is less studied than the VRPTW, lies in the side-constraints, which complicate the neighborhoods and invalidate many of the traditional VRPTW moves [18]. However, many practical applications naturally exhibit pickup and delivery constraints in their modeling. This includes dial-a-ride problems, airline scheduling, bus routing, tractor-trailer problems, helicopter support of offshore oil field platforms, and logistics and maintenance support [16]. *More generally, industrial vehicle routing problems are rarely pure and often feature side-constraints. Because of its practical relevance and its side-constraints, the PDPTW is a natural model to evaluate the robustness and scalability of various approaches with respect to side-constraints.*

This paper proposes a two-stage hybrid algorithm for the PDPTW. The overall structure of the algorithm is motivated by the recognition that minimizing the objective function directly may not be the most effective way to decrease the number of routes in vehicle routing problems (e.g., [1, 8]). Indeed, the objective function often drives the search toward solutions with low travel cost, which may make it difficult to reach solutions with fewer routes but higher travel cost. To overcome this limitation, our algorithm divides the search in two steps: (1) the minimization of the number of routes and (2) the minimization of total travel cost. This two-step approach makes it possible to design algorithms tailored to each sub-optimization.

Our algorithm uses two distinct local search procedures to exploit the specificities of each subproblem. The first step uses a very simple simulated annealing (SA) algorithm to minimize the number of routes. The SA algorithm only uses relocation of pairs of customers and one of its key aspects is a lexicographic evaluation function which minimizes the number of routes (primary criterion), maximizes the sum of the squares of the route sizes (secondary criterion), and minimizes travel cost of the routing plan (third criterion). The second criterion was also used successfully in other applications (e.g., graph coloring [9] and, more recently, vehicle routing [1]). The second step uses large neighborhood search (LNS) [19] to minimize total travel cost. It is motivated by our experience that LNS is particularly effective in minimizing total travel cost when given a solution that minimizes the number of routes, and when the problem is highly constrained [1]. The use of LNS for pickup and delivery problems was in fact suggested in the original LNS paper [19], because of its ability to handle side-constraints gracefully.

Experimental results on difficult PDPTW problems demonstrate the effectiveness of the algorithm. On the standard 100, 200, and 600 customers benchmarks [14], our algorithm produces 2, 25 (47%), and 46 (76%) new best solutions respectively, while matching or being close to best known solutions on the other instances. In several 600-customer instances, the algorithm decreases the number of vehicles by as much as 3. These results further confirm the effectiveness of two-stage approaches in vehicle routing, answers positively the open question in [19] on the potential of LNS for the PDPTW, and demonstrate the critical role of the first phase to boost LNS. Similarly, this research confirms that the structure of LNS makes it relatively easy to incorporate pickup and delivery constraints in our previous hybrid algorithm [1], validating Shaw's claim on the robustness of LNS wrt side constraints.

The rest of this paper is organized as follows. Section 2 specifies the PDPTW and describes the notations. Section 3 gives an overview of the overall algorithm. Section 4 presents the simulated annealing algorithm, while Section 5 describes the LNS algorithm for minimizing travel costs. Section 6 presents the experimental results. Section 7 discusses related work and Section 8 concludes the paper.

2 Problem Formulation

This section defines the pickup and delivery vehicle routing problem with time windows (PDPTW) and the various concepts used in this paper.

Customers The problem is defined in terms of N customers who are represented by the numbers $1, \ldots, N$ and a depot represented by the number 0. The set $\{0, 1, \ldots, N\}$ thus represents all the sites considered in the problem. We use *Customers* to represent the set of customers and *Sites* to represent the set of sites (The distinction between customers and sites simplifies the formalization of the problem and of the algorithm). We use $Customers^p$ and $Customers^d$ to denote the pickup and delivery customers respectively. The *travel cost* between sites i and j is denoted by c_{ij}. Travel costs satisfy the triangular inequality $c_{ij} + c_{jk} \geq c_{ik}$. The *normalized travel cost* c'_{ij} between sites i and j is defined as

$$c'_{ij} = c_{ij} \;/ \max_{i,j \in Sites} c_{ij}.$$

Every customer i has a *service time* $s_i \geq 0$. Given a pickup customer i, its delivery counterpart is denoted by $@i$. Every pickup customer has a *demand* $q_i \geq 0$ and its counterpart has demand $q_{@i} = -q_i$.

Vehicles The PDPTW is defined in terms of m identical vehicles. Each vehicle has a capacity Q.

Routes A vehicle route, or route for short, starts from the depot, visits a number of customers at most once, and returns to the depot. In other words, a route is a sequence $\langle 0, v_1, \ldots, v_n, 0\rangle$ or $\langle v_1, \ldots, v_n\rangle$ for short, where all v_i are different. The customers of a route $r = \langle v_1, \ldots, v_n\rangle$, denoted by *cust(r)*, is the set $\{v_1, \ldots, v_n\}$. We also use $route(c)$ to represent the route of customer c. The size of a route, denoted by $|r|$, is the number of customers $|cust(r)|$. The travel cost of a route $r = \langle v_1, \ldots, v_n\rangle$, denoted by *t(r)*, is the cost of visiting all its customers, i.e.,

$$t(r) = c_{0v_1} + c_{v_1v_2} + \ldots + c_{v_{n-1}v_n} + c_{v_n0}.$$

if the route is not empty ($n \geq 1$) and is zero otherwise.

