A NEW OPTIMIZATION ALGORITHM FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

MARTIN DESROCHERS
GÉRAD and École Polytechnique, Montreal, Canada

JACQUES DESROSIERS
GÉRAD and École des Hautes Etudes Commerciales, Montreal, Canada

MARIUS SOLOMON
Northeastern University, Boston, Massachusetts

(Received June 1990; revision received November 1990; accepted January 1991)

The vehicle routing problem with time windows (VRPTW) is a generalization of the vehicle routing problem where the service of a customer can begin within the time window defined by the earliest and the latest times when the customer will permit the start of service. In this paper, we present the development of a new optimization algorithm for its solution. The LP relaxation of the set partitioning formulation of the VRPTW is solved by column generation. Feasible columns are added as needed by solving a shortest path problem with time windows and capacity constraints using dynamic programming. The LP solution obtained generally provides an excellent lower bound that is used in a branch-and-bound algorithm to solve the integer set partitioning formulation. Our results indicate that this algorithm proved to be successful on a variety of practical sized benchmark VRPTW test problems. The algorithm was capable of optimally solving 100-customer problems. This problem size is six times larger than any reported to date by other published research.

The routing and scheduling of vehicles represents an important component of many distribution and transportation systems costs. The vehicle routing problem (VRP) involves the design of a set of minimum cost routes, originating and terminating at a central depot, for a fleet of vehicles which services a set of customers with known demands. Each customer is serviced exactly once and, furthermore, all the customers must be assigned to vehicles such that the vehicle capacities are not exceeded. The generic VRP and many of its practical occurrences have been studied intensively in the literature (Bodin et al. 1983, Magnanti 1981, and Laporte and Nobert 1987).

The vehicle routing problem with time windows (VRPTW) is a generalization of the VRP involving the added complexity of allowable delivery times, or time windows. In these problems, the service of a customer, involving pick-up (delivery) of goods or services, can begin within the time window defined by the earliest and the latest times when the customer will permit the start of service. Note that the times at which services begin are decision variables. This paper treats the hard window case, where if a vehicle arrives at a customer too early, it will wait. In addition, due dates cannot be violated. Time windows arise naturally in problems faced by business organizations that work on fixed time schedules. Specific examples of problems with hard time windows include bank deliveries, postal deliveries, industrial refuse collection and school bus routing and scheduling.

In this paper, we assume a homogeneous fleet. Furthermore, the number of vehicles used is free, i.e., the fleet size is determined simultaneously with the best set of routes and schedules rather than being fixed a priori. Note that even finding a feasible solution to the VRPTW when the number of vehicles is fixed is itself an NP-complete problem (Savelsbergh 1985).

While heuristics have been found to be very effective and efficient in solving a wide range of practical size VRPTW (see Desrochers et al. 1988, and Solomon, Baker and Schaffer 1988), optimal approaches have lagged considerably behind. In the literature, the largest problem solved to optimality involved four vehicles servicing 14 customers with tight time windows (Kolen, Rinnooy Kan and Trienekens 1987).

The contribution of this paper is the development

Subject classifications: Dynamic programming/optimal control: subproblem modeling. Programming, integer: column generation algorithm. Transportation: vehicle routing and scheduling.
Area of review: DISTRIBUTION, TRANSPORTATION AND LOGISTICS.

0030-364X/92/4002-0342 $01.25

of a new optimization algorithm for the VRPTW that is capable of optimally solving problems of a size far larger than any attempted to date in the literature. The paper is organized as follows. Section 1 reviews the literature. In Section 2, a set partitioning formulation for the VRPTW and the column generation approach are presented. Section 3 discusses the design of the subproblem. Section 4 introduces a set covering model. It also discusses the solution to the linear set covering model and the dynamic programming algorithms for the subproblem. The branch-and-bound approach for obtaining integer solutions is described in Section 5. In Section 6 we present the computation experiments. Finally, the last section states our conclusions.

1. LITERATURE REVIEW

We have witnessed significant advances made for different variants of the VRPTW. This fast growing body of research has been surveyed in Solomon and Desrosiers (1988) and Desrochers et al. (1988).

Optimal approaches, using dynamic programming, have been proposed for different variants of the single VRPTW to minimize the total distance traveled. For the traveling salesman problem, Christofides, Mingozzi and Toth (1981b) use state-space relaxations to reduce the number of states and obtain a lower bound. Problems with up to fifty moderately tight-time window constrained customers are considered. Psaraftis (1983) presents an $O(n^2 3^n)$ time algorithm (where n is the number of customers) to minimize a more general objective function, the total customer inconvenience, for the pick-up and delivery problem. The tractable problem size was limited to 8–10 customers. Desrosiers, Dumas and Soumis (1986) present an optimal dynamic programming algorithm that is capable of solving dial-a-ride problems involving up to forty customers (80 nodes) in less than six seconds on a CYBER 173.

Dynamic programming algorithms have also been used with great success to obtain integer optimal solutions to the shortest path problem with time window constraints (SPPTW). Desrosiers, Pelletier and Soumis (1983) developed a generalization of the Ford-Bellman-Moore dynamic programming algorithm approach to the classical shortest path problem. Desrochers and Soumis's (1988a) enhancement of this algorithm solved problems having up to 2,500 nodes and 250,000 arcs in less than one minute on a CYBER 173. Desrochers and Soumis (1988b) also present a primal-dual re-optimization algorithm which is particularly suitable when generating multiple disjoint routes in column generation schemes for VRPTW variants. Finally, Desrochers (1988) generalizes the above primal-dual approach to the shortest path problem with multiple resource constraints while Dumas, Desrosiers and Soumis (1989a) describe a shortest path algorithm for vehicle routing with paired pick-up and delivery stops and time-window constraints.

Column generation approaches for set partitioning formulations of several VRPTW variants have also been presented. In the case of the multitraveling salesman problem, such an approach to minimize a linear combination of the fleet size and total distance traveled is presented in Desrosiers, Soumis and Desrochers (1984). The columns are generated by solving an SPPTW. An updated version of this algorithm has recently solved to optimality problems with over 300 trips. A Lagrangian relaxation method proposed in Desrosiers, Soumis and Sauvé (1988) was successful to optimize the fleet size. The column generation scheme for the multitraveling salesman has been generalized to handle problems with several depots and different vehicle types in the context of dial-a-ride problems in Dumas, Desrosiers and Soumis (1989). A similar column generation approach has also been designed for the pick-up and delivery problem in the context of goods transportation with tight vehicle capacity constraints (Dumas, Desrosiers and Soumis 1991).

