Heuristic Approaches for the Fleet Size and Mix Vehicle Routing Problem with Time Windows

Article *in* Transportation Science · November 2007

DOI: 10.1287/trsc.1070.0190 · Source: DBLP

4 authors, including:

Mauro Dell'Amico
Università degli Studi di Modena e Reggio Emilia
115 PUBLICATIONS **4,456** CITATIONS

SEE PROFILE

Daniele Vigo
University of Bologna
178 PUBLICATIONS **13,740** CITATIONS

SEE PROFILE

TRANSPORTATION SCIENCE
Vol. 41, No. 4, November 2007, pp. 516–526
ISSN 0041-1655 | EISSN 1526-5447 | 07 | 4104 | 0516

informs

DOI 10.1287/trsc.1070.0190

Heuristic Approaches for the Fleet Size and Mix Vehicle Routing Problem with Time Windows

Mauro Dell'Amico
DISMI, Università di Modena e Reggio Emilia, Via Amendola 2, 42100, Reggio Emilia, Italy, dellamico.mauro@unimore.it

Michele Monaci
DEI, Università di Padova, Via Gradenigo 6/A, 35131, Padova, Italy, monaci@dei.unipd.it

Corrado Pagani
DISMI, Università di Modena e Reggio Emilia, Via Amendola 2, 42100, Reggio Emilia, Italy, pagani.corrado@unimore.it

Daniele Vigo
DEIS, Università di Bologna, Viale Risorgimento 2, 40136, Bologna, Italy, daniele.vigo@unibo.it

The fleet size and mix vehicle routing problem with time windows (FSMVRPTW) is the problem of determining, at the same time, the composition and the routing of a fleet of heterogeneous vehicles aimed to serve a given set of customers. The routing problem requires us to design a set of minimum-cost routes originating and terminating at a central depot and serving customers with known demands, within given time windows. This paper develops a constructive insertion heuristic and a metaheuristic algorithm for FSMVRPTW. Extensive computational experiments on benchmark instances show that the proposed method is robust and efficient, and outperforms the previously published results.

Key words: vehicle routing problem; time windows; heterogeneous fleet; metaheuristics
History: Received: February 2006; revision received: December 2006; accepted: January 2007.

1. Introduction

The fleet size and mix vehicle routing problem (FSMVRP) is a variant of the well-known vehicle routing problem (VRP) and consists of determining, at the same time, the composition and the routing of a fleet of heterogeneous vehicles in order to serve a given set of customers with known delivery demands. Because the temporal aspects of vehicle routing problems become increasingly important in real-world applications, in this paper the FSMVRP is extended by imposing time-windows constraints on the instant in which the customers have to be served; the resulting problem will be denoted as FSMVRPTW.

The fleet must be designed by appropriately selecting vehicles of different types. Each type of vehicles is defined by a loading capacity and a fixed cost for its use. An infinite number of vehicles of each type is available. The objective of FSMVRPTW is to minimize the total cost, which is composed of vehicle fixed costs and variable routing costs.

The FSMVRPTW is NP-hard because it reduces to the classical VRP when only one type of vehicles exists and all the time windows start at zero and terminate at a very large time instant. Both VRP and VRP with time windows (VRPTW) have generated a considerable amount of research in the last decades. Because of the difficulties in solving instances of practical interest, a great research effort on VRP and its variants has focused on heuristic algorithms. For recent surveys on VRP problems, the reader is referred to Toth and Vigo (2002).

In the last 20 years several algorithms have been designed for FSMVRP. In Renaud and Boctor (2002), a review of the papers published on this problem until 1999 is given. As to FSMVRPTW, in spite of its practical importance, many fewer results appeared in the literature. Liu and Shen (1999) designed the first heuristic for FSMVRPTW: It starts by determining a first feasible solution with a constructive method inspired by the savings algorithm by Clarke and Wright (1964) and by its adaptations for the FSMVRP proposed by Golden et al. (1984). Then, a second phase improves this solution by moving one customer either to another position in its route or to a different route. Liu and Shen tested their algorithm on a set of 168 benchmark instances obtained by modifying those proposed by Solomon (1987). Dullaert et al. (2002) designed a sequential insertion-based approach for FSMVRPTW by extending Solomon's sequential heuristic for VRPTW.

In this paper we present a constructive insertion heuristic for FSMVRPTW; this algorithm is embedded into a metaheuristic approach belonging to the class of the ruin and recreate framework introduced by Schrimpf et al. (2000).

The paper is organized as follows. In §2 the FSMVRPTW is formulated, whereas in §3 a mathematical model for FSMVRPTW is given. We describe our heuristic and metaheuristic algorithms in §§4 and 5, respectively. In §6 we report computational results on benchmark instances from the literature and finally, in §7 we draw some conclusions.

2. Problem Formulation

The FSMVRP can be defined as follows. Let $G = (V, A)$ be a directed graph where $V = N \cup \{0\}$ is the node set and $A = \{(i, j)\colon i, j \in V\}$ is the arc set. Let $N = \{1, \ldots, n\}$ be the set of customers, and let node 0 denote the depot. For each arc $(i, j) \in A$, we denote with d_{ij} the minimal traveling time from i to j. Each customer $i \in N$ is associated with a nonnegative demand q_i (for notation convenience, the depot is assigned a demand $q_0 = 0$), a service time s_i needed to load/unload quantity q_i, and a time window $[a_i, b_i]$. The service of customer i must start between a_i and b_i, i.e., in any feasible solution the vehicle serving customer i either arrives at i at time instant $t \in [a_i, b_i]$, or at a time instant $t < a_i$ and then waits $a_i - t$ time units before starting service.

To simplify the exposition, we assume that all time windows are within a given time horizon (e.g., a single day). We note that this implies $a_i \le b_i$ for each customer $i \in N$.

We are also given a fleet composed by H different vehicle types. Each vehicle of type h $(h = 1, \ldots, H)$ has capacity Q^h and a fixed cost F^h. Each type of vehicles is assumed to be available with infinite supply. Although in real-world applications routing costs may depend on the type of vehicle used, we consider the case in which this cost is identical for all the vehicles and equal to the sum of all the times spent along a route: (i) the traveling time, (ii) the service time, and (iii) the waiting time incurred when a vehicle arrives at a customer i before a_i. Note that the total service time is a constant for all feasible solutions; therefore, we will not explicitly consider it in the computation. The objective of the FSMVRPTW is to determine the optimal mix of the heterogeneous fleet of vehicles and their associated routes to minimize the sum of the vehicle fixed costs and of the routing costs subject to the following constraints:

1. Each route starts and ends at the depot.
2. Each route is assigned to exactly one vehicle.
3. The total demand of all customers served in a route cannot exceed the capacity of the vehicle assigned to that route.
4. Each customer is visited exactly once and its service starts within its time window.