Routing Plan A routing plan is a set of routes $\{r_1, \ldots, r_m\}$ ($m \leq N$) visiting every customer exactly once, i.e.,

$$\begin{cases} \bigcup_{i=1}^{m} cust(r_i) \;=\; Customers \\ cust(r_i) \cap cust(r_j) = \emptyset \;\; (1 \leq i < j \leq m) \end{cases}$$

Observe that a routing plan assigns a unique successor and predecessor to every customer. These successors and predecessors are sites. The successor and predecessor of customer i in routing plan σ are denoted by $succ(i, \sigma)$ and $pred(i, \sigma)$. For simplicity, our definitions often assume an underlying routing plan σ and we use i^+ and i^- to denote the successor and predecessor of i in σ.

Time Windows The customers and the depot have time windows. The time window of a site i is specified by an interval $[e_i, l_i]$, where e_i and l_i represent the earliest and latest arrival times respectively. Vehicles must arrive at a site before the end of the time window l_i. They may arrive early but they have to wait until time e_i to begin service. Observe that e_0 represents the time when all vehicles in the routing plan leave the depot and that l_0 represents the time when they must all return to the depot. The *departure time* of customer i, denoted by δ_i, is defined recursively as

$$\begin{cases} \delta_0 = 0 \\ \delta_i = \max(\delta_{i^-} + c_{i^- i}\ ,\ e_i) + s_i \quad (i \in \mathit{Customers}). \end{cases}$$

The earliest service time of customer i, denoted by a_i, is defined as

$$a_i = \max(\delta_{i^-} + c_{i^- i}\ ,\ e_i) \quad (i \in \mathit{Customers}).$$

The earliest arrival time of a route $r = \langle v_1, \ldots, v_n \rangle$, denoted by $a(r)$, is given by $\delta_{v_n} + c_{v_n 0}$ if the route is not empty and is e_0 otherwise. A routing plan satisfies the time window constraint for customer i if $a_i \leq l_i$. A routing plan σ satisfies the time window constraint for the depot if $\forall r \in \sigma \ : \ a(r) \leq l_0$.

Capacities The demand of a route r at customer c, denoted by $q(c)$, is the sum of demands of customers on r up to c, i.e.,

$$q(c) = \sum_{i \in cust(r)\ \&\ \delta_i \leq \delta_c} q_i.$$

The capacity constraint of a customer is satisfied if $q(c) \leq Q$.

Pickup and Deliveries The pickup and deliveries are represented by *precedence* and *coupling* constraints. The precedence constraint of $c \in \mathit{Customers}^p$ is satisfied if $d_c \leq \delta_{@c}$. Similarly, the coupling constraint of c is satisfied if $route(c) = route(@c)$.

The PDPTW A solution to the PDPTW is a routing plan $\sigma = \{r_1, \ldots, r_m\}$ satisfying the capacity constraints, time window constraints, and pickup and delivery constraints, i.e.,

$$\begin{cases} q(i) \leq Q & (i \in \mathit{Customers}) \\ a(r_j) \leq l_0 & (1 \leq j \leq m) \\ a_i \leq l_i & (i \in \mathit{Customers}) \\ route(i) = route(@i) & (i \in \mathit{Customers}^p) \\ \delta_i \leq \delta_{@i} & (i \in \mathit{Customers}^p) \end{cases}$$

```
Function PDPTWoptimize

1.  σ := RouteMinimize();
2.  return TravelCostMinimize(σ);
```

Fig. 1. The Two-Stage Hybrid Algorithm for Minimizing Routes and Travel Costs.

The size of a routing plan σ, denoted by $|\sigma|$, is the number of non-empty routes in σ, i.e., $\{r \in \sigma \mid cust(r) \neq \emptyset\}$. The PDPTW problem consists of finding a solution σ which minimizes the number of vehicles and, in case of ties, the total travel cost, i.e., a solution σ minimizing the objective function specified by the lexicographic order

$$f(\sigma) = \langle |\sigma|, \sum_{r \in \sigma} t(r) \rangle.$$

3 Overview of the Algorithm

Our algorithm is motivated by the recognition that minimizing the original objective function is not always the most effective way to approach the problem. Indeed, the objective function often drives the search towards solutions with low travel costs. The reduction in the number of routes occurs more as a side-effect of the travel cost minimization than as a primary feature of the search. In addition, focusing on travel cost may make it extremely difficult to reach solutions with fewer routes since it may require considerable degradation of the travel cost component of the objective function. To overcome this limitation, our algorithm separates the optimization into two stages: the minimization of the number of routes and the minimization of travel costs. Each of these two stages is optimized by an algorithm exploiting the underlying structure of the subproblem. (Of course, the second phase may sometimes reduce the number of vehicles as well as a side-effect of reducing travel distance.) The overall algorithm is depicted in Figure 1. The next two sections discuss each suboptimization in detail. Observe that two-stage algorithms has been very successful on the traditional VRPTW, where they have produced many new best solutions recently [1, 8].

4 Minimizing the Number of Routes

The first stage of our algorithm consists of minimizing the number of routes or, equivalently, the number of vehicles used in the routing plan. It uses simulating annealing [11] because of its success in reducing routes on the VRPTW and the overall simplicity of its implementation.

4.1 The Neighborhood

The SA neighborhood is based on a simple pair relocation operator, which is also used in [12, 14, 16]. Given a solution σ, $\mathcal{N}(\sigma)$ denotes the neighborhood of

σ, i.e., the set of feasible solutions that can be reached from σ by using pair relocation, which is defined as follows.

Pair Relocation For customers i, j, and k, first place i after j, i.e., remove arcs (i^-, i), (i, i^+), (j, j^+) and add arcs (i^-, i^+), (j, i), and (i, j^+). Second, place $@i$ after k, i.e., remove arcs $(@i^-, @i)$, $(@i, @i^+)$, (k, k^+), and add arcs $(@i^-, @i^+)$, $(k, @i)$, and $(@i, k^+)$.

