

A Branch-and-Cut Procedure for the Vehicle Routing Problem with Time Windows

Jonathan F. Bard • George Kontoravdis • Gang Yu

*Graduate Program in Operations Research, Department of Mechanical Engineering,
The University of Texas, Austin, Texas 78712-1063*

*Graduate Program in Operations Research, Department of Mechanical Engineering,
The University of Texas, Austin, Texas 78712-1063*

*Management Science and Information Systems, College of Business Administration,
The University of Texas, Austin, Texas 78712*

jbard@mail.utexas.edu • georgek@veltisto.com • yu@uts.cc.utexas.edu

This paper addresses the problem of finding the minimum number of vehicles required to visit a set of nodes subject to time window and capacity constraints. The fleet is homogeneous and is located at a common depot. Each node requires the same type of service. An exact method is introduced based on branch and cut. In the computations, ever increasing lower bounds on the optimal solution are obtained by solving a series of relaxed problems that incorporate newly found valid inequalities. Feasible solutions or upper bounds are obtained with the help of greedy randomized adaptive search procedure (GRASP). A wide variety of cuts is introduced to tighten the linear programming (LP) relaxation of the original mixed-integer program. To find violated cuts, it is necessary to solve a separation problem. A substantial portion of the paper is aimed at describing the heuristics developed for this purpose. A new approach for obtaining feasible solutions from the LP relaxation is also discussed. Numerical results for standard 50- and 100-node benchmark problems are reported.

The vehicle routing problem with time windows (VRPTW) can be defined as follows. Let $G_0 = (V, E)$ be a connected graph consisting of a set of $n+1$ nodes each of which can be reached only within a specified time interval or time window, and a set E of nonnegatively weighted arcs with associated traveling times. Each customer is denoted by an index i ($i = 1, \dots, n$) and imposes a service requirement in the form of a pickup. There is a single depot designated by the index 0. For easier reference, let $I = \{1, \dots, n\}$ be the set of customers and $I_0 = I \cup \{0\}$. For each $i \neq 0$, let $q_i > 0$, $\sigma_i \geq 0$, and $[a_i, b_i]$ be the demand, service time, and time window, respectively, for customer i .

The problem we address is to find the minimum number of tours, K^* , such that each node is reached within its time window and the accumulated service up to any node does not exceed a positive number Q (vehicle capacity). A secondary objective is to minimize the total distance traveled. An alternative objective that combines the two might be to minimize the lifecycle cost of providing service.

All problem parameters, such as customer demand and time windows, are assumed to be known with certainty. Moreover, each customer must be served by exactly one vehicle, so split service and multiple visits are not permitted. The tours, as defined in the problem statement, correspond to feasible routes

starting and ending at the depot. Because of the time component the routes are directed so that tour orientation is important. Any load that is picked up along a route must be taken to the depot. If the single service for all customers is a delivery, the problem remains the same except that each vehicle starts with a full load and returns to the depot empty.

Once the minimum number of tours K^* is determined the number of vehicles required is at most K^* since one vehicle is needed for each route in the worst case. If two routes r_i and r_j are time disjoint, then the same vehicle can handle both. In what follows we do not consider this case since it does not arise commonly in practice; therefore, the terms *routes* and *vehicles* are used interchangeably. It is assumed that the triangle inequality holds for all travel times between any subset of nodes.

Practical VRPTW instances are mainly solved by heuristics; however, methods that provide optimal or near-optimal solutions have become the subject of considerable research. This paper presents an exact branch-and-cut algorithm for solving the VRPTW. The proposed methodology combines several heuristics and lower bounding procedures into an exact scheme that can solve standard 50- and 100-node benchmark problems within reasonable computational time. Our main contribution centers on the integration of modern cutting plane techniques and feasibility heuristics for solving the VRPTW. The majority of the work reported on involved the development and implementation of several new heuristics for solving the so-called separation problem. Solutions to the latter provide valid inequalities that remove fractional solutions arising from the linear programming relaxation of the original mixed-integer linear program (MILP). Without efficient separation procedures, branch-and-cut is dominated by simple branch-and-bound.

The methodology incorporates the following types of cuts: capacity inequalities, comb inequalities, box inequalities, path-box inequalities, subtour elimination constraints, incompatible pair inequalities, and incompatible path inequalities. In each case, we present any new theory along with the heuristics proposed to solve the corresponding separation problem. Sufficient detail is provided to allow independent implementation.

The rest of the paper is organized as follows. A brief literature review is presented in §1. A mixed-integer linear programming formulation is presented in §2. In §3 we outline the general structure of a cutting-plane optimization method and highlight the valid inequalities for the VRPTW. Algorithmic details are presented in §4. We conclude with a discussion of the results and suggestions for future work.

1. Related Work

Survey papers on heuristic and exact methods for the VRPTW and related problems have been published by Bodin et al. (1983), Solomon and Desrosiers (1988), Desrochers et al. (1988), and Desrosiers et al. (1995). Exact methods for the VRPTW use one of the following three techniques: column generation, Lagrangian relaxation, or cutting planes such as branch and cut.

In their breakthrough work, Desrochers et al. (1992) used a column generation technique to solve many 50-node problems and several 100-node problems to optimality with the objective of minimizing the total distance traveled. The basic idea is to formulate the VRPTW as a set partitioning problem by considering all feasible routes implicitly. Because this number grows exponentially with the customer base, only a small fraction of the routes is included in the model. At each major iteration a check is performed to determine whether there exists a route not currently included that can reduce the overall distance. This is achieved by solving a relaxed shortest path subproblem with time window and capacity constraints. If such a route exists, its corresponding column is added to the model and the procedure is repeated until the optimal solution for the complete set of routes is found. To obtain integer solutions the above algorithm is embedded in a branch-and-bound scheme. Sol and Savelsbergh (1994) extended this approach to the pickup and delivery VRPTW.

Kohl (1995) and Kohl and Madsen (1997) used Lagrangian relaxation to remove from the constraint region the requirement that all customers be serviced. This allowed them to decompose the overall problem into individual time and capacity constrained shortest path subproblems, one for each vehicle. Although they were able to find good heuristic solutions relatively easy to these subproblems, finding the optimal

Lagrange multipliers required substantial computational effort. A bundle method was used for this purpose. Problems with up to 100 customers were solved.

Padberg and Rinaldi (1991) were among the first to develop a cutting plane method to solve large instances of the traveling salesman problem (TSP). A similar technique was used by Hoffman and Padberg (1993) to solve large airline crew scheduling instances. More recently, Araque et al. (1994) developed a branch-and-cut (B&C) algorithm for the identical customer vehicle routing problem, while Augerat (1995) presented a number of new valid inequalities for the symmetric capacitated vehicle routing problem (VRP). In the branch-and-cut approach, the problem is initially formulated as a MILP. The integrality constraints are then relaxed and the resulting linear program is optimized. If the solution is integer, then the algorithm terminates with the optimal solution to the original problem. If the solution is fractional, then valid inequalities or cuts are appended to the model and the procedure is repeated. When no more cuts can be identified by the separation schemes, or the effect of any new inequalities is marginal, branching is initiated.

In conjunction with the Lagrangian approach, Kohl and his colleagues (Kohl 1995, Kohl et al. 1999) also developed a two-path cut method for solving the VRPTW with the objective of minimizing distance. That work was extended by Rich (1999) in his thesis and then further elaborated on by Cook and Rich (1999). They used a cutting plane algorithm and parallel computations to solve instances with up to 200 customers.

2. MILP Formulation

In the model, t_i denotes the instant a vehicle leaves customer i and is treated as a variable. The travel time between i and j ($i, j \in I_0$) is denoted by τ_{ij} . We assume that the vehicle velocity is one so travel time is equal to the distance traveled. Without loss of generality, in what follows it is assumed that there is no time window associated with the depot and that σ_i is included in the travel time; i.e., $\tau_{ij} \leftarrow \tau_{ij} + \sigma_i$ for all customers (see Kontoravdis and Bard 1995). Moreover, if the time windows of i and j are such that $(a_j - b_i) > \tau_{ij}$, then τ_{ij} is replaced by $a_j - b_i$.

Let x_{ij} ($i, j \in I_0$) be a flow variable equal to one if a vehicle travels along arc (i, j) ; zero otherwise. Let y_i ($i \in I$) be the vehicle load at departure from customer i corresponding to the load that needs to come back to the depot; i.e., the load that is picked up from customers along the route up to and including i . The following mixed-integer linear programming model minimizes the number of flow variables emanating from the depot, and hence, the number of routes required:

$$\text{(VRPTW)} \quad \min \sum_{i=1}^n x_{0i} \quad (1)$$

subject to

$$\sum_{j \in I_0} x_{ij} = 1, \quad i \in I, \quad (2)$$

$$\sum_{j \in I_0} x_{ij} - \sum_{j \in I_0} x_{ji} = 0, \quad i \in I_0, \quad (3)$$

$$t_j \geq t_i + \tau_{ij}x_{ij} - T_{ij}(1 - x_{ij}), \quad i, j \in I, \quad (4)$$

$$y_j \geq y_i + q_j - Q_j(1 - x_{ij}), \quad i, j \in I, \quad (5)$$

$$q_i \leq y_i \leq Q, \quad i \in I, \quad (6)$$

$$a_i \leq t_i \leq b_i, \quad i \in I, \quad (7)$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in I_0, \quad (8)$$

where $T_{ij} = b_i - a_j$ and $Q_j = Q - q_j$. Any smaller values used for these parameters in Equations (4) and (5) could potentially cut off feasible solutions. To see this, simply set $x_{ij} = 0$ and make the appropriate substitutions.

Constraint (2) ensures that each customer is served by exactly one vehicle. It also implies that all the routes are pairwise disjoint so that a vehicle visits a customer only if it is to provide service. Constraint (3) is the flow conservation equation enforcing route continuity so that the constructed routes are tours (loops) rather than open paths. Constraint (4) defines the relation between flow variables and departure times. It also eliminates the formation of subtours not containing the depot because increasing departure times are required along a route. For example, consider a subtour consisting of three customers i_1, i_2, i_3 . Then x_{i_2, i_3} and x_{i_3, i_1} should all be equal to one. After substituting in (4) and adding we obtain $\tau_{i_2, i_3} + \tau_{i_3, i_1} \leq 0$ which is clearly infeasible for non-negative travel times.