The only work that we are aware of on exact methods for the VRPTW is that of Kolen, Rinnooy Kan and Trienekens (1987), Knudsen (1989) and Madsen (1990). Kolen, Rinnooy Kan and Trienekens extend the shortest q-path relaxation algorithm for the vehicle routing problem (Christofides, Mingozzi and Toth 1981b) to the VRPTW. With no branching required to obtain the solution of a four vehicle, 14-customer problem, the algorithm took 0.58 minute of DEC 20/60 CPU time. Knudsen uses Lagrangian relaxation and set partitioning/column generation to solve a 30-customer problem to optimality. Madsen proposes a Lagrangian relaxation for the computation of a lower bound for the VRPTW. Using variable splitting, this relaxation necessitates the integer solution of two types of subproblems: the SPPTW and the generalized assignment problem. The author has reported the optimal solution for a 31-customer problem.

The results presented in this paper have been used in Haouari, Dejax and Desrochers (1990) to model and solve two other vehicle routing problems: the fleet

size and mix vehicle routing problem with time windows and the multidepot vehicle routing problem with time windows.

2. THE SET PARTITIONING MODEL

Let $G = (N, A)$ be a network, where A is the set of route segments and N is the set of nodes or customers. Associated with each arc $(i, j) \in A$ is a cost c_{ij} and a duration t_{ij}. We assume that the service time at customer i is included in the duration of each arc (i, j). In this paper, the cost is taken to be the distance between i and j. The vehicle routing problem with time windows involves the design of a set of minimum cost routes originating and terminating at a central depot, d, for a fleet of vehicles which services a set of customers with known demands q_i. Each customer is serviced exactly once. The service of a customer, involving pick-up (delivery) of goods or services can begin at T_i within the time window defined by the earliest time, a_i, and the latest time, b_i, when the customer will permit the start of service. If a vehicle arrives at a customer too early, it will wait. In addition, due dates cannot be violated. Furthermore, all the customers must be assigned to vehicles such that the vehicle capacities, Q, are not exceeded.

The VRPTW on network G can be formulated as a set partitioning problem. For this, let R be the set of feasible routes for the VRPTW. Also let δ_{ir} be a constant that takes the value 1 if route $r \in R$ visits customer $i \in N\backslash\{d\}$ and 0 otherwise. Define c_r to be the cost of route r. The cost of a route is defined as the sum of the cost of the arcs of the route. Finally, let x_r be a binary variable equaling 1 if route r is used and 0 otherwise. The set partitioning problem selects a minimal cost set of routes satisfying the VRPTW constraints:

$$\min \sum_{r \in R} c_r x_r$$

$$\sum_{r \in R} \delta_{ir} x_r = 1, \quad i \in N\backslash\{d\}$$

$$x_r \in \{0, 1\}, \quad r \in R.$$

The columns represented by the variables correspond to the feasible routes. As their number is extremely large for all but very small sized problems, the set partitioning problem cannot be solved directly, i.e., by approaches involving exhaustive column enumeration; instead, we use a column generation method. The LP relaxation of the set partitioning problem is solved by column generation. Feasible columns are added as needed by solving a subproblem using dynamic programming. The solution obtained generally provides an excellent lower bound that is embedded in a branch-and-bound algorithm to solve the integer set partitioning problem.

3. DESIGNING THE SUBPROBLEM

A good subproblem must be able to efficiently price out all feasible routes. Such an ideal subproblem may not exist, hence, we may have to settle for a less efficient subproblem or a lesser quality lower bound. We will first describe a feasible route and a first dynamic programming model that is able to price out all feasible routes. The solution space of this model only contains feasible routes and the model is very time consuming to solve. We will also present two other dynamic programming models whose solution spaces contain more than the feasible route set but have a pseudopolynomial worst case complexity.

A path in the network G is defined as a sequence of nodes $(i_0, i_1, \ldots, i_K, i_{K+1})$, such that each arc (i_k, i_{k+1}) belongs to A. A path satisfying all the constraints to be described next is called a route. All routes start and end at the depot ($i_0 = i_{K+1} = d$). Let $\bar{c}_{ij}$ be the marginal cost associated with arc $(i, j) \in A$. The value of this marginal cost will be defined later. The marginal cost of a route is defined as the sum of the marginal cost of the arcs of the route. As established earlier, there are capacity and time window constraints on the routes. The sum of the node demands must be less than the vehicle capacity Q. Each node $i \in N$ has time window $[a_i, b_i]$ and a duration t_{ij} is associated with each arc $(i, j) \in A$. Recall that T_i is defined as the time at which service at node i can begin. If we set T_d to 0, we can then compute T_i for any route using the two relations $T_{i_k} + t_{i_k, i_{k+1}} \leq T_{i_{k+1}}$ and $a_{i_k} \leq T_{i_k} \leq b_{i_k}$ for $0 \leq k \leq K$.

The three models considered will be illustrated on the following four-node problem. The vehicle capacity, Q, is equal to six. The demand and time window of each node and the duration of each arc are specified in Figure 1. The costs are unspecified because we are mostly interested in enumerating the feasible solutions in each model.

We define $F_i(S, t)$ as the minimum marginal cost of the partial route going from the depot to node i, visiting *only once* all nodes in set S and ready to leave node i at time t or later. $F_i(S, t)$ can be computed by

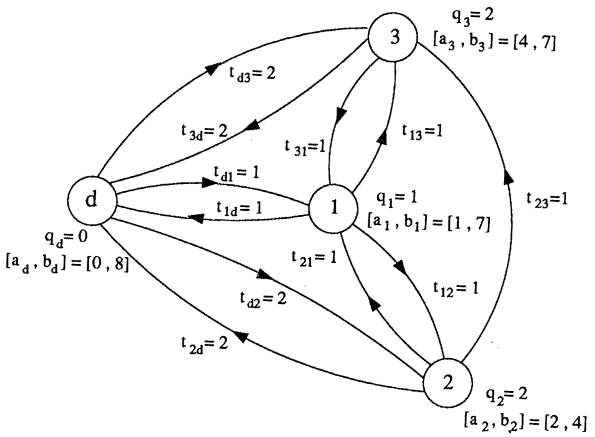

Figure 1. Demand and time window of nodes and arc duration.

solving the recurrence equations:

$$F_d(\phi, 0) = 0;$$

$$F_j(S, t) = \min_{(i,j)\in A} \{F_i(S - \{j\}, t') + \bar{c}_{ij} | t' + t_{ij} \leq t, \; a_i \leq t' \leq b_i \text{ and } \sum_{k\in S} q_k \leq Q\},$$

for all j, S, t such that $j \in N$, $S \subseteq N$, $a_j \leq t \leq b_j$.

A special case of this problem is the shortest weight-constrained path known to be NP-complete (Garey and Johnson 1979). Thus, our dynamic programming problem is NP-hard. Furthermore, no pseudo-polynomial algorithm is known for this problem.

This first model has the set of all feasible routes as a solution space. In our example, there are only eleven feasible routes: $(d, 1, d)$, $(d, 2, d)$, $(d, 3, d)$, $(d, 1, 2, d)$, $(d, 1, 3, d)$, $(d, 2, 1, d)$, $(d, 2, 3, d)$, $(d, 3, 1, d)$, $(d, 1, 2, 3, d)$, $(d, 2, 1, 3, d)$, and $(d, 2, 3, 1, d)$.