A Random Sub-Neighborhood An interesting feature of our SA algorithm is how it explores the neighborhood. Each iteration focuses on a (random) sub-neighborhood of $\mathcal{N}$ obtained by randomly choosing a customer c from *Customers* and by constructing all the pair relocations using c and $@c$. The sub-neighborhood is explored exhaustively to determine whether it contains a solution improving the best available routing plan. We denote by $\mathcal{N}(c, \sigma)$ the subset of $\mathcal{N}(\sigma)$ that can be reached by using pair relocation and customers c and $@c$.

4.2 The Evaluation Function

The evaluation function is another fundamental aspect of our simulated annealing algorithm. As mentioned earlier, the objective function $\langle |\sigma|, \sum_{r \in \sigma} t(r) \rangle$ is not always appropriate, since it may lead the search to solutions with a small travel cost and makes it impossible to remove routes. To overcome this limitation, our simulated algorithm uses a more complex lexicographic ordering

$$e(\sigma) = \langle |\sigma|, -\sum_{r \in \sigma} |r|^2, \sum_{r \in \sigma} t(r) \rangle.$$

especially tailored to minimize the number of routes. The first component is, of course, the number of routes. The second component maximizes $\sum_{r \in \sigma} |r|^2$ which means that it favors solutions containing routes with many customers and routes with few customers over solutions where customers are distributed more evenly among the routes. The intuition is to guide the algorithm into removing customers from some small routes and adding them to larger routes. Components of this type are used on many problems, a typical example being graph coloring [9]. The third component minimizes the travel cost of the routing plan.

4.3 The Simulated Annealing Algorithm

Figure 2 depicts the SA algorithm. The algorithm consists of a number of local searches (lines 2-22), each of which start from the best solution found so far and from the starting temperature. Each local search performs a number of iterations (lines 5-20) and decreases the temperature (line 21). These two steps are repeated until the time limit is exhausted or the temperature has reached its lower bound. Lines 6-19 describe one iteration and are most interesting. Lines 6-8 compute the sub-neighborhood

$$\mathcal{N}(c, \sigma) = \langle \sigma_1, \ldots, \sigma_s \rangle \text{ where } e(\sigma_i) \leq e(\sigma_j) \ (i < j)$$

```
    Function ROUTEMINIMIZE

1.  σ_b := GETINITIALSOLUTION();
2.  while (time < timeLimit) {
3.     σ := σ_b;
4.     t := startingTemperature;
5.     while (time < timeLimit & t > temperatureLimit) {
6.        for( i := 1; i ≤ maxIterations; i++) {
7.           c := RANDOM(Customers);
8.           ⟨σ_1, ..., σ_s⟩ := N(c,σ) where e(σ_i) ≤ e(σ_j) (i < j);
9.           if e(σ_1) < e(σ_b) then {
10.             σ_b := σ_1;
11.             σ := σ_1;
12.          } else {
13.             r := ⌊random([0,1])^β × s⌋;
14.             Δ := e(σ) − e(σ_r);
15.             if Δ ≥ 0 then
16.                σ := σ_r;
17.             else if random([0,1]) ≤ e^(Δ/t) then
18.                σ := σ_r;
19.          }
20.       }
21.       t := α × t;
22.    }
23. }
24. return σ_b;
```

Fig. 2. The Simulated Annealing Algorithm to Minimize the Number of Routes

for a random customer. Lines 9-11 select the solution σ_1 minimizing f in $\mathcal{N}(c, \sigma)$ if it improves the best solution found so far. These lines introduce an aspiration criterion [5] in the simulated annealing algorithm. Lines 13-18 are the core of the algorithm. Line 13 chooses a random element $\sigma_r \in \mathcal{N}(c, \sigma)$ and σ_r is selected as the next routing plan if it does not degrade the current solution (line 15) or with the traditional probability of simulated annealing otherwise (line 17). Observe also line 13 which biases the search towards "good" moves in $\mathcal{N}(c, \sigma)$ when $\beta > 1$.

5 Minimizing the Travel Cost

Our algorithm uses large neighborhood search (LNS) to minimize travel cost. LNS was proposed in [19] for the VRPTW, where it was shown particularly effective on the class 1 problems from the Solomon benchmarks, producing several improvements over the then best published solutions. However, the algorithm performed poorly on the class 2 benchmarks where it could not reduce the number of routes satisfactorily [19] (Our own experimental results confirm the findings of [19] on pickup and delivery problems). By separating the overall optimization in two stages, our algorithm directly addresses this LNS weakness and exploits its strength in minimizing travel cost. The rest of this section describes the LNS algorithm in detail. In general, the algorithm adapts the heuristics and strategies described in [19], although it departs on a number of issues which are critical to scale LNS to large-scale problems.

The Neighborhood and the Evaluation Function Given a solution σ, the neighborhood of LNS, denoted by $\mathcal{N}_R(\sigma)$, is the set of solutions that can be reached from σ by relocating at most p pairs of customers (where p is a parameter of the implementation). Since LNS also uses subneighborhoods and explores them in a specific order, we use additional notations. In particular, $\mathcal{N}_R(\sigma, S)$ denotes the set of solutions that can be reached from σ by relocating the customers in S. Also, given a partial solution σ with customers $Customers \setminus S$, $\mathcal{N}_I(\sigma, S)$ denotes the solutions that can be obtained by inserting the customers S in σ. Finally, LNS uses the original objective function, which involves the number of routes. This is important since, in some cases, minimizing travel costs makes it possible to decrease the number of routes further.

