A parallel cutting-plane algorithm for the vehicle routing problem with time windows *

William Cook and Jennifer L. Rich
Computational and Applied Mathematics
Rice University

May 28, 1999

Abstract

In the vehicle routing problem with time windows a number of identical vehicles must be routed to and from a depot to cover a given set of customers, each of whom has a specified time interval indicating when they are available for service. Each customer also has a known demand, and a vehicle may only serve the customers on a route if the total demand does not exceed the capacity of the vehicle. The most effective solution method proposed to date for this problem is due to Kohl, Desrosiers, Madsen, Solomon, and Soumis. Their algorithm uses a cutting-plane approach followed by a branch-and-bound search with column generation, where the columns of the LP relaxation represent routes of individual vehicles. We describe a new implementation of their method, using Karger's randomized minimum-cut algorithm to generate cutting planes. The standard benchmark in this area is a set of 87 problem instances generated in 1984 by M. Solomon; making using of parallel processing in both the cutting-plane generation and the branch-and-bound search, we solve 80 of these examples, including 10 that were previously unsolved in the literature. Our parallel implementation utilizes the *TreadMarks* distributed shared memory system.

A common element in many applications of vehicle routing is the notion of a time-window to specify that a customer's service must begin within a fixed period. These windows may arise, for example, from commitments made to customers or from the limited availability of certain resources. Applied case studies involving time-constrained routing can be found in Caseau and Koppstein [9], Cheng and Rich [8], Knight and Hoffer [24], Pullen and Webb [30], and elsewhere. Research efforts in this area have focused on a variety of simplified models that can be used as building blocks for the creation of tools to solve real-world instances. A survey of this work can be found in Desrosiers, Dumas, Solomon, and Soumis [15].

In our study we treat the *vehicle routing problem with time windows* (VRPTW) model that was proposed by Solomon [32]. In this model a number of identical vehicles must be

*This work was supported by ONR Grant N00014-98-1-0014 and by Texas ATP Grant 97-3604-010.

routed to and from a depot to serve a given set of customers. There are known travel times and travel costs between each pair of customers and between the customers and the depot. Each of the customers has a known demand for service along with a time interval specifying when its service can begin. The sum of the demands along each route must not exceed the capacity of a vehicle and the vehicles must arrive at the customers within the specified time windows. The objective is to find a solution that covers each of the customers and has minimum total travel cost. This model is studied in Bramel and Simchi-Levi [7], Desrochers, Desrosiers and Solomon [13], Halse [20], Kohl [25], Kohl, Desrosiers, Madsen, Solomon, and Soumis [26], Kohl and Madsen [27], Kolen, Rinnooy Kan, and Trienekens [28], Savelsbergh [31], and elsewhere.

The VRPTW provides a good starting point for modeling many applied problems, but the model itself can be a very difficult one to solve. Indeed, in 1984 Solomon [33] created a set of 87 test instances (the largest having 100 customers), but despite much attention, 17 of these instances have remained unsolved in the research literature. The main contribution of our paper is a new implementation of a linear programming (LP) based algorithm that succeeded in finding the optimal solution for 10 out of the 17 unsolved Solomon instances. Our approach is a refinement of the algorithm of Kohl, Desrosiers, Madsen, Solomon, and Soumis [26]. The new components consist of an improved search engine for locating k-path inequalities (including those with $k > 2$) and a parallel implementation of both the cutting-plane generator and the branch-and-bound search. Our cutting-plane procedure is based on Karger's [22] randomized algorithm for finding minimum cuts in graphs.

The paper is organized as follows. In Section 1 we give a formal definition of the VRPTW and describe the set-partioning formulation that is the basis of the most successful attacks to date on the problem. In Section 2 we describe the k-path inequalities used by Kohl, Desrosiers, Madsen, Solomon, and Soumis [26] and we present our procedure for finding violated cutting planes from this class. Some details of our implementation are given in Section 3 and our parallel code, based on the *TreadMarks* [1] distributed shared memory system, is described in Section 4. Finally, in Section 5 we report computational results for our algorithm on the Solomon test instances and in Section 6 we make some concluding remarks.

1 Set-Partitioning Formulation

The VRPTW can be naturally formulated in terms of a directed graph $G = (V, A)$, having node set V and arc set A. The customers, $\mathcal{C}$, are represented by nodes labeled 1 through n, and the depot is represented twice, by node 0 and node $n + 1$. The arcs A indicate the potential route segments between the customers and the depot. Each arc $ij \in A$ has an associated travel cost c_{ij} and a travel time t_{ij}. Node 0 is incident to outgoing arcs only and node $n + 1$ is only incident to incoming arcs. The route taken by a vehicle corresponds to a directed path in G that originates at node 0 and terminates at node $n + 1$, with service provided to a specified subset of the customers visited along the route. Each vehicle has

capacity q, and there is a demand d_i for each customer $i \in \mathcal{C}$. The sum of the demands of the customers served by a given route cannot exceed q.

The service at customer i must begin within a time window $[a_i, b_i]$, where a_i is the earliest time at which service may begin and b_i is the latest time at which service may begin. A vehicle is permitted to arrive to serve customer i prior to the start of i's time window, but it must then wait until time a_i before service may begin. For the depot, all vehicles must leave node 0 within the time window $[a_0, b_0]$ and return within the time window $[a_{n+1}, b_{n+1}]$. Given a fleet of vehicles $\mathcal{T}$, the objective is to minimize the total travel cost for a set of at most $|\mathcal{T}|$ routes that meet the service demands of each customer in $\mathcal{C}$. Since the model does not penalize routes based on total time, we may assume without loss of generality that $a_0 = b_0 = a_{n+1} = 0$.

The data is assumed to have certain characteristics. The constants q, d_i, c_{ij}, a_i, and b_i are nonnegative integers, and the travel times t_{ij} are positive integers. We also require that the triangle inequality holds for both the travel costs and travel times, that is for any $i, j, k \in V$ we have $c_{ij} + c_{jk} \geq c_{ik}$ and $t_{ij} + t_{jk} \geq t_{ik}$. These restrictions are not too harsh—they are usually satisfied in applied settings. They do, however, allow us to focus our attention on special types of routes. Firstly, we can now require that the routes correspond to simple directed paths from node 0 to node $n+1$, that is, we have no repeated nodes. Secondly, we can require that if a route visits a customer, then it serves that customer. These requirements are useful, for example, in modeling customer service times, since the travel time from i to j can include the service time at customer i, as well as a measure of the time it takes to travel the distance between the locations.