Constraint (5) tracks the load on a route at departure from a customer. It also eliminates subtours in a manner similar to that of Equation (4). Constraint (6) guarantees that the vehicle's capacity is not exceeded. The inequalities in (7) enforce the time window constraints while (8) defines the binary flow variables. The constants T_{ij} and Q_j in (4) and (5) can be replaced by any large numbers; however, computational experience has shown that these values should be as small as possible so the corresponding constraints are as tight as possible. The above formulation uses $O(n^2)$ binary variables and $O(n^2)$ constraints. A preprocessing step excludes all x_{ij} ($i, j \in I$) variables from the MILP formulation that correspond to infeasible vehicle trips.

3. Valid Inequalities

In this section we describe the principal classes of valid inequalities that can be used to reduce the solution space of (VRPTW) after relaxing the integrality constraints (8). We first define a few terms related to polyhedral theory (see Nemhauser and Wolsey 1988 for a complete description) and outline the general structure of a cutting-plane optimization method. We then present an overview of known valid inequalities for the traveling salesman problem and the capacitated vehicle routing problem. Our intent is to highlight only those valid inequalities that can be readily adapted to the VRPTW.

3.1. Polyhedra and Integer Programming

Polyhedral theory can be used to show that for every integer program (IP) there exists a corresponding linear program with the same optimal solution. More specifically, given $P = \{x \in \mathcal{R}_+^n : Ax \leq b\}$, where (A, b) is an integer matrix, and $S = P \cap Z^n$ then the $\text{conv}(S)$ (convex hull of S) is a rational polyhedron. The branch-and-cut approach exploits the above result by optimizing a series of linear problems whose polyhedral feasible regions correspond to tighter and tighter approximations of the convex hull of the initial problem. Let $\min\{cx : x \in S \subseteq Z_+^n\}$ be the initial integer program and $LP(S)$ the corresponding linear program after the integrality constraints are dropped. Starting

with $LP(S)$, valid inequalities for $\text{conv}(S)$ are iteratively added and the resulting formulation is optimized. The procedure terminates when the optimal solution over $\text{conv}(S)$ is reached. The convergence of a cutting plane method greatly depends on the type of valid inequalities added. Facets of $\text{conv}(S)$ are the most effective but it is not always possible to determine when a cut is a facet.

By definition valid inequalities do not remove any feasible integer points and hence are valid at any stage of a branch-and-bound algorithm. This contrasts with the more traditional cuts, such as Gomory fractional cuts, which may exclude feasible points during branch and bound.

3.2. TSP and VRP Valid Inequalities

We begin by introducing some terminology for the VRPTW that is used throughout the section. Similar definitions hold for the corresponding TSP and VRP formulations as well. Given an optimal solution to the relaxed (VRPTW) we define a weighted undirected graph $G_0 = (V, E)$, where $V = I_0$ and $E = \{(i, j) : i, j \in I_0 \text{ and } x_{ij} + x_{ji} > 0\}$. Each edge (i, j) has an associated weight $\omega_{ij} = x_{ij} + x_{ji}$. For $(S) \subseteq V$ we define $E(S) = \{(i, j) : i, j \in S\}$ as the set of edges in S , and $\delta(S) = \{(i, j) : i \in S \text{ and } j \notin S\}$ as the set of edges leaving S . For $e = (i, j) \in E$ we define $x_e = x_{ij} + x_{ji}$, and for $S \subseteq V$ we define $x(E(S)) = \sum_{e \in E(S)} x_e$. Similarly, for an edge set $A \subseteq E$ we define $x(A) = \sum_{e \in A} x_e$. The cut edges between two sets S_1 and S_2 are represented by (S_1, S_2) . Given an optimal solution to the relaxed (VRPTW) we also define a weighted directed graph $\vec{G}_0 = (V, A)$ where $V = I_0$ and $A = \{(i, j) : i, j \in I_0 \text{ and } x_{ij} > 0\}$. Each arc (i, j) has an associated weight $\omega_{ij} = x_{ij}$. The graph obtained from $G_0(\vec{G}_0)$ after removing the depot is denoted by $G(\vec{G})$. For $S \subseteq V$ we define λ_S to be a lower bound on the number of vehicles required to serve the customers in S .

Capacity Inequalities. The capacity inequalities, also known as subtour elimination constraints, guarantee that the number of vehicles visiting a set of customers S is no less than a lower bound λ_S on the number of vehicles required. For the undirected case, this can be written as

$$\sum_{e \in (S, V \setminus S)} x_e \geq 2\lambda_S. \quad (9)$$

The lower bound for the TSP is always one. For the VRP, one possibility for λ_5 is $\lceil \sum_{i \in S} q_i / Q \rceil$, which is basically a lower bound to the corresponding bin packing problem. It is known that (9) is a facet for the TSP (see Grötschel and Padberg 1985) but it is not necessarily a facet for the VRP.

Comb Inequalities. Let $H, W_1, \dots, W_k \subseteq V$. The node set (H, W_1, \dots, W_k) defines a *comb* in G if the following conditions are met:

$$\begin{aligned} |H \cap W_i| &\geq 1, \quad i = 1, \dots, k, \\ |W_i \setminus H| &\geq 1, \quad i = 1, \dots, k, \\ 2 \leq |W_i| &\leq |V| - 2, \quad i = 1, \dots, k, \\ W_i \cap W_j &= \emptyset, \quad i \neq j, \\ k &\geq 3 \text{ and odd.} \end{aligned}$$

The set H is the handle and the sets W_i are the teeth of the comb. Figure 1 illustrates a simple case.

Comb inequalities were first introduced for the TSP and are facet defining (see Grötschel and Padberg 1985 or Nemhauser and Wolsey 1988). The corresponding inequality is as follows:

$$\begin{aligned} x(E(H)) + \sum_{i=1}^k x(E(W_i)) \\ \leq |H| + \sum_{i=1}^k |W_i| - (3k + 1)/2. \end{aligned} \quad (10)$$

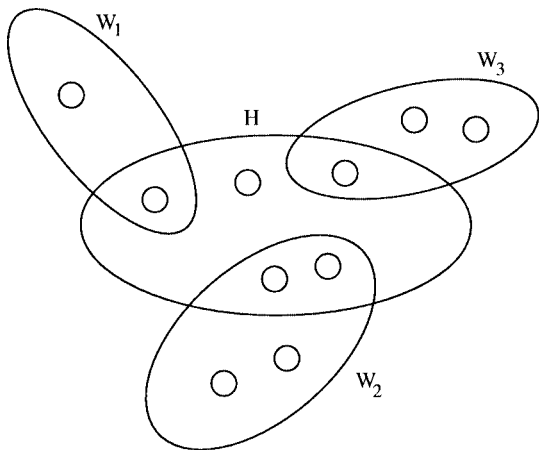


Figure 1 Example of a Comb

Constraint (10) is also valid for the VRP but specific variants have been developed to account for the depot and the vehicle capacity. If the depot belongs to a tooth, say W_1 , but not to the handle, then the following comb inequality (Cornuéjols and Harche 1993) is valid for the VRP and is a facet if certain necessary conditions are satisfied:

$$\begin{aligned} x(\delta(H)) &\geq (k + 1) - (x(\delta(W_1)) - 2\lambda_{V \setminus W_1}) \\ &\quad - \sum_{i=2}^k (x(\delta(W_i)) - 2). \end{aligned} \quad (11)$$

If the depot does not belong to the comb and the vehicle capacity is taken into account, then the following inequality is valid for the VRP (see Augerat 1995 and Laporte and Nobert 1984):

$$x(\delta(H)) \geq (k + 1) - \sum_{i=1}^k (x(\delta(W_i)) - 2\lambda_{W_i}) \quad (12)$$

which holds if all teeth satisfy the condition: $\lambda_{W_i \setminus H} + \lambda_{W_i \cap H} > \lambda_{W_i}$.

Box Inequalities. Let $H, W_1, \dots, W_k \subseteq V$. The node set (H, W_1, \dots, W_k) defines a *box* in G if the following conditions are met:

$$\begin{aligned} \sum_{\eta \in W_i} q_\eta &\leq Q, \quad i = 1, \dots, k, \\ W_i &\subset H, \quad i = 1, \dots, k, \\ W_i \cap W_j &= \emptyset, \quad i \neq j, \\ k &\geq 1. \end{aligned}$$

A box is similar to a comb with the difference being that each tooth, W_i , is a proper subset of the handle H . Moreover, all customers within a tooth must fit in one vehicle (see Figure 2). Given a box we define a bin packing problem with $k + |H \setminus \bigcup_{i=1}^k W_i|$ items and bin size Q . Each W_i is considered an item of size $\sum_{\eta \in W_i} q_\eta$, and each customer $\eta \in H \setminus \bigcup_{i=1}^k W_i$ is considered an item of size q_η . Let $\mathcal{B}_{H|W_1, \dots, W_k}$ be the optimal solution of this bin packing problem. Then the following is a valid inequality for VRP (see Augerat 1995):

$$x(\delta(H)) \geq 2\mathcal{B}_{H|W_1, \dots, W_k} - \sum_{i=1}^k (x(\delta(W_i)) - 2). \quad (13)$$

It is not known if (13) is a facet for the VRP.

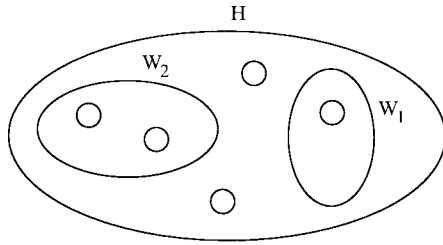


Figure 2 Example of a Box

Path-Box Inequalities. By combining the concept of a comb and a box one can define a *path-box* as follows (see Augerat 1995). Let $H, W_1, \dots, W_k \subseteq V$. The node set (H, W_1, \dots, W_k) defines a path-box in G if the following conditions are met:

$$\sum_{\eta \in W_i} q_\eta \leq Q, \quad i = 1, \dots, k,$$

$$W_i \cap H \neq \emptyset, \quad i = 1, \dots, k,$$

$$W_i \cap W_j = \emptyset, \quad i \neq j,$$

$$k \geq 1.$$

The difference between a comb and a path-box constraint is that in the case of the latter the demand of each tooth must fit in a vehicle and it may be entirely contained in the handle. A tooth that is contained within the handle is called a spot. Figure 3 shows an example of a path-box with two regular teeth W_2 and W_3 , and a spot W_1 . Given a path-box we define a bin packing problem in exactly the same way we did for a box with the additional constraint that no more than two teeth, not counting spots, can be included in

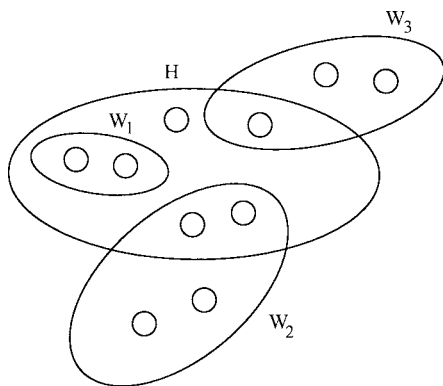


Figure 3 Example of a Path-Box

the same bin. Let $\mathcal{B}'_{H|W_1, \dots, W_k}$ be the optimal solution of this bin packing problem. Then the following is a valid inequality for VRP:

$$x(\delta(H)) \geq 2\mathcal{B}'_{H|W_1, \dots, W_k} - \sum_{i=1}^k (x(\delta(W_i)) - 2). \quad (14)$$

It is not known whether (14) is a facet for the VRP. From a computational point of view calculating $\mathcal{B}_{H|W_1, \dots, W_k}$ and $\mathcal{B}'_{H|W_1, \dots, W_k}$ for (13) and (14), respectively, requires the optimization of a bin packing subproblem. Because this may be time consuming, a bin packing lower bound can be used without affecting the validity of the constraints.

