TRANSPORTATION SCIENCE
Vol. 38, No. 4, November 2004, pp. 515–530
ISSN 0041-1655 | EISSN 1526-5447 | 04 | 3804 | 0515

DOI 10.1287/trsc.1030.0049

A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows

Russell Bent, Pascal Van Hentenryck
Brown University, Box 1910, Providence, Rhode Island 02912 {rbent@csbrown.edu, pvh@csbrown.edu}

The vehicle routing problem with time windows is a hard combinatorial optimization problem that has received considerable attention in the last decades. This paper proposes a two-stage hybrid algorithm for this transportation problem. The algorithm first minimizes the number of vehicles, using simulated annealing. It then minimizes travel cost by using a large neighborhood search that may relocate a large number of customers. Experimental results demonstrate the effectiveness of the algorithm, which has improved 10 (17%) of the 56 best published solutions to the Solomon benchmarks, while matching or improving the best solutions in 46 problems (82%). More important perhaps, the algorithm is shown to be very robust. With a fixed configuration of its parameters, it returns either the best published solutions (or improvements thereof) or solutions very close in quality on all Solomon benchmarks. Very preliminary results on the extended Solomon benchmarks are also given.

Key words: vehicle routing; large neighborhood search; simulated annealing
History: Received: September 2001; revision received: June 2002; accepted: August 2002.

Introduction

Vehicle routing problems are important components of many distribution and transportation systems, including such examples as bank deliveries, postal deliveries, school bus routing, and security patrol services. They have received considerable attention in the past decades. This paper considers the vehicle routing problem with time windows (VRPTW). Given a number of customers with known demands and a fleet of identical vehicles with known capacities, the problem consists of finding a set of routes originating and terminating at a central depot and servicing all the customers exactly once. The routes cannot violate the capacity constraints on the vehicles and, in addition, must meet the time windows of the customers, which specify the earliest and latest times for the start of service at a customer site. A standard objective of the VRPTW problem consists of minimizing the number of routes or vehicles (primary criterion) and the total travel cost (secondary criterion). Other objective functions have been considered in various papers; for example, optimality results often focus only on the second criterion. The VRPTW problem is NP-complete (Lenstra and Rinnooy Kan 1981), and instances involving 100 customers or more are very hard to solve optimally. Indeed, very few of the Solomon benchmarks (Solomon 1987) involving 100 customers have been solved optimally (see Fisher et al. 1997 and Kohl et al. 1999 for some recent results). As a consequence, local search techniques are often used to find good solutions in reasonable time.

Early work in local search on the VRPTW often utilized simple heuristics or metaheuristics, and an excellent summary can be found in Gendreau et al. (1997). In recent years, the focus of local search has shifted to more complicated metaheuristics to increase the power of the earlier techniques. These include simulated annealing (Chiang and Russell 1996), tabu search (Chiang and Russell 1997, Cordeau et al. 2001, De Backer et al. 2000, Rochat and Taillard 1995, Taillard et al. 1997), genetic/evolutionary algorithms (Gehring and Homberger 2001, Homberger and Gehring 1999, Potvin and Begio 1996), large neighborhood search (Shaw 1998), ant colony search (Gambardella et al. 1999), and variable neighbor search (Rousseau et al. 2002). Of particular interest are Homberger and Gehring (1999) and Shaw (1998), which include some techniques that are used and expanded in this paper. Research on the VRPTW is ongoing, and concurrent work in local search includes Berger et al. (2001), Bräysy (2001a, c), and Czech and Czarnas (2002). Some of this work focuses on finding solutions quickly using sophisticated heuristics, while other studies continue the work on metaheuristics. Concurrent work will be discussed in more detail shortly.

A New Algorithm. This paper presents a two-stage hybrid algorithm for the VRPTW. *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.* 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 travel cost.

This two-step approach makes it possible to design algorithms tailored to each suboptimization. The only prior two-stage algorithm of which we are aware is found in Gehring and Homberger (1999, 2001) and Homberger and Gehring (1999). Our algorithm uses two distinct local search procedures to exploit the specificities of each subproblem. Indeed, the first step of our algorithm uses simulated annealing to minimize the number of routes. *One critical aspect of our simulated annealing algorithm is its 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 minimal delay (Homberger and Gehring 1999) of the routing plan (third criterion).* The second criterion was also successfully used in other applications, e.g., graph coloring (Johnson et al. 1991). The second step of our algorithm uses a large neighborhood search (LNS) (Shaw 1998) to minimize total travel cost. *It is motivated by our belief that LNS is particularly effective in minimizing total travel cost when given a solution that minimizes the number of routes.* Note also that our implementation of LNS makes it very close to variable neighborhood search (Hansen and Mladenovic 1998).

Experimental results demonstrate the effectiveness of the algorithm. On the standard Solomon benchmarks, the algorithm improved the best published solutions in 10 of the 56 problems (17%) and matches or improves the best published results in 46 problems (82%). More important perhaps, the experimental results highlight the robustness of the algorithm. With a standard configuration of its parameters, the algorithm consistently returns either the best published solutions (or improvements thereof) or solutions that are very close in quality.

Recent Results. Since the submission of this paper, several very interesting algorithms have been developed independently and concurrently. These algorithms share some of our main motivations. In particular, they are often organized in several phases and separate the minimization of the number of vehicles and the minimization of travel times. In particular, Bräysy (2001a) proposes a fast algorithm that uses sophisticated insertion and merging heuristics to find an initial solution, an ejection chain heuristic to reduce the number of vehicles, and the Or-opt neighborhood (Or 1976) to minimize travel distance. The resulting algorithm almost always produces the same number of vehicles as ours. Its travel times are, in general, much higher than ours, but the overall algorithm is a very interesting approach to finding high-quality solutions quickly (e.g., under two minutes). Bräysy (2003) proposes a sophisticated four - stage algorithm that extends the previous approach with variable and large neighborhood search. The algorithm focuses on producing high-quality solutions, but its results seem to be weaker than ours on the Solomon benchmarks. Nonetheless, this algorithm is very effective in minimizing the total number of vehicles. Berger et al. (2001) propose an algorithm that runs two different genetic algorithms in parallel. The first algorithm tries to decrease the number of vehicles by minimizing constraint violations, while the second algorithm minimizes travel distances for a given number of vehicles. Of particular interest is the fact that some of the genetic operators use a version of LNS. The algorithm produces the same number of vehicles as our algorithm, on average, and ties three of the new solutions we found. In general, this algorithm and ours, which uses a similar experimental setting (i.e., 30-minute runs), are close in quality. Nonetheless, our algorithm improves six additional Solomon benchmarks (which have not been improved since), and hence it seems superior in quality. Finally, Gehring and Homberger (2001) generalize their earlier papers (Gehring and Homberger 1999, Homberger and Gehring 1999) by combining an evolutionary strategy (to minimize the number of vehicles) with a tabu search (to minimize travel distance). Our average number of vehicles is much better for each class of the Solomon benchmarks.

In summary, several authors seem to have converged to similar ideas in the last 18 months: (1) multistage approaches to separate the minimization of vehicles and the minimization of travel time and (2) the use of large neighborhood search. Our algorithm focuses on producing high-quality and robust solutions under reasonable CPU time constraints. It has produced many new best solutions to the Solomon benchmarks and has matched many of the existing ones (including those produced by algorithms developed independently and concurrently). Many (six) of our new solutions were not matched by these independent algorithms. Because of the high-quality of its solutions, its robustness, and its overall simplicity, we believe that our algorithm is an important addition to the set of tools with which to address the VRPTW.