In this setting we can describe the set-partioning formulation of the VRPTW proposed by Desrochers, Desrosiers, and Solomon [13]. Let $\hat{\mathcal{R}}$ denote the set of feasible routes (with no repeated nodes) for the VRPTW, and for each route $r \in \hat{\mathcal{R}}$, let β_{ir} be equal to 1 if r visits customer i, and 0 otherwise. The cost of a route, c_r, is defined to be the sum of the costs of the arcs in the route. Let $u = |\mathcal{T}|$. The VRPTW can be formulated as

$$\begin{array}{lll} \text{Minimize} & \sum_{r \in \hat{\mathcal{R}}} c_r y_r & \\ \text{subject to} & \sum_{r \in \hat{\mathcal{R}}} \beta_{ir} y_r = 1, & \text{for all } i \in \mathcal{C} \\ & \sum_{r \in \hat{\mathcal{R}}} y_r \leq u & \\ & y_r \geq 0 \text{ and integer}, & \text{for all } r \in \hat{\mathcal{R}}. \end{array} \tag{1}$$

In this integer programming (IP) model, the variables y_r that are set to 1 correspond to the routes of the vehicles in the solution.

We modify the basic formulation (1) in two ways. First, we add a simple lower bound on the number of vehicles required to provide service to the set of customers. The bound is $l = \lceil \sum_{i \in \mathcal{C}} d_i / q \rceil$, where $\lceil \alpha \rceil$ is the smallest integer greater than or equal to α. This bound is trivially satisfied by any integer solution, but it may improve the value of the LP relaxation. Second, we expand $\hat{\mathcal{R}}$ to include non-simple directed paths in G from node 0 to node $n+1$, but we do not include any path that contains a 2-cycle (that is, a sequence $i \to j \to i$). This

expanded set of routes, denoted by $\mathcal{R}$, weakens somewhat the LP relaxation (non-simple routes cannot appear in the integer programming solution, but they may appear in an LP solution), but it makes the LP easier to handle. To describe the modified formulation, we generalize the definition of β_{ir} to allow it to be a constant equal to the number of times route r visits customer i. The IP problem can then be written as

$$\begin{array}{lll} \text{Minimize} & \sum_{r\in\mathcal{R}} c_r y_r & \\ \text{subject to} & \sum_{r\in\mathcal{R}} \beta_{ir} y_r = 1, & \text{for all } \ i \in \mathcal{C} \\ & l \le \sum_{r\in\mathcal{R}} y_r \le u & \\ & y_r \ge 0 \text{ and integer}, & \text{for all } \ r \in \mathcal{R}. \end{array} \tag{2}$$

In the LP relaxation of (2), if π_i is the dual variable associated with the constraint requiring that customer $i \in \mathcal{C}$ is visited and π_0 is the dual variable associated with the bounds on the number of vehicles, then the reduced cost of a variable y_r is

$$c_r - \sum_{i\in\mathcal{C}\cup\{0\}} \pi_i \beta_{ir}.$$

This value corresponds to the cost of the route r using the modified arc costs $\bar{c}_{ij} = c_{ij} - \pi_i$. Therefore, given a dual solution π we can determine if there exists a variable in (2) having negative reduced cost by computing the minimum cost member of $\mathcal{R}$ using the $\bar{c}_{ij}$ arc costs. This problem is a variation of the *shortest path problem with time windows and capacity constraints* that is treated in Desrochers and Soumis [14], where a pseudo-polynomial-time dynamic-programming algorithm is presented. This algorithm is discussed further in Desrochers, Desrosiers, and Solomon [13] and Kohl [25], including the use of the Houck, Picard, Queyranne, and Vemuganti [21] method for eliminating the 2-cycles in the solutions.

Desrochers, Desrosiers, and Solomon [13] utilize the Desrochers-Soumis algorithm in an efficient *column-generation* procedure for solving the LP relaxation of (2). The approach is to start with a selected subset of routes, and repeatedly use the simplex algorithm to solve the LP and use the Desrochers-Soumis algorithm to obtain new routes to add to the formulation. For a general discussion of column generation we refer the reader to Wolsey [34].

We adopt the Desrochers, Desrosiers, and Solomon procedure in our study. The efficiency of the Desrochers-Soumis algorithm in the column-generation process is the reason we use the expanded set of routes $\mathcal{R}$ in (2); no such pseudo-polynomial algorithm is likely to be found for optimizing over the simple routes $\hat{\mathcal{R}}$ since Kohl [25] has shown that optimizing over $\hat{\mathcal{R}}$ is $\mathcal{NP}$-hard in the strong sense.

2 Cutting Planes

The LP relaxation of (2) provides a effective means for bounding the optimal value of the VRPTW. Indeed, in 34 of the 87 Solomon test instances, the value of the LP coincides with

the value of the VRPTW.[1] Moreover, for the remaining 46 instances with known optimal solutions (including those solved for the first time in our tests), the value of the LP is on average only 2.8% below the value of the VRPTW. The strength of this bound allows Desrochers, Desrosiers, and Solomon [13] to solve 50 of the 87 instances by incorporating the LP bound in a branch-and-bound algorithm for the VRPTW, where at each node of the search tree the column-generation algorithm is used to reoptimize after the branching step.

Taking advantage of recent improvements in LP solvers and improvements in computing hardware, it is possible to push the Desrochers, Desrosiers, and Solomon branch-and-bound algorithm a bit further as we indicate in Table 1. It is clear, however, that further improvements in computing power alone will not significantly extend the range of instances that can be solved. It is therefore natural to consider the use of cutting planes to tighten the LP relaxation, with the of aim of reducing the size of the bound-and-bound search needed to reach an optimal solution. This approach was proposed and tested by Kohl, Desrosiers, Madsen, Solomon, and Soumis [26]; we extend their work in our study.

Time Limit (300Mhz Pentium II)	Number Solved
10 Seconds	41 out of 87
100 Seconds	54 out of 87
1,000 Seconds	62 out of 87
10,000 Seconds	64 out of 87

Table 1: Instances Solved with Branch-and-Bound

In a cutting-plane approach we add inequalities to (2) that are valid for all integer solutions, but are not satisfied by the optimal LP solution. The inequalities we use as cutting planes have a very simple form: they state that a given set of customers S must be visited by at least k vehicles, where k is an integer that is no larger than $k(S)$, the least number of vehicles needed to serve each customer in S. These inequalities are a natural generalization of the subtour cuts introduced for the traveling salesman problem (TSP) by Dantzig, Fulkerson, and Johnson [12]; they were studied in the context of vehicle routing by Laporte, Nobert, and Desrochers [29] and dubbed *k-path* cuts by Kohl, Desrosiers, Madsen, Solomon, and Soumis [26] in their work on the VRPTW.