3.3. VRPTW Valid Inequalities

All VRP inequalities presented above are valid for (VRPTW) as well. We now show how some of these inequalities can be strengthened by taking advantage of the VRPTW structure and then present some additional constraints.

Subtour Elimination Constraints. A feasible solution to (VRPTW) consists of a set of disjoint directed cycles all containing the depot. If the depot is not taken into account, the resulting partial solution is a collection of open directed paths. Therefore, a cycle in G indicates an infeasibility that should be eliminated. Inequality (9) eliminates such cycles. Depending on the density of G and the cardinality of a subtour S it might be preferable to use the following equivalent inequality to keep the corresponding linear programming model sparse:

$$\sum_{e \in E(S)} x_e \leq |S| - \lambda_S. \quad (15)$$

Unlike the standard VRP, the orientation of a tour is important because of the time window constraints and the two types of service. Directed subtours in \vec{G} can be eliminated by the following inequality:

$$\sum_{(i, j) \in (S, V \setminus S)} x_{ij} \geq \lambda_S. \quad (16)$$

Constraint (16) is similar to (9) for the undirected case and requires that the number of vehicles leaving a set of customers S must be greater than or equal to a lower bound on the number of vehicles needed

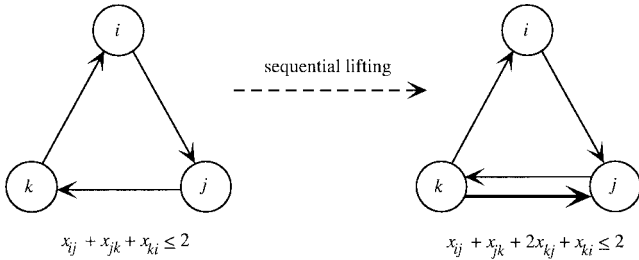


Figure 4 Example of Lifting Directed Subtour Inequalities

to serve S . The following proposition shows that (15) and (16) can be used interchangeably depending whether G or \vec{G} is used.

LEMMA 1. Inequality (16) defined over \vec{G} is violated if and only if inequality (15) defined over G is violated.

If the lower bound on the number of vehicles is not taken into account in (15), then a sequential lifting technique can be applied to raise some of the left-hand side coefficients (see Grötschel and Padberg 1985). The resulting inequality is a facet for the corresponding asymmetric TSP. Figure 4 demonstrates the lifting of a three node cycle. The bold arc (k, j) in the graph on the right is given a weight of two. Symmetry provides for two other possibilities in this case.

For the general case of a subtour with $k \geq 3$ customers, $\{i_1, i_2, \dots, i_k\}$, the following two inequalities are valid for the asymmetric TSP and will be used for (VRPTW):

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2 \sum_{j=2}^{k-1} x_{i_j i_1} + \sum_{j=3}^{k-1} \sum_{h=2}^{j-1} x_{i_j i_h} \leq k-1 \quad (17)$$

and

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2 \sum_{j=3}^k x_{i_1 i_j} + \sum_{j=4}^k \sum_{h=3}^{j-1} x_{i_j i_h} \leq k-1. \quad (18)$$

Constraints (17) and (18) are called \vec{D}_k and \vec{D}_k^- inequalities, respectively (see Fischetti and Toth 1997).

Comb Inequalities. The following proposition introduces a valid comb inequality for (VRPTW) which is similar to the one obtained by Laporte and Nobert (1984). The inequality is also valid for the VRP.

PROPOSITION 1. For any comb in G inequality (19) is valid for (VRPTW):

$$\sum_{e \in E(H)} x_e + \sum_{i=1}^k \sum_{e \in E(W_i)} x_e \leq |H| + \sum_{i=1}^k |W_i| - \frac{1}{2} \sum_{i=1}^k (\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)}). \quad (19)$$

PROOF. First observe that Constraints (2) and (3) correspond to the following constraint for nodes in the undirected graph G :

$$\sum_{e \in \delta(v)} x_e = 2 \quad \text{for } v \in V \setminus \{0\}. \quad (20)$$

By multiplying (20) by $\frac{1}{2}$ and summing over all $v \in H$ we obtain

$$\sum_{e \in E(H)} x_e + \frac{1}{2} \sum_{e \in \delta(H)} x_e \leq |H|. \quad (21)$$

Note that the summation over $e \in \delta(H)$ consists of two components as shown by (22):

$$\sum_{e \in \delta(H)} x_e = \sum_{i=1}^k \sum_{e \in \delta(H) \cap E(W_i)} x_e + \sum_{i=1}^k \sum_{e \in \delta(H) \setminus \bigcup_{l=1}^k E(W_l)} x_e. \quad (22)$$

The definition of G requires $x_e \geq 0$, or equivalently

$$-\frac{1}{2} x_e \leq 0 \quad \text{for } e \in E. \quad (23)$$

Now add (23) to (21) for all $e \in \delta(H) \setminus \bigcup_{i=1}^k E(W_i)$:

$$\sum_{e \in E(H)} x_e + \frac{1}{2} \sum_{i=1}^k \sum_{e \in \delta(H) \cap E(W_i)} x_e \leq |H|. \quad (24)$$

The subtour elimination constraints for $W_i, H \cap W_i$, and $W_i \setminus H$ are the following:

$$\sum_{e \in E(W_i)} x_e \leq |W_i| - \lambda_{W_i} \quad \text{for } i = 1, \dots, k, \quad (25)$$

$$\sum_{e \in E(H \cap W_i)} x_e \leq |H \cap W_i| - \lambda_{H \cap W_i} \quad \text{for } i = 1, \dots, k, \quad (26)$$

$$\sum_{e \in E(W_i \setminus H)} x_e \leq |W_i \setminus H| - \lambda_{W_i \setminus H} \quad \text{for } i = 1, \dots, k. \quad (27)$$

Multiplying (25), (26), and (27) by $\frac{1}{2}$, summing over i and adding to (24) gives (19). \square

If all the lower bounds that appear in the right-hand side of (19) are equal to one, then the comb inequality for (VRPTW) reduces to

$$\sum_{e \in E(H)} x_e + \sum_{i=1}^k \sum_{e \in E(W_i)} x_e \leq |H| + \sum_{i=1}^k |W_i| - 3k/2.$$

In this case (10) is stronger than (19) by $\frac{1}{2}$. However, if at least one of the lower bounds is larger than one, then (19) is stronger by at least $\frac{1}{2}$. The following proposition establishes the condition under which (19) is stronger than (12) and vice versa.

PROPOSITION 2. *For any comb in G such that $\lambda_{W_i \setminus H} + \lambda_{W_i \cap H} > \lambda_{W_i}$:*

- (i) *If $\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} = 3$ for each $i = 1, \dots, k$, then (12) is stronger than (19) by one.*
- (ii) *If $\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} = 4$ for exactly one tooth while it is three for all other teeth, then (12) and (19) are equivalent.*
- (iii) *In all other cases (19) is stronger than (12) by at least one.*

PROOF. We first express (19) in a form similar to (12) by multiplying both sides of the inequality by two and using the fact that $2x(S) = 2|S| - x(\delta(S))$ for $S \in V_0$. This gives

$$\begin{aligned} & -x(\delta(H)) + \sum_{i=1}^k (-x(\delta(W_i))) \\ & \leq -\sum_{i=1}^k (\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)}) \end{aligned}$$

or

$$x(\delta(H)) \geq -\sum_{i=1}^k [x(\delta(W_i)) - (\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)})]. \quad (28)$$

Now replace $\lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)}$ with $\lambda_{W_i} + (\lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} - \lambda_{W_i})$ in (28) to get

$$\begin{aligned} x(\delta(H)) & \geq \sum_{i=1}^k (\lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} - \lambda_{W_i}) \\ & \quad - \sum_{i=1}^k (x(\delta(W_i)) - 2\lambda_{W_i}) \end{aligned}$$

or

$$x(\delta(H)) \geq \xi - \sum_{i=1}^k (x(\delta(W_i)) - 2\lambda_{W_i})$$

where $\xi = \sum_{i=1}^k (\lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} - \lambda_{W_i})$. Therefore, comparing (19) and (12) reduces to comparing ξ with $k + 1$.

- (i) $\lambda_{W_i} + \lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} = 3$ for each $i = 1, \dots, k$. Because each lower bound is at least one each item in this equation must be exactly one implying that $\lambda_{H \cap W_i} + \lambda_{W_i \setminus (H \cap W_i)} - \lambda_{W_i} = 1$, hence $\xi = k$.
- (ii) Following the same steps as in (i) gives $\xi = k + 1$.
- (iii) Following the same steps as in (i) gives $\xi > k + 1$. \square

Incompatible Pair Inequalities. Let i and j be two customers. If j can be the immediate successor of i in a feasible route, then i and j define a feasible vehicle transition. We represent this symbolically by $i \rightarrow j$. Similarly, if there is no feasible route with i and j in consecutive positions then i and j define an infeasible transition denoted by $i \nrightarrow j$. We use $i \leftrightarrow j$ to denote the feasibility of both $i \rightarrow j$ and $j \rightarrow i$. Similarly, $i \nleftrightarrow j$ means that $i \nrightarrow j$ and $j \nrightarrow i$ and i, j define an incompatible pair of nodes. Because the triangle inequality holds for travel times between any $i, j \in I_0$, each member of an incompatible pair of nodes in G must belong to a different connected component; i.e., there should not be a path between i and j in G .