Christofides, Mingozzi and Toth (1981a) have proposed a state-space relaxation for dynamic programs in order to obtain easy to compute lower bounds on the value of their objective function. We can apply this technique by defining a mapping of the original state-space (S, t) onto the new state-space (q, t) and by defining correct new transitions. Each state (S, t) is mapped on state $(\sum_{k\in S} q_k, t)$ and $G_i(q, t)$ is the minimum marginal cost of the partial path going from the depot to node i, having accumulated a load q and ready to leave this node at time t or later. The new recurrence equations are:

$$G_d(0, 0) = 0;$$

$$G_j(q, t) = \min_{(i,j)\in A} [G_i(q', t') + \bar{c}_{ij} | t' + t_{ij} \leq t, \; a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q],$$

for all j, q, t such that $j \in N$, $q_j \leq q \leq Q$ and $a_j \leq t \leq b_j$.

This new dynamic programming problem has the multiple knapsack problem as a special case and thus is also NP-hard. However, there are pseudopolynomial algorithms to solve it. One of them is described in Section 4.2.

This second model has a larger solution space than the first model. In our example, there are twenty-two solutions for this model, the previous eleven feasible routes and the following eleven paths: $(d, 1, 2, 1, d)$, $(d, 1, 3, 1, d)$, $(d, 2, 1, 2, d)$, $(d, 3, 1, 3, d)$, $(d, 1, 2, 1, 2, d)$, $(d, 1, 2, 1, 3, d)$, $(d, 1, 2, 3, 1, d)$, $(d, 1, 3, 1, 3, d)$, $(d, 2, 1, 2, d)$, $(d, 2, 1, 3, 1, d)$, and $(d, 3, 1, 3, 1, d)$. Note that most of these nonelementary paths contain a cycle of the form (i, j, i). These cycles are known as 2-cycles. We will now consider a third model in which the 2-cycles are eliminated.

A 2-cycle elimination procedure was first proposed by Houck et al. (1980) for a path relaxation of the traveling salesman problem. The basic idea is to keep two partial paths from the depot to each node i. These are the best and the next best partial paths, respectively. The state space remains the same as in the second model, and $H_i(q, t)$ is the minimum marginal cost of the partial path going from the depot to node i, having accumulated a load q and ready to leave this node at time t or later. Define $p_i(q, t)$ as the predecessor of node i in the path associated with marginal cost $H_i(q, t)$. Let $H_i'(q, t)$ be the marginal cost of the best partial path going from the depot to node i, having accumulated a load q, ready to leave this node at time t or later and not having $p_i(q, t)$ as the last node visited. Obviously, $H_i(q, t) \leq H_i'(q, t)$. The recurrence equations for the subproblem with 2-cycle elimination are:

$$H_d(0, 0) = 0;$$

$$H_j(q, t) = \min_{(i,j)\in A} \{[H_i(q', t') + \bar{c}_{ij} | j \neq p_i(q', t'), \; t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q], \; [H_i'(q', t') + \bar{c}_{ij} | j = p_i(q', t'), \; t' + t_{ij} \leq t, a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q]\},$$

for all j, q, t such that $j \in N$, $q_j \leq q \leq Q$ and $a_j \leq t \leq b_j$.

$$H_j'(q, t) = \min_{(i,j)\in A} \{[H_i(q', t') + \bar{c}_{ij} | j \neq p_i(q', t'), \; i \neq p_j(q, t), \; t' + t_{ij} \leq t, \; a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q], \; [H_i'(q', t') + \bar{c}_{ij} | j = p_i(q', t'), \; i \neq p_j(q, t), \; t' + t_{ij} \leq t, \; a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q]\},$$

for all j, q, t such that $j \in N$, $q_j \leq q \leq Q$ and $a_j \leq t \leq b_j$.

This third dynamic programming problem also has the multiple knapsack problem as a special case and thus is NP-hard. In this case also, there are pseudo-polynomial algorithms to solve it. One of them is described in Section 4.2. This third model also has a larger solution space than the first model. However, it is smaller than the solution space of the second model. In our example, there are twelve solutions for this model, the eleven feasible routes and the path: $(d, 1, 2, 3, 1, d)$.

For the path relaxation of the TSP, Houck et al. have shown that for an n node problem, the ratio of the number of feasible paths with 2-cycles to the number of feasible paths without 2-cycles is $(1 + 1/(n-3))^{n-3}$. This ratio is worth approximately $e = 2.718$ for n large enough. Of course, this result cannot be extended to our case, but it is indicative of the reduction in the number of feasible paths in a model when the 2-cycles are eliminated.

An interesting parallel can be made between the bounding procedures utilized in our method versus those of Kolen, Rinnooy Kan and Trienekens (1987). The third dynamic programming model described above is similar to the first level of Kolen, Rinnooy Kan and Trienekens' two-level bounding mechanism. However, the second level of our model is the linear relaxation of a set partitioning problem, while theirs is a second state-space relaxation. This state-space relaxation is essentially a row agregation, where each column of the set partitioning model is reduced to the respective route demand. Hence, this agregation is equivalent to a knapsack constraint.

4. A SET COVERING TYPE MODEL

The VRPTW has been formulated as a set partitioning problem in Section 2. However, for implementation reasons, we are not going to use this model directly. Instead, for column generation we are going to use a set covering type model because the subproblem can generate routes containing cycles. Furthermore, the linear relaxation of the set covering type model is numerically far more stable than that of the set partitioning model. The solution derived from this set covering type model is made feasible for the set partitioning model by using the branch-and-bound scheme described in Section 5.

To introduce the set covering type model, let γ_{ir} be a constant taking an integer value if route $r \in R$ visits customer $i \in N \setminus \{d\}$ and 0 otherwise. The constant γ_{ir} indicates that a customer i can be visited more than once by route r. Let c_r be the cost of route r and take x_r as a binary variable equaling 1 if route r is used and 0 otherwise. Finally, let X_d and X_c be two additional integer variables. The variable X_d is defined as the number of routes, while the variable X_c represents the total distance traveled. Note that X_c is integer only if c_r is integer for all r, $r \in R$. We satisfy this condition by taking c_{ij} to be integer for $(i, j) \in A$. We can now present our set covering type model formally. The mathematical formulation is:

$$\min \sum_{r\in R} c_r x_r \tag{1}$$

$$\sum_{r\in R} \gamma_{ir} x_r \geq 1, \; i \in N\setminus\{d\}, \tag{2}$$

$$\sum_{r\in R} x_r - X_d = 0, \tag{3}$$

$$\sum_{r\in R} c_r x_r - X_c = 0, \tag{4}$$

$$x_r \in \{0, 1\}, \quad r \in R, \tag{5}$$

$$X_d, X_c \geq 0, \text{ integer.} \tag{6}$$

The objective function and constraints (2) and (5) form a set covering problem which selects a minimal cost (distance) set of routes such that for each customer i there is at least one route visiting that customer. Constraints (3) and (4) ensure that the number of routes and the total distance traveled, respectively, are integers.