Structure of the Paper. The rest of this paper is organized as follows. Section 1 describes the problem formulation and specifies the notations used in the paper. Section 2 gives an overview of the overall algorithm. Section 3 presents the simulated annealing algorithm for minimizing routes, while §4 describes

the LNS algorithm for minimizing travel costs. Section 5 presents the experimental results. Section 6 discusses related work, and §7 concludes the paper. The appendix contains the improvements over the best solutions found during the course of this research, as well as preliminary results on the extended Solomon benchmarks.

1. Problem Formulation and Definitions

This section defines the VRPTW 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 zero. The set $\{0, 1, \ldots, N\}$ thus represents all the sites considered in the problem. We also 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). 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

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

Every customer i has a *demand* $q_i \geq 0$ and a *service time* $s_i \geq 0$.

Vehicles. The VRPTW problem 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 $\{v_1, \ldots, v_n\}$. The size of a route, denoted by $|r|$, is the number of customers $|cust(r)|$. The demand of a route, denoted by $q(r)$, is the sum of the demands of its customers; i.e.,

$$q(r) = \sum_{c \in cust(r)} q_c.$$

A route satisfies its capacity constraint if

$$q(r) \leq Q.$$

The travel cost of a route $r = \langle v_1, \ldots, v_n\rangle$, denoted by $t(r)$, is the cost of visiting all of its customers, i.e.,

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

if the route is not empty ($n \geq 1$) and is 0 otherwise.

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

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

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 be serviced. 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

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

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

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

The earliest arrival time of a route $r = \langle v_1, \ldots, v_n\rangle$, denoted by $a(r)$, is given by $\delta_{v_n} + c_{v_n0}$ 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 $\forall r \in \sigma\colon a(r) \leq l_0$. The latest arrival time for customer i that does not violate the time window constraints of i and the customers served after i on its route, denoted by z_i, is defined recursively as

$$\begin{cases} z_0 = l_0 \\ z_i = \min(z_{i^+} - c_{ii^+} - s_i, l_i) \quad (i \in Customers). \end{cases}$$

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

$$\begin{cases} q(r_j) \leq Q & (1 \leq j \leq m) \\ a(r_j) \leq l_0 & (1 \leq j \leq m) \\ a_i \leq l_i & (i \in Customers). \end{cases}$$

The size of a routing plan σ, denoted by $|\sigma|$, is the number of nonempty routes in σ; i.e.,

$$\{r \in \sigma \mid cust(r) \neq \varnothing\}.$$

The VRPTW 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

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

2. Overview of the Algorithm

As mentioned in the introduction, our algorithm is motivated by the recognition that minimizing the objective function

$$\left\langle |\sigma|, \sum_{r \in \sigma} t(r) \right\rangle$$

is not always the most effective way to approach the problem. Indeed, the objective function often drives the search toward 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 because it may require considerable degradation of the travel-cost component of the objective function. The situation is further exacerbated by the discovery of more effective algorithms for minimizing travel cost.

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. The overall algorithm is depicted in Figure 1. The next two sections discuss each suboptimization in detail. Observe also that Gehring and Homberger (1999) and Homberger and Gehring (1999) were the only papers in which a two-stage algorithm was proposed when this research was initiated.

```
Function VRPTWOptimize

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

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

3. Minimizing the Number of Routes

As mentioned, 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 a simulated annealing algorithm (Kirkpatrick et al. 1983) with a number of interesting features that are now reviewed. Our use of simulated annealing was motivated by its success in problems such as graph coloring (Johnson et al. 1991), where the goal is to minimize the size of a set (e.g., the number of colors), and the overall simplicity of its implementation.

3.1. The Neighborhood

The neighborhood of our simulated annealing algorithm is based on the traditional move operators described, for instance, in De Backer et al. (2000) and Kindervater and Savelsbergh (1997): two-exchange, Or-exchange, relocation, crossover, and exchange. We describe these moves informally for completeness. See Kindervater and Savelsbergh (1997) for a comprehensive overview as well as incremental data structures and algorithms to compute them efficiently.

Two-Exchange. For two customers i and j on the same route where i is visited before j, remove arcs (i, i^+), (j, j^+); add arcs (i, j), (i^+, j^+); and reverse the orientation of the arcs between i^+ and j.

Or-Exchange. Remove a sequence of one, two, or three customers from a route, and reinsert the sequence elsewhere on the same or on a different route.

Relocation. For customers i and j, place i after j; i.e., remove arcs (i^-, i), (i, i^+), (j, j^+), and add arcs (i^-, i^+), (j, i), (i, j^+).

Exchange. Exchange the positions of customers i and j; i.e., remove (i^-, i), (i, i^+), (j^-, j), (j, j^+), and add (i^-, j), (j, i^+), (j^-, i), (i, j^+).

Crossover. Exchange the successors of customers i and j; i.e., remove (i, i^+), (j, j^+), and add (i, j^+), (j, i^+).

Given a solution σ, $\mathcal{N}(\sigma)$ denotes the neighborhood of σ, i.e., the set of solutions that can be reached from σ by using one of these move operators. We also denote by *operators* the set of move operators {two-exchange, Or-exchange, relocation, exchange, crossover}.

A Random Subneighborhood. One of the interesting features of our simulated annealing algorithm is how it explores the neighborhood. Indeed, each iteration of the algorithm focuses on a (random) subneighborhood of $\mathcal{N}$ obtained by randomly choosing a move operator o from *operators* and a customer c from *Customers*, and by constructing all the moves using operator o and customer c. The subneighborhood will

be explored exhaustively to find whether it contains a solution improving the best available routing plan and to choose the next move. We denote by $\mathcal{N}(o, c, \sigma)$ the subset of $\mathcal{N}(\sigma)$ that can be reached by using move operator o and customer c.

3.2. The Evaluation Function

The evaluation function is another fundamental aspect of our simulated annealing algorithm. As mentioned earlier, the objective function

$$\left\langle |\sigma|, \sum_{r\in\sigma} t(r) \right\rangle$$

is not always appropriate, because it may lead the search to solutions with a small travel cost and make it impossible to remove routes. To overcome this limitation, our simulated algorithm uses a more complex lexicographic ordering

$$e(\sigma) = \left\langle |\sigma|, -\sum_{r\in\sigma} |r|^2, mdl(\sigma) \right\rangle$$

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 in many algorithms, a typical example being graph coloring (Johnson et al. 1991). The third component minimizes the minimal delay of the routing plan. This concept was introduced by Homberger and Gehring (1999) in the context of evolutionary algorithms. It favors solutions where customers on the smallest route can be relocated on other routes with no constraint violations or with time window violations that are as small as possible. Minimizing minimal delay thus favors solutions where customers can be relocated more easily over solutions where relocation is difficult. More precisely, the minimal delay is defined as follows.

Definition 1 (Minimal Delay). The minimal delay of a solution σ, denoted by $mdl(\sigma)$, is defined as

$$mdl(\sigma) = mdl(r, \sigma) \quad \text{where } |r| = \min_{r'\in\sigma} |r'|.$$

$$mdl(r, \sigma) = \sum_{i\in cust(r)} mdl(i, r, \sigma).$$

$$mdl(i, r, \sigma) = \begin{cases} 0 & \text{if } \mathcal{N}(relocation, i, \sigma) \neq \varnothing \\ \infty & \text{if } \forall r' \in r : r \neq r' : q(r') + q_i > Q. \\ \min_{j\in Customers\setminus cust(r)} mdl(i, j, r, \sigma) & \\ \quad \text{otherwise.} & \end{cases}$$

$$mdl(i, j, r, \sigma) = \max(\delta_j + c_{ji} - l_i, 0) + \max(\delta_i + c_{ij^+} - z_{j^+}, 0).$$

In other words, the minimal delay of a solution σ is the minimal delay of the route with the smallest number of customers. The delay of a route is the summation of the delay of its customers. The minimal delay of a customer i is zero if i can be relocated on another route, ∞ if i cannot be relocated without violating the capacity constraints of the vehicle or the minimal time window violations induced by relocating i after a customer j on another route. The time window violation is given by the summation of the violation of the time window of i and the violation of the time window of the successors of j.

3.3. The Simulated Annealing Algorithm

Figure 2 depicts the simulated annealing algorithm. The algorithm consists of a number of local searches (Lines 3–23), each of which starts from the best solution found so far and from the starting temperature. Each local search performs a number of iterations (Lines 6–21) and decreases the temperature (Line 22). These two steps are repeated until the time limit is exhausted or the temperature has reached its lower bound. Lines 7–20 describe one iteration and are most interesting. Lines 7–9 compute the subneighborhood

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

for a random move operator and a random customer. Lines 10–12 select the solution σ_1 minimizing f in $\mathcal{N}(o, c, \sigma)$ if it improves the best solution found so far. These lines introduce an aspiration criterion (Glover 1989) in the simulated annealing algorithm. Lines 14–19 are the core of the algorithm. Line 14 chooses a random element $\sigma_r \in \mathcal{N}(o, c, \sigma)$, and σ_r is selected as the next routing plan if it does not degrade the current solution (Line 16), or with the traditional probability of simulated annealing otherwise (Line 18). Observe also Line 14

14. $r := \lfloor \text{RANDOM}([0, 1])^\beta \times s \rfloor$;

which biases the search toward "good" moves in $\mathcal{N}(o, c, \sigma)$ when $\beta > 1$.

4. Minimizing the Travel Cost

Our algorithm uses an LNS to minimize travel cost. LNS was proposed in (Shaw 1998) for vehicle routing problems. It was shown to be particularly effective on the Class 1 problems from the Solomon benchmarks, producing several improvements over the then-best published solutions. However, the algorithm performs poorly on the Class 2 benchmarks, where it could not reduce the number of routes satisfactorily (Shaw 1998). (Our own experimental results,