Let $i \nleftrightarrow j$ be an incompatible pair of nodes, and let \mathcal{P} be a path $\{i, k_1, \dots, k_{|\mathcal{P}|-2}, j\}$ in G . Then the following is a valid inequality:

$$x_{i, k_1} + x_{k_1, i} + \dots + x_{k_{|\mathcal{P}|-2}, j} + x_{j, k_{|\mathcal{P}|-2}} \leq |\mathcal{P}| - 2. \quad (29)$$

Constraint (29) can be lifted by reducing its right-hand side when the number of vehicles required to service the customers in \mathcal{P} is more than two; i.e.,

$$\begin{aligned} & x_{i, k_1} + x_{k_1, i} + \dots + x_{k_{|\mathcal{P}|-2}, j} + x_{j, k_{|\mathcal{P}|-2}} \\ & \leq |\mathcal{P}| - \max\{2, \lambda_{\mathcal{P}}\}. \end{aligned} \quad (30)$$

If the cardinality of \mathcal{P} is at least four, then (29) can also be lifted by considering the directed counterpart of \mathcal{P} . Let i_1, i_2, i_3, i_4 be four consecutive nodes in \mathcal{P} . We identify two cases.

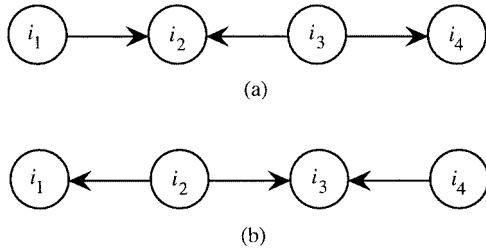


Figure 5 Special Cases for Incompatible Pair Inequalities
 Note. The above two lifting cases apply to (29) only, not (30).

Case 1. If $x_{i_1 i_2}, x_{i_3 i_2}, x_{i_3 i_4} > 0$ and $x_{i_2 i_1}, x_{i_2 i_3}, x_{i_4 i_3}$ either do not exist or are zero (see Figure 5a), then the left-hand side portion of (29) that corresponds to i_1, \dots, i_4 can be replaced by $x_{i_1 i_2} + 2x_{i_3 i_2} + x_{i_3 i_4}$.

Case 2. If $x_{i_2 i_1}, x_{i_2 i_3}, x_{i_4 i_3} > 0$ and $x_{i_1 i_2}, x_{i_3 i_2}, x_{i_3 i_4}$ either do not exist or are zero (see Figure 5b), then the left-hand side portion of (29) that corresponds to i_1, \dots, i_4 can be replaced by $x_{i_2 i_1} + 2x_{i_2 i_3} + x_{i_4 i_3}$.

Inequality (30) is useful only when $\lambda_{\mathcal{P}} = 1$. If this is not the case, it is dominated by the subtour elimination Constraint (15) defined over the set of nodes in \mathcal{P} . This is illustrated in Figure 6 where $i \leftrightarrow j, \mathcal{P} = \{i, h, k, j\}$ and $\lambda_{\mathcal{P}} = 2$. The arcs with solid lines correspond to flow variables included in (30), the dashed arcs are the additional variables included in (15) while the dotted arcs entering and leaving node j are provided for flow balance. The incompatible pair inequality is

$$x_{ih} + x_{hi} + x_{hk} + x_{kh} + x_{kj} + x_{jk} \leq 2$$

while the subtour elimination constraint is

$$x_{ih} + x_{hi} + x_{hk} + x_{kh} + x_{kj} + x_{jk} + x_{ik} + x_{ki} + x_{hj} + x_{jh} \leq 2.$$

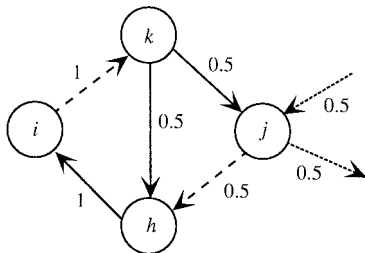


Figure 6 Case Where Subtour Elimination Constraint Dominates Incompatible Pair Inequality

Incompatible Path Inequalities. The incompatible path inequalities are similar to incompatible pair inequalities but they take into account the underlying directed graph. If $i \leftrightarrow j, j \rightarrow i$ and the triangle inequality holds for the travel times between any $i, j \in I_0$, then any feasible solution cannot contain a direct path, $\vec{\mathcal{P}} \in I$, from i to j . If such a path exists in a fractional solution, then the following incompatible path inequality is valid:

$$x_{i, k_1} + x_{k_1, k_2} + \dots + x_{k_{|\vec{\mathcal{P}}|-2}, j} \leq |\vec{\mathcal{P}}| - 2. \quad (31)$$

Inequality (31) can be strengthened by taking into account the number of vehicles needed to serve customers in $\vec{\mathcal{P}}$, giving

$$x_{i, k_1} + x_{k_1, k_2} + \dots + x_{k_{|\vec{\mathcal{P}}|-2}, j} \leq |\vec{\mathcal{P}}| - \max\{2, \lambda_{\vec{\mathcal{P}}}\}. \quad (32)$$

For the undirected case, valid inequalities for cutting off infeasible fractional paths were developed by Fischetti et al. (1998) in their work on the orienteering problem. They used the term *path inequalities*. The orienteering problem is variant of the prize collecting TSP where each city i has a value v_i but now the edge costs are zero. The objective is to find a subtour that maximizes the weighted sum of the cities visited within some fixed amount of time, t_0 . In a related paper, Ascheuer et al. (1999) developed infeasible path elimination constraints for the asymmetric TSP.

4. Branch-and-Cut Algorithm

In this section we present a branch-and-cut methodology for (VRPTW). The overall algorithm is first outlined and then a detailed explanation of each component is given.

4.1. Algorithm Outline

A branch-and-cut approach employs a combination of cutting planes and implicit enumeration to solve integer programs. The basic idea is to identify violated inequalities (preferably facets) that are valid throughout the search tree. Let x, t, y be vectors representing the flow, time, and load variables of (VRPTW). The following generic model expressed in matrix form is equivalent to (1)–(7):

$$z = \min fx \quad (33)$$

Branch-and-Cut Procedure

Input: (VRPTW) and a known incumbent \bar{z}

Output: Optimal solution, z^* , of (VRPTW)

Step 1. Set $z^* = \bar{z}$

Let Ω be the set of enumeration nodes that still need to be explored.

Set Ω to contain the root node of the search tree.

Step 2. **If** $\Omega = \emptyset$ **Then**

Report z^* and **Stop**; otherwise, select enumeration node $\omega \in \Omega$ and **Goto** Step 3.

Step 3. Solve relaxation of corresponding $LP(\mathcal{F})$ defined as

$z_{LP}^k(\mathcal{F}) = \min\{fx : (x, t, y) \in \bar{P}^k\}$. Let (x^k, t^k, y^k) be the optimal solution.

Step 4. **If** $LP(\mathcal{F})$ is infeasible **Then**

Set $\Omega = \Omega \setminus \{\omega\}$ and **Goto** Step 2.

Step 5. **If** $z_{LP}^k(\mathcal{F}) \geq z^*$ **Then**

Set $\Omega = \Omega \setminus \{\omega\}$ and **Goto** Step 2.

Step 6. **If** optimal solution of $LP(\mathcal{F})$ is integer **Then**

Set $z^* = \min\{z^*, z_{LP}^k(\mathcal{F})\}$, $\Omega = \Omega \setminus \{\omega\}$ and **Goto** Step 2.

Step 7. Solve separation problem and let $\mathcal{F}^k \in \mathcal{F}$ be the set of violated inequalities.

Put $\bar{P}^k \leftarrow \bar{P}^k \cap \{(x, t, y) \in \mathcal{R}^\eta : a_1x + a_2t + a_3y \leq a_0 \text{ for } (a_1, a_2, a_3, a_0) \in \mathcal{F}^k\}$.

Let κ be the number of identified violated valid inequalities.

Step 8. **If** $\kappa > 0$ **Then**

Goto Step 3

Step 9. Create new node $\hat{\omega}$ by branching, add it to Ω , put $k \leftarrow k + 1$ and **Goto** Step 3

Figure 7 Branch-and-Cut Outline

$$P = \begin{cases} A_1x \leq a, \\ B_1x + B_2t \leq b, \\ C_1x + C_2y \leq c, \\ D_1y \leq d, \\ x \in X \subset \mathcal{R}^{(n+1) \times (n+1)}, \\ t \in T \subset \mathcal{R}^n, \\ y \in Y \subset \mathcal{R}^n. \end{cases} \quad (34)$$

Using (33) and (34), (VRPTW) can be stated as

$$\min\{fx : (x, t, y) \in \text{conv}(P)\}, \quad (35)$$

where $\text{conv}(P)$ is the convex hull of P . Let \mathcal{F} be a complete system of facet defining inequalities for (VRPTW); i.e.,

$$\mathcal{F} = \{(a_1, a_2, a_3, a_0) \in \mathcal{R}^\eta : a_1x + a_2t + a_3y \leq a_0, (x, t, y) \in P\}, \quad (36)$$

where $\eta = (n+1) \times (n+1)$. Combining (35) and \mathcal{F} , the linear programming relaxation of (VRPTW), denoted

by $LP(\mathcal{F})$, can be written as

$$z_{LP}(\mathcal{F}) = \min fx$$

subject to

$$a_1x + a_2t + a_3y \leq a_0, \quad \forall (a_1, a_2, a_3, a_0) \in \mathcal{F},$$

$$(x, t, y) \in \bar{P} \equiv P \cup \{0 \leq x \leq 1\}.$$

Given a point $(x, t, y) \in \mathcal{R}^\eta$ the separation problem consists of either showing that (x, t, y) satisfies all the facet defining inequalities in \mathcal{F} or finding at least one $(a_1, a_2, a_3, a_0) \in \mathcal{F}$ such that $a_1x + a_2t + a_3y > a_0$.