Optimizing the LP using the current columns constitutes the first phase of the column generation procedure. In the second phase, a subproblem is solved to find the minimal marginal cost column. If a column with negative marginal cost is found, this variable is added to the known ones and we return to the first phase. Otherwise, the current solution is optimal. Section 4.1 describes the first phase of the procedure, while Section 4.2 describes the second phase.

4.1. The Linear Set Covering Solution

The simplex method is used to solve the linear relaxation of the set covering type problem. The XMP code of Marsten (1981) is used for this purpose. Initially, the problem contains $|N| - 1$ columns (an identity matrix), i.e., one vehicle for each customer.

The simplex method gives the dual variables π_i, $i \in N\backslash\{d\}$, π_d and π_c associated with constraints (2), (3) and (4), respectively, necessary for the solution to the subproblem. It also enables easy reoptimization each time new columns are generated from the subproblem. Therefore, from linear programming theory, we obtain the marginal cost $\bar{c}_r$ of a route r that is given by:

$$\bar{c}_r = c_r - \sum_{i \in N\backslash\{d\}} \pi_i \gamma_{ir} - \pi_d - \pi_c c_r.$$

Since the cost of a route is defined as $\sum_{k=0}^{K} c_{i_k,i_{k+1}}$ for a route $(i_0, i_1, \ldots, i_K, i_{K+1})$, $\bar{c}_r$ becomes

$$\bar{c}_r = \sum_{k=0}^{K} c_{i_k,i_{k+1}} - \sum_{k=1}^{K} \pi_{i_k} - \pi_d - \pi_c \sum_{k=0}^{K} c_{i_k,i_{k+1}},$$

$$\bar{c}_r = \sum_{k=0}^{K} [(1 - \pi_c)c_{i_k,i_{k+1}} - \pi_{i_k}] \quad \text{as } i_0 = d.$$

Therefore, we can define the marginal cost $\bar{c}_{ij}$ of an arc (i, j) as

$$\bar{c}_{ij} = (1 - \pi_c)c_{ij} - \pi_i, \text{ for all } (i, j) \in A.$$

Solving the LP relaxation is accelerated by generating several columns simultaneously. This is possible as the one-time solution of a subproblem by dynamic programming not only produces the minimum marginal cost column but also many other columns of negative marginal cost. The routes (columns) selected to be added to the LP relaxation are almost disjoint as this enhances the efficient discovery of integer solutions. Furthermore, as set covering characterizations of vehicle routing applications exhibit high degeneracy, we further accelerate the convergence of the simplex method by a perturbation strategy on the right-hand side of constraints (2).

4.2 Solving the Subproblem

A primal-dual algorithm developed for the resolution of the shortest path problem with resource constraints (Desrochers 1988) computes the cost $G_j(q, t)$ (or $H_j(q, t)$ and $H'_j(q, t)$) by progressive refinement of lower and upper bounds on its value. The algorithm requires the creation of two sets of labels at each node. The first set includes the labels associated with feasible paths and defines primal solutions which provide an upper bound on the cost of efficient solutions at each node. The second set includes the labels associated with lower bounds on the cost of a path ending at node j with a given state value. The algorithm modifies the sets of labels until the optimal solution is obtained. For a set S of states at node j, labels are created through a "pulling" process. Labels associated with feasible paths from the origin to node j are obtained by extending all feasible paths from the origin to node i for which the addition of arc (i, j) allows arrival at node j in a state belonging to S. All the new labels at a given iteration are created at a single node, in contrast with other dynamic programming approaches requiring updating at several nodes per iteration. With proper implementation, this method has a worst case complexity $O(Q^2(\sum_{i \in N}(b_i + 1 - a_i))^2)$.

We first present a pulling algorithm used to solve the second model described in Section 3. Let $\underline{G}_j(q, t)$ and $\bar{G}_j(q, t)$ be a lower bound and an upper bound on $G_j(q, t)$, respectively. P_j is defined as the set of all states at node j having equal lower and upper bounds. $\bar{P}_j$ is the complement of P_j, i.e., the set of all states at node j such that $\underline{G}_j(q, t) < \bar{G}_j(q, t)$.

Definition. *Given* (q_1, t_1) *and* $(q_2, t_2) \in \mathbb{R}^2$, (q_1, t_1) *is said to be lexicographically smaller than* (q_2, t_2) *if and only if* $q_1 < q_2$, *or* $q_1 = q_2$ *and* $t_1 < t_2$.

The Pulling Algorithm

Step 1. **Initialization.** Initialize the lower bounds $\underline{G}_j(q, t) = -\infty$, the upper bounds $\bar{G}_j(q, t) = \infty$ for all j, q and t such that $j \in N\backslash\{d\}$, $0 \leq q \leq Q$, and $a_j \leq t \leq b_j$. Initialize also $\bar{G}_d(0, 0) = \underline{G}_d(0, 0) = 0$. Finally, initialize the sets $P_j = \emptyset$, $j \in N\backslash\{d\}$ and $P_d = \{(0, 0)\}$.

Step 2. **Search for a state (q, t) to be treated.** Find the state (q, t) of minimum lexicographic value from the set $W = \cup_{j \in N}(\bar{P}_j)$. If $W = \emptyset$, stop.

Step 3. **Treatment of state (q, t) of node j.** Compute new lower and upper bounds:

$$\underline{G}_j(q, t) = \min_{(i,j) \in A} \{\underline{G}_i(q', t') + \bar{c}_{ij} | t' + t_{ij} \leq t, \; a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q\}.$$

$$\bar{G}_j(q, t) = \min_{(i,j) \in A} \{\bar{G}_i(q', t') + \bar{c}_{ij} | t' + t_{ij} \leq t, \; a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q\}.$$

Update set P_j. Return to Step 2.

We now describe a pulling algorithm with 2-cycle elimination used to solve the third model of Section 3. Let $\underline{H}_j(q, t)$ and $\bar{H}_j(q, t)$ be a lower bound and an upper bound on $H_j(q, t)$, respectively.

$\bar{H}'_j(q, t)$ is an upper bound on $H'_j(q, t)$. P_j is defined as the set of all states at node j having equal lower and upper bounds. $\bar{P}_j$ is the complement of P_j; i.e., the set of all states at node j such that $\underline{H}_j(q, t) < \bar{H}_j(q, t)$.