```
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.           o := random(Operators);
8.           c := random(Customers);
9.           ⟨σ_1, ..., σ_s⟩ := N(o,c,σ) where e(σ_i) ≤ e(σ_j) (i < j);
10.          if e(σ_1) < e(σ_b) then {
11.             σ_b := σ_1;
12.             σ := σ_1;
13.          } else {
14.             r := ⌊random([0,1])^β × s⌋;
15.             Δ := e(σ) − e(σ_r);
16.             if Δ ≥ 0 then
17.                σ := σ_r;
18.             else if random([0,1]) ≤ e^{Δ/t} then
19.                σ := σ_r;
20.          }
21.       }
22.       t := α × t;
23.    }
24. }
25. return σ_b;
```

Figure 2 The Simulated Annealing Algorithm to Minimize the Number of Routes

in fact, confirm the findings in Shaw 1998.) 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 follows the heuristics and strategies described in Shaw (1998), although it departs on a number of issues that seem important experimentally.

4.1. The Neighborhood and the Evaluation Function

Given a solution σ, the neighborhood of the LNS algorithm, denoted by $\mathcal{N}_R(\sigma)$, is the set of solutions that can be reached from σ by relocating at most p customers (where p is a parameter of the implementation). Because the LNS algorithm also uses subneighborhoods and explores the neighborhood in 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*$\backslash S$, $\mathcal{N}_I(\sigma, S)$ denotes the solutions that can be obtained by inserting the customers S in σ. Finally, the LNS algorithm uses the original objective function

$$\left\langle |\sigma|, \sum_{r\in\sigma} t(r) \right\rangle$$

as an evaluation function. Observe that the evaluation function still involves the number of routes. This is important because in some cases, minimizing travel costs makes it possible to decrease the number of routes.

4.2. 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 the 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

```
Function TravelCostMinimize(σ_b)

1.  for(l := 1; l ≤ maxSearches; l++)
2.     for(n := 1; n ≤ p; n++)
3.        for(i := 1; i ≤ maxIterations; i++) {
4.           S := SelectCustomers(σ_b, n);
5.           select σ_c ∈ N_R(σ_b, S) such that f(σ_c) = min_{σ∈N_R(σ_b,S)} f(σ);
6.           if f(σ_c) < f(σ_b) then {
7.              σ_b := σ_c;
8.              i := 1;
9.           }
```

Figure 3 The LNS Algorithm to Minimize Travel Cost

customers S of size n; 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 (Hansen and Mladenovic 1998). It remains to describe how to select customers and how to implement Line 5 in the above algorithm.

4.3. Selecting Customers to Relocate

The LNS algorithm uses the same strategy as in Shaw (1998) to select the customers to relocate. The implementation is depicted in Figure 4. It first selects a customer randomly (Line 1) and iterates Lines 3–6 to remove the $n-1$ remaining customers. Each such iteration selects a customer from S (the already selected customers) and ranks the remaining customers according to a relatedness criterion (Lines 3–4). The new customer to insert is selected in Line 5 and, once again, the algorithm biases the selection toward related neighbors. The relatedness measure is defined as in Shaw (1998):

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

where $v_{ij} = 1$ if customers i and j are on different routes, and is 0 otherwise.

4.4. The Exploration Algorithm

Our LNS algorithm uses a branch-and-bound algorithm to explore the selected subneighborhood. 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 whose best insertion degrades the objective function the most (this heuristic is also used in Shaw 1998). The algorithm then explores all the partial solutions obtained by inserting 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

$$\text{Bound}(\sigma, S) \leq \min_{\sigma' \in \mathcal{N}_I(\sigma, S)} f(\sigma').$$

It remains to discuss the lower bound and how to keep the computation times reasonable.

Bounding. The bounding function used in our LNS algorithm is novel and returns the cost of a minimum spanning k-tree (Fisher et al. 1997) on the insertion graph with the depot as distinguished vertex, generalizing the well-known one-tree bound of the traveling salesman problem. The insertion graph vertices are the customers. Given a solution σ over customers $C = \bigcup_{r \in \sigma} cust(r)$ 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, and

(3) all the feasible edges connecting a customer from C and a customer from S.

```
Function SelectCustomers(σ, n)

1.  S := { random(Customers) };
2.  for(i := 2; i ≤ n; i++) {
3.     c := random(S);
4.     ⟨c_0, ..., c_{N−i}⟩ := Customers \ S such that relateness(c, c_i) ≤ relateness(c, c_j) (i ≤ j);
5.     r := := ⌊random([0, 1])^β × |Customers \ S|⌋;
6.     S := S ∪ {c_r};
7.  }
```

Figure 4 Selecting Customers in the LNS Algorithm

```
Function DFSEXPLORE(σ_c, S, σ_b)