Given a nonempty set $S \subseteq \mathcal{C}$, let $\delta(S)$ denote the set of outgoing arcs from S, that is $\delta(S) = \{ij \in A : i \in S, j \notin S\}$. For a route $r \in \mathcal{R}$, let μ_r^S denote the number of arcs r has in $\delta(S)$, counting multiplicities when an arc appears more than once. With this notation, a k-path inequality in the set-partitioning formulation can be written as

$$\sum_{r \in \mathcal{R}} \mu_r^S y_r \geq k$$

for an appropriate choice of k.

[1] This does not agree with the count of 27 instances reported in Desrochers, Desrosiers, and Solomon [13]. The difference is mostly likely due to the manner in which the Euclidean arc costs are computed.

Subtour Cuts

For any nonempty set S, we can clearly set $k = 1$ since S must be visited by at least one vehicle. We refer to these 1-path cuts as *subtour cuts*, since they are a directed version of the inequalities used by Dantzig, Fulkerson, and Johnson. Subtour cuts can be used to correct the obvious flaws that appear in the LP solutions due to the fact that we permit non-simple directed paths in our model. An important point is that violated inequalities from this class can be detected efficiently using techniques borrowed from the TSP. To this end, for each arc $ij \in A$ and each route $r \in \mathcal{R}$, let γ_{ij}^r denote the number of times ij appears in r. For a given LP solution $(y_r : r \in \mathcal{R})$, let

$$x_{ij} = \sum_{r \in \mathcal{R}} \gamma_{ij}^r y_r \tag{3}$$

for each arc $ij \in A$. A subtour inequality is violated if an only if

$$\sum_{ij \in \delta(S)} x_{ij} < 1.$$

Therefore, we can determine if a violated subtour cut exists by using maximum flow techniques in the directed graph G, where we use the x_{ij} values as arc capacities. In our implementation, we use the maximum-flow code of Anderson and Setubal [2] for this purpose.

2-Path Cuts

Now consider the class of inequalities having $k = 2$. These inequalities, together with subtour cuts, were used in the computational study of Kohl, Desrosiers, Madsen, Solomon, and Soumis [26]. Given a solution y to the LP and the corresponding x-vector described in (3), the set $S \subseteq \mathcal{C}$ determines a 2-path cut if we have $k(S) \geq 2$ while $\sum_{ij \in \delta(S)} x_{ij} < 2$.

To determine if $k(S) \geq 2$ (or equivalently, $k(S) > 1$), we apply the same 2-step approach used in [26]. The first step is to check if there is sufficient vehicle capacity on a single vehicle to visit all of the customers in S. If not, then clearly more than one vehicle is needed and it becomes unnecessary to consider step 2. The second step is more complicated, as it requires determining if there exists a feasible route that leaves the depot, visits every customer in S once, and returns to the depot. This is precisely the *traveling salesman problem with time windows* (TSPTW) feasibility problem. The TSPTW is studied in Ascheuer, Fischetti, and Grötschel [3], Baker [4], Balas and Simonetti [5], Christofides, Mingozzi, and Toth [10], and Dumas, Desrosiers, Gélinas, and Solomon [17]. Despite the fact that the feasibility problem is $\mathcal{NP}$-hard, so long as the size of S is relatively small, the problem can be solve efficiently by dynamic programming using the method proposed by Dumas, Desrosiers, Gélinas, and Solomon [17]. If it is concluded that the TSPTW is infeasible for a set S, then clearly more than one vehicle is required, implying that the corresponding 2-path inequality is valid.

To identify sets $S \subseteq \mathcal{C}$ as candidates for 2-path cuts, Kohl, Desrosiers, Madsen, Solomon, and Soumis [26] use a greedy heuristic to build maximal sets having $\sum_{ij \in \delta(S)} x_{ij} < 2$.

These authors also describe (but do not test) an exact separation algorithm that involves building the family of all minimal sets that require more than one vehicle, and then checking whether or not the corresponding 2-path inequalities are violated. In the computational tests reported in [26], the heuristic method was chosen due to the anticipated large running time for the exact procedure. We propose below a natural alternative to the algorithms of [26].

Consider the undirected graph $G' = (V', E)$ obtained from G by identifying the two copies of the depot node and by identifying each pair of directed arcs ij and ji in A. For each edge $e = (i, j) \in E$, let $w_e = x_{ij} + x_{ji}$, where x is the vector defined in (3). For a set of nodes $R \subseteq V'$, let

$$\delta_{G'}(R) = \{e \in E \mid e \text{ has an end in } R, \text{ and an end in } V' \setminus R\}.$$

The total weight of a set of edges $F \subseteq E$ is denoted by $w(F)$. Since the x-vector satisfies the condition that the flow into a set of customers equals the flow out of the set, then clearly $w(\delta_{G'}(R))$ is exactly double the value of $\sum_{ij \in \delta(R)} x_{ij}$. Thus, identifying a set of customers S such that $\sum_{ij \in \delta(S)} x_{ij} < 2$ amounts to finding a set in G' satisfying $w(\delta_{G'}(S)) < 4$.

If the graph G' is built from an x-vector corresponding to an LP solution that does not contain any violated subtour inequalities, then the minimum value of any cut in the graph is 2. Thus, our goal is to find every set S of customers in the graph G' such that the value of the cut is less than double the value of the minimum cut. The random contraction algorithm of Karger [22] can be used to accomplish this task. In an implementation-oriented paper by Karger and Stein [23], the authors prove that Karger's algorithm on an undirected graph with n nodes finds with high probability all cuts with weight within a multiplicative factor α of the minimum cut in $O(n^{2\alpha} log^3 n)$ time. We use this algorithm with $\alpha = 2$ to generate the relevant sets S such that $S \subseteq \mathcal{C}, |S| > 1$, and $w(\delta_{G'}(S)) < 4$. For each generated set S we check to determine if $k(S) \geq 2$, unless the set S contains another set that has already been determined as valid.

Algorithm 1 *2-path Separation Routine*

(1) Build the undirected graph G' from the x-vector.

(2) Use Karger's algorithm with $\alpha = 2$ to find
$\Gamma = \{S : S \subseteq \mathcal{C}, |S| > 1, w(\delta_{G'}(S)) < 4\}$.

(3) Sort the sets in Γ from smallest to largest.

(4) For each $S \in \Gamma$, if S does not contain a known valid set $S_1 \in \Gamma$ corresponding to a 2-path inequality and $k(S) \geq 2$ then add S to list of valid inequalities.

The implementation of Karger's algorithm that is used in this work was written by

Sanjeeb Dash (Rice University). In our applications, running the random contraction algorithm the necessary number of iterations to achieve the goal of probabilistically generating all of the desired sets is too costly; we found that computationally, setting the number of iterations at $3n^2$ consistently worked well with our data.

