IIE Transactions (1998) **30**, 821–834

A branch and cut algorithm for the VRP with satellite facilities

JONATHAN F. BARD[1], LIU HUANG[2], MOSHE DROR[3] and PATRICK JAILLET[4]

[1]*Graduate Program in Operations Research, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063, USA. Email: jbard@mail.utexas.edu*
[2]*DSC Communications, 11009 Metric Blvd. Austin, Texas 78758, USA. Email: luhang@austin.dsccc.com*
[3]*College of Business and Public Administration, University of Arizona, Tucson, AZ 85721, USA. Email: mdror@bpa.arizona.edu*
[4]*Department of Management Science and Information Systems, The University of Texas at Austin, Austin, TX 78712, USA and Department of Mathematics, ENPC, Paris, France. Email: jaillet@athena.bus.utexas.edu*

Received February 1997 and accepted October 1997

An important aspect of the vehicle routing problem (VRP) that has been largely overlooked is the use of satellite facilities to replenish vehicles during a route. When possible, satellite replenishment allows the drivers to continue making deliveries until the close of their shift without necessarily returning to the central depot. This situation arises primarily in the distribution of fuels and certain retail items. When demand is random, optimizing customer routes *a priori* may result in significant additional costs for a particular realization of demand. Satellite facilities are one way of safeguarding against unexpected demand. This paper presents a branch and cut methodology for solving the VRP with satellite facilities subject to capacity and route time constraints. We begin with a mixed-integer linear programming formulation and then describe a series of valid inequalities that can be used to cut off solutions to the linear programming relaxation. Several separation heuristics are then outlined that are used to generate the cuts. Embedded in the methodology is a VRP heuristic for finding good feasible solutions at each stage of the computations. Results are presented for a set of problems derived from our experience with a leading propane distributor.

1. Introduction

The vehicle routing problem (VRP) finds application in virtually all areas of transportation, from mail collection to airport baggage handling. The particular application that we have in mind involves the restocking of a commodity for customers geographically distributed throughout a service area. Commodities of interest include home heating oil, propane, and automotive parts, to name a few. In each situation, it is assumed that a fleet of vehicles is available at a central depot to service customers over the course of the day. When a driver depletes his supply of commodity, he must return to the depot for replenishment and then continue making deliveries until the end of his shift. A unique aspect of the problem that we investigate is the presence of satellite facilities which the drivers can visit during the day to load up their vehicles with more commodity. In other words, it is not necessary to return to the central depot for replenishment.

The vehicle routing problem with satellite facilities (VRPSF) is a major component of the inventory routing problem (IRP) in which a large number of consumers rely on a central supplier to provide them with a given commodity on a regular basis ([1,2]). The IRP involves the specification of the day of the week on which a particular customer will be visited as well as the determination of a routing sequence to restock his storage tank or bin. The objective is to minimize the annual delivery cost while attempting to ensure that no customer stocks out at any time. In the first [3] of two related papers, we developed an integrated framework for assigning customers to days of the week and solving the resultant routing problems. A key part of this work was the derivation of the incremental cost of serving a customer on a particular day. These costs allowed us to calculate the optimal service interval for each delivery point.

In a companion paper [4], we developed three heuristics to solve the VRPSF. In the integrated methodology, after the daily tours are derived, a parametric analysis is conducted to investigate the tradeoff between distance and annual cost. This leads to the efficient frontier from which the decision maker is free to choose the most attractive alternative. For real-time decision making, the methodology is embedded in a rolling horizon framework where the solution for the first week is implemented and then the problem is re-solved for the next two-week period, and so on.

The purpose of this paper is to present an exact procedure for solving the VRPSF that combines heuristics with polyhedral theory in a branch and cut framework. The approach is grounded in the seminal work of Grötschel and Padberg [5], and Padberg and Rinaldi [6] for the traveling salesman problem (TSP) where instances of up to 1000 customers were solved using a branch and cut algorithm. The basic idea of the approach is to combine implicit enumeration with cutting planes. In this context, the primary goal is to identify *valid inequalities* or cuts at each node of the search tree that do not eliminate any feasible integer points in the underlying constraint region but provide a tighter approximation of the convex hull of the set of all feasible points. A critical aspect of the algorithm is a procedure for finding good feasible solutions or tight upper bounds. For this purpose, we use our modified Clarke–Wright VRPSF heuristic.