The Algorithm At a high level, the LNS algorithm can be seen as a local search where each iteration selects a neighbor σ_c in $\mathcal{N}_R(\sigma_b)$ and accepts the move if $f(\sigma_c) < f(\sigma_b)$. It can be formalized as follows:

```
for(i := 1;i ≤ maxIterations; i++) {
    SELECT σ_c ∈ N_R(σ_b);
    if f(σ_c) < f(σ_b) then
        σ_b := σ_c;
}
```

In practice, it is important to refine and extend the above algorithm in three ways. The first modification consists of exploring the neighborhood by increasing number of allowed relocations. The second change generalizes the algorithm to a sequence of local searches. The third modification consists of exploring the subneighborhood $\mathcal{N}_R(\sigma_b, S)$ more exhaustively to find its best solution. The overall algorithm is depicted in Figure 3. Observe line 2 which adds another loop, line 4 which selects a set of customers S of size $2n$, line 5 which selects a best neighbor in $\mathcal{N}_R(\sigma_b, S)$, and line 8 which reinitializes the number of allowed iterations. In fact, the algorithm is now very close to variable neighborhood search [6]. It remains to describe how to select customers and how to implement line 5 in the above algorithm.

Selecting Customers to Relocate The LNS algorithm adapts the traditional customer selection [19] to the PDPTW. The implementation is depicted in Figure 4. It first selects a customer pair randomly (lines 1-2) and iterates lines 4-7 to remove the $n - 1$ remaining customer pairs. Each such iteration selects a pickup customer from S (the already selected customers) and ranks the remaining pickup customers according to a relatedness criterion (lines 4-5). The new

```
Function TravelCostMinimize(σ_b)

for(l := 1; l ≤ maxSearches; l++)
    for(n := 1; n ≤ p; n++)
        for(i := 1; i ≤ maxIterations; i++) {
            S := SelectCustomers(σ_b, n);
            SELECT σ_c ∈ N_R(σ_b, S) SUCH THAT f(σ_c) = min_{σ∈N_R(σ_b,S)} f(σ);
            if f(σ_c) < f(σ_b) then {
                σ_b := σ_c;
                i := 1;
            }
```

Fig. 3. The LNS Algorithm to Minimize Travel Cost

```
Function SelectCustomers(σ, n)

1.  c := { RANDOM(Customers^p) };
2.  S := {c, @c};
3.  for(i := 2; i ≤ n; i++) {
4.      c := RANDOM(S ∩ Customers^p);
5.      ⟨c_0, ..., c_{N/2 - i}⟩ := Customers^p \ S SUCH THAT
                        relateness(c, c_i) ≥ relateness(c, c_j)  (i ≤ j);
6.      r := ⌊random([0, 1])^β × |Customers^p \ S|⌋;
7.      S := S ∪ {c_r, @c_r};
8.  }
```

Fig. 4. Selecting Customers in the LNS Algorithm

customer to insert is randomly selected in line 6 and, once again, the algorithm biases the selection toward related neighbors. The relatedness measure is defined as in [19]:

$$relateness(i, j) = \frac{1}{c'_{ij} + v_{ij}}$$

where $v_{ij} = 1$ if $route(i) \neq route(j)$ and is zero otherwise.

The Exploration Algorithm Our LNS algorithm uses a branch and bound algorithm to explore the selected sub-neighborhood. The algorithm is depicted in Figure 5. If the set of customers to insert is empty, the algorithm checks whether the current solution improves the best solution found so far. Otherwise, it selects the customer pair whose best insertion degrades the objective function the most. The algorithm then explores all the partial solutions obtained by inserting c and $@c$ by increasing order of their travel costs. Also, observe that only the partial solutions whose lower bounds are better than the best solution found so far are explored by the algorithm. The lower bound satisfies the inequality $Bound(\sigma, S) \leq \min_{\sigma' \in \mathcal{N}_I(\sigma,S)} f(\sigma')$.

```
Function LDSExplore(σ_c, S, σ_b, d, dmax)

if d > dmax then {
    if S = ∅ then {
        if f(σ_c) < f(σ_b) then σ_b := σ_c;
    } else {
        c := arg-max_{c∈S} min_{σ∈N_I(σ,{c,@c})} f(σ);
        S_c := S \ {c, @c};
        ⟨σ_0, ..., σ_k⟩ := N_I(σ, {c, @c}) WHERE f(σ_i) ≤ f(σ_j) (i ≤ j);
        for(i := 1; i ≤ k; i++) {
            if Bound(σ_i, S_c) < f(σ_b) then {
                LDSExplore(σ_i, S_c, σ_b, d, dmax);
                d := d + 1;
            }
        }
    }
}
```

Fig. 5. The Branch and Bound Algorithm with a Limited Discrepancy Strategy.

The bounding function is the cost of a minimum spanning k-tree [4] on the insertion graph with the depot as distinguished vertex, generalizing the well-known 1-tree bound of the traveling salesman problem. The insertion graph vertices are the customers. Given a solution σ over customers $C = \cup_{r \in \sigma} cust(r)$ and a set S of vertices to insert, the insertion graph edges come from three different sets:

1. the edges already in σ;
2. all the edges between customers in S;
3. all the feasible edges connecting a customer from C and a customer from S.

For large-scale problems, finding the best reinsertion is too time-consuming. Our algorithm uses limited discrepancy search (LDS) [7] to explore only a small part of the search tree. More precisely, it only uses one LDS phase which allows up to d discrepancies. Note that the tree is not binary and the heuristic selects the insertion points by increasing lower bounds.

Observe also that the neighborhood $\mathcal{N}_I(\sigma, \{c, @c\})$ is of size $O(N^2)$. On large-scale problems or on problems with wide time windows, the computation cost of maintaining this neighborhood during branching can become quite expensive. To overcome this difficulty, our algorithm only maintains the y best feasible insertion points found initially (where y is an implementation parameter). This approximation is critical to scale LNS to large-scale problems.