1.  if S = ∅ then {
2.     if f(σ_c) < f(σ_b) then σ_b := σ_c;
3.  } else {
4.     c := arg-max_{c∈S} min_{σ∈N_I(σ,{c})} f(σ);
5.     S_c := S \ {c};
6.     ⟨σ_0, ..., σ_k⟩ := N_I(σ, {c}) WHERE f(σ_i) ≤ f(σ_j) (i ≤ j);
7.     for(i := 1; i ≤ k; i++)
8.        if BOUND(σ_i, S_c) < f(σ_b) then
9.           DFSEXPLORE(σ_i, S_c, σ_b);
10. }
```

Figure 5 The Branch-and-Bound Algorithm for the Neighborhood Exploration

More precisely, the insertion graph is defined as follows.

DEFINITION 2 (INSERTION GRAPH). Let σ be a partial solution over customers C and S be the set of customers to insert (*Customers* $= C \cup S$). The insertion graph is the graph G (*Customers*, E), where

$$E = E_\sigma \cup E_S \cup E_c;$$

$$E_\sigma = \{(i, i^+) \mid i \in C\};$$

$$E_S = \{(i, j) \mid i, j \in S\};$$

$$E_c = \big\{(pred(j, \sigma'), j) \mid j \in S \,\&\, pred(j, \sigma') \in C \,\&\, \sigma' \in \mathcal{N}_I(\sigma, \{j\})\big\} \cup \big\{(j, succ(j, \sigma')) \mid j \in S \,\&\, succ(j, \sigma') \in C \,\&\, \sigma' \in \mathcal{N}_I(\sigma, \{j\})\big\}.$$

Incomplete Search. For a large number of customers, finding the best reinsertion may be too time consuming. Our algorithm uses limited discrepancy search to explore only a small part of the search tree. Limited Discrepancy Search (LDS) (Harvey and Ginsberg 1995) is a search strategy relying on a good heuristic for the problem at hand. Its basic idea is to explore the search tree in waves, and each successive wave allows the heuristic to make more mistakes. Wave 0 simply follows the heuristic. Wave 1 explores the solutions that can be reached by assuming that the heuristic made one mistake. More generally, wave i explores the solutions that can be reached by assuming that the heuristic makes i mistakes.

Figure 6 illustrates these waves graphically on a binary tree. The figure describes the successive waves used in exploring the tree. The nodes visited in a given wave are colored black and those visited in a previous waves are colored grey. By exploring the search tree according to the heuristic, LDS may reach good solutions (and thus an optimal solution) much faster than depth-first and best-first search for some applications. Its strength is its ability to explore diverse parts of the search tree containing good solutions that are only reached much later by depth-first search. Our implementation uses one phase of limited discrepancy search that allows up to d discrepancies. Figure 7 depicts the algorithm. Observe that in the LNS algorithm, the tree is not binary and the heuristic selects the insertion points by increasing lower bounds.

5. Experimental Results

This section describes experimental results on our algorithm. The algorithm was implemented in C++ and the entire code is less than 4,500 lines. The core of the algorithm is about 2,000 lines. They include about 350 lines for the simulated annealing algorithm, 300 lines for the LNS algorithm, and about 1,300 lines for the data structures. All results are given on a Sun Ultra 10, 440 MHZ, 256 MB RAM using a Sun C++ compiler. All numbers used were double precision floating points. Our experimental results use the standard Solomon benchmarks available at http://w.cba.neu.edu/~msolomon/problems.htm. See Solomon (1987) for their descriptions.

The rest of this section is organized as follows. Section 5.1 compares our best solutions with the best

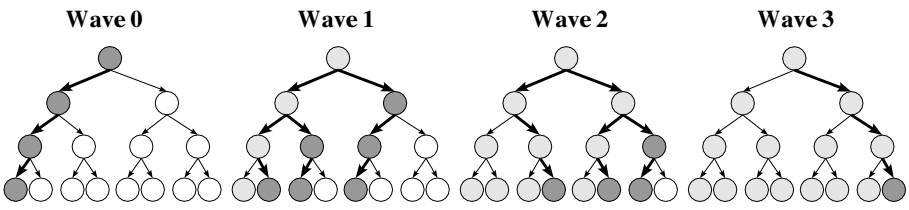

Figure 6 The Successive Waves of LDS

```
Function LDSEXPLORE(σ_c, S, σ_b, d)