Without loss of generality, we assume that vehicle types are numbered according to a nondecreasing order of fixed-cost values F^h and that, for each customer u, there exists (at least) one type h of vehicles such that $Q^h \ge q_u$, i.e., each customer can be served by a suitable vehicle. Note that, under this assumption, a (trivial) feasible solution to FSMVRPTW always exists: Associate each customer u to a different route performed by a vehicle of type h with $Q^h \ge q_u$. Moreover, if the routing costs are equal for all types of vehicles, we can further assume that $F^{h'} > F^h$ iff $Q^{h'} > Q^h$, because otherwise vehicles of type h' can be removed from the instance; indeed, in any feasible solution, vehicles of type h' can be replaced by vehicles of type h without increasing the cost of the solution.

3. A Mathematical Formulation for FSMVRPTW

In order to define a mathematical formulation for FSMVRPTW, let us define the set K of distinct vehicles obtained by defining n vehicles of type h for each $h \in H$. For each $k \in K$, let $\tilde{Q}^k$ and $\tilde{F}^k$ denote the capacity and the cost of vehicle k, respectively.

Our formulation is a generalization of the three-index formulation of VRP (see Toth and Vigo 2002). Hence, the routing part of the problem is modelled through two sets of binary variables: (i) variables x_{ij}^k taking value 1 iff arc (i, j) is traversed by vehicle k; (ii) variables y_i^k taking value 1 iff customer i is served by vehicle k. In order to select a suitable set of vehicles, we introduce binary variables z^k, whose value is 1 if vehicle $k \in K$ is used and 0 otherwise. Handling time windows and duration of the routes requires the definition of the following sets of variables: (i) variables t_i^k to indicate the minimum time instant in which vehicle k can possibly arrive at each node $i \in V$; (ii) variables τ_i to indicate the minimum time instant in which service at customer i can start; (iii) variable π^k to determine the time instant in which vehicle k, if used, starts its route. Observe that, for each vehicle k, the starting time and ending time of its route are given by the two variables π^k and t_0^k.

Using these variables and a very large positive constant M (which can be set to $\max_{i \in N}\{b_i + s_i\} + \max_{(i,j) \in A} d_{ij}$), problem FSMVRPTW can be formulated as follows:

$$\min \sum_{k \in K} (\tilde{F}^k z^k + t_0^k - \pi^k) - \sum_{i \in N} s_i \tag{1}$$

$$\text{subject to} \sum_{k \in K} y_i^k = 1, \quad i \in N, \tag{2}$$

$$y_i^k = \sum_{j \in V\colon (j,i) \in A} x_{ji}^k, \quad i \in N,\ k \in K, \tag{3}$$

$$y_i^k = \sum_{j \in V:\,(i,j)\in A} x_{ij}^k, \quad i \in N,\ k \in K, \tag{4}$$

$$\sum_{i \in V} q_i y_i^k \le \tilde{Q}^k z^k, \quad k \in K, \tag{5}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij}^k \ge y_l^k, \quad S \subseteq N,\ l \in S,\ k \in K, \tag{6}$$

$$t_j^k \ge \tau_i + s_i + d_{ij} - M(1 - x_{ij}^k), \quad (i,j) \in A:\ i \in N,\ k \in K, \tag{7}$$

$$t_j^k \ge \pi^k + d_{0j} - M(1 - x_{0j}^k), \quad k \in K,\ j \in N, \tag{8}$$

$$t_0^k \ge \pi^k, \quad k \in K, \tag{9}$$

$$\tau_i \ge t_i^k, \quad i \in N,\ k \in K, \tag{10}$$

$$a_i \le \tau_i \le b_i, \quad i \in N, \tag{11}$$

$$x_{ij}^k \in \{0,1\}, \quad (i,j) \in A,\ k \in K, \tag{12}$$

$$y_i^k \in \{0,1\}, \quad i \in N,\ k \in K, \tag{13}$$

$$z^k \in \{0,1\}, \quad k \in K, \tag{14}$$

$$t_i^k \ge 0, \quad i \in V,\ k \in K, \tag{15}$$

$$\tau_i \ge 0, \quad i \in N, \tag{16}$$

$$\pi^k \ge 0, \quad k \in K. \tag{17}$$

Constraint (2) proposes that each customer must be visited by exactly one vehicle, whereas constraints (3) and (4) propose that if vehicle k visits customer i, it must enter and leave, respectively, the associated node. Inequalities (5) state the capacity constraint for each used vehicle $k \in K$, and inequalities (6) give the connectivity requirement on each route. Constraints (7), (10), and (11) define the time instant for the service at each customer i, and impose the time-windows constraints. On the other hand, constraint (8) defines the time instant in which each vehicle $k \in K$ starts its route. Note that when a given vehicle k is not used, variables π^k and t_0^k are unconstrained, but (9) states that $t_0^k - \pi^k \ge 0$, and the objective function will set them to a common value in any optimal solution.

The very large number of variables in model (2)–(17) and the presence in the constraints of very large M values make it impossible (or extremely hard) to obtain the exact solution of the problem. Hence, we face FSMVRPTW by means of a heuristic algorithm, which is outlined in the next section.

4. Constructive Heuristic

In this section we present a multistart constructive heuristic for FSMVRPTW, denoted as M_ParallelRegret_TW, which executes the constructive heuristic ParallelRegret_TW several times, with different values of some input parameters. We first describe our algorithm ParallelRegret_TW (§4.1), and then provide details on the multistart strategy (§4.2).

4.1. ParallelRegret_TW Algorithm

The constructive heuristic we present can be classified as a parallel-route-construction approach and implements the *regret procedure* presented in Christofides, Mingozzi, and Toth (1979) and Potvin and Rousseau (1993).

The algorithm is executed by giving as input a partial solution consisting of a set of feasible routes that does not serve all customers. This partial solution is completed by iteratively assigning each unrouted customer either to the existing routes or to a new one. The choice of the next customer to be assigned is performed by associating with each unrouted customer u a score $\theta(u)$ that takes into account the urgency of assigning customer u. The computation of $\theta(u)$ is based on a penalty $\xi(u,r)$, which is related to the cost of assigning customer u to a specific route r.