Branch-and-cut algorithms are based on iteratively solving the separation problem. If the solution of this problem identifies violated inequalities, then they are added to the formulation and the corresponding linear program is reoptimized. Figure 7 summarizes the steps for the overall branch-and-cut approach at a particular iteration k , where $\bar{P}^k \leftarrow \bar{P}$. Step 1 initializes the optimal solution and the set of unexplored nodes of the search tree. Step 2 specifies the termination criterion and selects a node of the search tree for further processing. In our implementation, the node with

the largest objective function value is always chosen. The relaxed linear program (LP) is solved at Step 3. Steps 4 through 6 perform the necessary bookkeeping associated with infeasible, nonoptimal and feasible integer solutions. Step 7 involves the solution of the separation problem. If new violated inequalities are identified, they are added to the model and the LP relaxation is reoptimized. If no new violations are found, a new node is created (Step 9).

The valid inequalities generated at Step 7 greatly depend on the order that the various separation routines are invoked. In our implementation, subtour elimination constraints were identified first, followed by the search for incompatible pair and path inequalities, \vec{D}_k and \overleftarrow{D}_k -inequalities, and finally comb constraints. If no new cuts were found and the current enumeration node, ω , was the root, then a heuristic was used to find violated two-path inequalities (see next section).

4.2. Separation Heuristics for Subtour Elimination Constraints

To begin, we present four heuristics for identifying subsets of nodes that violate (15) or (16). Each was implemented as a subroutine in our code. We then outline a simple heuristic for finding violations of the \overleftarrow{D}_k , \vec{D}_k inequalities in (17) and (18).

The first heuristic works on the undirected graph G and identifies node subsets S for which $x(\delta(S)) < 2\lambda_S$. If $\lambda_S = 1$, then violations of (15) can be found by calculating the min-cut of graph G . If the min-cut ($S : V \setminus S$) is less than two, then S violates (15). Gomory and Hu (1961) provide an $O(n^4)$ algorithm for finding the min-cut of a graph, which solves $n - 1$ max-flow problems along the way. In fact, their algorithm finds the minimum cut between each pair of nodes in the graph which is really more information than we need because we only want to identify a subset S that violates (15). Padberg and Rinaldi (1990a) offer an improved version of the Gomory-Hu algorithm that has the same worst-case complexity but runs faster in practice. For large TSPs, they suggest the use of heuristics because of the slow running time of the exact procedure. Nagamochi and Ibaraki (1992) proposed a simple algorithm for finding the min-cut on an unweighted graph. Frank (1994) then modified it to

achieve slightly greater efficiency. We have conceived a version of Frank's algorithm for weighted directed graphs that has complexity $O(|V||E|)$. In the code, all cuts found at the intermediate steps are evaluated for possible violation of (15) although they might not be the graph's minimum cut. We note that the min-cut algorithm, while exact for identifying violated TSP subtour elimination constraints, is only a heuristic for the VRP. The reason is that a minimum cut on the support graph might not necessarily produce a violation of (15) although a violation might exist somewhere else in the graph, say for a different set S with a larger value of λ_S . Because λ_S always equals one for the TSP, if a violation exists, the min-cut algorithm will always find it.

The second approach applies a graph shrinking heuristic on G similar to that of Araque et al. (1994) to identify sets S that are most likely to violate (15). The shrinking procedure works as follows. Two nodes of G are contracted into one *supernode* by eliminating the edge connecting them. The nodes incident to the contracted nodes become incident to the supernode (see Figure 8). The supernode is assigned a weight that is equal to the sum of the weights of the edges connecting the nodes that form it. Each supernode is considered as the set S in (15). The shrinking process is repeated until the resulting graph consists of one or more disconnected supernodes. The order that nodes are combined to form supernodes determines which constraints are identified. The weight of a supernode corresponds to the left-hand side of (15). Therefore, if one does not consider the lower bound on the number of vehicles, the larger the weight of a supernode the more likely it is for the corresponding subtour constraint to be violated.

In our implementation, a candidate list of high weight edges is constructed and then an edge is

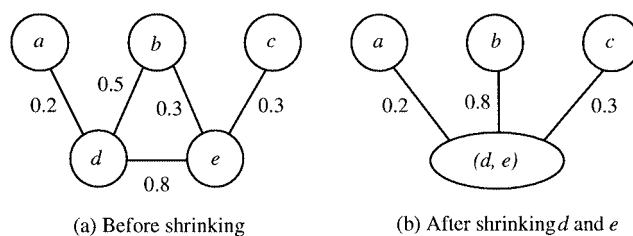


Figure 8 Node Shrinking Procedure

arbitrarily selected for contraction. The shrinking heuristic is applied several times so that different subtour constraints are identified. Randomization serves as a search diversification mechanism. By repeating the randomized shrinking heuristic as little as 10 times a considerably larger number of violated subtour inequalities are found as compared to the deterministic approach. The complexity of the procedure is $O(|V||E|\log|E|)$ because the shrinking routine is repeated at most $|V| - 1$ times and contracting two nodes into a supernode involves $O(|E|)$ edges and $O(\log|E|)$ effort to keep the graph edges sorted. If G is sparse, then the complexity reduces to $O(|V|^2\log|E|)$.

The third heuristic exploits the structure of a feasible solution. The undirected graph G_0 that corresponds to a feasible integer solution consists of one connected component with the depot as the only articulation point of the graph. Similarly, G for a feasible integer solution is a disconnected graph consisting of connected components each of which is an open path. Each path corresponds to a route. For a fractional solution, each component of G_0 not containing the depot violates (15). Also, each biconnected component, S , of G is considered a candidate for a violation of (15).

The fourth heuristic involves finding violated *two-path* inequalities. This type of inequality was introduced by Kohl (1995) and is defined as a subtour elimination constraint of the form (15) such that the set of customers S needs exactly two vehicles to be serviced ($\lambda_S = 2$) but is currently being serviced by one vehicle. In our implementation, we used one of the heuristics proposed by Kohl. We found that it worked well for relatively constrained problems despite its exponential worst-case complexity.

Our investigation of the \vec{D}_k and \overleftarrow{D}_k inequalities led us to consider the nodes of each biconnected component of G as a subtour that might violate (17) or (18). To increase the likelihood of finding a violation, the nodes of each biconnected component are numbered such that the third term of (17) or (18) is maximized. To accomplish this, the node with the highest overall incoming (outgoing) edge weight is selected to be the first node of the set when testing for violated \vec{D}_k inequalities (\overleftarrow{D}_k inequalities).

4.3. Separation Heuristics for Comb Inequalities

To find a comb (as well as box and path-box inequalities) we first identify a set of candidate handles and then for each handle, an appropriate set of teeth is calculated. The procedure is outlined in Figure 9. To begin, a *fractional* graph \mathcal{G} is derived from G after removing all edges of weight less than ε or greater than $1 - \varepsilon$ (Step 1), where ε is a small constant. The biconnected components of \mathcal{G} are found next and serve as building blocks for the subsequent steps (Step 2). A set of candidate handles is constructed by using the biconnected components of \mathcal{G} (Step 3). Each biconnected component with at least three nodes is a handle candidate. In addition, the union of two biconnected components that share a common articulation point is also a handle candidate. For each handle H , a set of teeth candidates is derived by using one of the following two methods. The first identifies edges in G that belong to the cutset $(H, V \setminus H)$ and have weight at least $1 - \varepsilon$. The vertices incident to such an edge become a tooth candidate (Step 4a). The second method considers each element ν of H and constructs a tooth that includes ν and consists of either one more node that lies outside H or a biconnected component of \mathcal{G} that intersects H at ν (Step 4b.1–4b.3). Intersecting teeth are discarded (Step 4c). If the resulting teeth set along with H comprise a comb and either (12) or (19) is violated, then the corresponding inequality is appended to the model (Step 5).

The procedure is not exact. Its complexity depends on the number of biconnected components in \mathcal{G} , which is usually a small constant for instances of large or moderate size. This results in $O(|E|)$ complexity. The same approach outlined in Figure 9 was used to find box and path-box inequalities.

4.4. Separation Heuristic for Incompatible Path Inequalities

The underlying weighted directed graph, \vec{G} , is used to identify paths that violate (32). We define $\vec{G}' = (V \setminus \{0\}, A)$. Given \vec{G}' we define $\vec{\mathcal{G}}'$ by replacing the weight, ω_e , of each edge $e \in A$ by $1 - \omega_e$. Let $i, j \in I, i \rightarrow j, j \rightarrow i$ and $\vec{\mathcal{P}}$ be a directed path from i to j in $\vec{\mathcal{G}}'$. The weight of a directed path $\vec{\mathcal{P}}$ is denoted by $\omega_{\vec{\mathcal{P}}}$.

Procedure for Comb Inequalities

Input: Graph \vec{G} and constant ε

Output: Violated comb inequalities

- Step 1 Let \mathcal{G} be a graph derived from \vec{G} after removing all edges with weights less than ε or greater than $1 - \varepsilon$.
- Step 2 Find set \mathcal{B} of bi-connected components of \mathcal{G} .
- Step 3 Let \mathcal{H} be a set of handle candidates constructed as follows:
Each bi-connected component of \mathcal{G} with at least three nodes belongs to \mathcal{H} ;
The union of any two bi-connected components of \mathcal{G} that share a common articulation point belongs to \mathcal{H} .
- Step 4 **For** each $H \in \mathcal{H}$ find set \mathcal{T} of teeth candidates as follows
- Step 4a Each edge of the cut set $(H, V \setminus H)$ in G with at least $1 - \varepsilon$ weight belongs to \mathcal{T}
- Step 4b **For** each node $\nu \in H$ **Do**
- Step 4b.1 Let $T_1 = \{(\nu, i) : i \notin H\}$ and $e^* \in T_1$ such that $\omega_{e^*} = \max_{e \in T_1} \{\omega_e\}$
- Step 4b.2 Let $T_2 = \{S : S \in \mathcal{B} \text{ and } S \cap H = \{\nu\}\}$ and $B^* \in T_2$ such that $\omega_{B^*} = \max_{B \in T_2} \{\omega_B\}$
- Step 4b.3 **If** $\omega_{e^*} \geq \omega_{B^*}$ **Then** $\mathcal{T} = \mathcal{T} \cup e^*$
Else $\mathcal{T} = \mathcal{T} \cup B^*$
- Step 4c Eliminate from \mathcal{T} intersecting teeth.
- Step 5 **If** $|\mathcal{T}| \geq 3$ and (12) or (19) is violated **Then** add corresponding comb inequality.

Figure 9 **Comb Inequality Identification**

and the violation of (32) for a particular path $\vec{\mathcal{P}}$ is denoted by $\nu_{\vec{\mathcal{P}}}$. We first present a lemma that relates the weight of an incompatible path with the violation of the corresponding valid inequality (32).