1.  if d ≥ 0 then {
2.     if S = ∅ then {
3.        if f(σ_c) < f(σ_b) then σ_b := σ_c;
4.     } else {
5.        c := arg-max_{c∈S} min_{σ∈N_I(σ,{c})} f(σ);
6.        S_c := S \ {c};
7.        ⟨σ_0, ..., σ_k⟩ := N_I(σ, {c}) WHERE f(σ_i) ≤ f(σ_j) (i ≤ j);
8.        for(i := 1; i ≤ k; i++) {
9.           if BOUND(σ_i, S_c) < f(σ_b) then {
10.             LDSEXPLORE(σ_i, S_c, σ_b, d);
11.             d := d + 1;
12.          }
13.       }
14.    }
15. }
```

Figure 7 The Branch-and-Bound Algorithm with a Limited Discrepancy Strategy

published solutions that use the objective function described here. Section 5.2 reports the best results for minimizing routes and compares them with other approaches. Section 5.3 gives the robustness results. The new solutions to the Solomon benchmarks and results on the extended Solomon benchmarks are reported on the authors' website. In reporting the results, we use the following abbreviations to denote existing algorithms: B = (Bräysy 2003), BBB = (Berger et al. 2001), CC = (Czech and Czarnas 2002), CLM = (Cordeau et al. 2001), CR = (Chiang and Russell 1997), DDS = (Desrochers et al. 1992), DFS = (De Backer et al. 2001), GH = (Gehring and Homberger 1999), GTA = (Gambardella et al. 1999), H = (Homberger 2000), HG = (Homberger and Gehring 1999), IKM = (Ibaraki et al. 2001), IKP = (Ioannou et al. 2001), PB = (Potvin and Begio 1996), RGP = (Rousseau et al. 2002), RT = (Rochat and Taillard 1995), S = (Shaw 1998), SSS = (Schrimpf et al. 2000), TBG = (Taillard et al. 1997), TOS = (Thangiah et al. 1994).

5.1. Best Published Results

Tables 1 and 2 report our best results and compare them to the best published results. The column *data* gives the names of the benchmark; the column *best* gives the best published solutions and their sources; the column *SA + LNS* describes the best solution found by our algorithm; and the last two columns report the deviation from the best solutions, both in absolute terms and in percentage. Finally, bold-faced values indicate achievement of best published solutions; italicized/starred results indicate an improvement on the best published results of which we are aware.

The results indicate that our algorithm improved 10 (17%) of the best published solutions to the Solomon benchmarks, while matching or improving the best solutions in 46 benchmarks (82%). The algorithm was able to obtain the minimum number of vehicles published on all instances. In addition, on all benchmarks but two, the algorithm produced solutions that are less than 1% from the best published solutions and, in a couple of cases, it improved the best published solutions by more than 2%. *These results seem to indicate the benefits of decomposing the optimization in two stages, the effectiveness of simulated annealing to minimize routes, and the benefits of LNS to minimize travel cost.*

5.2. Minimizing Routes

Table 3 compares our algorithm to other metaheuristics with respect to the number of routes. It gives the average number of vehicles for the best solution in each class of Solomon's problems. The best results are marked in bold. The results show that our algorithm, together with HG, always produces the best results. *Observe that our algorithm and HG use fundamentally different local search techniques, and yet they both produce the best results. Hence, these results seem to indicate the benefits of using a separate stage to minimize the number of our routes.* Observe also that concurrent and independent work (e.g., Berger et al. 2001; Bräysy 2001a, c) has confirmed the value of separating the minimization of the number of vehicles and the minimization of travel distance by getting high-quality results as well.

5.3. Robustness

Robustness is a fundamental and desirable property of local search algorithms. An algorithm is robust if it performs well on large classes of problems with the same parameter configurations. This section studies the robustness of our algorithm.

Table 1 Solomon Benchmarks: Comparison with Best Published Results, Class 1

Data	Best			SA + LNS		Compare	
c101	828.94	10	RT	**828.937**	**10**	0	0%
c102	828.94	10	RT	**828.937**	**10**	0	0%
c103	828.06	10	RT	**828.065**	**10**	0	0%
c104	824.78	10	RT	**824.777**	**10**	0	0%
c105	828.94	10	PB	**828.937**	**10**	0	0%
c106	828.94	10	RT	**828.937**	**10**	0	0%
c107	828.94	10	RT	**828.937**	**10**	0	0%
c108	828.94	10	RT	**828.937**	**10**	0	0%
c109	828.94	10	PB	**828.937**	**10**	0	0%
r101	1,645.79	19	H	1,650.80	**19**	5.0	0.3%
r102	1,486.12	17	RT	**1,486.12**	**17**	0	0%
r103	1,292.68	13	S	**1,292.68**	**13**	0	0%
r104	1,007.31	9	S	**1,007.31**	**9**	0	0%
r105	1,377.11	14	RT	**1,377.11**	**14**	0	0%
r106	1,252.03	12	RT	**1,252.03**	**12**	0	0%
r107	1,104.66	10	S	**1,104.66**	**10**	0	0%
r108	963.99	9	S	*960.876*[1]	*9**	−3.1	−0.3%
r109	1,194.73	11	HG	**1,194.73**	**11**	0	0%
r110	1,118.84	10	H	**1,118.84**	**10**	0	0%
r111	1,096.72	10	RGP	1,096.73	**10**	0.01	0%
r112	982.14	9	GTA	991.245	**9**	9.1	0.9%
rc101	1,696.94	14	TBG	1,696.95	**14**	0.01	0%
rc102	1,554.75	12	TBG	**1,554.75**	**12**	0	0%
rc103	1,261.67	11	S	**1,261.67**	**11**	0	0%
rc104	1,135.48	10	S	**1,135.48**	**10**	0	0%
rc105	1,633.72	13	RGP	*1,629.44*[1]	*13**	−4.3	−0.3%
rc106	1,427.13	11	CLM	*1,424.73*[1]	*11**	−2.4	−0.2%
rc107	1,230.48	11	S	**1,230.48**	**11**	0	0%
rc108	1,139.82	10	TBG	**1,139.82**	**10**	0	0%

[1] BBB achieved this result in concurrent unpublished work.

Tables 4 and 5 depict the results for a specific configuration of our algorithm. The results correspond to five runs of our algorithm. For simulated annealing, the parameters are 2,000 for starting temperature, 0.95 for cooling factor α, 2,500 iterations per each temperature, 0.01 minimum temperature, and 10 for the simulated annealing determinism factor β. For LNS, the parameters are 35 for the maximum customers to remove p, 1,000 iterations without improvement before removing one more customer, 15 for the determinism factor β, and 4 discrepancies. The allowed time is split one-third for SA and two-thirds for LNS. Bold-faced numbers indicate matches with the best published results. Italicized numbers indicate results better than the best published solutions. Italicized and starred numbers indicate results better than the best published solutions and equal to the best results we found. Where different numbers of vehicles were discovered, the number of times each vehicle result is obtained is indicated next to the average results. There are a number of interesting observations to be drawn from these results.

Best Results. The algorithm finds the best published result (or an improvement thereof) *in all 5 runs* in 14 problems (25%) after 30 minutes and in 16 problems (29%) after 120 minutes. Furthermore, the best published result (or an improvement thereof) is achieved at least once in 23 problems (41%) after 30 minutes and in 30 problems after 120 minutes (54%). In the 30-minute runs, the algorithm improves the best published results in two cases with the standard configuration and it is almost always within 5% of the best published solutions. In the 120-minute runs, the algorithm improves the best published results in four cases with the standard configuration and is always within 3.5% of the best published solutions except in one case (5.6%). In general, giving more time to the algorithm helps produce better solutions, although this is not always true (because simulated annealing gives extremely random starting solutions).

Average Results. The average results are harder to compare systematically because all five runs do not always produce the best number of routes. However, it can be seen that they are never very far from the best solutions. For the 30-minute runs, they are, in general, within 2% of the best solutions on Class 1 and within 6% on Class 2. For the 120-minute runs, they are always within 2% and almost always within 1%

Table 2 Solomon Benchmarks: Comparison with Best Published Results, Class 2

Data	Best			SA + LNS		Compare	
c201	591.56	3	PB	**591.557**	**3**	0	0%
c202	591.56	3	PB	**591.557**	**3**	0	0%
c203	591.17	3	RT	**591.173**	**3**	0	0%
c204	590.60	3	PB	**590.599**	**3**	0	0%
c205	588.88	3	PB	**588.876**	**3**	0	0%
c206	588.49	3	PB	**588.493**	**3**	0	0%
c207	588.29	3	RT	**588.286**	**3**	0	0%
c208	588.32	3	RT	**588.324**	**3**	0	0%
r201	1,252.37	4	HG	**1,252.37**	**4**	0	0%
r202	1,191.70	3	RGP	1,195.30	**3**	3.6	0.3%