6 Experimental Results

This section reports preliminary experimental results on the algorithm. All results are given on a 1.2Ghz AMD Athlon Thunderbird K7 processor running

	Best			SA/LNS				Best			SA/LNS		
	V	TD	Pub	V	TD	Time		V	TD	Pub	V	TD	Time
lc101	10	828.937	LL	10	828.937	0.00	lc201	3	591.557	LL	3	591.557	0.00
lc102	10	828.937	LL	10	828.937	0.00	lc202	3	591.557	LL	3	591.557	0.00
lc103	9	1082.35	SAM	**9**	**1035.35**	0.02	lc203	3	585.564	LL	3	591.173	0.00
lc104	9	860.011	SAM	9	860.011	0.33	lc204	3	590.599	SAM	3	590.599	4.47
lc105	10	828.937	LL	10	828.937	0.00	lc205	3	588.876	LL	3	588.876	0.00
lc106	10	828.937	LL	10	828.937	0.00	lc206	3	588.493	LL	3	588.493	0.00
lc107	10	828.937	LL	10	828.937	0.01	lc207	3	588.286	LL	3	588.286	0.00
lc108	10	826.439	LL	10	826.439	0.00	lc208	3	588.324	LL	3	588.324	0.00
lc109	9	1027.60	SAM	**9**	**1000.6**	42.57							
lr101	19	1650.80	LL	19	1650.80	0.00	lr201	4	1253.23	SAM	4	1253.23	0.01
lr102	17	1487.57	LL	17	1487.57	0.01	lr202	3	1197.67	LL	3	1197.67	0.01
lr103	13	1292.68	LL	13	1292.68	0.01	lr203	3	949.396	LL	3	949.396	0.13
lr104	9	1013.39	LL	9	1013.39	0.00	lr204	2	849.05	LL	2	849.05	0.53
lr105	14	1377.11	SAM	14	1377.11	0.00	lr205	3	1054.02	LL	3	1054.02	0.01
lr106	12	1252.62	LL	12	1252.62	0.00	lr206	3	931.625	LL	3	931.625	0.78
lr107	10	1111.31	LL	10	1111.31	0.00	lr207	2	903.056	LL	2	903.056	0.01
lr108	9	968.966	LL	9	968.966	0.00	lr208	2	734.848	LL	2	734.848	0.01
lr109	11	1208.96	SAM	11	1208.96	0.00	lr209	3	930.586	SAM	3	930.586	12.97
lr110	10	1159.35	LL	10	1159.35	0.00	lr210	3	964.224	LL	3	964.224	0.04
lr111	10	1108.90	LL	10	1108.9	0.00	lr211	2	884.294	LL	2	913.837	1.23
lr112	9	1003.77	LL	9	1003.77	0.00							
lrc101	14	1708.70	SAM	14	1708.70	0.00	lrc201	4	1406.94	SAM	4	1406.94	0.14
lrc102	12	1558.07	SAM	12	1558.07	0.00	lrc202	3	1374.27	LL	3	1374.27	0.01
lrc103	11	1258.74	LL	11	1258.74	0.00	lrc203	3	1089.07	LL	3	1089.07	0.01
lrc104	10	1128.40	SAM	10	1128.40	0.01	lrc204	3	818.67	SAM	3	818.663	0.18
lrc105	13	1637.62	SAM	13	1637.62	0.00	lrc205	4	1302.20	LL	4	1302.20	0.05
lrc106	11	1424.73	SAM	11	1424.73	0.00	lrc206	3	1159.03	SAM	3	1159.03	0.01
lrc107	11	1230.14	SAM	11	1230.14	0.00	lrc207	3	1062.05	SAM	3	1062.05	0.05
lrc108	10	1147.43	SAM	10	1147.43	0.00	lrc208	3	852.758	LL	3	852.758	0.11

Table 1. 100 Customers

Linux, using g++ with the -O flag, and double precision floating-point numbers. The results are rounded to six significant digits. Our experimental results use the standard PDPTW benchmarks available at

http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks.html

See [14] for their descriptions. For prior results, we use the abbreviations LL=[14] and SAM=[21].

Our algorithm was run with a fixed configuration on all benchmarks, which is necessarily suboptimal, in order to demonstrate the robustness of the algorithm across many different problems. Simulated annealing was allowed to run for 5 minutes, with initial temperature of 2000, cooling factor of 0.95, 2500 iterations per temperature, a minimum temperature of 0.01, and $\beta = 10$. LNS was run with a maximum customer pairs removed of 18, 500 attempts for each removal size, 15 as the relatedness determinism, 3 discrepancies, and 15 initial insertion points maintained for each pair removed. LNS is allowed 60 minutes to find a solution (90 minutes for the 600-customer benchmarks) although, in practice, it finds the best solution much quicker in many cases.

Tables 1, 2, and 3 report the experimental results for 100, 200, and 600 customers. The tables compare our algorithm with the best known solutions on these standard benchmarks. For each benchmark, we give the number of vehicles and the travel cost of the best known solution, as well as the best solutions found by our algorithm among 5 runs (10 for the 600-customer instances). We also report the time in minutes taken by LNS to the best solution (the simulating