In the next section, we discuss the literature on the IRP and mention some of the more efficient exact algorithms for the VRP. This is followed in Section 3 by the presentation of a mixed-integer linear programming (MILP) model of the VRPSF. In Section 4, the complete solution methodology is presented. This includes an outline of polyhedral theory, the steps of the branch and cut algorithm, and the procedures used to solve the separation problems accompanying cut identification. Our computational experience is summarized in Section 5 followed by a discussion of the results in Section 6.

2. Related work

2.1. *Inventory routing problem*

Finding better solutions to the IRP has motivated the work presented here. When satellite facilities are included in the model as they are in many real-life settings, the routing subproblem arguably becomes the most difficult component of the IRP. Not only are routes restricted by capacity and time, but now a decision has to be made regarding where and when to replenish a vehicle. This has the effect of converting a symmetric problem into an asymmetric problem and thus significantly increasing the dimensionality of the mathematical formulation.

Previous to our work with satellite facilities, Dror *et al*. [2] investigated the basic IRP and compared several different computational schemes. They started with a set of customers, where each customer has a storage tank of known capacity and an individual demand function. The objective is to minimize annual delivery costs while guarding against stockouts. Daily consumption rates are assumed to be independent, identically distributed normal random variables with known parameters. The problem is formulated as a two-stage integer program that handles customer selection and scheduling of deliveries by route, truck, and day of the week. The primary solution approach proposed is based on a hierarchical decomposition. Some computation results were presented.

The IRP has both a long-term and short-term component. From a practical point of view, routing has to be done weekly, so in addressing this aspect of the problem Dror *et al*. defined two subsets of customers and two sets of corresponding cost coefficients. The first set of customers are those who must be restocked during the given planning period. The second set are those who are restocked only if there exists a cost savings opportunity to do so. Their solution procedure over an annual time base consisted of a sequence of weekly solutions to the (single period) IRP. The corresponding VRPs were then solved with heuristics that included several exchange procedures for improving the initial routes.

Taking a slightly different approach, Jaillet *et al*. derived [3] the expected total cost of restocking a customer over a given time period, usually a year. It is then possible to determine the optimal frequency d^* to restock a customer by identifying the minimum point on the corresponding cost curve. Our procedure for assigning customers to days of the week over the planning horizon is based on d^*. If the best day to visit a customer is within the planning horizon, the customer is selected. Routing then follows.

2.2. *Exact algorithms for the VRP*

In Laporte [7], several exact procedures for solving VRPs are reviewed. The first is a branch-and-bound scheme due to Laporte *et al*. [8] that uses the solution to a related assignment problem as a lower bound. By exploiting the relationship between the VRP and the m-TSP, and devising effective branching rules, the authors were able to solve to optimality randomly generated asymmetric capacitated VRPs with up to 260 nodes.

The second is due to Christofides *et al*. [9] who investigated the symmetric VRP defined on a planar graph. The algorithm is based on the k-degree center tree relaxation of the m -TSP, where m is fixed. They successfully solved instances ranging in size from 10 to 25 nodes. The third algorithm mentioned uses a dynamic programming formulation, but resorts to a state-space relaxation to obtain lower bounds [7]. The authors investigated problems containing from 10 to 25 nodes and achieved solutions varying from 0 to 7% below the optimum.

In ground breaking work by Desrochers *et al*. [10], the VRP with time windows was modeled as a set partitioning problem and solved with a column generation scheme. The key to the approach was the development of an efficient heuristic for solving a constrained shortest path subproblem that arose at each major iteration. Computational testing showed success on instances with up to 100 customers.

Also addressing the VRP with time windows, Fisher and Jaikumar [11] proposed a three-index vehicle flow

formulation, and used Benders decomposition as the solution methodology. They reported computational results for VRPs ranging from 50 to 199 customers. A two-index formulation was investigated by Laporte *et al.* [12] who optimally solved VRPs containing up to 60 nodes.

3. Mathematical model

The VRPSF that we investigate is defined by a set I of n customers with known nonnegative demand for a given commodity. Each customer must be serviced by exactly one of m homogeneous vehicles located initially at a central depot. We associate with the set I and the depot a directed graph $G = (V, A)$, where the set of nodes V corresponds to the customer set I, the depot 0, and a set of $s \geq 0$ satellite facilities. The set $A \in V \times V$ corresponds to the arcs between the nodes in V. The depot and the s satellite facilities hold an unlimited supply of the given commodity (however, for modeling purposes a limit is placed on the number of replenishment visits as discussed below). The major distinction between the depot and a satellite facility is that the depot serves as the origin and final destination of each vehicle.