LEMMA 2. Let $i, j \in I, i \rightarrow j, j \rightarrow i$ and a path $\vec{\mathcal{P}}$ in $\vec{\mathcal{G}}$ form i to j . Then

$$\nu_{\vec{\mathcal{P}}} = (\max\{2, \lambda_{\vec{\mathcal{P}}}\} - 1) - \omega_{\vec{\mathcal{P}}}. \quad (37)$$

PROOF. Let $\vec{\mathcal{P}} = i, k_1, \dots, k_\psi, j$. We consider two cases, depending on the value of $\lambda_{\vec{\mathcal{P}}}$.

Case 1. $\lambda_{\vec{\mathcal{P}}} = 1$.

The constraint corresponding to (32) is as follows:

$$x_{i, k_1} + x_{k_1, k_2} + \dots + x_{k_\psi, j} \leq |\vec{\mathcal{P}}| - 2. \quad (38)$$

By expressing the cardinality of $\vec{\mathcal{P}}$ as a function of ψ we obtain

$$x_{i, k_1} + x_{k_1, k_2} + \dots + x_{k_\psi, j} \leq (\psi + 2) - 2. \quad (39)$$

The left-hand side of (39) contains $\psi + 1$ terms; therefore, by multiplying both sides of (39) by -1 and then adding $\psi + 1$ to both sides we obtain

$$(1 - x_{i, k_1}) + (1 - x_{k_1, k_2}) + \dots + (1 - x_{k_\psi, j}) \geq 1. \quad (40)$$

The left-hand side of (40) is $\omega_{\vec{\mathcal{P}}}$. Also, the violation of (38) is the difference between the right- and left-hand side; i.e., $\nu_{\vec{\mathcal{P}}} = 1 - \omega_{\vec{\mathcal{P}}}$.

Case 2. $\lambda_{\vec{\mathcal{P}}} > 1$.

The derivation is almost identical to that of the first case. The constraint corresponding to (32) is

$$x_{i, k_1} + x_{k_1, k_2} + \dots + x_{k_\psi, j} \leq |\vec{\mathcal{P}}| - \lambda_{\vec{\mathcal{P}}}.$$

After substituting $\psi + 2$ for $|\vec{\mathcal{P}}|$, multiplying both sides by -1 , and adding $\psi + 1$ to both sides we obtain

$$\omega_{\vec{\mathcal{P}}} \geq \lambda_{\vec{\mathcal{P}}} - 1$$

which leads to $\nu_{\vec{\mathcal{P}}} = (\lambda_{\vec{\mathcal{P}}} - 1) - \omega_{\vec{\mathcal{P}}}$. \square

Our separation procedure for finding incompatible path inequalities is based on finding the shortest directed path in $\vec{\mathcal{G}}$ between nodes i and j where $i \rightarrow j$ and $j \rightarrow i$ (see Figure 10). The complexity of the procedure depends on how many pairs of nodes, (i, j) , exist such that $i \rightarrow j$ and $j \rightarrow i$. In our implementation, Step 2 is preceded by finding the shortest path between all pairs of nodes, resulting into $O(|V|^3)$ complexity.

Procedure for Incompatible Path Inequalities

Input: Graph \bar{G}'

Output: Violated incompatible pair inequalities

Step 1 Construct \bar{G}' from \bar{G}' by replacing the weight, ω_e , of each edge with $1 - \omega_e$.

Step 2 For each $i, j \in I$ such that $i \not\leftrightarrow j$ and $j \rightarrow i$ Do

Step 2a Let $\bar{\mathcal{P}}^*$ be the shortest path from i to j in \bar{G}' .

Step 2b If $\nu_{\bar{\mathcal{P}}^*} > 0$ Then

Step 2c Add incompatible path inequality for path $\bar{\mathcal{P}}^*$.

Figure 10 Incompatible Path Inequality Identification

The proposed procedure is exact as shown by the following lemma.

LEMMA 3. Let $i, j \in I, i \not\leftrightarrow j, j \rightarrow i$ and $\bar{\mathcal{P}}^*$ be the shortest path from i to j in \bar{G}' . Then there exists no other path, $\bar{\mathcal{P}}$, from i to j such that $\lambda_{\bar{\mathcal{P}}} \leq \lambda_{\bar{\mathcal{P}}^*}$ and $\nu_{\bar{\mathcal{P}}} \leq \nu_{\bar{\mathcal{P}}^*}$.

PROOF. The proof is based on Lemma 2. Let $\bar{\mathcal{P}}$ be a path between i and j , then

$$\omega_{\bar{\mathcal{P}}} \geq \omega_{\bar{\mathcal{P}}^*} \quad (41)$$

which leads to

$$1 - \omega_{\bar{\mathcal{P}}} \leq 1 - \omega_{\bar{\mathcal{P}}^*} \quad (42)$$

after multiplying by -1 and adding 1 to both sides of the inequality.

Case 1. $\lambda_{\bar{\mathcal{P}}^*} = 1$.

The right-hand side of (42) is $\nu_{\bar{\mathcal{P}}^*}$. If $\lambda_{\bar{\mathcal{P}}} = 1$, then the left-hand side of (42) is $\nu_{\bar{\mathcal{P}}}$; i.e., $\nu_{\bar{\mathcal{P}}} \leq \nu_{\bar{\mathcal{P}}^*}$.

Case 2. $\lambda_{\bar{\mathcal{P}}^*} > 1$.

Case 2a. $\lambda_{\bar{\mathcal{P}}} = 1$. Then the left-hand side of (42) is $\nu_{\bar{\mathcal{P}}}$. Also the right-hand side cannot be more than $(\lambda_{\bar{\mathcal{P}}^*} - 1) - \omega_{\bar{\mathcal{P}}^*}$ which is $\nu_{\bar{\mathcal{P}}^*}$.

Case 2b. $1 < \lambda_{\bar{\mathcal{P}}} \leq \lambda_{\bar{\mathcal{P}}^*}$. Given that $\lambda_{\bar{\mathcal{P}}} \leq \lambda_{\bar{\mathcal{P}}^*}$ then

$$\lambda_{\bar{\mathcal{P}}} - 1 \leq \lambda_{\bar{\mathcal{P}}^*} - 1. \quad (43)$$

Adding (41) and (43) produces

$$(\lambda_{\bar{\mathcal{P}}} - 1) - \omega_{\bar{\mathcal{P}}} \leq (\lambda_{\bar{\mathcal{P}}^*} - 1) - \omega_{\bar{\mathcal{P}}^*}$$

implying $\nu_{\bar{\mathcal{P}}} \leq \nu_{\bar{\mathcal{P}}^*}$. \square

4.5. Separation Heuristic for Incompatible Pair Inequalities

The separation heuristic for finding incompatible pair inequalities; i.e., violations of (29) or (30), is similar to

the previously described algorithm for incompatible path inequalities. An outline is contained in Figure 11. We start with graph G and derive a new graph \bar{G}' by replacing the weight, ω_e , of each edge $e \in G$ by $\omega_{\max} - \omega_e$, where ω_{\max} is the maximum edge weight in G . Each incompatible pair $i \leftrightarrow j$ of G is considered and if i and j belong in the same connected component in G , then a path connecting them may violate (30). To increase the chance of finding a violated inequality the shortest path between i and j in \bar{G}' is considered. In our implementation, we first find the shortest paths between all pairs of nodes before proceeding to Step 2 of the heuristic. This intermediate step determines the complexity of the procedure which is $O(n^3)$ (the Floyd-Warshal algorithm was used; see Floyd 1962). Finding all-to-all shortest paths is justified especially for highly constrained problems because the number of incompatible node pairs is relatively high.

4.6. Time and Load Constraints

Early testing indicated that constraints (4) and (5) were almost always loose in the solution to the first LP relaxation. The number of these constraints is $O(n^2)$ so their inclusion in the model greatly increases

Procedure for Incompatible Pair Inequalities

Input: Graph G

Output: Violated incompatible pair inequalities

Step 1 Construct \bar{G}'

Step 2 For each $i \not\leftrightarrow j$ Do

Step 2a If i, j belong to the same connected component of G Then

Step 2b Let \mathcal{P} be the shortest path between i and j in \bar{G}'

Step 2c If \mathcal{P} violates (30) Then

Add incompatible pair inequality.

Figure 11 Incompatible Pair Inequality Identification

Ordering Heuristic

Input: LP solution of (VRPTW) and lower bound λ on the number of vehicles

Output: Feasible solution

Step 1 Sort customers in ascending order based on optimal LP solution time variables (t_i).

Step 2 Select seed customers and initialize λ routes.

Step 3 **While** $U \neq \emptyset$ **Do**

Step 3a **If** $\exists i \in T$ and $j \in U$ such that $x_{ij} = 1$ **Then**

Let $u = j$ and $\rho^* =$ route of i .

Else

Let $u \in U$ such that $t_u = \min_{i \in U} \{t_i\}$.

Let ρ^* such that $c_{\rho^*,u} = \min \{c_{\rho,u}\}$.

Step 3b Append u to end of route ρ^* and set $U = U \setminus u$.

Figure 12 Heuristic for Finding Feasible Solutions

the computational effort. To reduce this effort, we omitted (4) and (5) at the first iteration of the B&C algorithm (see Figure 7). At Step 7, violations were identified and added back into the formulation.

We had initially tried replacing (4) and (5) with a set of surrogate constraints obtained for a given $j \in I$ by summing each inequality over $i \in I(j)$, where $I(j) = \{i \in I : x_{ij} \text{ is feasible for } j \in I\}$. In the case of (4), the resultant inequalities are

$$|I(j)|t_j \geq \sum_{i \in I(j)} (t_i + \tau_{ij}x_{ij} - T_{ij}(1 - x_{ij})), \quad j \in I.$$

This approach provided no benefit when compared to solving the full model so it was abandoned in favor of omitting (4) and (5) altogether.

4.7. Lower and Upper Bounds

Effective lower and upper bounding procedures can accelerate the solution process by eliminating part of the solution space. In our computational study we used the lower and upper bounding heuristics of Kontoravdis and Bard (1995). In addition, the following heuristics were developed to convert fractional LP solutions to feasible solutions.