The Pulling Algorithm With 2-Cycle Elimination

Step 1. **Initialization.** Initialize the lower bounds $\underline{H}_j(q, t) = -\infty$, the upper bounds $\bar{H}_j(q, t) = \infty$, $\bar{H}'_j(q, t) = \infty$, and the predecessors $p_j(q, t) =$ nil, for all j, q and t such that $j \in N\backslash\{d\}$, $0 \leq q \leq Q$, and $a_j \leq t \leq b_j$. Initialize also $\bar{H}_d(0, 0) = \underline{H}_d(0, 0) = 0$. Finally, initialize the sets $P_j = \varnothing$, $j \in N\backslash\{d\}$ and $P_d = \{(0, 0)\}$.

Step 2. **Search for a state (q, t) to be treated.** Find the state (q, t) of minimum lexicographic value from set $W = \cup_{j\in N}(\bar{P}_j)$. If $W = \varnothing$, stop.

Step 3. **Treatment of state (q, t) of node j.** Compute new lower and upper bounds:

$$\begin{aligned}\bar{H}_j(q, t) = \min_{(i,j)\in A} \{&[\bar{H}_i(q', t') + \bar{c}_{ij} | j \neq p_i(q', t'),\\ &t' + t_{ij} \leq t,\ a_i \leq t' \leq b_i,\\ &a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q],\\ &[\bar{H}'_i(q', t') + \bar{c}_{ij} | j = p_i(q', t'),\\ &t' + t_{ij} \leq t,\ a_i \leq t' \leq b_i,\\ &a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q]\}.\end{aligned}$$

Set the predecessor $p_j(q, t)$.

$$\begin{aligned}\bar{H}'_j(q,t) = \min_{(i,j)\in A} \{&[\bar{H}_i(q',t') + \bar{c}_{ij} | j \neq p_i(q',t),\\ &i \neq p_j(q,t), t' + t_{ij} \leq t,\\ &a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q],\\ &[\bar{H}'_i(q',t') + \bar{c}_{ij} | j = p_i(q',t'),\\ &i \neq p_j(q,t), t' + t_{ij} \leq t, a_i \leq t' \leq b_i\\ &\text{and } q' + q_j \leq q]\},\end{aligned}$$

$$\begin{aligned}\underline{H}_j(q, t) = \min_{(i,j)\in A} \{&\underline{H}_i(q', t') + \bar{c}_{ij} | t' + t_{ij} \leq t,\\ &a_i \leq t' \leq b_i \text{ and } q' + q_j \leq q\}.\end{aligned}$$

Update set P_j. Return to Step 2.

5. THE BRANCH-AND-BOUND SCHEME

When the subproblem does not generate any more negative marginal cost columns, the simplex algorithm provides the optimal solution of the linear relaxation of the set covering type formulation. If the solution is integer and each customer is covered exactly once, the solution is also optimal for the VRPTW. Otherwise, the linear relaxation of the set covering type model may be fractional or some customers may be covered more than once. In either case, a branch-and-bound tree must be explored, and additional columns might be generated at each branch. In practice, however, overcovering has never arose. Furthermore, this condition is guaranteed not to occur if the cost matrix satisfies the strict triangle inequality. The design of the branch-and-bound strategies is of extreme importance in a column generation scheme because branching decisions are taken with reference to the fractional solutions of the generated columns, and these decisions must be compatible with the subproblem structure for the generation of new columns.

The branching strategies designed for the VRPTW can be separated into two categories. At the first level, the integrality of the number of vehicles used and the total distance traveled is required. This branching level involves all the columns of the set covering formulation. At the next level, branching decisions are taken locally on the arcs of the network. This second branching level only involves very few columns.

At the first level, if the number of vehicles used is fractional, say $X_d = v$, two branches are created: one with $X_d \leq \lfloor v \rfloor$ and the other with $X_d \geq \lceil v \rceil$. In each case, the dual variable π_d associated with constraint (3) is adequately transferred to the subproblem. Next, in each branch, if the solution is still fractional, it is possible to add a cut on the total distance traveled (which is minimized in the objective function). In fact, if X_c is fractional, say $X_c = c$, the cut on the total distance traveled is equivalent to restricting X_c to be greater or equal to $\lceil c \rceil$. The dual variable, π_c, associated with constraint (4) is also easily transferred to the subproblem. This first level process is repeated everywhere it is needed in the branch-and-bound tree. Each time the branching strategy on the number of vehicles or the cut on the total distance is utilized, the current solution becomes infeasible. Reoptimization may be carried out with Phases I and II of the simplex algorithm or with the dual simplex algorithm.

At the second level, branching decisions are taken on the arcs of the subproblem network. Note that it is possible to fix a fractional variable x_r at one. This information is easily transferred to the subproblem by fixing the value of the flow on each arc of the route at one, that is, by removing from the network the other arcs incident to the nodes visited by route r. However, it is impossible to directly fix variable x_r at zero. Rather, it is necessary to explore many combinations of the flow values of the arcs of the route. Such a strategy is described in Desrosiers, Soumis and

Desrochers. In the actual solution procedure for the VRPTW, we have chosen a single arc branching strategy. We next describe this strategy.

Columns that contain cycles are chosen first. For each column, a score is given to the arcs incident to nodes visited more than once (e.g., four arcs for a node visited twice). The score of an arc is a function of the total flow value on that arc, that is, a combination of the flow from all the nonzero basic variables. The arc with the best score is chosen and two branches are created, at 0 and 1, respectively. If there exists no columns that contain cycles, fractional variables are next examined. Some of them are discarded if each arc of the column has a total flow value of one. For the others, a score (which depends on the variable value) is calculated. Next, for the set of columns with good scores, each arc is evaluated by a score (which depends on the combined flow value on the arc). The arc with the best score is chosen and two 0–1 branches are created.

Finally, if arc (i, j) is fixed at 0, it is simply removed from the subproblem network. The cost of all columns generated that use that arc are penalized. Then, the solution of the set covering type formulation is still feasible but at a very high cost. After optimization over the current columns, new columns are generated as needed. If arc (i, j) is fixed at 1, arcs $(i, k) \in A$, $k \neq j$ and $(l, j) \in A$, $l \neq j$ are removed from the subproblem network. Moreover, columns that use arcs $(i, k) \in A$, $k \neq j$ or arcs $(l, j) \in A$, $l \neq i$ are also penalized and new columns are generated as needed.

6. COMPUTATIONAL EXPERIMENTS

6.1. Time Window Reduction

As stated in Section 4.2, the worst case complexity of the dynamic programming algorithms is

$$O(Q^2(\textstyle\sum_{i\in N}(b_i + 1 - a_i))^2).$$

Hence, in this case, complexity is a function of time window width and vehicle capacity. To improve the efficiency of these algorithms, we reduce the time windows' width using four conditions:

1. minimal arrival time from predecessors:
   $a_k = \max\{a_k, \min\{b_k, \min_{(i,k)\in A}\{a_i + t_{ik}\}\}\}$;
2. minimal arrival time to successors:
   $a_k = \max\{a_k, \min\{k_k, \min_{(k,j)\in A}\{a_j - t_{kj}\}\}\}$;
3. maximal departure time from predecessors:
   $b_k = \min\{b_k, \max\{a_k, \max_{(i,k)\in A}\{b_i + t_{ik}\}\}\}$;
4. maximal departure time to successors:
   $b_k = \min\{b_k, \max\{a_k, \max_{k,j)\in A}\{b_j - t_{kj}\}\}\}$.