To distinguish the individual visits to a satellite facility it is necessary to introduce a unique node for each potential visit (including reloading at the depot). Accordingly, facility α is associated with n_α nodes in V rather than just 1 node. Define:

$$\bar{n} = n + \sum_{\alpha=0}^{s} n_\alpha.$$

In the extended graph that results, the total number of nodes in V (including the depot) is $\bar{n} + 1$. A feasible vehicle delivery sequence starts at the depot, visits a subset of customers dropping off the required amount of commodity to each, reloads perhaps at one of the satellite facilities, continues to another subset of customers, reloads again at a satellite facility, and at the end of the route returns to the depot. An additional requirement for feasibility is that the delivery sequence be completed within T hours.

Indices and sets

- I = set of customers; $I_0 = I \cup \{0\}$, where 0 is the depot;
- i, j = customer indices; $i, j \in I = \{1, \ldots, n\}$;
- F = set of satellite facilities;
- α = index for satellite facilities or the depot when it is being used for reloading; $\alpha \in F = \{1, \ldots, s\} \cup \{0\}$;
- F_α = set of nodes added to graph G corresponding to potential visits to satellite facility $\alpha \in F \cup \{0\}$; F_0 corresponds to visits to the depot for reloading.

Parameters

- d_{ij} = distance from node i to node j, where a node can be a customer, the depot, or a satellite facility;
- τ_{ij} = travel time from node i to node j, where a node can be a customer, the depot, or a satellite facility;
- m = number of vehicles available each day;
- n_α = upper bound on the number of times satellite facility $\alpha \in F$ may be visited; $n_\alpha = |F_\alpha|$ (If $n_\alpha = 0$ then F_α is logically the empty set.);
- Q = capacity of a vehicle;
- $\hat{Q}$ = maximum amount of commodity remaining on a vehicle before a refill is permitted; $\hat{Q} < Q$;
- T = maximum time permitted for a route;
- q_i = demand for customer i;
- p_i = service time at customer i; for $i \in F$, p_i is the vehicle refill time (Although this value may depend on the residual stock, it is treated as a constant.).

Variables

- x_{ij} = binary variable equal to 1 if node i immediately precedes node j; 0 otherwise;
- t_j = time service begins at node j;
- y_j = load on a vehicle remaining to be delivered just before arriving at customer j.

The mathematical formulation for the VRPSF is:

$$\min \quad \sum_{\substack{i,j=0 \\ i \neq j}}^{\bar{n}} d_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{\substack{j=0 \\ j \neq i}}^{\bar{n}} x_{ij} = 1 \quad i = 1, \ldots, n \tag{2}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} \leq 1 \quad i \in F_0 \cup \cdots \cup F_s \tag{3}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{\bar{n}} x_{ji} - \sum_{\substack{i=0 \\ i \neq j}}^{\bar{n}} x_{ij} = 0 \quad j = 0, \ldots, \bar{n} \tag{4}$$

$$\sum_{j=1}^{n} x_{0j} \leq m \tag{5}$$

$$t_j \geq t_i + \tau_{ij} x_{ij} - T_{ij}(1 - x_{ij}) \quad i \neq j; \quad i \in I \cup F_0 \cup \cdots \cup F_s; \; j \in I_0 \cup F_0 \cup \cdots \cup F_s \tag{6}$$

$$\max_{l \in I}(2\tau_{0l} + p_l) \leq t_0 \leq T \tag{7}$$

$$\tau_{0j} \leq t_j \leq T - (p_j + \tau_{j0}) \quad j = 1, \ldots, n \tag{8}$$

$$\min_{l \in I}(\tau_{0l} + p_l + \tau_{lj}) \leq t_j \leq \max_{l \in I}(T - p_j - \tau_{jl} - p_l - \tau_{l0}) \quad j \in F_0 \cup \cdots \cup F_s \tag{9}$$

$$y_j \le y_i - \bar{q}_i x_{ij} + \bar{Q}_i(1 - x_{ij}) \; i \ne j$$
$$i \in I \cup F_0 \cup \cdots \cup F_s; \; j \in I_0 \cup F_0 \cup \cdots \cup F_s \quad (10)$$
$$q_j \le y_j \le Q \quad j = 1, \ldots, n \quad (11)$$
$$0 \le y_j \le \hat{Q} \quad j \in F_0 \cup \cdots \cup F_s \quad (12)$$
$$x_{ij} \in \{0, 1\} \quad \forall i, j \quad (13)$$