Since Karger's algorithm generates relevant sets in polynomial time, it is reasonable to expect the overall separation routine to be relatively fast, so long as the TSP with time windows feasibility problem (TSPTW) can be solved efficiently. The exact dynamic programming approach to solving the TSPTW is capable of very quickly solving instances, so long as the size of S is small (less than 10-15 nodes). It is not always necessary, though, to solve the TSPTW with the exact algorithm, since we may first run a heuristic to see if we can quickly find a feasible solution. In this work a very simple insertion heuristic is used for this purpose. The use of the heuristic prior to the dynamic programming algorithm is optional, but our tests indicate significant savings in terms of time are achieved by doing so.

k-Path Cuts

While the validity of the k-path inequality for $k \geq 3$ is clear, prior to this work, the implementation of a separation routine for these inequalities appeared to be unreasonable. However, by modifying the new 2-path cutting-plane approach outlined above, we have a natural separation routine for k-path inequalities where $k \geq 3$.

Clearly, Karger's algorithm can be applied to generate the relevant sets S for any size k. By setting $\alpha = k$, sets S for which $\sum_{ij \in \delta(S)} x_{ij} < k$ are found. Thus, the only other modification of the cutting-plane algorithm must be to the routine for checking if $k(S) \geq k$. We first check if $k-1$ vehicles have a total capacity large enough to handle the sum of the demands. If so, then it must be determined if there exists a set of no more than $k-1$ routes which cover the set S. This is no longer a TSPTW-feasibility problem, but rather a VRPTW-feasibility problem. In this smaller VRPTW instance, the objective is to minimize the number of vehicles required, rather than the total distance traveled. Thus, an optimization method for the VRPTW must be called from within the cutting-plane procedure on a smaller instance of the VRPTW with a modified objective function. If the problem is determined to be infeasible, or the minimum number of vehicles required is greater than or equal to k, then the set corresponds to a valid k-path inequality.

One obvious choice for an optimization method for the smaller VRPTW is the set-partitioning and column-generation method used to solve the initial problem. Since such an approach may not always provide a fast solution to the smaller VRPTW, the separation routine can be transformed into a heuristic by enforcing a limit on the amount of time spent on solving any single smaller VRPTW. If a solution to the smaller instance is not determined within such a time limit, then the set is discarded, and the next set is considered.

As in the 2-path case, the optimization method for solving the smaller VRPTW is only necessary if a heuristic is unable to generate a feasible solution for the instance. Thus, the

Algorithm 2 *k-Path Separation Routine*

(1) Build the undirected graph G' from the x-vector.

(2) Use Karger's algorithm with $\alpha = k$ to find $\Gamma = \{S : S \subseteq \mathcal{C}, |S| \geq k, w(\delta_{G'}(S)) < 2k\}$.

(3) Sort the sets in Γ from smallest to largest.

(4) For each $S \in \Gamma$, if S does not contain a known valid set $S_1 \in \Gamma$ corresponding to a valid k-path inequality and $k(S) \geq k$ can be determined, then add S to the list of valid inequalities.

heuristic TSPTW solution method is applied first, since a feasible solution to the TSPTW clearly indicates that the corresponding k-path inequality where $k \geq 2$ is not valid. This will typically work well on the very small sets contained in Γ. For a set S which cannot be visited by a single vehicle, a heuristic for the VRPTW is needed. In our implementation, we use two fast heuristics that are described in Solomon [33], namely the nearest-neighbor algorithm and the "insertion-criterion (i)" heuristic. If either of these algorithms generates a feasible solution to the smaller VRPTW with $k-1$ or fewer vehicles, then $k(S) < k$ and the current set is not a valid k-path inequality. While testing the optimization method for the smaller VRPTW, we found that computationally, solving the VRPTW instance with the objective of minimizing the number of vehicles was far more time consuming than using the objective of minimizing total distance traveled. Therefore, we optimized with the latter objective first, as an additional heuristic before finally having to call the optimization algorithm with the modified objective function to determine if the set actually yields a k-path cut. Although not addressed in our work, these results indicate that it is likely that the overall routine would improve with better heuristics and optimization methods for the smaller VRPTW. To our knowledge, there has not been a focus on trying to solve small VRPTW's. The k-path separation routine, though, provides motivation for future research in this area.

3 Implementation Details

We have incorporated the k-path cut generator in a branch-and-bound algorithm for the VRPTW. A few details are given below concerning the implementation of the algorithm. The CPLEX callable library [11], version 5.0 is used to solve all of the linear programming problems that arise. The primal simplex algorithm is used to solve the LPs within the set partitioning and column-generation scheme, and the dual simplex algorithm is used to re-optimize after adding cutting planes and before generating additional columns. As is the

standard in the vehicle routing literature, solutions that the LP solver claims to be optimal are assumed to be optimal and are not verified with exact arithmetic.

At the start of our main procedure, we apply the time window reductions described in Desrochers, Desrosiers, and Solomon [13]. These reductions serve to tighten the time window at a node as much as possible based on the earliest and latest possible arrival and departure times from all other nodes.

After this preprocessing, we begin the column-generation phase of the algorithm. We require generated columns to have a reduced cost of less than -10^{-4}; this value is sufficiently large to work with the default settings of the LP solver. Even with this tolerance, however, a large number of columns of can be generated in a given subproblem. To handle this, we follow the implementation of Kohl, Desrosiers, Madsen, Solomon, and Soumis [26] and set a limit of at most 20 columns on the number to be added from each subproblem solved.

The next phase of the algorithm is the cutting-plane generation. Our code requires the user to specify the maximum k ($k \geq 0$) for which k-path cutting planes are to be sought. If $k \geq 1$, then the cutting strategy checks for subtour cuts first by calling the maximum-flow algorithm. (We require that cutting planes be violated by at least 10^{-7} to be added to the LP.) If no subtour cuts are found and $k \geq 2$, then the 2-path separation routine is called. If no 2-path cuts are found and $k \geq 3$, then the 3-path separation routine is called, and so on in this way until whatever value of k is specified by the user. The largest value of k for which we present results is six. Some experiments applying a global cutting strategy at nodes within the branching tree were performed, but the results indicate that the additional time required to do so is not accompanied by a significantly smaller search tree. Thus, in the computational results, cutting is only applied at the top of the branching tree.

In the case of $k \geq 3$, the k-path separation routine requires a limit on the amount of time to invest in attempting to solve the smaller VRPTW's. Unless otherwise mentioned in the results section, this time limit is set at 600 seconds. In most cases, this limit is sufficiently large to allow the optimal solution for the smaller VRPTW to be obtained.