The first is based on the time variables t_i which must be increasing along a route in a feasible solution. This property is exploited to build routes (see Figure 12). To begin, customers are placed in ascending order of their arrival times (Step 1). They are then appended to partial routes starting with those customers having relatively early arrival times. Given a

lower bound λ on the number of vehicles, the λ customers with the earliest arrival times are considered as seed customers to start routes (Step 2). A minimum time (distance) between seed customers is required so that the first customers of the routes are as dispersed as possible. After the initialization step, customers are appended sequentially to the end of existing routes. Let U be the set of customers that are yet to be routed. A customer is selected to be routed based on two criteria (Step 3a). Let T be the set of last customers in the existing partial routes. If there is a $j \in U$ and an $i \in T$ such that $x_{ij} = 1$ in the LP solution, then j is appended after i . Otherwise, the customer u with the earliest arrival time is selected and appended to the end of the partial route ρ^* (Step 3b); ρ^* is chosen to minimize the following cost:

$$c_{\rho,u} = \delta_1(t_u^{(\rho)}/b_u) + \delta_2 t_u^{(\rho)} \tag{44}$$

where $t_u^{(\rho)}$ is the arrival time at u if it is appended to the end of route ρ . If u cannot be feasibly inserted in any existing route, then a new route is introduced. The cost function (44) consists of two components that are appropriately weighted by positive constants δ_1 and δ_2 , such that $\delta_1 + \delta_2 = 1$. The first component provides an estimate of the urgency to visit a particular customer by comparing the arrival time with the end of the corresponding time window. The second component accounts for the time that is allocated for visiting a customer.

The second heuristic is based on iteratively rounding flow variables and reoptimizing the resultant

model. Given a constant γ , a binary variable x_{ij} is rounded up to one if $x_{ij} \geq 1 - \gamma$ and it is rounded down to zero if $x_{ij} < \gamma$. If the resulting model is either infeasible or nonoptimal at a particular iteration, then the heuristic fails. Otherwise, it continues until either an all integer solution is obtained which is better than the incumbent, or no more flow variables can be rounded. In the former case, the heuristic terminates successfully, while in the latter case the corresponding LP solution can be provided to the ordering heuristic to obtain a feasible solution.

The gap between upper and lower bounds can help reduce the number of binary variables (Nemhauser and Wolsey 1988). Let \underline{z} and \bar{z} be a lower and upper bound, respectively, on the optimal LP solution. We denote by \bar{c}_{ij} the reduced cost of the binary flow variable x_{ij} . Then, if x_{ij} is one and $\bar{c}_{ij} < -(\bar{z} - \underline{z})$, then there exists an optimal IP solution for which x_{ij} is equal to one. Similarly, if x_{ij} is zero and $\bar{c}_{ij} > (\bar{z} - \underline{z})$, then there exists an optimal IP solution for which x_{ij} is equal to zero. The above conditions are evaluated at every iteration of the branch-and-cut algorithm so that as many flow variables as possible are fixed.

4.8. Preprocessing

When the time window and/or capacity constraints are tight, a large number of binary variables can be eliminated by identifying incompatible node pairs. When these constraints are not particularly restrictive the number of flow variables can still be reduced by eliminating candidates that are not likely to be part of an optimal solution. The sparsified problem is then solved and at optimality a check is made to see if the objective function can be improved by reinstating any of the eliminated variables. In our implementation, we eliminate a transition from i to j if the direct time from i to j is more than τ^{\max}/ϕ , where τ^{\max} is the maximum time between any two nodes and ϕ is the sparsification factor. The latter is set so that the resulting time matrix contains approximately 20–30% of the $n(n+1)$ transitions. Upon termination to ensure that the optimal solution of the sparsified problem is also the optimal solution of the original problem, the following check is performed. Let z^* be the optimal solution of the sparsified problem and let z_{LP} be the solution of the relaxed sparsified LP at the root node

of the search tree. For each edge (i, j) that is excluded due to sparsification, the following should be true for the optimal solution of the sparsified problem to be the optimal solution of the original problem:

$$\bar{c}_{ij} > z^* - z_{LP}. \quad (45)$$

If a discarded edge does not satisfy (45) then it is added to the formulation and the branch-and-cut procedure is repeated, starting from the current basis.

A second preprocessing step involves reducing customer time windows. Narrow time windows cause many transitions to become infeasible, thus reducing the number of integer variables. In our study we used the time window reduction techniques introduced by Desrochers et al. (1992) and Kontoravdis and Bard (1995). Both procedures are invoked at the beginning of the branch-and-cut algorithm. In addition, each time a flow variable is permanently fixed the window reduction routines are called to check if any time windows can be further reduced.

5. Computational Experience

The proposed branch-and-cut approach was tested on the six problem sets (R1, C1, RC1, R2, C2, RC2) developed by Solomon (1987). Each data set contains from 8 to 12 100-customer problems over a Euclidean 100×100 grid. Customer locations are determined by a random uniform distribution for problem sets R1 and R2, but are restricted to be within clusters for sets C1 and C2. Sets RC1 and RC2 have a combination of clustered and randomly placed customers. Sets R1, C1 and RC1 have a short scheduling horizon with tight time windows; the opposite is true for R2, C2 and RC2. Additional data sets were derived by considering only the first 50 customers of R1, C1, RC1, R2, C2 and RC2 sets. In all instances, only a single type of service is included.

Travel time between customers is equal to the corresponding distances. Both travel time and distance is truncated to one decimal digit. All runs were done on a SUN SPARC-10 workstation. The branch-and-cut algorithm was implemented in C++ and compiled by the SUN C++ compiler with the optimization option (-O) enabled. The CPLEX callable library version 4.0 was used as the LP solver. The quality of a solution is

measured in terms of the minimum number of vehicles. Distance traveled was considered as a secondary objective. It was handled by applying a two-opt distance reduction procedure to the solution associated with the minimum number of vehicles.

The algorithm was terminated either upon obtaining the optimal solution or reaching the maximum allowed CPU time limit which was set to one hour. Various algorithm parameters were set as follows (see Kontoravdis 1997 for a complete description of the implementation details): Branching was done on the time variables after preliminary results showed that this was a more effective strategy than branching on a single flow variable. All violated valid inequalities found by the separation heuristics were added to the formulation regardless of the degree of violation. A valid inequality was considered inactive if its slack was 0.5 or more, and was purged from the model after being inactive for 10 iterations. The randomized variation of the graph contraction heuristic for finding violated subtours (see §4.2) was terminated if either no new inequality was found within five repetitions or a maximum of 200 iterations was reached. A number of flow variables was initially fixed to zero by using the procedure described in §4.7 with the sparsification factor set to five.

To initialize the branch-and-cut algorithm we first obtained a lower and upper bound on the optimal solution by using the lower bounding procedures and the GRASP described in (Kontoravdis and Bard 1995). In a number of instances, this was all that was necessary. Our general strategy, though, was to invoke these procedures at each node of the search tree.

Before presenting the results we need to make a few comments on the issues involved with comparing the proposed branch-and-cut scheme with previous work. It is particularly difficult to compare computation times across different hardware platforms and various compiler and software environments. Because of this we only report our computational results. Another issue relates to the solutions obtained. Comparisons are not straightforward because different methods for calculating time, distance, and the cost of a solution have been used. More specifically, Desrochers et al. (1992) calculated cost using one significant decimal digit and travel time with no decimal digits. In both

cases the remaining decimals were truncated. Kohl (1995) and Kohl and Madsen (1997) on the other hand calculated both cost and travel time using one significant decimal digit and then truncated. We followed their convention. Different calculation conventions have minor impact on most of Solomon's data sets in terms of the solution space. However, for a few data sets, for example R101 and R102, variations of the numerical accuracy lead to totally different optimal solutions due to the fact that some routes that are optimal under one convention are not feasible under the other.

Readers interested in computations based on parallel processing are referred to the work of Rich (1999), and Cook and Rich (1999) who solve the same set of problems as well as instances containing up to 200 customers. Their general approach is similarly based on cutting planes.

5.1. Results for 50-Node Problems

We now present our computations for the 50-node R and RC problems. Results for the C problems are not presented here because all of these instances were solved optimally during the preprocessing step where lower and upper bounds are calculated. Tables 1 and 2 summarize the results. *LB* is the starting lower bound. *Nodes* gives the number of search tree nodes explored before reaching optimality and *Time* is the number of CPU seconds including preprocessing and initialization. The column *Presolver* shows whether the branch-and-cut algorithm was invoked or the problem was optimally solved by the preprocessor. *All cuts* provides the number of valid inequalities identified by the separation procedures until the optimal solution was found. Finally, *Active cuts* corresponds to the number of valid inequalities that were still in the LP formulation upon termination of the branch-and-cut algorithm. For the problems that were solved in the preprocessing step there is no corresponding entry in the *All cuts* and *Active cuts* columns.

All 50-node instances, except RC201, were solved optimally. Most of the R1 and RC1 instances were previously solved by Kohl with the objective of minimizing the distance traveled rather than the number of vehicles. Our solution times were comparable to his. Note that the only real difference between

Table 1 Results for the 50-Node R Problems

Data Set	LB	Optimal	Distance	Nodes	Presolver	Time	All Cuts	Active Cuts
r101.50	11	11	1,156.0	0	Y	12		
r102.50	9	10	933.6	1	N	23	1	1
r103.50	8	8	925.8	0	Y	9		
r104.50	5	6	662.9	49	N	949	3,831	805
r105.50	7	8	1,012.1	18	N	552	2,355	1,275
r106.50	6	7	837.0	75	N	166	1,620	938
r107.50	5	6	788.0	20	N	94	570	521
r108.50	5	6	652.6	67	N	317	1,375	877
r109.50	6	7	827.5	86	N	514	1,744	1,392
r110.50	5	7	781.9	107	N	1,614	956	748
r111.50	5	7	738.0	92	N	993	789	428
r112.50	5	6	699.9	38	N	860	1,750	991
r201.50	2	2	1,014.9	0	Y	15		
r202.50	2	2	895.9	0	Y	13		
r203.50	2	2	784.7	0	Y	16		
r204.50	1	2	560.0	1	N	37	54	19
r205.50	2	2	870.7	0	Y	12		
r206.50	2	2	787.4	0	Y	14		
r207.50	2	2	680.3	0	Y	13		
r208.50	1	2	547.4	17	N	459	1,687	307
r209.50	2	2	761.2	0	Y	7		
r210.50	2	2	826.4	0	Y	11		
r211.50	1	2	642.8	8	N	721	914	683

Table 2 Results for the 50-Node RC Problems

Data Set	LB	Optimal	Distance	Nodes	Presolver	Time	All Cuts	Active Cuts
rc101.50	6	8	944.6	54	N	457	1,591	607
rc102.50	5	7	828.4	86	N	1,593	2,018	781
rc103.50	5	6	714.4	90	N	232	869	184
rc104.50	5	5	579.2	0	Y	13		
rc105.50	6	8	858.8	99	N	1,473	2,256	593
rc106.50	5	6	724.4	68	N	935	1,097	429
rc107.50	5	6	693.1	61	N	589	635	158
rc108.50	5	6	662.0	19	N	175	416	182
rc201.50	2				N			
rc202.50	2	2	884.8	0	Y	11		
rc203.50	1	2	710.7	1	N	41	264	192
rc204.50	1	2	560.4	1	N	29	146	120
rc205.50	2	2	1,114.0	1	N	97	374	248
rc206.50	2	2	891.1	0	Y	9		
rc207.50	1	2	779.1	1	N	52	109	92
rc208.50	1	2	610.8	16	N	809	1,189	471

the two problems is the objective function term so both formulations have the same complexity. If the goal were to minimize distance, one would simply replace $\min \sum_{i=1}^n x_{0i}$ in (1) with $\min \sum_{i=0}^n \sum_{j=0}^n \tau_{ij} x_{ij}$. As mentioned earlier, distance minimization was a secondary objective in our work, and was addressed by a two-opt heuristic. Our solutions for R1 and RC1 had an average gap of 7% and 3.3%, respectively, from the optimal solutions reported by Kohl.