The above conditions are applied sequentially at each node. The set of nodes is examined cyclically until no further reductions are possible. Generally, two or three cycles are sufficient. At the end of this process, some arcs can be eliminated. Conditions 2 and 3 were derived independently by Cyrus (1988). In some cases, similar conditions may be applied to the capacity constraints at some nodes. These time conditions will be now illustrated on a small network having four nodes and three arcs.

In Figure 2, we compute the earliest arrival time at each node. At nodes 2 and 3, the earliest arrival time is greater than the start of the time window, thus, we can increase the time window start by one unit at these nodes. In Figure 3, we compute the earliest departure time from each node inducing no waiting time at its successors. At node 3, the latest such departure time is greater than the start of the time window, so we can increase the window start by two units at this node. In Figure 4, the latest arrival time

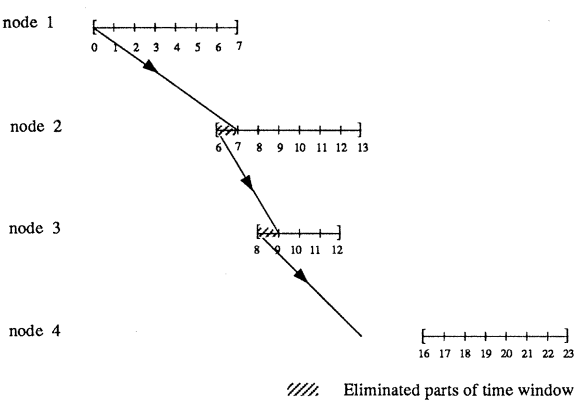

Figure 2. Example of time window reduction using rule 1.

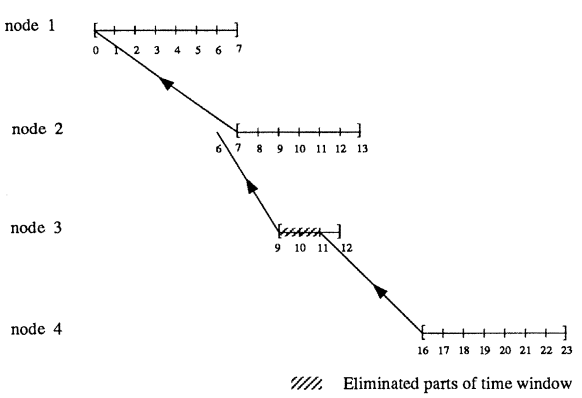

Figure 3. Example of time window reduction using rule 2.

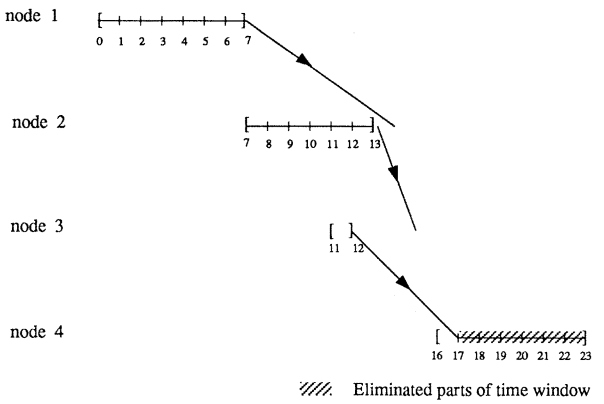

Figure 4. Example of time window reduction using rule 3.

at each node is computed. It is impossible to arrive at node 4 after time value 17, and we can reduce the time window end by six units at this node. In Figure 5, we compute the latest departure time of each node that is legal for all its successors. At nodes 1 and 2, the latest such departure time is smaller than the time window end. We can reduce the time window end by one unit at node 1 and by four units at node 2. Note that further reductions are still possible in this example.

6.2. Description of the Test Problems

To conduct the computational experiments we have used three of the six benchmark problem sets developed by Solomon (1987). The actual data are available from the authors. The geographical data are randomly generated by a random uniform distribution in problem set R1, clustered in problem set C1, and a mix of randomly generated data and clusters in problem set RC1. All the test problems are 100-customer euclidean problems. Travel times between customers are equal to the corresponding distances truncated to one decimal place. All values are then multiplied by 10 to obtain integers. The customer demands are relatively small compared to the vehicle capacity. These problems have a short scheduling horizon. In problem sets R1 and RC1 the time windows have a uniformly distributed, randomly generated center and a normally distributed random width. The time windows' width is then reduced in a preprocessing step, as illustrated in the previous section. In problem set C1, however, the time windows are positioned around the arrival times at customers of a 3-opt, cluster-by-cluster routing solution. The time windows' width is derived and reduced as above. Depending on the problem, the density of the time windows is 25, 50, 75, or 100%. From a given problem in sets R1, C1, or RC1, we have generated additional test problems by considering only the first 25 or the first 50 customers. Therefore, a total of 87 problems were tested.

6.3. Computational Results

To analyze the behavior of the VRPTW optimization algorithm described in the previous sections, this algorithm was programmed in FORTRAN using the GENCOL software for column generation (Sansó et al. 1990). The algorithm was compiled using the F77 compiler (optimize option) and run on a SUN SPARK 1 workstation. Two versions of the algorithm were implemented. They differ in the respect that the second version involves the elimination of 2-cycles. We present only the computational results for the version with 2-cycle elimination which performs much better than the other. An initial solution is obtained by assigning one vehicle per customer. Computational results are presented in Tables I–III. In these tables, for each problem presented in column one, the next two columns describe the network size after preprocessing (the number of nodes and arcs). Columns four, five and six highlight the quality of the solution, i.e., the LP bound, the value of the optimal integer solution divided by 10, and the number of vehicles used. Finally, some characteristics of the column generation and the branch-and-bound procedures (the number of nodes explored, the number of columns generated and total CPU time in seconds) are presented in columns 7–9.

As can be seen from Tables I–III, the algorithm was able to optimally solve 29 problems involving 25 customers out of the 29 tested, 14 problems with 50 customers and 7 problems with 100 customers.

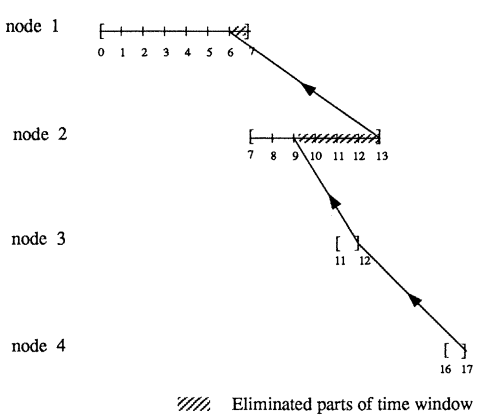

Figure 5. Example of time window reduction using rule 4.