If an integer solution is not obtained after solving the LP relaxation and generating cuts, then branching is applied. We apply the same branching rules as in Kohl, Desrosiers, Madsen, Solomon, and Soumis [26]. At each node in the search tree, branching is performed on the number of vehicles, if fractional; otherwise, branching is performed on a single arc value x_{ij}. The branching arc ij is selected as the one maximizing the quantity $c_{ij} \cdot \min(x_{ij}, 1 - x_{ij})$. In terms of the set partitioning model, if $x_{ij} = 0$, then the cost of each route in the LP which uses arc (i, j) is increased (by adding a penalty), and the arc is eliminated from the subproblem network. If $x_{ij} = 1$, then the cost of each route in the LP visiting i or j without using arc (i, j) is increased, and the arc (i, j) is made to be the only feasible way of leaving node i or visiting node j in the subproblem. For more details on branching within the set-partitioning and column-generation model, see Desrochers, Desrosiers, and Solomon [13] and Kohl [25].

4 Parallel Computation

The solution procedure described in this paper contains two distinct portions that have been implemented in parallel using the *TreadMarks* [1] distributed shared memory system. First, each of the calls to the k-path separation routine for $k \geq 2$ at the root node of the branching tree is implemented in parallel. Second, after all of the violated valid inequalities that could be determined are added to the LP relaxation, the branch-and-bound procedure is then performed in parallel. The combination of these two parallel portions enabled the overall algorithm to solve a total of 10 out of the 17 unsolved Solomon instances.

In the parallel implementation of the k-path separation routine, the relevant sets S are generated and stored in the set Γ using Karger's algorithm on a single processor. Once the set Γ is known, each set $S \in \Gamma$ must be evaluated to determine if $k(S) \geq k$. If $k = 2$, this may require solving a TSPTW; if $k > 2$, this may require solving a smaller VRPTW. In either instance, the evaluation of each set S can be done independently of the other sets. Thus, the sets contained in Γ are divided evenly into the sets $\Gamma_1, \Gamma_2, \ldots, \Gamma_p$, where p is the number of processors. Each processor i then considers each of the sets contained in Γ_i, recording only those sets that are valid. Since some processors may finish evaluating their assigned sets sooner than others, we implement a *task-stealing* procedure in which a processor i that finishes evaluating the sets in Γ_i may remove and process unevaluated sets from another processor's list of sets until all of the sets have been considered. After all of the sets on all processors have been evaluated, a single processor collects the list of valid sets from each machine and adds the corresponding cutting planes to the LP.

The parallel implementation of the cutting-plane procedure differs from the sequential version in one way. In the sequential algorithm, only the minimal valid sets are added to the LP formulation. In other words, a set S is not checked for validity if it contains a subset that has already been determined to be valid. In the parallel version, this would involve checking each set S against the list of valid sets over all processors. This would involve considerable communication between the processors, which undermines the benefits of parallelization. Adding only the minimal valid sets to the LP formulation, though, is a restriction made only to limit the number of rows added at a time, not because it is required for correctness. Computational experiments could not determine if adding all of the valid sets is better or worse than adding only the minimal valid sets. Thus, it is advantageous in the parallel implementation for each processor to either check each set S only against the valid sets found on that processor, or against none at all. The result in either case is that the final list of valid sets may include non-minimal valid sets.

At the end of the cutting-plane phase of the algorithm, a single processor has maintained an LP relaxation for the VRPTW. If the optimal solution to this problem is integer-feasible, then the algorithm is complete; otherwise, branch-and-bound with column generation must be performed. To parallelize this process, all of the processors must have access to a global heap of active nodes and a best known upper bound. At the beginning of the branch-and-bound routine, one processor starts to process the root node of the branching tree,

while all of the other processors are told to "wait" until the list of active nodes is not empty. Processing a node involves first removing a node from the heap of active nodes, then solving the LP relaxation using column generation. If an integer solution is found, then the best upper bound is updated and the heap is pruned; otherwise, a branching variable is determined and the appropriate new nodes are created. If the new nodes are feasible, then they are added to the set of active nodes. The heap is a minimal heap that is indexed on the objective value for each active node. It there exists a waiting processor, then when a node is added to the heap a message is sent to the waiting processor that tells the processor that the list of active nodes is no longer empty; otherwise, the node is simply inserted into the heap. Processors continue to remove nodes from the heap and process them until the heap is empty and an integer solution has been found, or until the time limit has been reached. Further information on implementing a parallel branch-and-bound algorithm using *TreadMarks* can be found in Bixby, Cook, Cox, and Lee [6].

5 Computational Results

Our computational study is carried out on the 87 instances in "Problem Set 1" of the well-known Solomon problems [33]. These instances are the standard benchmark for testing exact solution techniques for the VRPTW. The arc costs and the travel times for the problem set are determined by the Euclidean distance between pairs of (x, y) coordinates; we adopt the rounding convention used in Kohl [25], Kohl, Desrosiers, Madsen, Solomon, and Soumis [26], and Kohl and Madsen [27] for computing these distances. The test instances consist of three classes, "c", "r", and "rc." The c-instances are clustered, the r-instances are random, and the rc-instances are random-clustered.

Our sequential experiments are performed on a 300 MHz Pentium II with 256 megabytes of memory, and our parallel experiments are performed on a cluster of 32 of these workstations, linked via a switched 100 megabit ethernet network. Running times are reported not in CPU time, but in total elapsed time. All of the algorithms are implemented in the ANSI C programming language and compiled with the GNU gcc 2.7.2.1 complier, using the –O option.

The sequential version of the algorithm using only subtour and 2-path cutting planes prior to branching succeeds in solving 73 out of 87 of the Solomon benchmark instances. Three of these, specifically $r103.100, r106.100$, and $r112.50$, were previously unsolved in the literature. Table 2 reports the statistics for the r- and rc-instances having 50 and 100 customers using the 2-path routine. Since all of the c-instances and all of the 25-customer instances can be solved using the described branch-and-bound method without adding any cutting planes, we do not provide the specific statistics for those 47 examples. The column designated LB(1) refers to the value of the LP relaxation prior to the addition of valid inequalities. The value of the lower bound after all cutting planes have been inserted into the LP is denoted as LB(2). The optimal integer objective value is given as OPT. In those instances where cutting planes are added to the LP, the "Closed" column indicates the