We are not aware of any previous optimal results for the R2 and RC2 instances. Kohl and Madsen only report results for the C1 data sets. Moreover, no attempt was made by Kohl in his thesis to solve any of the R2, C2, or RC2 instances.

5.2. Results for 100-Node Problems

Like the 50-node problems all 100-node C problems were solved optimally in the preprocessing step. In contrast, only a few of the 23 R and 16 RC instances were solved. The results are shown in Table 3. None of the loosely constrained problems R2 and RC2 was solved within the one hour CPU time limit and so are not listed. Our best feasible solutions for these instances can be found in (Kontoravdis and Bard 1995).

In terms of distance, our solutions were an average of 6.15% from the optimal solutions reported by Kohl. This does not take into account problem R103 because Kohl does not report an optimal solution for it. Desrochers et al. (1992) reported optimal solutions for problems R101 and R102. However, due to the fact that they truncated the travel time they obtained an optimal solution for R101, which required one less vehicle than did ours and Kohl's.

Table 3 Results for the 100-Node R and RC Problems

Data Set	LB	Optimal	Distance	Nodes	Presolver	Time	All Cuts	Active Cuts
r101.100	18	19	1,672.4	2	N	392	709	450
r102.100	16	17	1,544.7	7	N	284	542	358
r103.100	13	13	1,389.2	0	Y	34		
r105.100	10	14	1,460.3	128	N	2,713	3,724	1,562
rc101.100	10	15	1,719.2	73	N	548	1,877	695
rc105.100	12	14	1,694.0	87	N	726	1,263	838

6. Discussion

We have presented a new optimization method for minimizing the fleet size of a temporally constrained VRP. Our results indicate that problems with 50 customers can be routinely solved, while larger instances can also be solved if they are sufficiently constrained. The proposed method is general enough to be used for solving other routing problems as well. To minimize travel time or distance, for example, only a few minor modifications are needed in the objective function.

As can be seen from Tables 1–3, the number of cuts added to a problem is anywhere from one to five times the number of cuts in the final LP. This relationship remained fairly constant when we varied the order in which the separation problems were solved. Nevertheless, it would be useful to have some idea of the relative importance of each valid inequality. One way to gauge their effect is to repeatedly solve a set of benchmark instances and use a different class of cuts at each repetition. We initially tried this approach for a small subset of the problems in Table 1 but were unable to solve any of them. This is pretty much in line with Augerat's computational findings which showed that adding only one type of valid inequalities substantially reduces the power of the approach. Consequently, we used the following indicators to assess the importance of each cut: number of inequalities generated, number of active inequalities at the final solution, average slack of active inequalities, and corresponding computation time.

Typical behavior can be inferred by examining the algorithm's performance on the 50-node problem labeled r105.50 in Table 1. In this case, the subtour elimination constraints represented 19% of the overall 3831 inequalities found and 20% of the active inequalities of the final LP. By way of contrast, the incompatible pair inequalities represented 13% of the active constraints in the final LP, the incompatible path inequalities 9%, the combs 4%, and the \vec{D}_k and \overleftarrow{D}_k inequalities 2%. Also, 17% of the departure time constraints (4) and 17% of the pickup constraints (5) were active at the optimum. For those instances that were not presolved, the optimal solution was uncovered, on average, about half way through the search tree. The bulk of the work was in repeatedly solving the LP

relaxations. This consumed approximately 70% of the effort.

The overall results indicated that the subtour elimination constraints were by far the most effective while the combs were the least. This was expected as we used four different heuristics to eliminate subtours and only one for combs. A randomized version of the comb identification procedure would most likely increase their computational impact. Similarly, adding a more sophisticated separation routine for the \vec{D}_k and \overleftarrow{D}_k inequalities should improve the overall performance of the branch-and-cut algorithm. One possibility in this regard would be to do a depth-first search on the undirected support graph to identify all cycles. The directed version of each cycle would then be reconstituted and sequentially tested for a violation of (17) or (18). For a cycle consisting of k nodes, up to k tests could be performed for each constraint. When the support graph is sparse, this procedure should work well but may become too enumerative for densities exceeding 30 or 40%.

References

- Araque, J. R., G. Kudva, T. Morin, J. F. Pekny. 1994. A branch-and-cut algorithm for vehicle routing problems. *Ann. Oper. Res.* **50** 37–59.
- Ascheuer, N., M. Fischetti, M. Grötschel. 1999. Solving the asymmetric travelling salesman problem with time windows by branch and cut. Preprint SC 99-31. *Konard-Zuse-Zentrum für Informationstechnik*, Berlin, Germany.
- Augerat, P. 1995. Approche polyédrale du problème de tournées de véhicules. Ph.D. thesis, Institut National Polytechnique de Grenoble, Grenoble, France.
- Bodin, L. B., B. L. Golden, A. A. Assad, M. O. Ball. 1983. Routing and scheduling of vehicles and crews—The state of the art. *Comput. Oper. Res.* **10** 63–211.
- Cook, W., J. L. Rich. 1999. A parallel cutting plan algorithm for the vehicle routing problem with time windows. Technical report, Computational and Applied Mathematics, Rice University, Houston, TX.
- Cornuéjols, G., F. Harche. 1993. Polyhedral study of the capacitated vehicle routing. *Math. Programming* **60** 21–52.
- Desrochers, M., J. Desrosiers, M. M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* **40** 342–354.
- , J. K. Lenstra, M. W. P. Savelsbergh, F. Soumis. 1988. Vehicle routing with time windows: Optimization and approximation. B. L. Golden, and A. A. Assad, eds. *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam, The Netherlands, 65–84.

- Desrosiers, J., Y. Dumas, M. M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M. O. Ball, T. L. Magnati, C. L. Monma, and G. L. Nemhauser, eds. *Handbooks in Operations Research and Management Science: Network Routing*, vol. 8. North-Holland, Amsterdam, The Netherlands, 35–139.
- Fischetti, M., P. Toth. 1997. A polyhedral approach to the asymmetric travelling salesman problem. *Management Sci.* **43** 1520–1536.
- , J. J. Salazar González, P. Toth. 1998. Solving the orienteering problem through branch-and-cut. *INFORMS J. Comput.* **10** 133–148.
- Floyd, R. 1962. Algorithm 97, shortest path. *Comm. ACM* **5** 345.
- Frank, A. 1994. On the edge-connectivity algorithm of Nagamochi and Ibaraki. Working paper, ARTEMIS-IMAG, Université de Grenoble, Grenoble, France.
- Gomory, R. E., T. C. Hu. 1961. Multiterminal network flows. *SIAM J. Appl. Math.* **9** 551–570.
- Grötschel, M., M. W. Padberg. 1985. Polyhedral theory. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds. *The Traveling Salesman Problem*. Wiley, Chichester, U.K., 251–306.
- Hoffman, K. L., M. W. Padberg. 1993. Solving airline crew scheduling problems by branch-and-cut. *Management Sci.* **39** 657–683.
- Kohl, N. 1995. Exact methods for time constrained routing and related scheduling problems. Ph.D. thesis, Technical University of Denmark, Lyngby, Denmark.
- , O. B. G. Madsen. 1997. An optimization algorithm for the vehicle routing problem with time windows based on Lagrangian relaxation. *Oper. Res.* **45** 395–406.
- , J. Desrosiers, O. B. G. Madsen, M. M. Solomon, F. Soumis. 1999. Two-path cuts for the vehicle routing problem with time windows. *Transportation Sci.* **33** 101–116.
- Kontoravdis, G. 1997. The vehicle routing problem with time windows. Ph.D. thesis, Graduate Program in Operations Research, The University of Texas, Austin, TX.
- , J. F. Bard. 1995. A GRASP for the vehicle routing problem with time windows. *ORSA J. Comput.* **7** 10–23.
- Laporte, G., Y. Nobert. 1984. Comb inequalities for the vehicle routing problem. *Methods Oper. Res.* **51** 271–276.
- Nagamochi, H., T. Ibaraki. 1992. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.* **5** 54–66.
- Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and Combinatorial Optim.* Wiley, New York.
- Padberg, M. W., G. Rinaldi. 1990a. An efficient algorithm for the minimum capacity cut problem. *Math. Programming* **47** 19–36.
- , ———. 1990b. Facet identification for the symmetric traveling salesman polytope. *Math. Programming* **47** 219–257.
- , ———. 1991. A branch-and-cut algorithm for the resolution of large scale symmetric traveling salesman problems. *SIAM Rev.* **33** 60–100.
- Rich, J. L. 1999. A computational study of vehicle routing problem applications. Ph.D. thesis, Computational and Applied Mathematics, Rice University, Houston, TX.
- Sol, M., M. W. P. Savelsbergh. 1994. A branch-and-price algorithm for the pickup and delivery problem with time windows. Memorandum COSOR 94-22. Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands.
- Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Oper. Res.* **35** 254–265.
- , J. Desrosiers. 1988. Time window constrained routing and scheduling problems. *Transportation Sci.* **22** 1–13.

Received: February 2000; revision received: August 2000; accepted: October 2000.