r203	942.64	3	HG	*941.408*	*3**	−1.2	−0.1%
r204	848.91	2	SSS	*825.519*	*2**	−23.4	−2.8%
r205	994.42	3	RGP	**994.42**	**3**	0	0%
r206	906.14	3	SSS	914.627	**3**	8.5	0.9%
r207	914.39	2	CR	*893.328*	*2**	−21.1	−2.3%
r208	726.823	2	GTA	**726.823**	**2**	0	0%
r209	909.16	3	H	**909.163**	**3**	0	0%
r210	939.37	3	DFS	951.294	**3**	11.9	1.3%
r211	904.32	2	SSS	*892.713*	*2**	−11.6	−1.3%
rc201	1,406.94	4	CLM	1,412.45	**4**	5.5	0.4%
rc202	1,377.089[1]	3	GTA	1,387.38	**3**	10.3	0.7%
rc203	1,051.82[2]	3	SSS	1,064.14	**3**	12.3	1.2%
rc204	798.464	3	GTA	**798.464**	**3**	0	0%
rc205	1,302.42	4	HG	*1,297.65*	*4**	−4.8	−0.4%
rc206	1,146.32	3	H	**1,146.32**	**3**	0	0%
rc207	1,062.05	3	CLM	*1,061.14*	*3**	−0.9	−0.1%
rc208	829.69	3	RGP	*828.141*[3]	*3**	−1.5	−0.2%

[1] CC achieved a result of 1,367.09 subsequent to this work.
[2] CC achieved a result of 1,049.62 subsequent to this work.
[3] IKM achieved this result in concurrent unpublished work.

on Class 1 and almost always within 5% on Class 2. It is also interesting to compare the average results in 120 minutes and the best results in 30 minutes. These results are, in fact, quite similar in quality, which is a good indication of the robustness of the algorithm.

Route Minimization. Table 6 reports the average number of vehicles required over five runs and compares these results with other approaches where the papers provided averages across independent runs of their programs. The results are clustered by problem classes. The best results are in bold. CPU time is given in the column headers or underneath the results. Prior results are given in the first five columns. Concurrent and independent results are given in the next two columns. Note that comparing times is misleading, as some prior or concurrent results were achieved on less powerful machines. The final column gives the best-possible value for each class, i.e., the average number of vehicles for the class if the best published number of vehicles is achieved for each benchmark.

Note that after 30 minutes our algorithm beats the average number of vehicles of any prior results using this metric. On the nontrivial r2 class, our algorithm achieves the best-possible value inferred from the

Table 3 Solomon Benchmarks: Route Reduction Comparison

Data	RT	TBG	CR	CLM	HG	DFS	GTA	S	GH	SA + LNS[1]
c1	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
c2	**3**	**3**	**3**	**3**	**3**	**3**	**3**	—	**3**	**3**
r1	12.25	12.17	12.17	12.08	**11.92**	12.5	12	12	12.41	**11.92**
r2	2.91	2.82	**2.73**	**2.73**	**2.73**	3	**2.73**	—	2.82	**2.73**
rc1	11.88	**11.5**	11.88	**11.5**	**11.5**	12	11.63	11.75	11.88	**11.5**
rc2	3.38	3.38	**3.25**	**3.25**	**3.25**	3.38	**3.25**	—	**3.25**	**3.25**

[1] Concurrent work, Berger et al. (2001) and Bräysy (2003), were able achieve the same results. Bräysy (2003) was able to achieve the same results on very short runs, except for r1, which was 12.0. However, the average travel distance of our implementation, when compared against the implementations with the same number of vehicles, is better.

Table 4 Solomon Benchmarks Class 1: Robustness Results

		30 CPU minutes					120 CPU minutes				
Data	Veh	Best	Cmp (%)	Avg	Cmp (%)	Worst	Best	Cmp (%)	Avg	Cmp (%)	Worst
c101	**10**	**828.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
c102	**10**	**828.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
c103	**10**	**828.065**	0.0	**828.065**	0.0	**828.065**	**828.065**	0.0	**828.065**	0.0	**828.065**
c104	**10**	**824.777**	0.0	**824.777**	0.0	**824.777**	**824.777**	0.0	**824.777**	0.0	**824.777**
c105	**10**	**828.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
c106	**10**	**828.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
c107	**10**	**822.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
c108	**10**	**828.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
c109	**10**	**828.937**	0.0	**828.937**	0.0	**828.937**	**828.937**	0.0	**828.937**	0.0	**828.937**
r101	**19**	1,650.8	0.3	1,650.8	0.3	1,650.8	1,650.8	0.3	1,650.8	0.3	1,650.8
r102	**17**	**1,486.12**	0.0	**1,486.12**	0.0	**1,486.12**	**1,486.12**	0.0	**1,486.12**	0.0	**1,486.12**
r103	**13**						**1,292.68**	0.0	1,296.17	0.3	1,297.62
	14	1,213.62		1,214.48		1,217.92					
r104	**9**						1,017.52	1.0	1,017.52 1	1.0	
	10	981.232		984.13		989.803			987.38 4		989.056
r105	**14**	1,387.14	0.7	1,401.83	1.8	1,426.17	**1,377.11**	0.0	1,380.85	0.3	1,387.14
r106	**12**	1,257.96	0.4	1,270.19	1.5	1,292.16	1,257.96	0.5	1,258.31	0.5	1,259.71
r107	**10**	1,114.78	0.9	1,119.28 4	1.3		**1,104.66**	0.0	1,111.39	0.6	1,114.29
	11			1,072.12 1		1,072.12					
r108	**9**	966.86	0.3	979.75 4	1.6		966.118	0.2	968.04	0.4	973.424
	10			961.359 1		961.359					
r109	**11**	1,197.42	0.2	1,219.9 4	2.1		1,197.42	0.2	1,218.54 4	2.0	
	12			1,166.24 1		1,166.24			1,153.89 1		1,153.89
r110	**10**	1,126.63	0.7	1,130.76 4	1.1		1,119.14	0.0	1,125.66	0.6	1,127.94
	11			1,114.28 1		1,114.28					
r111	**10**	1,096.74	0.0	1,107.78 4	1.0		1,096.73	0.0	1,097.49	0.0	1,100.55
	11			1,063.3 1		1,063.3					
r112	**9**						992.754	1.1	1,001.54 3		
	10	966.793		971.79		986.753			967.95 2		968.94
rc101	**14**	1,697.43	0.1	1,697.43 1	0.1		1,296.95	0.0	1,697.21 4	0.0	
	15			1,624.51 4		1,627.29			1,623.58 1		1,623.58
rc102	**12**	**1,554.75**	0.0	**1,554.75**	0.0	**1,554.75**	**1,554.75**	0.0	**1,554.75** 4	0.0	
	13								1,477.54 1		1,477.54
rc103	**11**	**1,261.67**	0.0	1,267.47	0.5	1,278.55	**1,261.67**	0.0	1,267.17	0.4	1,270.72
rc104	**10**	**1,135.48**	0.0	1,144.97	0.8	1,156.05	**1,135.48**	0.0	1,141.15	0.5	1,159.43
rc105	**13**	1,635.9	0.1	1,638.24 2	0.3		*1,629.44**	−0.3	1,636.86 3	0.2	
	14			1,553.03 3		1,563.76			1,541.23 2		1,542.27
rc106	**11**						*1,424.73**	−0.2	1,435.82 4	0.6	
	12	1,376.26		1,378.52		1,387.57			1,376.25 1		1,376.25
rc107	**11**	1,230.95	0.0	1,231.85	0.1	1,232.26	1,230.95	0.0	1,231.84	0.1	1,232.26
rc108	**10**	**1,139.82**	0.0	1,162.00	1.9	1,193.45	**1,139.82**	0.0	1,156.04	1.4	1,187.76

published results. Once again, the results indicate the robustness of our algorithm.

Finally, observe that the algorithms developed independently achieve excellent robustness results as well. In general, algorithm BBB produces slightly better results than ours. Algorithm B is excellent for route minimization. As mentioned, its main weakness is in minimizing travel distance, but it is an excellent candidate for quickly producing high-quality solutions.

Summary. Overall, the algorithm appears to be very robust, performing well on all instances of the benchmarks. The algorithm is robust, both with respect to route minimization and travel cost minimization, on these benchmarks. This is one of the strengths of the algorithm, together with its ability to produce excellent solutions on all benchmarks.

6. Discussion and Related Work

Prior Work. This paper presented a two-stage hybrid local search algorithm for the vehicle routing problem with time windows. When this research was started, Gehring and Homberger (1999) and Homberger and Gehring (1999) were the only other papers presenting a two-stage algorithm for vehicle routing. The algorithm in Homberger and Gehring (1999) is not hybrid, however, and uses the same evolutionary metaheuristic with two evaluation functions. Their evolutionary metaheuristic uses the uniform order-based crossover of Davis (1991), and their mutation operators are Or-opt from Or (1976) (generalized so that sequences of customers can be moved to other vehicles), λ-interchange from Osman (1993), and

Table 5 Solomon Benchmarks Class 2: Robustness Results

		30 CPU minutes					120 CPU minutes				
Data	Veh	Best	Cmp (%)	Avg	Cmp (%)	Worst	Best	Cmp (%)	Avg	Cmp (%)	Worst
c201	**3**	**591.557**	0.0	**591.557**	0.0	**591.557**	**591.557**	0.0	**591.557**	0.0	**591.557**
c202	**3**	**591.557**	0.0	614.04	3.8	703.993	**591.557**	0.0	**591.557**	0.0	**591.557**
c203	**3**	**591.173**	0.0	656.844	11.1	753.137	**591.173**	0.0	607.11	2.7	670.834
c204	**3**	**590.599**	0.0	619.72	4.9	672.158	**590.599**	0.0	**590.599**	0.0	**590.599**
c205	**3**	**588.876**	0.0	**588.876**	0.0	**588.876**	**588.876**	0.0	**588.876**	0.0	**588.876**
c206	**3**	**588.493**	0.0	607.99	3.2	685.964	**588.493**	0.0	**588.493**	0.0	**588.493**
c207	**3**	**588.286**	0.0	607.78	3.3	685.758	**588.286**	0.0	**588.286**	0.0	**588.286**
c208	**3**	**588.324**	0.0	**588.324**	0.0	**588.324**	**588.324**	0.0	**588.324**	0.0	**588.324**