Table I
Computational Results on Problem Set R1

Problem	Nodes	Arcs	LP Solution	IP Solution	Vehicles	B-B Nodes Explored	Columns Generated	CPU Time
R101	25	227	617.1	617.1	8	1	78	5.8
	50	817	1034.6	1035.2	12	11	1395	66.7
	100	3272	1604.5	1607.7	18	23	8596	1064.2
R102	25	404	546.4	547.1	7	3	165	20.3
	50	1437	904.6	904.6	11	1	591	67.8
	100	5336	1434.0	1434.0	17	1	2126	756.9
R103	25	510	454.6	454.6	5	1	246	22.2
	50	1859	765.4	772.5	9	173	37330	8939.1
	100							
R104	25	559	416.9	416.9	4	1	419	46.0
	50							
	100							
R105	25	303	526.0	530.5	6	7	554	22.6
	50	1076	891.7	899.3	9	33	4458	362.6
	100							
R106	25	458	457.3	467.4	5	15	2451	205.2
	50	1642	783.3	785.2	8	5	1688	386.4
	100							
R107	25	542	423.0	424.3	4	21	3191	304.1
	50	1995	703.2	711.1	7	79	19816	7362.1
	100							
R108	25	586	396.2	397.2	4	3	1041	307.4
	50							
	100							
R109	25	411	441.3	441.3	5	1	207	14.4
	50							
	100							
R110	25	530	425.4	429.5	4	5	655	64.3
	50	2026	692.4	697.0	7	53	15341	4906.1
	100							
R111	25	526	423.5	428.8	4	15	2794	330.0
	50							
	100							
R112	25	647	383.9	393.0	4	21	2643	623.3
	50							
	100							

The method proved most successful on the clustered data (problem set C1) where it was able to optimally solve 20 of the 27 problems including 5 out of the 9 problems with 100 customers. Two additional 100-customer problems were solved optimally on R1. The problem set RC1 proved difficult to solve. We conjecture that most of this difficulty could be removed by employing a more sophisticated branching scheme. We elected not to include computational results for the problems not optimally solved. The empty lines are there as a challenge for future research. The reader can also observe that problems with an average of up to 10 customers per vehicle were solved in the numerical experiment.

The results also reveal that the LP relaxation of the set covering type model provides an excellent primal lower bound which in turn allows the efficient derivation of an optimal solution by branch and bound. For 27 problems out of the 87 attempted, the lower bound is equal to the optimal value. For the other problems, the average integrality gap is 1.5% (the maximum gap is 12.1%.) Therefore, solving the subproblem with integer variables results in a small integrality gap, allowing the implicit exploration of a part of the integrality gap of the set covering type model.

In terms of computational time, the width of the time windows relative to the scheduling horizon and their density had a strong effect on the algorithm. The column generation method was more efficient for the more tightly constrained problems and those with higher time window density. In these cases, the subproblems generating feasible routes are easy to solve

Table II
Computational Results on Problem Set C1

Problem	Nodes	Arcs	LP Solution	IP Solution	Vehicles	B-B Nodes Explored	Columns Generated	CPU Time
C101	25	332	191.3	191.3	3	1	498	18.6
	50	1217	362.4	362.4	5	1	870	67.1
	100	4515	827.3	827.3	10	1	3768	434.5
C102	25	473	190.3	190.3	3	1	534	79.7
	50	1732	361.4	361.4	5	1	1554	330.2
	100	6581	827.3	827.3	10	1	4038	1990.8
C103	25	569	190.3	190.3	3	1	447	134.7
	50	2165	361.4	361.4	4	1	1769	896.0
	100							
C104	25	604	186.9	186.9	3	1	485	223.9
	50							
	100							
C105	25	357	191.3	191.3	3	1	531	25.6
	50	1333	362.4	362.4	5	1	1142	99.1
	100							
C106	25	336	191.3	191.3	3	1	447	20.7
	50	1272	362.4	362.4	5	1	1170	91.3
	100	5423	827.3	827.3	10	1	3681	724.8
C107	25	377	191.3	191.3	3	1	514	31.7
	50	1443	362.4	362.4	5	1	1725	170.6
	100	5621	827.3	827.3	10	1	4970	1010.4
C108	25	431	191.3	191.3	3	1	513	43.1
	50	1643	362.4	362.4	5	1	1605	245.6
	100	6344	827.3	827.3	10	1	4989	1613.6
C109	25	482	189.4	191.3	3	3	1747	585.4
	50							
	100							

Table III
Computational Results on Problem Set RC1

Problem	Nodes	Arcs	LP Solution	IP Solution	Vehicles	B-B Nodes Explored	Columns Generated	CPU Time
RC101	25	277	405.1	461.1	4	99	6399	225.4
	50							
	100							
RC102	25	408	346.0	346.0	3	1	255	18.1
	50							
	100							
RC103	25	518	332.1	332.8	3	5	953	103.0
	50							
	100							
RC104	25	564	305.9	306.6	3	6	1273	177.9
	50							
	100							
RC105	25	372	411.0	411.3	4	5	488	37.4
	50							
	100							
RC106	25	388	338.3	345.5	3	27	2064	248.4
	50							
	100							
RC107	25	513	293.3	298.3	3	3	473	113.9
	50							
	100							
RC108	25	617	280.3	294.5	3	11	1126	256.0
	50							
	100							

by dynamic programming since the sets of feasible states are relatively small. Furthermore, even in such cases, the LP relaxation provides excellent bounds. In contrast, the bounds produced by other relaxations generally deteriorate with increases in the tightness of the time window constraints. Finally, the number of columns generated is quite large since many negative marginal cost columns are introduced in the model each time the subproblem is solved. This column generating procedure reduces the total CPU time.

7. CONCLUSIONS

In this paper, we have presented the development of a new optimization algorithm which uses a column generation approach for a set partitioning formulation for the VRPTW. Our results indicate that this algorithm proved to be very successful on a variety of practical sized benchmark VRPTW test problems. The algorithm was capable of optimally solving problems of a size six times larger than any reported to date by other published research. We consider the performance of our method on the test problems indicative of its performance in general.

Many optimal approaches proposed to date for simpler variants of the VRPTW suffer from the drawback that the size of the problems they are capable of solving decreases dramatically with increases in the size or complexity of the constraints. The column generation approach, however, is capable of solving much larger problems with a greater number of complex constraints by incorporating the hardest constraints at key points of the solution approach. This approach is robust as the LP relaxation gives very good lower bounds even when the time windows are widened or their number decreased. This approach is also very flexible as near-optimal solutions can be obtained to much larger problems by early termination of the algorithm. Note, however, that the set partitioning model (solved either directly or by column generation) stops being competitive in environments involving many customers to be visited on the same route. This is because, in such cases, the density of the LP increases and degeneracy becomes a problem.