We now describe in detail how these scores are computed. Given a route r, let $U_r = \{i_0, i_1, \ldots, i_{n(r)-1}, i_{n(r)}\}$ with $i_0 = i_{n(r)} = 0$, denote the sequence of nodes visited by the route. Moreover, let $\bar{n}$ denote the total number of customers already inserted in the current partial solution, and let $g(u,r,p)$ be the minimum arrival time at customer u if it is served immediately after customer i_p in route r. Given a route r, an insertion point p is declared infeasible for customer u if either $g(u,r,p) > b_u$ or if serving u after i_p delays the arrival at some customer i_k $(k = p+1, \ldots, n(r)-1)$ after b_{i_k}. The feasibility check of the insertion has been implemented according to the parametric approach of Savelsbergh (1992).

Let $\Phi(r,u)$ denote the set of feasible insertion points for customer u in route r. Score $\xi(u,r)$ is the weighted sum of three elements: (i) the additional routing cost associated to the insertion of customer u in route r without taking into account the vehicle capacity; (ii) the additional cost denoted as $Chg\_V(u,r)$ (see below) to be paid when the vehicle serving route r must be changed to also visit customer u; (iii) the waiting time to be paid when a vehicle arrives at a customer u before a_u (i.e., before the customer time window).

Formally, we have

$$\xi(u,r) = \min_{p \in \Phi(r,u)} \left\{ \left(d_{i_p u} + d_{u i_{p+1}} - d_{i_p i_{p+1}} \right) + \alpha \frac{\bar{n}}{n} Chg\_V(u,r) + \beta \frac{\bar{n}}{n} \max(0, a_u - g(u,r,p)) \right\}, \tag{18}$$

where α and β are tuning parameters. Observe that both the second term and the third term have been multiplied by the ratio between $\bar{n}$ and the total number of customers n in order to increase their relative importance as soon as the number of routed

customers increases. Further, note that the first term always has a nonnegative value because the distances d_{ij} satisfy the triangular inequality.

To formally describe cost $Chg\_V(u, r)$, let h be the type of vehicle serving route r and let h' be the cheapest type of vehicle we must use when u is added to the route. The cost is

$$Chg\_V(u, r) = \left(1 - \frac{\bar{n}}{n}\right) C_R + \frac{\bar{n}}{n} C_A, \tag{19}$$

where C_R is the *relative* fixed-cost increase

$$C_R = F^{h'} \sum_{i \in U_r \cup \{u\}} \frac{q_i}{Q^{h'}} - F^h \sum_{i \in U_r} \frac{q_i}{Q^h} \tag{20}$$

and C_A denotes the *absolute* fixed-cost increase

$$C_A = F^{h'} - F^h. \tag{21}$$

Note that C_A is zero if the vehicle type h does not change. Moreover, we assume that both costs are infinite if the total demand of the route, including customer u, exceeds the largest vehicle capacity. The ratio $\bar{n}/n$ is used in the computation of score Chg_V to give different emphasis to the two costs C_R and C_A during the evolution of the algorithm. In particular, solutions containing almost all customers have a score mainly determined by C_A, whereas solutions with few customers have scores mainly depending on C_R—thereby favoring those insertions that increase the vehicle capacity.

We now have all the ingredients necessary to formally define score $\theta(u)$ used to select the next customer to be inserted in a route. Score $\theta(u)$ includes: (a) the increase of penalty ξ that occurs when customer u is not assigned to its preferred route; (b) the distance of u from the depot; (c) the width of the time window of u.

We have

$$\theta(u) = \left(\operatorname{smin}_r \xi(u, r) - \min_r \xi(u, r)\right) + \gamma_d d_{0u} - \gamma_w (b_u - a_u), \tag{22}$$

where "smin" denotes the second minimum value, and γ_d and γ_w are two weighing parameters. Criterion (a), which is often called a *regret* criterion, considers the difficulty of assigning customer u in relation to the current partial solution, whereas criteria (b) and (c) depend on the absolute difficulty of the customer. We assume that whenever a customer u can be inserted into a unique route, then smin takes a very large value, so imposing that u must be assigned immediately.

The overall algorithm ParallelRegret_TW, depicted in Figure 1, works as follows. For each unrouted customer u, we compute penalty $\xi(u, r)$ for each route r, and hence score $\theta(u)$ according to (22). Then we iteratively assign each unrouted customer to a route. At each iteration, customer $\bar{u}$ having the maximum θ value is selected and assigned either to the best possible position in the route corresponding to the minimum $\xi(\bar{u}, r)$ value, or to a new empty route if the former choice is not possible or the latter one uses a vehicle with smaller fixed cost. (Note that we implemented this test by means of (19), which disregards the routing costs because a new route with a single customer will be modified, with high probability, during the next iterations; therefore its current routing cost could be misleading.) Finally, the score vector θ is updated and a new iteration is performed. The algorithm ends when all customers are routed.

```
ALGORITHM PARALLELREGRET_TW.
input: S_0 [partial solution], α, β, γ_d, γ_w [parameters];
output: S [updated solution], z [value of the new solution];
  S = S_0;
  for each unrouted customer u do
    for each route r do compute the score ξ(u, r) [see (18)];
    define as r'(u) and r''(u) the routes corresponding to the first
      and second minimum of ξ(u, r)
    compute the score θ(u) [see (22)];
  endfor
  while the set of unrouted costumers is not empty do
    determine the customer ū with maximum θ(u) value;
    if (Chg_V(u, r'(ū)) > min_h{F^h: Q^h ≥ q_ū}) then
      initialize a new route r̂ in S with customer ū;
    else
      set r̂ = r'(ū) and let p̄ be the best feasible insertion point
      for customer ū in r̂: insert ū in position p̄ of route r̂,
      possibly changing the associated vehicle;
    endif
    update S and its value z;
    for each unrouted customer u do
      update score ξ(u, r̂) and r'(u), r''(u);
      if (r'(u) = r̂ or r''(u) = r̂) then
        update score θ(u);
      endif
    endfor
  endwhile
end.
```

Figure 1 Pseudocode of Algorithm ParallelRegret_TW

4.2. Multistart ParallelRegret_TW Algorithm

To find a good initial solution, we execute the ParallelRegret_TW algorithm presented in the previous section several times by initializing it with different partial solutions, each composed by ν single customer routes, where ν iteratively assumes the integer values in an interval $[\nu_{\min}, \nu_{\max}]$. When the algorithm is used to build a solution from scratch, we use for $\nu_{\min}$ the sum of the demands of the customers multiplied by 1.5 and divided by the maximum capacity of the available vehicles, and we set $\nu_{\max} = 2\nu_{\min}$. If instead the algorithm is used to complete a partial solution,

we set $\nu_{\min} = 0$ and $\nu_{\max}$ to a value depending on the number of the unrouted customers (see Figure 3).