r201	**4**	1,287.67	2.8	1,300.26	3.8	1,317.98	1,254.72	0.2	1,271.48	1.5	1,284.68
r202	**3**	1,237.04	3.8	1,261.89 2	5.9		1,199.17	0.6	1,228.12	3.1	1,245.4
	4			1,135.3 3		1,166.0					
r203	**3**	967.822	2.7	985.32	4.5	1,026.83	963.66	2.2	972.94	3.2	995.084
r204	**2**	*833.883*	−1.7	860.03 3	1.3		*838.06*	−1.3	857.11	1.0	871.655
	3			793.73 2		798.701					
r205	**3**	1,036.83	4.3	1,050.06	5.6	1,061.8	1,008.55	1.4	1,041.31	4.7	1,070.27
r206	**3**	956.289	5.5	981.85	8.4	1,018.26	927.724	2.4	955.70	5.5	977.019
r207	**2**	*901.091*	−1.5	923.73 4	1.0		*893.328**	−2.3	*908.51*	−0.6	920.876
	3			866.577 1		866.577					
r208	**2**	737.369	1.5	758.773	4.4	773.315	**726.823**	0.0	749.17	3.1	773.681
r209	**3**	943.709	3.8	955.90	5.1	980.098	941.318	3.5	955.30	5.1	970.167
r210	**3**	967.996	3.0	982.66	4.6	1,006.61	968.661	3.1	975.75	3.9	979.958
r211	**2**	913.752	1.0	934.30 4	3.3		908.062	0.0	923.53	2.1	943.14
	3			809.538 1		809.538					
rc201	**4**	1,466.02	4.2	1,481.45	5.1	1,519.08	1,426	1.4	1,438.44	2.2	1,459.07
rc202	**3**	1,387.38	0.7	1,424.73 2	3.5		1,387.38	0.7	1,411.00 4	2.5	
	4			1,238.38 3		1,301.23			1,162.8 1		1,162.8
rc203	**3**	1,097.31	4.3	1,109.04	5.4	1,125.8	1,068.08	1.5	1,078.96	2.6	1,099.70
rc204	**3**	841.282	5.4	850.46	6.4	865.928	818.208	2.5	833.82	4.4	851.993
rc205	**4**	1,322.64	1.6	1,353.91	4.0	1,395.88	1,312.9	0.8	1,325.77	1.8	1,347.59
rc206	**3**	1,187.28	3.6	1,217.93	6.2	1,239.49	1,170.52	2.1	1,215.85	6.1	1,242.71
rc207	**3**	1,093.75	2.9	1,111.6	4.7	1,130.36	1,070.85	0.8	1,096.06	3.2	1,115.05
rc208	**3**	875.605	5.5	900.61	8.5	914.755	875.977	5.6	900.85	8.6	942.997

2-opt* from Potvin and Rousseau (1995). Their evaluation function to minimize routes is a lexicographic function with three components. Their second component is the size of the smallest route, while ours is the sum of the squares of the route sizes. Their third component is the minimal delay of the routing plan. Interestingly, Homberger and Gehring's paper (1999) is hybrid and combines an evolutionary strategy (to minimize the number of vehicles) with a tabu search (to minimize travel distance). As discussed earlier, our average number of vehicles is much better for each class of the Solomon benchmarks. Their motiva-

Table 6 Solomon Benchmarks: Robustness of the Route Minimization

			GTA		BBB		SA + LNS				Best
Data	RT	TBG	1,800	S 3,600	1,800	B	1,800	3,600	5,400	7,200	possible
c1	**10**	**10**	**10**	—	**10**	**10**	**10**	**10**	**10**	**10**	10
	540	2,929				—					
c2	**3**	**3**	**3**	—	**3**	**3**	**3**	**3**	**3**	**3**	3
	1,200	3,275				—					
r1	12.58	12.33	12.38	12.33	12.17	**11.92**	12.25	12.2	12.07	12.03	11.92
	1,300	13,774				17,286					
r2	3.09	3.00	3.00	—	**2.73**	**2.73**	2.85	2.76	2.75	**2.73**	2.73
	4,900	3,372				15,558					
rc1	12.38	11.9	11.92	11.95	11.88	11.50	11.8	11.7	11.65	11.63	11.5
	2,600	11,264				12,876					
rc2	3.62	3.38	3.33	—	**3.25**	**3.25**	3.3	3.35	3.33	3.28	3.25
	1,300	1,933				7,788					

tion for using two-stage algorithms is similar to ours: The recognition that minimizing travel costs may not always be most effective for minimizing the number of routes. *One of the contributions of this paper is to provide evidence of the benefits of a two-stage approach for vehicle routing with time windows. Indeed, the fundamentally different nature of these two two-stage algorithms, together with their effectiveness, seem to indicate that the two-stage approach has benefits across metaheuristics*.

The first stage of our algorithm uses a novel simulated annealing algorithm. The algorithm uses traditional moves operators described in De Backer et al. (2000) and Kindervater and Savelsbergh (1997): 2-exchange, Or-exchange, relocation, crossover, and exchange. A critical aspect of the simulated annealing is the lexicographic evaluation function. Its second component, maximizing the sum of the squares of route sizes, was inspired by some graph-coloring algorithms (Johnson et al. 1991). Its third component is the minimal delay of Homberger and Gehring (1999). Our simulated annealing algorithm also includes some greedy components typical of tabu search (Glover 1989), including an aspiration criterion and a bias toward good solutions in the random process. These greedy aspects were shown to be beneficial experimentally. Of course, simulated annealing was used for solving vehicle routing problems in the past. In particular, Chiang and Russell (1996) describe a simulated annealing where the neighborhood is defined by the λ-interchange mechanism of Osman (1993) and the k-node interchange mechanism of Christofides and Beasley (1984). The algorithm in Chiang and Russell (1996) makes use of a tabu list within the simulated annealing process and uses a weighted objective function incorporating total time along with number of vehicles and travel cost. *Probably the main contribution here is the novel evaluation function and the additional evidence that minimal delay is a fundamental concept in minimizing the number routes.*

The second stage of our algorithm uses the LNS technique pioneered by Shaw (1998). In that paper, LNS was shown to be very effective on Class 1 of the Solomon benchmarks. No results were given on Class 2, as LNS could not reduce the number of routes satisfactorily (Shaw 1998, p. 426) because the Class 2 benchmarks have a high number of customers per route. This fact was also confirmed by our own experimental results. Our implementation adds a restarting strategy, making our algorithm essentially similar to a variable neighborhood search (Hansen and Mladenovic 1998). It also adds a more precise lower bound based on minimal spanning k-trees. Both of these components were shown to have benefits experimentally, especially as far as robustness is concerned. Of course, there is clearly much room left for improvements in implementations of LNS. *Probably the main contribution here is to show that LNS is particularly effective for minimizing travel cost across all Solomon benchmarks when given routing plans minimizing the number of routes.*

There are, of course, many other algorithms for vehicle routing. See, for instance, Gendreau et al. (1997) for a good overview of techniques for solving vehicle routing problems using local search, Chiang and Russell (1997) and Rochat and Taillard (1995) for tabu search algorithms, Gambardella et al. (1999) for an ant colony metaheuristic, De Backer et al. (2000) for guided local search on top of tabu search, and Taillard et al. (1997) for the problem with soft time windows.

Concurrent and Independent Work. As mentioned in the introduction, many researchers seem to have converged to related approaches in the last 18 months. Most of the new algorithms are hybrid and use several phases, separate the minimization of vehicles and the minimization of travel times, and include some notion of large neighborhood search. In particular, Bräysy (2001a) proposes a fast algorithm that uses sophisticated insertion and merging heuristics to find an initial solution, an ejection chain heuristic to reduce the number of vehicles, and the Or-opt neighborhood (Or 1976) to minimize travel distance. The resulting algorithm almost always produces the same number of vehicles as ours, but its travel times are higher than ours. This algorithm seems to be a very interesting approach to finding high-quality solutions quickly (e.g., under two minutes). Bräysy (2003) proposes a sophisticated four-stage algorithm that extends the previous approach with variable and large neighborhood search. The algorithm focuses on producing high-quality solutions, but its results seem to be weaker than ours on the Solomon benchmarks. Nonetheless, this algorithm is very effective in minimizing the total number of vehicles. Berger et al. (2001) propose an algorithm that runs two different genetic algorithms in parallel. The first algorithm tries to decrease the number of vehicles by minimizing constraint violations, while the second algorithm minimizes travel distances for a given number of vehicles. Of particular interest is the fact that some genetic operators use a version of LNS. The algorithm produces the same number of vehicles as our algorithm on average and ties three of the new solutions we found. In general, this algorithm and ours, which uses a similar experimental setting (i.e., 30-minute runs), are very close in quality. Nonetheless, our algorithm improves six additional Solomon benchmarks and hence seems superior in quality.