	Best			SA/LNS				Best			SA/LNS		
	V	TD	Pub	V	TD	Time		V	TD	Pub	V	TD	Time
lc1_2_1	20	2704.57	LL	20	2704.57	0.00	lc2_2_1	6	1931.44	SAM	6	1931.44	0.00
lc1_2_2	19	2764.56	LL	19	2764.56	0.05	lc2_2_2	6	1881.40	SAM	6	1881.40	0.09
lc1_2_3	18	2772.18	SAM	**17**	**3134.08**	26.78	lc2_2_3	6	1845.54	SAM	**6**	**1844.33**	2.05
lc1_2_4	17	2708.90	SAM	**17**	**2693.41**	14.29	lc2_2_4	6	1767.12	SAM	6	1778.54	5.63
lc1_2_5	20	2702.05	LL	20	2702.05	0.00	lc2_2_5	6	1891.21	LL	6	1891.21	0.00
lc1_2_6	20	2701.04	LL	20	2701.04	0.00	lc2_2_6	6	1857.78	SAM	6	1857.78	0.05
lc1_2_7	20	2701.04	LL	20	2701.04	0.05	lc2_2_7	6	1850.13	SAM	6	1850.13	0.01
lc1_2_8	20	2689.83	SAM	20	2689.83	0.09	lc2_2_8	6	1824.34	LL	6	1824.34	3.87
lc1_2_9	18	2724.24	LL	18	2724.24	0.36	lc2_2_9	6	1854.21	SAM	6	1854.21	1.32
lc1_2_10	18	2741.56	LL	18	2741.56	1.00	lc2_2_10	6	1817.45	SAM	6	1817.45	0.27
lr1_2_1	20	4819.12	SAM	20	4819.12	2.07	lr2_2_1	5	4073.10	SAM	5	4073.10	1.58
lr1_2_2	18	4228.21	SAM	**17**	**4666.09**	1.86	lr2_2_2	4	3796.16	LL	**4**	**3796.00**	7.36
lr1_2_3	15	3761.52	LL	**15**	**3657.19**	3.53	lr2_2_3	4	3100.03	SAM	4	3100.38	46.49
lr1_2_4	11	2968.57	SAM	**10**	**3146.06**	21.41	lr2_2_4	3	2754.96	SAM	3	2956.15	30.14
lr1_2_5	17	4331.14	SAM	**16**	**4760.18**	5.22	lr2_2_5	4	3438.39	SAM	4	3438.39	2.46
lr1_2_6	15	4068.74	SAM	**14**	**4175.16**	2.03	lr2_2_6	4	3201.54	SAM	4	3208.53	16.74
lr1_2_7	13	3190.75	SAM	**12**	**3851.36**	7.12	lr2_2_7	3	3190.75	LL	3	3337.28	41.52
lr1_2_8	10	2718.23	SAM	**9**	**2871.67**	41.18	lr2_2_8	3	2295.44	SAM	3	2407.66	39.59
lr1_2_9	15	4224.35	SAM	**14**	**4411.54**	37.14	lr2_2_9	4	3198.44	SAM	4	3198.44	1.59
lr1_2_10	12	3654.80	LL	**11**	**3744.95**	4.70	lr2_2_10	3	3447.42	SAM	3	3478.67	44.10
lrc1_2_1	19	3606.06	SAM	19	3606.06	0.06	lrc2_2_1	7	2997.06	SAM	**6**	**3690.10**	10.80
lrc1_2_2	16	3621.30	SAM	**15**	**3681.36**	47.48	lrc2_2_2	6	2674.16	SAM	**6**	**2666.01**	0.41
lrc1_2_3	14	3255.33	SAM	**13**	**3161.75**	27.06	lrc2_2_3	5	2620.85	SAM	**5**	**2523.59**	53.78
lrc1_2_4	10	2890.02	SAM	**10**	**2655.27**	10.67	lrc2_2_4	4	2202.89	SAM	4	2795.7	4.94
lrc1_2_5	16	3750.52	SAM	**16**	**3715.81**	2.20	lrc2_2_5	5	2785.75	SAM	**5**	**2776.93**	2.86
lrc1_2_6	17	3368.66	SAM	17	3368.66	0.97	lrc2_2_6	5	2707.75	SAM	5	2707.96	2.51
lrc1_2_7	16	3326.18	SAM	**15**	**3417.16**	17.17	lrc2_2_7	5	2546.77	SAM	**4**	**3050.03**	16.67
lrc1_2_8	14	3164.50	LL	**14**	**3087.62**	14.99	lrc2_2_8	4	2442.04	SAM	**4**	**2401.84**	40.99
lrc1_2_9	15	3100.88	SAM	**14**	**3129.65**	20.97	lrc2_2_9	4	2209.94	SAM	4	2750.30	23.75
lrc1_2_10	13	2884.71	SAM	**13**	**2833.85**	56.06	lrc2_2_10	4	2059.16	SAM	**3**	**2699.55**	31.46

Table 2. 200 Customers

annealing time being fixed). Bold-face entries indicate improvement over the best known solution.

The tables indicate that our algorithm produces very high-quality solutions across the board. For 100 customers, it produces two new best solutions and matches 54 (93%). For 200 customers, it improves 28 (47%) best solutions and matches 24 (40%). For 600 customers, it produces 46 new solutions (77%), while matching 5 more (8%). Since previous work does not report computation times, it is impossible to make comparisons. Most 100-customer instances are solved quickly, spending little time in LNS in almost all instances. On the 200-customer instances, the variation in running time is much larger and can range from a few seconds to almost an hour. The 600-customer instances spend significant amount of time in LNS. Note that these times are comparable to those of our state-of-the-art VRPTW algorithm [1].

In summary, these preliminary results are extremely encouraging and demonstrate that the approach produces very high-quality results in reasonable times.