For a given ν value, we select eight different sets of "seed" customers and, for each such set, we run algorithm ParallelRegret_TW several times, with different parameters, thus obtaining several potentially different solutions for each value of ν.

To select the ν seeds, we adopt a spatial criterion similar to the geometric method of Fisher and Jaikumar (1981): We partition the plane into ν cones with origin at the depot, and we try to choose one customer inside each cone. To do this, we consider one cone at a time, and for each unselected customer u, we compute the angular distance, e.g., $ray(u)$, between u and the ray bisecting the cone. In each of the eight runs, we associate with each customer u a weight $w(u)$ depending on one of the eight combinations of the following three customer characteristics: (i) width of the time window, (ii) demand, and (iii) distance from the depot. We use a binary vector $\delta = (\delta_1, \delta_2, \delta_3)$ to define the overall criterion:

$$w(u) = 100ray(u) + \delta_1(b_u - a_u) - \delta_2 5q_u - \delta_3 5d_{0u}. \quad (23)$$

Note that each of the eight values of δ determine a different weighting strategy and a different choice of the seeds. The customer with minimum w is selected as the seed of the current cone and the weights are recomputed with respect to the next cone.

Starting from each set of seed customers, we run ParallelRegret_TW 64 times, one for each combination of the parameters $\alpha \in \{0.0, 0.5, 1.0, 1.5\}$, $\beta \in \{0.0, 0.2\}$, $\gamma_d \in \{0.0, 0.5, 1.0, 1.5\}$, and $\gamma_w \in \{0.0, 0.2\}$. Details on the choice of these parameters can be found in §6.1, but here it is interesting to note that the performances of the algorithm mainly depend on the values of α and γ_d, so we gave more tentative values to these parameters.

The resulting algorithm, denoted as M_Parallel-Regret_TW. is described in Figure 2.

```
ALGORITHM M_PARALLELREGRET_TW.
input: S_0 [partial solution, possibly empty], ν_min, ν_max;
output: S* [updated solution], z* [value of the new solution];
  z* = ∞, S* = ∅;
  for ν = ν_min to ν_max do
    for each of the eight values of binary vector δ do
      partition the plane into ν identical cones with origin on the
        depot;
      Seed = ∅;
      for each cone do
        compute w(u) for each customer u ∉ Seed ∪ S_0 [see (23)];
        find ū: w(ū) = min{w(u): u ∉ Seed ∪ S_0}
        Seed = Seed ∪ {ū};
      endfor
      let R̄ = {ν single customer routes defined by Seed};
      associate each route in R̄ with the cheapest feasible vehicle;
      set S = S_0 ∪ R̄;
      for each set of parameters α, β, γ_d, γ_w do
        (S', z') = ParallelRegret_TW(S, α, β, γ_d, γ_w);
        if z' < z* then z* = z', S* = S';
      endfor
    endfor
  endfor
end.
```

Figure 2 Detailed Description of Algorithm M_ParallelRegret_TW

5. Ruin and Recreate Approach

The ruin and recreate (RR) approach is a new class of algorithms introduced by Schrimpf et al. (2000), who used this framework to solve VRPTW instances with encouraging results. Recently, Pisinger and Ropke (2007) and Ropke and Pisinger (2007) applied their adaptive large neighborhood search (ALNS) to solve different types of VRPs with competitive results. The ALNS framework is an extension, based on the ruin and recreate paradigm, of the large neighborhood search proposed by Shaw (1998).

The basic idea of the RR approach is to obtain new solutions of an optimization problem by considerable destruction (*ruin*) of an existing feasible solution, followed by a rebuilding (*recreate*) procedure to obtain a new complete solution. The main elements of an RR approach are:

- one (or more) constructive heuristic;
- some ruin strategies;
- a decision rule for solution acceptance.

The *constructive heuristic* algorithm is used both to obtain an initial feasible solution and to complete a partial solution during the recreate step. A *ruin strategy* consists of selecting parts of the current solution by means of an appropriate criterion and removing them from it. In general, more than one of such strategies may be available, and the one to be applied to a given solution is randomly selected. The partial solution is transformed into a new feasible solution with the rebuilding procedure, and the new solution is accepted according to the *decision rule*. If the new solution is accepted, it becomes the current solution and a new ruin step is performed; otherwise, a new ruin step is applied to the previous solution until the new solution is accepted. The process is iterated until a prescribed stopping criterion is met. When applied to a routing problem, the procedure can be specialized as follows:

1. Construct a feasible solution $\mathcal{S}^*$ and let z^* be its value;
2. **while** a stopping criterion is not met **do**
   $\mathcal{S}_0 = \mathcal{S}^*$;
3. **for each** Ruin strategy **do**
   $\mathcal{S} = \mathcal{S}_0$;
   3.1. using the selected strategy, select a set $\bar{U}$ of customers and remove them from $\mathcal{S}$ **[Ruin step];**

3.2. using a constructive heuristic, appropriately reinsert each customer of set $\bar{U}$ in $\mathcal{S}$ **[Recreate step];**
3.3. let z be the cost of $\mathcal{S}$: **if** $(z < z^*)$, **then** $z^* = z$, $\mathcal{S}^* = \mathcal{S}$;
endfor
endwhile.

According to the description above, our RR algorithm performs a sequence of iterations until a given time limit has been reached. Each iteration is executed, starting from the best current solution $\mathcal{S}^*$, and includes several RR steps. Indeed, many partial solutions are obtained from the same starting solution by removing a proper set of customers (see §5.1) and completing the resulting solution according to the recreate strategy described in §5.2.

The RR approach and the classical local search (LS) paradigm share some similitude, but also present many differences that we will briefly highlight in the following. Both methods make use of a systematic perturbation of the current solution that leads to a new solution. LS is based on the concept of *neighborhood*, which is a set of solutions obtained from the current one with a small modification. For VRPs, this modification may be, e.g., the movement of a single customer to a different route. To choose the next solution to be considered, LS usually explores all or almost all of the neighborhood. In other words, we can say that LS performs a deep evaluation of solutions close to the current one and then moves to a neighboring solution. Ruin and recreate, on the other hand, is based on the destruction of a large part of the current solution followed by a rebuild of a complete solution. Therefore, RR does not explore solutions close to the current one, but evaluates solutions that may be far from the current one in the solution space. In the LS literature the attempt to continue the search from distant solutions is known as a *diversification strategy* and is rarely applied during the evolution of the search. With this setting, RR is a pure systematic use of a diversification strategy.