These results, in fact, provide further evidence of the main theses of this paper: the benefits of optimizing the number of vehicles and travel time independently, the value of hybrid approach, and the potential of large neighborhood search.

It seems fair to conclude that at this point there is no single algorithm for the VRPTW that is appropriate for all purposes. Our algorithm focuses on producing very high-quality solutions under reasonable time constraints. It was instrumental in improving 10 Solomon benchmarks (only 4 of which were tied by independent and concurrent work) and is extremely robust. These computational results, together with its overall conceptual simplicity, make it a significant contribution to the set of tools for approaching the VRPTW.

7. Conclusion

This paper proposed a two-stage hybrid algorithm for multiple vehicle routing with capacity and time window constraints. The algorithm first minimizes the number of vehicles using a simulated annealing algorithm. It then minimizes travel cost by using a large neighborhood search that possibly relocates a large number of customers. Experimental results demonstrate the effectiveness of the algorithm, which has improved 10 (27%) of the 56 best published solutions to the Solomon benchmarks, while matching or improving the best solutions in 46 benchmarks (82%). More important perhaps, the algorithm, with a fixed configuration of its parameters, is shown to be very robust, returning either the best published solutions (or improvements thereof) or solutions very close in quality on all Solomon benchmarks. It is also conceptually simple to understand and to implement. These results seem to indicate the benefits of using a two-stage approach, of using simulated annealing to minimize the number of routes, and of using LNS for minimizing travel costs.

It is also important to mention that concurrent and independent work (e.g., Berger et al. 2001; Bräysy 2001a, b; 2003) have confirmed the main theses of this paper: the benefits of optimizing the number of vehicles and travel time independently, the value of hybrid approach, and the potential of large neighborhood search. This new generation of algorithms has, we believe, significantly enhanced our understanding of the local search approaches to the VRPTW.

Acknowledgments

The first author is supported by a National Defense Science and Engineering Graduate (NDSEG) fellowship from the American Society of Engineering Education (ASEE). The second author is partly supported by National Science Foundation ITR Award DMI-0121495.

References

Berger, J., M. Barkaoui, O. Bräysy. 2001. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. Working paper, Defense Research Establishment Valcartier, Val Belair, Canada.

Bräysy, O. 2001a. Five local search algorithms for the vehicle routing problem with time windows. Working paper, SINTEF Applied Mathematics, Department of Optimization, Norway.

Bräysy, O. 2001b. Local search and variable neighborhood search algorithms for the VRPTW. Acta Wasaensia 87, Mathematics 8, Operational Research, Universitas Wasaensis, Vaasa, Oslo, Finland.

Bräysy, O. 2003. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS J. Comput.* **15**(4) 347–368.

Chiang, W., R. Russell. 1996. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Ann. Oper. Res.* **63** 3–27.

Chiang, W., R. Russell. 1997. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS J. Comput.* **9** 417–430.

Christofides, N., J. Beasley. 1984. The period routing problem. *Networks* **14** 237–246.

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

Czech, Z., P. Czarnas. 2002. A parallel simulated annealing for the vehicle routing problem with time windows. *Proc. 10th Euromicro Workshop Parallel, Distributed Network-based Processing*, Canary Islands, Spain, 376–383.

Davis, L. 1991. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

De Backer, B., V. Furnon, P. Shaw, P. Kilby, P. Prosser. 2000. Solving vehicle routing problems using constraint programming and metaheuristics. *J. Heuristics* **6** 501–523.

Desrochers, M., J. Desrosiers, M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* **40** 342–354.

Fisher, M., K. Joernsten, O. Madsen. 1997. Vehicle routing with time windows: Two optimization algorithms. *Oper. Res.* **45**(3) 488–492.

Gambardella, L., E. Taillard, G. Agazzi. 1999. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. D. Corne, M. Dorigo, F. Glover, eds. *New Ideas in Optimization*. McGraw-Hill, London, U.K., 63–76.

Gehring, H., J. Homberger. 1999. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. *Proc. EUROGEN99—Short Course on Evolutionary Algorithms Engrg. Comput. Sci.*, Reports of the Department of Mathematical Information Technology Series A, Collections, A2/1999, University of Jyvaskyla, Jyväskylä, Finland, 57–64.

Gehring, H., J. Homberger. 2001. A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific J. Oper. Res.* **18** 35–47.

Gendreau, M., G. Laporte, J. Potvin. 1997. Vehicle routing: Modern heuristics. E. Aarts, J. Lenstra, eds. *Local Search in Combinatorial Optimization*, Ch. 9. John Wiley & Sons Ltd., New York, 311–336.

Glover, F. 1989. Tabu search. *ORSA J. Comput.* **1** 190–206.

Hansen, P., N. Mladenovic. 1998. An introduction to variable neighborhood search. S. Voss, S. Martello, I. H. Osman, C. Roucairol, eds. *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, New York, 433–458.

Harvey, W., M. Ginsberg. 1995. Limited discrepancy search. *Proc. 14th Internat. Joint Conf. Artificial Intelligence*, Montreal, Canada.

Homberger, J., H. Gehring. 1999. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR* **37** 297–318.

Homberger, W. 2000. *Verteilt-Parallele Metaheuristiken zur Tourenplanung*. Gaber, Wiesbaden, Germany.

Ibaraki, T., M. Kubo, T. Masuda, T. Uno, M. Yagiura. 2001. Effective local search algorithms for the vehicle routing problem with general time windows. *Proc. 4th Metaheuristics Internat. Conf.*, Porto, Portugal.

Ioannou, G., M. Kritikos, G. Prastacos. 2001. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *J. Oper. Res. Soc.* **52** 523–537.

Johnson, D., C. Aragon, L. McGeoch, C. Schevon. 1991. Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning. *Oper. Res.* **39**(3) 378–406.

Kindervater, G., M. Savelsbergh. 1997. Vehicle routing: Handling edge exchanges. E. Aarts, J. Lenstra, eds. *Local Search in Combinatorial Optimization*, Ch. 10. John Wiley & Sons Ltd., New York, 337–360.

Kirkpatrick, S., C. Gelatt, M. Vecchi. 1983. Optimization by simulated annealing. *Science* **220** 671–680.

Kohl, N., J. Desrosiers, O. Madsen, M. Solomon, F. Soumis. 1999. 2-path cuts for the vehicle routing problem with time windows. *Transportation Sci.* **33** 101–116.

Lenstra, J., A. H. G. Rinnooy Kan. 1981. Complexity of vehicle routing and scheduling problems. *Networks* **11** 221–227.

Or, I. 1976. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. Ph.D. thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL.

Osman, I. 1993. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res.* **40**(1) 421–452.

Potvin, J., S. Begio. 1996. The vehicle routing problem with time windows—Part II: Genetic search. *INFORMS J. Comput.* **8** 165–172.

Potvin, J., J. Rousseau. 1995. An exchange heuristic for routing problems with time windows. *J. Oper. Res. Soc.* **46** 1433–1446.

Rochat, Y., E. Taillard. 1995. Probabilistic diversification and intensification in local search for vehicle routing. *J. Heuristics* **1** 147–167.

Rousseau, L., M. Gendreau, G. Pesant. 2002. Using constraint-based operators to solve the vehicle routing problem with time windows. *J. Heuristics* **8** 43–58.

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.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. *Proc. Principles Practice Constraint Programming*, Pisa, Italy, 417–431.

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

Taillard, E., P. Badeau, M. Gendreau, F. Geurtin, J. Potvin. 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Sci.* **31** 170–186.

Thangiah, S., I. Osman, T. Sun. 1994. Hybrid genetic algorithms, simulated annealing and tabu search methods for vehicle routing problems with time windows. Technical Report UKC/OR94/4, Institute of Mathematics & Statistics, University of Kent, Canterbury, U.K.