In addition to removing some of the computational barriers that existed to date for the generic VRPTW, our results will also lead to further standardization of the computational experiment to be conducted in this field. The optimal solutions to realistic sized problems obtained here can be used to benchmark the effectiveness of future approaches.

The approach presented in this paper can easily be extended to even more difficult problem variants. Multiple time windows per customer can be modeled by using as many nodes as time windows for each customer. Only one node will be visited among the nodes associated with a given customer. Two other extensions have been solved in Haouari, Dejax and Desrochers (1990). These are the heterogeneous fleet problem and the multiple depot problem.

ACKNOWLEDGMENT

We thank Ms. Sylvie Gélinas who programmed the 2-cycle elimination and Ms. Margaret Maclure who assembled the various column generation codes and performed the computational study. The two first authors were supported by grants from the Canadian Government (NSERC) and the Québec Government (FCAR).

REFERENCES

BODIN, L., B. GOLDEN, A. ASSAD AND M. BALL. 1983. Routing and Scheduling of Vehicles and Crews: The State of the Art. *Computs. and Opns. Res.* **10,** 62–212.

CHRISTOFIDES, N., A. MINGOZZI AND P. TOTH. 1981a. State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems. *Networks* **11,** 145–164.

CHRISTOFIDES, N., A. MINGOZZI AND P. TOTH. 1981b. Exact Algorithms for the Vehicle Routing Problem Based on the Spanning Tree and Shortest Path Relaxations. *Math. Prog.* **20,** 255–282.

CYRUS, J. P. 1988. The Vehicle Scheduling Problem: Models, Complexity and Algorithms. Ph.D. Thesis, Technical University of Nova Scotia.

DESROCHERS, M. 1988. An Algorithm for the Shortest Path Problem With Resource Constraints. Cahiers du GÉRAD G-88-27, École des H.E.C., Montreal, Canada.

DESROCHERS, M., AND F. SOUMIS. 1988a. A Generalized Permanent Labeling Algorithm for the Shortest Path Problem With Time Windows. *INFOR* **26,** 191–212.

DESROCHERS, M., AND F. SOUMIS. 1988b. A Reoptimization Algorithm for the Shortest Path Problem With Time Windows. *Eur. J. Opnl. Res.* **35,** 242–254.

DESROCHERS, M., J. K. LENSTRA, M. W. P. SAVELSBERGH AND F. SOUMIS. 1988. Vehicle Routing With Time Windows: Optimization and Approximation. In *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad (eds.). North-Holland, Amsterdam, 65–84.

DESROSIERS, J., Y. DUMAS AND F. SOUMIS. 1986. A Dynamic Programming Solution of the Large-Scale

Single-Vehicle Dial-a-Ride Problem With Time Windows. *Am. J. Math. and Mgmt. Sci.* **6,** 301–325.

DESROSIERS, J., J. PELLETIER AND F. SOUMIS. 1983. Plus Court Chemin Avec Contraintes d'Horaires. *RAIRO* **17,** 357–377 (in French).

DESROSIERS, J., F. SOUMIS AND M. DESROCHERS. 1984. Routing With Time Windows by Column Generation. *Networks* **14,** 545–565.

DESROSIERS, J., F. SOUMIS AND M. SAUVÉ. 1988. Lagrangian Relaxation Methods for Solving the Minimum Fleet Size Multiple Traveling Salesman Problem With Time Windows. *Mgmt. Sci.* **34,** 1005–1022.

DUMAS, Y., J. DESROSIERS AND F. SOUMIS. 1989. Large Scale Multi-Vehicle Dial-a-Ride Problems. Cahiers du GÉRAD G-89-30, École des H.E.C., Montreal, Canada.

DUMAS, Y., J. DESROSIERS AND F. SOUMIS. 1991. The Pick-up and Delivery Problem With Time Windows. *Eur. J. Opnl. Res.* **54,** 7–22.

GAREY, M. R., AND D. S. JOHNSON. 1979. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

HAOUARI, M., P. DEJAX AND M. DESROCHERS. 1990. Modelling and Solving Complex Vehicle Routing Problems Using Column Generation. Working Paper, LEIS, École Centrale, Paris.

HOUCK JR., D. J., J.-C. PICARD, M. QUEYRANNE AND R. R. VEMUGANTI. 1980. The Travelling Salesman Problem as a Constrained Shortest Path Problem: Theory and Computational Experience. *Opsearch* **17,** 93–109.

KNUDSEN, K. G. 1989. Routing With Time Windows. Master Thesis, Aarhus University, Denmark.

KOLEN, A. W. J., A. H. G. RINNOOY KAN AND H. W. J. M. TRIENEKENS. 1987. Vehicle Routing With Time Windows. *Opns. Res.* **35,** 266–273.

LAPORTE, G., AND Y. NOBERT. 1987. Exact Algorithms for the Vehicle Routing Problem. *Ann. Discr. Math.* **31,** 147–184.

MAGNANTI, T. 1981. Combinatorial Optimization and Vehicle Fleet Planning: Perspectives and Prospects. *Networks* **11,** 179–214.

MADSEN, O. B. G. 1990. Lagrangean Relaxation and Vehicle Routing. IMSOR, The Technical University of Denmark.

MARSTEN, R. E. 1981. The Design of the XMP Linear Programming Library. *ACM Trans. Math. Software* **7,** 481–497.

PSARAFTIS, H. 1983. An Exact Algorithm for the Single-Vehicle Many-to-Many Dial-A-Ride Problem With Time Windows. *Trans. Sci.* **17,** 351–357.

SANSÓ, B., M. DESROCHERS, J. DESROSIERS, Y. DUMAS AND F. SOUMIS. 1990. Modeling and Solving Routing and Scheduling Problems: GENCOL User Guide. GERAD, 5255 Decelles, Montreal, Canada.

SAVELSBERGH, M. W. P. 1985. Local Search in Routing Problems With Time Windows. *Ann. Opns. Res.* **4,** 285–305.

SOLOMON, M. M. 1987. Algorithms for the Vehicle Routing and Scheduling Problem With Time Window Constraints. *Opns. Res.* **35,** 254–265.

SOLOMON, M. M., AND J. DESROSIERS. 1988. Time Window Constrained Routing and Scheduling Problems. *Trans. Sci.* **22,** 1–13.

SOLOMON, M. M., E. BAKER AND J. SCHAFFER. 1988. Vehicle Routing and Scheduling Problems With Time Window Constraints: Efficient Implementations of Solution Improvement Procedures. In *Vehicle Routing: Methods and Studies,* B. L. Golden and A. A. Assad (eds.). North-Holland, Amsterdam, 85–105.