5.1. Ruin Strategy

In our problem, the choice of the vehicles to be used is crucial and, broadly speaking, may be seen as a variant of a bin-packing problem. Therefore, our ruin strategy was inspired by previous works in the packing area and, in particular, by the unified tabu search framework proposed by Lodi, Martello, and Vigo (1999) for the two-dimensional bin-packing problem, in which the neighborhood is searched through moves that attempt to change the packing of a subset of items to empty a specific *target bin*.

As discussed in the previous section, at each iteration of the overall algorithm, several attempts to destroy the current solution are performed, each corresponding to a set of customers to be removed. All of these sets include a customer belonging to a *target route* and the remaining customers belonging to other routes.

The choice of the target route is made using a criterion that tries to identify a route that could be served by a cheaper vehicle if a small number of customers is removed from it. Given a route r, let TL_r be its current load and h the vehicle type currently assigned to it. We select as the target the route $\bar{r}$ that minimizes the *load excess*

$$LE(r) = TL_r - Q^{h-1}, \tag{24}$$

which represents the nonnegative quantity of goods that cannot be delivered to the customers served by r using a vehicle of type $h-1$.

In our RR implementation we introduced a simple tabu criterion to avoid selecting a route twice. At each ruin step we store the selected route $\bar{r}$ in a tabu list with infinite length (so that $\bar{r}$ cannot be further selected during the entire execution of the algorithm). Due to this tabu we can fail to find a target route. In this case, we prematurely terminate the algorithm. We used a straightforward implementation of the tabu list, but the time spent to check the tabu status was generally negligible in our computational experiments.

Given the target route $\bar{r}$, each of its customers, in turn, is considered a target customer. The target customer is used to initialize the set of customers to be removed.

The other customers to be removed are selected with the following procedure. For each target customer $\bar{u}$ of $\bar{r}$ and for each route $r \neq \bar{r}$ we compute the extra distance associated with the best insertion of $\bar{u}$ in r. Then, we perform one or two iterations by increasing an index k from 2 to $\min(3, \#r)$, where $\#r$ is the total number of routes in the current solution. For each value of k, we define $K(\bar{u})$ as the set of the $2k$ routes with smallest extra distance, and we iteratively destroy all k-tuple of routes in $K(\bar{u})$ (i.e., we define a partial solution in which all the customers of the selected routes, as well as the target customer $\bar{u}$, are unrouted). Each partial solution is then transformed into a feasible solution through the recreate strategy described in the next section.

5.2. Recreate Strategy

Starting from a partial solution obtained with the ruin step of the previous section, we apply algorithm M_ParallelRegret_TW described in §4.2 to complete it. Because this step is performed several times during the improving phase, we speed up the computation by using only one of the tentative values for the two less important parameters β and γ_w. In particular,

```
    Algorithm Target_RR_TW.
input: a FSMVRPTW instance
output: 𝒮* [solution], z* [value of the solution]
  [initialization phase]
  define ν_min and ν_max [see §4.2];
  𝒮_0 = ∅, (𝒮*, z*) = M_ParallelRegret_TW(𝒮_0, ν_min, ν_max);
  [improving phase]
  while not time limit do
    𝒮_0 = 𝒮*;
    determine the non-tabu route r̄ minimizing the load excess
      [see (24)];
    if no such route exists then stop;
    store r̄ in the tabu list;
    for each customer ū in the target route r̄ do
      remove customer ū from route r̄ of 𝒮_0;
      for k = 2 to min(3, number of routes in 𝒮_0) do
        let K(ū) be the set of 2k routes with smallest extra
        distance associated with the best insertion of ū;
        for each k − tuple of routes in K(ū) do
          [Ruin step]
          𝒮 = 𝒮_0;
          remove all customers of the k selected routes from 𝒮;
          [Recreate step]
          ν_max = min(2k, |{removed customers}|);
          (𝒮, z) =M_ParallelRegret_TW(𝒮, 0, ν_max);
          if (z < z*) then z* = z, 𝒮* = 𝒮;
        endfor;
      endfor;
    endfor;
  endwhile;
end.
```

Figure 3 Detailed Description of Algorithm RR_TW

we have chosen the average of the two values used in the construction of the starting solution by setting $\beta = \gamma_w = 0.1$.

5.3. Overall Algorithm

The overall algorithm, denoted as Target_RR_TW and detailed in Figure 3, starts by computing an initial solution by using the constructive heuristic algorithm M_ParallelRegret_TW described in §4.2, with all the combinations of the four parameters. This solution is refined through an improving phase that alternates ruin and recreate steps until either a time limit is reached or no target route can be found for the ruin step because of the tabu status of all routes.

Figure 4 shows one iteration of the ruin and recreate algorithm when applied to instance 03 of Golden et al. (1984) for the FSMVRP with 20 customers; this instance was chosen for the sake of clarity: It has no time windows and has Euclidean distances as costs. The demand of each customer is reported close to the corresponding node, and the total demand of each route is indicated in a box. There are five types of vehicles, having capacities $Q = \{20, 30, 40, 70, 120\}$ and costs $F = \{20, 35, 50, 120, 225\}$. Starting from an initial solution (Figure 4(a)), the target route is $\bar{r} = 6$. Next we consider what happens when the target customer is $\bar{u} = 6$, k takes value 2, and the two routes to be ruined are 4 and 5 (see Figures 4(b) and 4(c)). In the recreate step all unrouted customers are included in the two new routes 4′ and 5′ shown in Figure 4(d). In the new solution, the target route and route 4′ are now assigned to smaller vehicles with capacities 20 and 30, respectively, thus leading to an overall fixed-cost saving equal to 45 units. At the same time, the overall routing cost evolves from a value of 388.53 to a final value of 371.03.

6. Computational Results

The algorithm presented in §5 was coded in ANSI C++ and run on a Pentium 600 MHz personal computer on the benchmark instances proposed by Liu and Shen (1999), derived from the VRPTW instances by Solomon (1987).

Solomon's benchmark for VRPTW consists of 56 Euclidean instances, each having 100 customers with coordinates generated according to three different methods: Data set R consists of randomly generated values; data set C of clustered points; data set RC is a mixture of random and clustered values. Each of the three sets consists of two subsets. Subsets R1, C1, and RC1 have a short scheduling horizon and small vehicle capacities; hence, only a small number of customers can be served by a route. Subsets R2, C2, and RC2, on the other hand, may have routes with a larger number of customers. For each of the Solomon's instances, Liu and Shen (1999) introduced three different subclasses of instances, characterized by different vehicle costs: type a with large vehicle cost, type b with medium vehicle cost, and type c with small vehicle cost. Combining each VRPTW instance with the three classes of vehicle costs, a total of 168 instances is obtained.