Problem	LB(1)	LB(2)	OPT	Closed	Nodes	Cuts	Veh	Cols	Time(s)
r101.50	1043.367	1044.000	1044.0	1.000	1	1	12	292	1.67
r101.100	1631.150	1634.000	1637.7	0.435	9	1	20	1360	37.51
r102.50	909.000	909.000	909.0		1	0	11	418	1.43
r102.100	1466.600	1466.600	1466.6		1	0	18	1320	34.92
r103.50	765.950	767.300	772.9	0.194	49	2	9	2051	47.37
r103.100	**1206.312**	**1206.376**	**1208.7**	**0.027**	**39**	**2**	**14**	**4860**	**741.60**
r104.50	616.500	620.758	625.4	0.478	74	12	6	3382	489.87
r104.100									
r105.50	892.120	893.650	899.3	0.213	8	2	9	609	5.51
r105.100	1346.142	1349.316	1355.3	0.347	71	7	15	2892	251.37
r106.50	791.367	793.000	793.0	1.000	1	5	8	499	4.19
r106.100	**1226.440**	**1227.404**	**1234.6**	**0.118**	**760**	**4**	**13**	**10781**	**13975.23**
r107.50	704.438	705.880	711.1	0.216	42	3	7	2262	67.07
r107.100									
r108.50									
r108.100									
r109.50	775.096	776.231	786.8	0.097	140	8	7	1955	91.88
r109.100									
r110.50	692.577	694.150	697.0	0.356	3	3	7	666	12.16
r110.100									
r111.50	691.812	692.642	707.2	0.054	199	6	7	3955	477.35
r111.100									
r112.50	**607.219**	**612.389**	**630.2**	**0.225**	**2895**	**15**	**6**	**18052**	**43675.00**
r112.100									
rc101.50	850.021	944.000	944.0	1.000	1	16	8	604	6.98
rc101.100	1584.094	1616.921	1619.8	0.919	7	23	14	1568	136.23
rc102.50	719.902	813.037	822.5	0.908	303	21	7	5123	671.32
rc102.100									
rc103.50	643.133	710.112	710.9	0.988	4	15	6	1403	44.55
rc103.100									
rc104.50	543.750	543.750	545.8		17	0	5	2855	58.89
rc104.100									
rc105.50	754.443	853.675	855.3	0.984	15	17	8	1432	38.83
rc105.100	1471.160	1509.733	1513.7	0.907	36	16	15	3288	502.27
rc106.50	664.433	716.501	723.2	0.886	7	14	6	941	29.61
rc106.100									
rc107.50	591.476	631.588	642.7	0.783	71	7	6	2843	139.08
rc107.100									
rc108.50	538.957	587.120	598.1	0.814	8	7	6	1579	56.80
rc108.100									

Table 2: r- and rc-instances: Subtour and 2-path cutting planes added at root node of search tree.

proportion of the gap between OPT and LB(1) closed by the insertion of valid inequalities. The number of nodes in the search tree is provided in the "Nodes" column, and the number of cutting planes added to the LP is given in the "Cuts" column. The number of vehicles used in the optimal solution is found in the "Veh" column and the total number of columns in the final LP is provided in the "Cols" column. Finally, the computation time in total elapsed seconds is displayed in the last column. A blank row indicates that the code is unable to find the optimal solution within a time limit of 50,000 seconds. Those instances that were previously unsolved in the literature that our algorithm solved are printed in boldface.

The sequential algorithm that looks for 3-path cutting planes, in addition to subtour and 2-path inequalities, generates optimal solutions for the same set of problems. In these instances, the time limit on solving each of the smaller VRPTW problems within the 3-path cutting-plane routine is set at 600 seconds. A limit of 60 seconds was tested as well, but we found that many of the smaller VRPTW's were not solved within this smaller time limit, while only one failure occurred with the larger limit. The value of LB(2) improved with the addition of 3-path cuts in six of the r-instances (r101.100, r103.50, r104.50, r105.100, r106.100, and r111.50) and seven of the rc-instances (rc101.100, rc102.50, rc103.50, rc105.100, rc106.50, rc107.50 and rc108.50). Clearly, in those examples which do not yield any 3-path cuts, the additional time spent in the cutting-plane portion of the code provides no benefits. Unfortunately, for the thirteen instances that do find 3-path cuts, the improvement of the lower bound prior to branching does not yield similar improvements in time. However, it is clear that the 3-path separation routine can be applied without an unreasonable cost in terms of time. This can be seen in Table 3. The table indicates the

Problem Set	Num nodes	Num solved	Num w/ cuts	k=2 Average gab closed	k=2 Average time	k=3 Average gap closed	k=3 Average time
r	25	12	7	0.382 (0.654)	1.56	0.382 (0.654)	3.63
	50	11	10	0.348 (0.383)	4079.41	0.356 (0.391)	4170.24
	100	5	4	0.186 (0.232)	3008.13	0.252 (0.315)	3148.62
rc	25	8	3	0.267 (0.713)	1.46	0.267 (0.713)	3.52
	50	8	7	0.795 (0.909)	130.76	0.822 (0.939)	615.71
	100	2	2	0.913 (0.913)	319.25	0.920 (0.920)	502.545
c	25	9	0	0.000 (0.000)	1.60	0.000 (0.000)	1.63
	50	9	1	0.007 (0.067)	25.22	0.007 (0.067)	25.57
	100	9	1	0.111 (1.000)	118.81	0.111 (1.000)	118.90

Table 3: Average gap closed and average time for those instances solved by the sequential algorithm.

total number of instances in each set that are solved and the total number of instances out of those solved that added cutting planes to the linear programming problem. We give both the average gap closed over all instances in the test set, as well as the average gap

Problem	Time (seconds) for k=3 Number of Processors			
	1	4	8	16
r101.50	1.66	1.97	2.10	2.18
r101.100	117.07	140.39	112.46	119.31
r102.50	1.44	1.46	1.49	1.51
r102.100	34.93	35.92	36.13	36.16
r103.50	329.49	298.82	294.29	299.44
r103.100	838.29	488.11	542.98	563.96
r104.50	822.42	367.22	954.16	940.14
r104.100	–	–	–	–
r105.50	14.94	9.73	9.23	10.59
r105.100	354.62	280.72	189.61	370.14
r106.50	4.18	4.56	4.51	4.75
r106.100	14398.17	4311.20	2418.88	1424.43
r107.50	82.45	47.81	55.31	56.56
r107.100	–	21363.82	9966.76	6231.27
r108.50	–	32467.77	14688.51	9605.05
r108.100	–	–	–	–
r109.50	95.92	42.47	43.69	43.14
r109.100	–	–	–	34,456.37
r110.50	30.34	20.74	19.98	20.88
r110.100	–	–	–	–
r111.50	930.51	307.05	193.41	186.08
r111.100	–	–	–	–
r112.50	43559.24	7589.11	3794.89	1730.38
r112.100	–	–	–	–

Table 4: Summary of times for r-instances on 1, 4, 8, and 16 processors using subtour, 2- and 3-path cutting planes.

calculated only for those instances that added cutting planes (this latter value is given in parentheses).

Note that the average time for solving the 50 and 100 customer instances in set r is much larger than the corresponding times for the c and rc sets. This is due to the fact that $r112.50$ and $r106.100$ took considerably longer than the other solved examples of comparable size. Based on the results in Table 3, it appears that the cutting-plane procedure is most effective for the rc-problem set.