7 Discussion and Related Work

This paper originated as an attempt to generalize our hybrid algorithm for the VRPTW to pickup and delivery problems. *The hope was to validate the claim in*

	Best			SA/LNS				Best			SA/LNS		
	V	TD	Pub	V	TD	Time		V	TD	Pub	V	TD	Time
lc1_6_1	60	14095.6	LL	60	14095.6	0.01	lc2_6_1	19	7977.98	SAM	19	7977.98	0.88
lc1_6_2	59	14164.0	LL	**58**	**14379.5**	1.96	lc2_6_2	19	8483.50	SAM	**19**	**8253.67**	19.06
lc1_6_3	54	15920.6	SAM	**51**	**14569.3**	46.45	lc2_6_3	18	7500.13	SAM	**18**	**7436.50**	64.37
lc1_6_4	48	13567.5	SAM	48	13750.6	89.21	lc2_6_4	18	8513.88	LL	18	9479.88	89.99
lc1_6_5	60	14086.3	LL	60	14086.3	0.82	lc2_6_5	19	8596.84	LL	**19**	**8047.37**	53.37
lc1_6_6	60	14090.8	LL	60	14090.8	0.51	lc2_6_6	19	8328.40	SAM	**19**	**8237.58**	53.36
lc1_6_7	60	14083.8	LL	60	14083.8	0.82	lc2_6_7	19	8704.89	SAM	**19**	**8038.56**	48.81
lc1_6_8	59	14670.4	SAM	**59**	**14554.3**	11.32	lc2_6_8	18	8147.00	LL	19	7855.38	88.57
lc1_6_9	56	14993.4	LL	**55**	**14648.1**	85.44	lc2_6_9	19	8258.20	SAM	19	8304.29	43.55
lc1_6_10	57	15337.7	LL	**54**	**14870.3**	59.96	lc2_6_10	18	7963.86	SAM	**18**	**7853.27**	55.24
lr1_6_1	59	24149.1	SAM	**59**	**22838.3**	53.04	lr2_6_1	12	18842.4	SAM	**12**	**18840.8**	23.63
lr1_6_2	46	22854.4	SAM	**45**	**20985.7**	55.46	lr2_6_2	11	20243.4	LL	11	22348.2	59.90
lr1_6_3	37	19975.6	LL	**37**	**18685.9**	82.16	lr2_6_3	10	17855.1	SAM	**10**	**16657.5**	59.69
lr1_6_4	28	14717.3	SAM	**28**	**14199.9**	86.05	lr2_6_4	7	14595.6	SAM	**7**	**14223.2**	82.71
lr1_6_5	42	21750.6	SAM	**40**	**22188.8**	78.88	lr2_6_5	11	15907.5	SAM	**10**	**21250.1**	88.26
lr1_6_6	37	20376.7	SAM	**35**	**20406.2**	59.73	lr2_6_6	10	19160.3	SAM	**9**	**21722.8**	89.44
lr1_6_7	31	16709.3	SAM	**28**	**16963.8**	86.42	lr2_6_7	8	16778.0	LL	**8**	**16262.0**	59.80
lr1_6_8	21	12978.3	SAM	**21**	**12620.1**	88.01	lr2_6_8	8	11671.2	SAM	**6**	**13344.1**	38.08
lr1_6_9	37	21821.2	SAM	**34**	**21273.3**	88.05	lr2_6_9	10	18791.2	SAM	**9**	**18853.4**	58.22
lr1_6_10	30	19120.7	LL	**29**	**18373.9**	59.09	lr2_6_10	8	19070.6	SAM	**8**	**18869.2**	17.08
lrc1_6_1	54	18251.2	SAM	**53**	**17930.0**	24.34	lrc2_6_1	17	13172.6	SAM	**17**	**13111.6**	22.16
lrc1_6_2	47	16736.9	SAM	**45**	**16040.3**	32.52	lrc2_6_2	15	11587.8	SAM	**15**	**11463.0**	55.74
lrc1_6_3	39	15525.2	SAM	**36**	**14407.6**	54.82	lrc2_6_3	13	12428.64	SAM	**11**	**15167.3**	78.21
lrc1_6_4	27	12138.4	SAM	**25**	**11308.6**	89.82	lrc2_6_4	11	8282.80	SAM	**8**	**12512.5**	89.42
lrc1_6_5	49	17368.4	SAM	**47**	**16803.9**	87.75	lrc2_6_5	15	12401.5	SAM	**15**	**12309.7**	46.47
lrc1_6_6	48	17869.8	SAM	**45**	**17126.4**	89.60	lrc2_6_6	13	12679.3	SAM	14	12894.1	72.36
lrc1_6_7	42	16020.3	SAM	**40**	**15493.5**	59.15	lrc2_6_7	12	12998.4	SAM	12	13851.5	38.01
lrc1_6_8	37	15626.0	LL	**36**	**15352.6**	58.93	lrc2_6_8	12	10898.3	SAM	12	11877.8	89.35
lrc1_6_9	37	15342.6	SAM	**37**	**15253.7**	71.08	lrc2_6_9	11	11917.2	SAM	11	14810.5	56.60
lrc1_6_10	34	14137.5	SAM	**33**	**13830.5**	59.25	lrc2_6_10	10	13165.4	SAM	**9**	**12874.8**	73.08

Table 3. 600 Customers

[19] that LNS should handle side-constraints gracefully. The algorithm presented here keeps the two-stage approach of the original algorithm, but it differs in several important ways. First, the SA algorithm was no longer able to use the wealth of moves available for the VRPTW. It is now based on a single move, pair relocation, which is also used in other algorithms for the PDPTW [12, 16]. *However, despite its simplicity, the SA algorithm boosts the quality of LNS significantly, since LNS cannot decrease the number of vehicles sufficiently on many benchmarks.* The LNS adaptation to the PDPTW was less drastic. The key idea is to select and reinsert pickup and delivery customers in pairs. Additional approximations, i.e., maintaining only a subset of the insertion points, was also necessary to obtain high-quality solutions on large-scale problems and problems with many customers.