The results of our algorithm are compared with those of the heuristics by Liu and Shen (1999) and by Dullaert et al. (2002), referred to as L-S and DJSV, respectively. Because in these two papers different ways of evaluating the solution cost are adopted, we describe the results of our algorithm by means of two different tables.

Table 1 compares our algorithm with that of Liu and Shen (1999), where the solution cost is given by the total fixed cost of the used vehicles and the total schedule time, excluding the (constant) service time at the customers. The results are presented in a way similar to that adopted in Liu and Shen (1999), namely by giving the average results both for each group of instances (i.e., R1a, R1b, ..., RC2c) and for each of the six subsets (i.e., R1, C1, ..., RC2). The last row of the table reports the average results over all the instances. The first two columns identify the data set and the number of instances it includes. The next

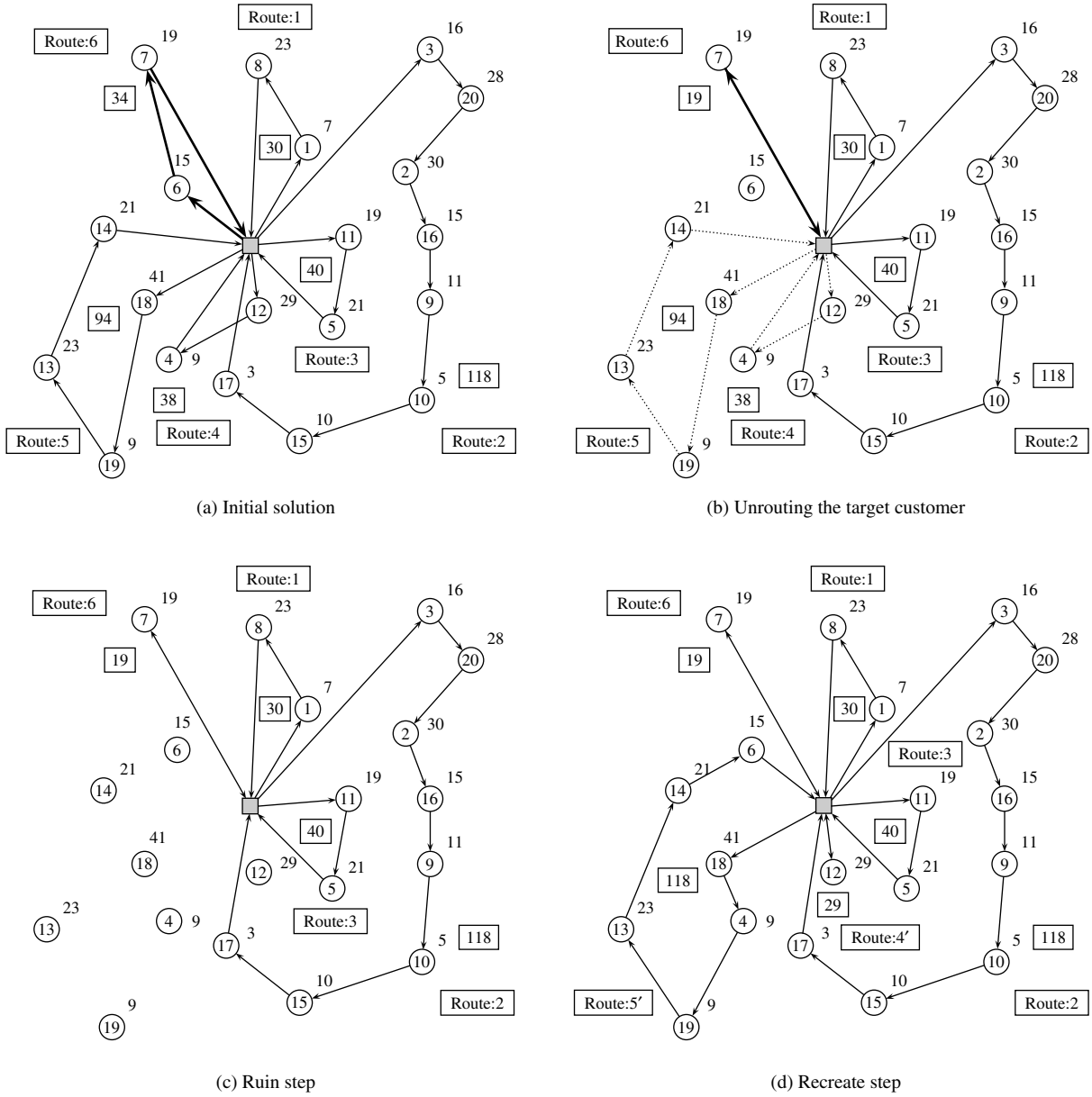

Figure 4 Evolution of Ruin and Recreate Steps

six columns report the results of Algorithm L-S without and with the improving phase, giving the average solution value (label "Val."), the average percentage error (label "Δ%"), and the average computing time (label "Time"), when available. The percentage error is computed with respect to the best average solution value known for each group of instances, which is always obtained by our Target_RR_TW algorithm with a time limit of 920 CPU seconds. The computing times of Algorithm L-S are expressed in seconds on a Pentium 233 MHz. In order to have a raw comparison with our algorithm we report in the last row of the table the average computing time on the original Pentium 230 and (in parentheses) the same value scaled accordingly to Dongarra (2006), with respect to our Pentium 600. The last eight columns of Table 1 refer to the results of our constructive M_ParallelRegret_TW algorithm and to the ruin and recreate algorithm Target_RR_TW. For the latter, we report the results obtained with two different time limits that include both the constructive and improving phases. We have set the first time limit to the average (scaled) computing time of Algorithm L-S with improving phase, i.e., to 230 seconds. The longer time limit was set to 920 seconds, i.e., four times the first one. We remind the reader that the total computing time for Target_RR_TW may be less than the imposed time limit, because the Target_RR_TW algo-