Results for the parallel version of the algorithm relative to the sequential version are provided in Table 4. The value of LB(2) may vary depending on which cutting planes are added to the LP formulation. In general, though, the LB(2) values remain close to the ones obtained using the sequential algorithm. Only the elapsed time in seconds for the algorithm using subtour, 2- and 3-path cutting planes for the 50 and 100 customer r-instances is reported in Table 4. Note that optimal solutions for three additional instances ($r107.100$, $r108.50$, and $r109.100$) are found. Specific statistics on these examples are found

in Table 5. In some of the instances in Table 4, the times do not decrease as expected with the addition of more machines. For example, in the case of $r104.50$, the increase in time from 4 machines to 8 machines occurred because adding more machines increases the number of non-minimal valid sets that are found and added as cutting planes. In this case, the new inequalities lead to different LP relaxations which yield better cuts and ultimately a better lower bound. Thus, the longer runtime is the consequence of a more successful cut-generation phase. In those instances in which the time stays relatively constant regardless of an increase in the number of processors, we believe that the runtime cannot be reduced further, as our algorithm is written. The problem lies in the fact that if k is greater than or equal to 3, then there may exist a difficult smaller VRPTW that must be solved. No matter how many processors are used, a single machine will have to solve this difficult problem, and all other machines will have to wait until it has done so. Thus, there exists a minimal amount of time that is required to complete the cut-generation phase of the problem based on the time it takes to solve these smaller VRPTW's on a single machine. If the branching tree is large enough, the runtime in the search tree portion of the code will be reduced with the addition of more processors.

For those instances that remained unsolved after using 16 processors, but showed a relatively small optimality gap, we ran the parallel version of the code on 32 processors and increased the maximum value of k to 6. (In these examples, $k = 5$ is the largest value for which a violated cut was actually detected.) In this way, three additional instances were solved. Also a clear indication that $r110.100$ would solve was obtained, but due to a technical issue with *TreadMarks* we were forced to use only 16 machines and solve using an increased time limit. Thus, a total of 10 previously unsolved instances from the data set are

Problem	k	Procs	LB(1)	LB(2)	OPT	Gap closed	Nodes	Cuts	Veh	Time(s)
r107.100	6	32	1051.844	1052.927	1064.6	0.085	3,201	9	11	4,050.73
r108.50	6	32	588.926	595.625	617.7	0.233	2,507	40	6	6,359.41
r109.100	6	32	1130.587	1133.164	1146.9	0.158	13,832	35	13	19,135.05
r110.100	3	16	1048.482	1049.939	1068.0	0.075	90,448	9	12	457,024.92
r111.100	6	32	1032.028	1032.075	1048.7	0.003	28,503	4	12	41,879.97
rc102.100	6	32	1403.646	1439.547	1457.4	0.668	26,632	65	14	42,150.49
rc103.100	6	32	1218.495	1242.491	1258.0	0.607	9,236	77	11	84,438.90

Table 5: Results from the parallel version of the algorithm for 7 previously unsolved instances.

solved. The 7 examples which were solved using the parallel implementation are provided in Table 5. The value in the column marked "k" refers to the maximum value of k for which the separation routine is called. The column marked "Procs" indicates the number of processors used to obtain the optimal solution. The other column headings remain the same.

The best results obtained for the still unsolved instances on 16 processors with a $30,000$

second time limit are given in Table 6. The column marked "Number of cuts" indicates the number of valid inequalities by type: subtour/2-path/3-path/4-path/5-path/6-path. The LB(3) column indicates the lower bound achieved on a single machine prior to termination. In other words, the "lower bound" is not actually valid since a different machine may have

Problem	LB(1)	LB(2)	LB(3)	Number of cuts	nodes
*rc104.100	1094.333	1098.505	1109.4	2/17/14/0/0/0	6069
rc106.100	1308.781	1332.824	1356.1	1/27/36/0/0/0	17904
rc107.100	1170.689	1180.995	1201.2	1/17/10/0/0/0	10457
*rc108.100	1063.011	1085.781	1093.3	10/12/11/0/0/0	387
r104.100	949.103	951.135	962.3	2/8/1/0/0/0	10184
r108.100	907.162	909.472	919.4	2/8/1/0/0/0	8905
r112.100	919.192	922.337	935.1	5/29/3/0/0/0	3279

Table 6: Results on unsolved instances using 16 processors.

been working on a node from the heap that has a lower value; it does, however, give a sense of how much the branch-and-bound tree has accomplished. In the case of the two examples marked with an asterisk (*), the code could not even complete the cut-generation phase of the algorithm before the end of the time limit. This is due to the presence of many (at least 50) very difficult to solve smaller VRPTW's, each of which took the full time limit of 600 seconds to determine that the instance could not be solved. To avoid this phenomenon specifically for these two instances, the time limit on the smaller VRPTW's was reduced to 60 seconds. The results in Table 6 marked with an asterisk refer to the code using the reduced time limit on the smaller VRPTW's.

Thus far, we have only considered Problem Set 1 of the Solomon test instances. In [33], Solomon also proposed a second collection of examples that are much less tightly constrained than those in Problem Set 1. No optimization results have been reported in the literature for these "Problem Set 2" instances, and they pose difficult challenges for the algorithms currently available, including our implementation. We did however make a small amount of progress on this set of examples, and in Table 7 we report the optimal values for 20 out of the 24 Problem Set 2 c-instances. The running times varied between 2 seconds and 25,000 seconds on a single processor. We also solved 8 out of the 33 r-instances and 2 out of the 24 rc-instances. The optimal values for these 10 examples are given in Table 8. A major difficulty we encountered with the r and rc instances is in our implementation of the Dumas, Desrosiers, Gélinas, and Solomon TSPTW algorithm, where the storage required for the state space in the loosely constrained instances quickly exceeded the limits of our workstation.

As a final test, we ran our code on the 200-customer Problem Set 1 "Extended Solomon" instances generated by Homberger [18]. We succeeded in solving 8 out of the 30 instances, using 4 processors and setting $k = 3$. The results for the solved instances are reported in Table 9.

Problem	OPT	Problem	OPT	Problem	OPT	Problem	OPT
c201.25	214.7	c203.25	214.7	c205.25	214.7	c207.25	214.5
c201.50	360.2	c203.50	359.8	c205.50	359.8	c207.50	359.6
c201.100	589.1	c203.100		c205.100	586.4	c207.100	585.8
c202.25	214.7	c204.25	213.1	c206.25	214.7	c208.25	214.5
c202.50	360.2	c204.50		c206.50	359.8	c208.50	350.5
c201.100	589.1	c204.100		c206.100	586.0	c208.100	

Table 7: Optimal values for c-instances of Problem Set 2.