Single vehicle pickup and delivery problems were first introduced by [17] in 1980. Small instances of multiple vehicle problems with time windows were introduced and solved optimally in [3]. A good survey of the various models and techniques utilized in early work on pickup and delivery problems can be found in [18]. More recent advances on multiple vehicle problems has focused on metaheuristics including tabu search and simulated annealing. Tabu search was used in [12, 16] to minimize another objective function, i.e., total schedule duration, in pickup and delivery problems. They use pair relocation as one of their neighborhood operators to move customers between routes. They also introduced pair exchange operators and a single customer relocation within the same route. A tabu search/simulated annealing hybrid was successfully used in [14] to solve the PDPTW. This algorithm was compared in detail in the experimental section. Excellent results were also produced in [21] but the report is not available unfortunately. A squeaky wheel algorithm was also proposed in [15], but it does not seem to be competitive with the two earlier algorithms in solution quality.

8 Conclusion

This paper proposed a two-stage hybrid algorithm for pickup and delivery vehicle routing problems with multiple vehicles and time windows (PDPTW). The algorithm minimizes the number of vehicles using simulated annealing in the first stage, and minimizes travel cost using LNS in the second stage. Experimental results show the effectiveness of the approach which produced many new best solutions on instances with 100, 200, and 600 customers.

More precisely, the results demonstrate that the two-stage approach boosts the solution quality of LNS significantly, that a simple simulated annealing algorithm is excellent in reducing the number of vehicles, and that LNS, with appropriate reductions in its underlying search space, is very effective in optimizing travel cost. The paper also settles positively the open issue in the original LNS paper, which advocated the use of LNS for the PDPTW because of its ability to handle side-constraints gracefully. More generally, these results seem to indicate that a two-step approach, combining SA and LNS, should produce high-quality results for vehicle routing problems with additional side-constraints.

There are many open issues that deserve attention. As research moves to large-scale problems involving several hundreds or thousands of customers, scaling the algorithms raise new interesting challenges that were not systematically studied here. It is indeed unlikely that the same algorithmic configuration would perform effectively on all instances. It would be interesting to study the impact of various decisions on the behaviour of the algorithm and to study how to tune these decisions dynamically during search. It is also clear that a unique algorithm does not exist for all purposes. It would be interesting to study algorithms producing high-quality results for the PDPTW in short times, even if there is some decrease in solution quality and robustness. Finally, it is of great interest to evaluate the approach on complex problems with additional side-constraints. Obviously, progress in that respect will strongly depend on the availability of such complex instances.

Acknowledgments

This work is partly supported by an NDSEG fellowship from (ASEE) and an NSF ITR DMI-01210495 award.

References

1. Bent, R. and Van Hentenryck, P. A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Transportation Science* (to appear).

2. Chiang, W. and Russell, R. Simulated Annealing Metaheuristics for the Vehicle Routing Problem with Time Windows. *Annals of Operations Research*, 63:3–27 (1996).
3. Dumas, Y., Desrosiers, J. and Soumis, F. The Pickup and Delivery Problem with Time Windows *European Journal of Operational Research*, 54:7–22. (1991).
4. Fisher, M., Joernsten, K., and Madsen, O. Vehicle routing with time windows: Two optimization algorithms. *Operations Research*, 45(3):488–492 (1997).
5. Glover, F. Tabu Search. *Orsa Journal of Computing*, 1:190–206 (1989).
6. Hansen, P. and Mladenovic, N. An introduction to variable neighborhood search. In Voss, S., Martello, S., Osman, I. H., and Roucairol, C., editors, *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer Academic Publishers (1998).
7. Harvey, W. and Ginsberg, M. Limited Discrepancy Search. In *Proceedings of IJCAI-95*, Montreal, Canada (1995).
8. Homberger, J. and Gehring, H. Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows. *INFOR*, 37:297–318 (1999).
9. Johnson, D., Aragon, C., McGeoch, L., and Schevon, C. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research*, 39(3):378–406 (1991).
10. Kindervater, G. and Savelsbergh, M. Vehicle Routing: Handling Edge Exchanges. In Aarts, E. and Lenstra, J., editors, *Local Search in Combinatorial Optimization*, chapter 10, pages 337–360. John Wiley & Sons Ltd (1997).
11. Kirkpatrick, S., Gelatt, C., and Vecchi, M. Optimization by Simulated Annealing. *Science*, 220:671–680 (1983).
12. Lau, H. and Liang, Z. Pickup and Delivery with Time Windows: Algorithms and Test Case Generations In *Proceedings of the 13th IEEE Conf. on Tools with Artificial Intelligence (ICTAI)*, 333-340 (2001).
13. Lenstra, J. and Rinnooy Kan, A. H. G. Complexity of Vehicle Routing and Scheduling Problems. *Networks*, 11:221–227 (1981).
14. Li, H. and Lim, A. A Metaheuristic for the Pickup and Delivery Problem with Time Windows In *13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 160–170 (2001).
15. Lim, H., Lim, A., and Rodrigues, B. Solving the Pickup and Delivery Problem with Time Windows Using Squeaky Wheel Optimization with Local Search In *American Conference on Information Systems (AMCIS)*, (2002)
16. Nanry, W. and Barnes, J. Solving the Pickup and Delivery Problem with Time Windows Using Reactive Tabu Search *Transportation Research Part B*, 34:107–121 (2000).
17. Psarafis, H. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-A-Ride Problem *Transportation Science*, 14:130–154 (1980).
18. Savelsbergh, M and Sol, M. The General Pickup and Delivery Problem *Transportation Science*, 29 (1):107–121 (1995).
19. Shaw, P. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Principles and Practice of Constraint Programming*, pages 417–431 (1998).
20. Solomon, M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35 (2):254–265 (1987).
21. Unpublished Results SINTEF Applied Mathematics-Department of Optimisation, Technical Report in Progress *http://www.sintef.no/static/am/opti/projects/top/vrp/benchmarks* (2003).