Table 1 Comparison with Algorithm L-S

		L-S									Target_RR_TW				
		No impr.			Impr.			M_ParallelRegret_TW			Time limit 230			Time limit 920	
Data set	Nr.	Val.	Δ%	Time	Val.	Δ%	Time	Val.	Δ%	Time	Val.	Δ%	Time	Val.	Time
R1a	12	4,562	9.12		4,398	5.19		4,362.11	4.28	81.99	4,223.83	0.98	230.79	4,182.89	921.95
R1b	12	2,149	11.49		2,054	6.56		2,021.19	4.54	63.70	1,944.52	0.57	230.83	1,933.45	921.57
R1c	12	1,788	10.68		1,700	5.23		1,692.46	5.18	65.60	1,619.35	0.64	230.85	1,609.12	921.53
R1	36	2,833.00	10.04	524	2,717.33	5.54	531	2,691.92	4.53	70.43	2,595.90	0.81	230.82	2,575.15	921.68
C1a	9	8,042	11.25		8,007	10.76		8,137.01	12.92	70.11	7,334.23	1.78	230.76	7,205.93	921.67
C1b	9	2,626	10.12		2,485	4.20		2,451.87	2.61	54.47	2,411.93	0.94	230.99	2,389.46	921.27
C1c	9	1,870	14.75		1,705	4.62		1,662.69	2.28	48.87	1,626.98	0.08	230.98	1,625.68	921.28
C1	27	4,179.33	11.51	429	4,065.67	8.48	435	4,083.86	9.18	57.82	3,791.05	1.36	230.91	3,740.36	921.40
RC1a	8	5,429	6.08		5,184	1.29		5,213.51	2.19	58.67	5,142.38	0.79	230.87	5,102.02	921.36
RC1b	8	2,342	8.25		2,235	3.30		2,305.10	6.27	48.19	2,190.73	1.00	230.80	2,169.08	920.86
RC1c	8	1,926	7.93		1,849	3.61		1,917.49	7.50	47.61	1,803.73	1.12	230.81	1,783.73	921.09
RC1	24	3,232.33	6.96	464	3,089.33	2.23	470	3,145.37	4.21	51.49	3,045.61	0.91	230.83	3,018.27	921.10
R2a	11	3,855	8.01		3,809	6.73		3,838.52	6.10	52.71	3,661.32	1.20	230.92	3,617.79	921.24
R2b	11	1,915	10.88		1,797	4.05		1,830.57	6.74	53.69	1,767.03	3.04	230.77	1,714.96	896.98
R2c	11	1,589	10.64		1,513	5.35		1,533.00	5.33	53.84	1,482.60	1.87	230.79	1,455.36	921.03
R2	33	2,453.00	9.31	523	2,373.00	5.75	529	2,400.70	6.10	53.41	2,303.65	1.81	230.83	2,262.70	913.08
C2a	8	7,058	12.61		6,717	7.17		6,642.78	9.47	54.23	6,209.26	2.32	230.77	6,068.39	921.30
C2b	8	2,054	8.24		1,970	3.81		1,918.23	2.80	53.61	1,895.23	1.56	230.76	1,866.07	921.15
C2c	8	1,373	7.58		1,288	0.92		1,276.10	0.88	54.22	1,268.47	0.28	230.81	1,264.97	683.24
C2	24	3,495.00	11.05	438	3,325.00	5.65	444	3,279.04	6.93	54.02	3,124.32	1.89	230.78	3,066.47	841.90
RC2a	8	5,381	13.21		5,273	10.94		5,318.88	13.40	47.33	4,849.99	3.40	231.02	4,690.33	921.18
RC2b	8	2,432	12.80		2,324	7.79		2,341.76	7.45	46.57	2,189.08	0.45	230.91	2,179.36	921.14
RC2c	8	2,066	12.96		1,978	8.15		1,945.01	12.19	48.06	1,791.03	3.31	230.83	1,733.71	920.90
RC2	24	3,293.00	13.06	527	3,191.67	9.58	533	3,201.88	11.65	47.32	2,943.37	2.64	230.92	2,867.80	921.07
Avg.		3,192.07	10.27	488 (226)	3,074.11	6.17	494 (230)	3,079.92	6.90	56.71	2,919.94	1.52	230.85	2,876.34	908.38

rithm terminates if it fails in finding a non-tabu target route (see, e.g., set C2c and the second time limit). The best overall results are always obtained with the longer time limit.

Table 1 shows that our constructive algorithm M_ParallelRegret_TW outperforms Algorithm L-S without improvement, with respect to both the quality of the solutions found and the computing times. Moreover, our constructive heuristic produces solutions that are very close to those of Algorithm L-S with improvement and that require a shorter computing time (one-fourth of the running time of L-S).

Algorithm Target_RR_TW with a time limit of 230 seconds always produces solutions that are much better than those of Algorithm L-S with improvement. The average percentage gap is reduced by a factor of four, that is, from 6.17% (for L-S) to 1.52% (for our algorithm). Better solutions are obtained with the longer time limit.

An analysis of the impact of the improving phase of algorithm Target_RR_TW on the final solution value is illustrated in Figure 5. We show how the solution value changes with the running time, giving the average percentage improvement, computed on all 168 instances, with respect to the value of the solution provided by M_ParallelRegret_TW. The improving phase is monotonically increasing with the computing time and does not show saturation effects in the 920 seconds we considered. The figure also shows that our solutions are already better than those of Liu and Shen (1999) after 60 seconds of computation.