Problem	OPT	Problem	OPT	Problem	OPT	Problem	OPT
r201.25	463.3	r202.50	698.5	r206.25	374.4	rc202.25	338.0
r201.50	791.9	r203.25	391.4	r210.25	404.6		
r202.25	410.5	r205.25	393.0	rc201.25	360.2		

Table 8: Optimal values for r- and rc-instances of Problem Set 2.

Problem	OPT	Time (Seconds)
c1.200	2698.6	106.98
c2.200	2694.3	6121.15
c5.200	2694.9	140.49
c6.200	2694.9	190.80
c7.200	2694.9	228.97
c8.200	2684.0	11,850.68
r1.200	4667.2	3587.01

Table 9: Results on 200-customer extended Solomon instances, using 4 processors and a 20,000 second time limit.

6 Conclusions

Karger's algorithm is an efficient method for searching for subtour-like cuts in the VRPTW and it should have an impact on solution methods for other classes of vehicle routing problems. In particular, a Karger-driven approach can make effective use of an array of processors, as we did in our work with the *TreadMarks* distributed shared memory system. However, the fact that not all of the (relatively small) Solomon instances could be solved in our tests indicates that much work remains to be done in this area. We carried out experiments with variations of the TSP blossom and comb inequalities, but discovered that very few were found and even fewer improved the LP bound (and so we did not report these results in our computational tests). A weakness of the blossoms and combs we tested is that they do not make direct use of the time windows on the nodes; an area that would appear to be of direct practical value would be further research into TSP-like inequalities that incorporate time-window information.

Acknowledgements

We would like to thank Oli Madsen for kindly helping us with our column-generation routine, Alan Cox and Eyal Delara for their help in working with the *TreadMarks* system, and Sanjeeb Dash for providing us with his implementation of Karger's algorithm.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: shared memory computing on networks of workstations. *IEEE Computer* **29** (1996) 18–28.

[2] R. J. Anderson and J. C. Setubal. Goldberg's algorithm for maximum flow in perspective: a computational study. In: *Network Flows and Matching*, D. S. Johnson and C. C. McGeoch (editors), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 12, American Mathematical Society, Providence, 1993, pp. 1–18.

[3] N. Ascheuer, M. Fischetti, and M. Grötschel. A polyhedral study of the asymmetric travelling salesman problem with time windows. *ZIB Preprint* SC-97-11, ZIB Berlin, Germany, 1997.

[4] E. Baker. An exact algorithm for the time constrained traveling salesman problem. *Operations Research* **31** (1983) 938–945.

[5] E. Balas and N. Simonetti. Linear time dynamic programming algorithms for some new classes of restricted TSP's. *Management Science Research Report* 617, Graduate School of Industrial Administration, Carnegie Mellon University, 1996.

[6] R. Bixby, W. Cook, A. Cox, and E. Lee. Parallel Mixed Integer Programming. *Technical Report* CRPC–TR95554, Center for Research on Parallel Computation, Rice University, 1995.

[7] J. Bramel and D. Simchi-Levi. On the effectiveness of set covering formulations for the vehicle routing problem with time windows. *Operations Research* **45** (1997) 295–301.

[8] E. Cheng and J. L. Rich. A home health care routing and scheduling problem. *Technical Report* 98–04, Computational and Applied Mathematics, Rice University, 1998.

[9] Y. Caseau and P. Koppstein. A cooperative-architecture expert system for solving large time/travel assignment problems. In: *Database and Expert Systems Applications, Proceedings of the International Conference in Valencia, Spain, 1992*, A. Min Tjoa and I. Ramos (editors), Springer-Verlag, Wien, 1992, pp. 197–202.

[10] N. Christofides, A. Mingozzi, and P. Toth. State space relaxation procedures for the computation of bounds to routing problems. *Networks* **11** (1981) 145–164.

[11] *Using the CPLEX Callable Library,* Version 5.0. CPLEX Optimization, Incline Village, Nevada, 1997.

[12] G. B. Dantzig, R. Fulkerson, and S. M. Johnson, Solution of a large-scale traveling salesman problem. *Operations Research* **2** (1954) 393–410.

[13] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40** (1992) 342–354.

[14] M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR* **26** (1988) 191–211.

[15] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. In: *Network Routing*, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser (editors), North-Holland, Amsterdam, 1995, pp. 35–139.

[16] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks* **14** (1984) 545–565.

[17] Y. Dumas, J. Desrosiers, E. Gélinas, and M. M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research* **43** (1995) 367–371.

[18] J. Homberger. Extended Solomon's VRPTW instances. Available electronically at: `http://www.fernuni-hagen.de/WINF/touren/menuefrm/probinst.htm`.

[19] M. L. Fisher, K. O. Jörnsten, and O. B. G. Madsen. Vehicle routing with time windows: two optimization algorithms. *Operations Research* **45** (1997) 488–492.

[20] K. Halse. *Modeling and solving complex vehicle routing problems.* Ph.D. Thesis, IMSOR, Technical University of Denmark, Lyngby, 1992.

[21] D. J. Houck, Jr., J–C. Picard, M. Queyranne, and R. R. Vemuganti. The travelling salesman problem as a constrained shortest path problem: theory and computational experience. *Opsearch* **17** (1980) 93–109.

[22] D. R. Karger. Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. In: *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM-SIAM, 1993, pp. 84–93.

[23] D. R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In: *Proceedings of the 25th ACM Symposium on the Theory of Computing*, ACM Press, 1993, pp. 757–765.

[24] K. Knight and J. Hofer. Vehicle scheduling with timed and connected calls: a case study. *Operational Research Quarterly* **19** (1968) 299–310.

[25] N. Kohl. *Exact methods for time constrained routing and scheduling problems*, Ph.D. Thesis, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1995.

[26] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis. k-path cuts for the vehicle routing problem with time windows. *Technical Report* IMM-REP-1997-12, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1997.

[27] N. Kohl and O. B. G. Madsen. An optimization algorithm for the vehicle routing problem with time windows based on Lagrangean relaxation. *Operations Research* **45** (1997) 396–406.

[28] A. W. J. Kolen, A. H. G. Rinnooy Kan, and H. W. J. M. Trienekens. Vehicle routing with time windows. *Operations Research* **35** (1987) 266–273.

[29] G. Laporte, Y. Nobert, and M. Desrochers. Optimal routing under capacity and distance restrictions. *Operations Research* **33** (1985) 1050–1073.

[30] H. Pullen and M. Webb. A computer application to a transport scheduling problem. *The Computer Journal* **10** (1967) 10–13.

[31] M. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research* **4** (1984) 285–305.

[32] M. Solomon. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Report* 83-05-03, Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1983.

[33] M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* **35** (1987) 254–265.

[34] L. A. Wolsey. *Integer Programming*. Wiley, New York, 1998.