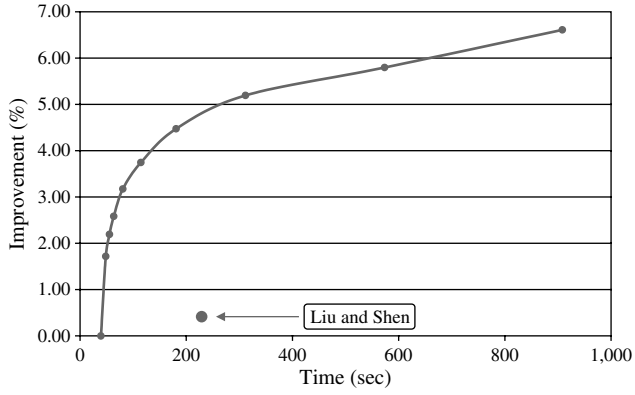

Figure 5 Improvement of Algorithm Target_RR_TW

Table 2 Comparison with Algorithm DJSV

		DJSV	M_ParallelRegret_TW	
Data set	Nr.	Val.	Val.	Δ% (%)
R1a	12	1,548.53	1,469.60	5.10
R1b	12	1,557.38	1,375.88	11.65
R1c	12	1,557.85	1,341.58	13.88
R1	36	1,554.59	1,395.69	10.22
C1a	9	1,166.09	934.71	19.84
C1b	9	1,126.01	884.48	21.45
C1c	9	1,155.45	865.77	25.07
C1	27	1,149.18	894.99	22.12
RC1a	8	1,665.04	1,618.13	2.82
RC1b	8	1,680.55	1,523.10	9.37
RC1c	8	1,689.92	1,508.00	10.77
RC1	24	1,678.50	1,549.74	7.67
R2a	11	1,426.52	1,271.70	10.85
R2b	11	1,431.49	1,269.48	11.32
R2c	11	1,419.81	1,269.58	10.58
R2	33	1,425.94	1,270.25	10.92
C2a	8	821.38	616.70	24.92
C2b	8	821.38	616.70	24.92
C2c	8	811.16	616.70	23.97
C2	24	817.97	616.70	24.61
RC2a	8	1,800.82	1,648.53	8.46
RC2b	8	1,741.97	1,612.39	7.44
RC2c	8	1,741.75	1,571.35	9.78
RC2	24	1,761.51	1,610.75	8.56
Avg.		1,406.20	1,232.03	13.72

Table 2 compares our approach with the constructive Algorithm DJSV proposed by Dullaert et al. (2002), which solves a problem in which the objective function includes only the total schedule time (i.e., it excludes the service times at the customers and the vehicle fixed costs). Dullaert et al. (2002) do not give computing times for their algorithm, because they observed that their method is simpler and faster than the Algorithm L-S. We have already observed that the computing times for our algorithm M_ParallelRegret_TW are considerably shorter than those of Algorithm L-S, as well. For each group of instances, Table 2 gives the average solution value (label "Val.") for both algorithms DJSV and M_ParallelRegret_TW, and the percentage gap between the two solution values (label "Δ%"). One can see that our constructive heuristic largely outperforms Algorithm DJSV by improving the solution value, on average, by 13.72%.

6.1. Parameter Tuning and Sensitivity Analysis

In this section we describe the method we used to tune the parameters of the constructive algorithm M_ParallelRegret_TW and discuss the robustness of the overall RR algorithm when these parameters change.

In order to determine the effective sets of values of the four parameters α, β, γ_d, γ_w of algorithm M_ParallelRegret_TW, we chose as a subset of representative problems the 28 instances of sets R1b, RC1a, and C2c from Liu and Shen (1999), and used a simple tuning process by varying one parameter at a time and fixing the remaining ones to zero. In particular, for each parameter, we considered all values in $[0, 2]$ with stepsize equal to 0.1. The result of this analysis showed that the influence of parameters α and γ_w on the solution quality is much greater than that of parameters β and γ_d. Therefore, we decided to select four values for the first two parameters and two for the last two parameters. By considering the values giving the best average results we selected values $\{0.0, 0.5, 1.0, 1.5\}$ for α and γ_w and values $\{0.0, 0.2\}$ for β and γ_d. Therefore, we defined a total of 64 vectors of parameters by considering all combinations of such values.

In order to analyze the sensitivity of algorithm Target_RR_TW to the change of the values of α, β, γ_d, γ_w, we have solved all the benchmark instances by giving random values to these parameters. More precisely, at each execution of algorithm ParallelRegret_TW, we randomly generated the value of each parameter uniformly in the range $[0, 2]$. The average percentage error of this method, over all 168 instances of our test bed, was 3.9% with a time limit of 230 seconds and reduces to 2.1% by allowing a longer time limit of 920 seconds, confirming the quite robust behaviour of the approach.

7. Conclusions

This paper describes an insertion-based parallel approach for problem FSMVRPTW and a metaheuristic procedure that adopts the RR paradigm for the improvement of the current solution.

The main idea underlying the insertion-based parallel approach is the use of dynamic weight parameters that change values with the progressive insertion of the customers to give more or less importance to a certain insertion criterion.

The metaheuristic procedure behaves effectively for problems with heterogeneous fleet because the ruin step systematically allows for the frequent combination of two or more routes into a longer route requiring a larger vehicle and the splitting of a given route into a few smaller routes.

Computational testing on a test bed from the literature revealed that this new heuristic significantly outperforms the previous published heuristics for the same problem.

Acknowledgments

The authors thank the Ministero dell'Istruzione, dell'Università e della Ricerca, and the Consiglio Nazionale delle Ricerche, Italy, for the support given to this project. They

also thank two anonymous referees and an associate editor for constructive comments that considerably improved this paper.

References

Christofides, N., A. Mingozzi, P. Toth. 1979. The vehicle routing problem. N. Christofides, A. Mingozzi, P. Toth, C. Sandi, eds. *Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 315–338.

Clarke, G., J. Wright. 1964. Scheduling of vehicles from a central depot to a number of delivery point. *Oper. Res.* **12** 568–581.

Dongarra, J. J. 2006. Performance of various computers using standard linear equations software. Technical Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN.

Dullaert, W., G. K. Janssens, K. Sorensen, B. Vernimmen. 2002. New heuristics for the fleet size and mix vehicle routing problem with time windows. *J. Oper. Res. Soc.* **53** 1232–1238.

Fisher, M. L., R. Jaikumar. 1981. A generalized assignment heuristic for the vehicle routing problem. *Networks* **11** 109–124.

Golden, B., A. Assad, L. Levy, F. Gheysens. 1984. The fleet size and mix vehicle routing problem. *Comput. Oper. Res.* **11** 49–66.

Liu, F. H., S. Y. Shen. 1999. The fleet size and mix vehicle routing problem with time windows. *J. Oper. Res. Soc.* **50** 721–732.

Lodi, A., S. Martello, D. Vigo. 1999. Heuristic and meta-heuristic approaches for a class of two-dimensional bin packing problems. *INFORMS J. Comput.* **11** 345–357.

Pisinger, D., S. Ropke. 2007. A general heuristic for vehicle routing problems. *Comput. Oper. Res.* **34**(8) 2403–2435.

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

Renaud, J., F. Boctor. 2002. A sweep-based algorithm for the fleet size and mix vehicle routing problem. *Eur. J. Oper. Res.* **140** 618–628.

Ropke, S., D. Pisinger. 2007. A unified heuristic for a large class of vehicle routing problems with backhauls. *Eur. J. Oper. Res.* **171** 750–775.

Savelsbergh, M. W. P. 1992. *Computer Aided Routing*. CWI Tract 75, Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands.

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

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. *Principles and Practice of Constraint Programming—CP98*, Vol. 1520. *Lecture Notes in Computer Science*. Springer, New York, 417–431.

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

Toth, P., D. Vigo. 2002. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematic Monographs on Discrete Mathematics and Applications, Philadelphia, PA.