where

$$T_{ij} = \begin{cases} T - \tau_{0j} - (p_i + \tau_{i0}) & \text{if } i \in I \\ T - \tau_{0j} - \min_l(p_i + \tau_{il} + p_l + \tau_{l0}) & \\ \quad \text{if } i \in F_0 \cup \cdots \cup F_s,\ i \ne j;\ j \in I_0 \cup F_0 \cup \cdots \cup F_s & \end{cases}$$

$$\bar{q}_i = \begin{cases} q_i & \text{if } i \in I, \\ -Q & \text{if } i \in F_0 \cup \cdots \cup F_s \end{cases}$$

$$\bar{Q}_i = \begin{cases} Q - q_i & \text{if } i \in I, \\ Q & \text{if } i \in F_0 \cup \cdots \cup F_s \end{cases}$$

The objective function (1) minimizes the total distance traveled per day. Equation (2) ensures that each customer has exactly one successor which might be another customer, a satellite facility, or the depot. If the successor were the depot, index j would be either 0 or an element of F_0, depending on whether the route was terminating or the vehicle was stopping to reload. Constraint (3) ensures that each (copy of a) satellite facility will have at most one successor. Equation (4) provides continuity of flow ensuring that the number of arrivals at a node is equal to the number of departures. When j is a customer or a satellite facility other than the depot ($j \in I \cup F_0 \cup \cdots \cup F_s$), each summation term must be equal to 1 implying that each node has exactly one predecessor; when j is the depot ($j = 0$), the summations will be at most m as specified by constraint (5); that is, no more than m vehicles can be used on any day. Note that vehicles and routes are considered to be synonymous.

Constraint (6) is introduced primarily for accounting purposes and tracks the time service begins at node $j \in I_0 \cup F_0 \cup \cdots \cup F_s$. Taken together with the bounds constraints (7) and (8) assure that each route is completed by time T. At t_0, corresponding to $j = 0$, all vehicles must have returned to the depot. Because the t_j variables are increasing, (6) also guarantees that no customer node is visited twice or is part of an isolated cycle. Thus no subtours can form.

In (7), the lower bound on t_0 is calculated by finding the maximum amount of time it takes to reach any customer, service that customer and then return to the depot; the upper bound is the closing time of the routes. In (8), the lower bound on t_j is determined by the time it takes to go from the depot to customer j; the upper bound is the difference between the closing time of the route and the minimum service plus travel time from customer j to the depot. In (9), the lower bound is determined by the minimal amount of time it would take to travel from the depot to a customer, provide service, and then travel to a satellite facility; the upper bound is determined by finding the latest time it would be possible to depart site j, visit another customer l, and then return to the depot arriving no later than T. When j does not represent the first time satellite facility α is visited ($j \in F_\alpha \setminus \{1\}$), the lower bounds in (9) can be strengthened with additional calculations. The parameter T_{ij} is derived in a similar manner by determining the largest possible value t_i can take when i and j are not on the same route, and by noting that $t_j \ge \tau_{0j}$.

The load on a vehicle just prior to visiting node j is tracked by constraint (10). If i is a satellite facility and j is a customer, then the value of y_i is immaterial because $\bar{q}_i = -Q$ permitting the vehicle to take on a full load. This implies that $y_j = Q$. If i and j are not on the same route then $\bar{Q}_i$ represents the largest y_j can be, depending on whether i is a customer or a satellite facility. Constraint (11) bounds y_j when j is a customer; when j is a satellite facility, y_j is restricted in (12) to be less than or equal to a prespecified value $\hat{Q}\,(< Q)$. The above formulation of the daily problem differs from previous representations [2] in that it accounts for satellite facilities and obviates the need to distinguish between routes and vehicles.

4. Solution methodology

The LP relaxation of problem (1)–(13) is obtained by replacing Equation (13) with the constraint $0 \le x_{ij} \le 1$ for all i, j. In the branch and cut approach, this problem is solved at the root node of a search tree and if the solution is integral the algorithm terminates with the optimal solution to the original MILP. If the solution is fractional then valid inequalities or cuts are identified by solving a *separation problem*. These cuts are added to the constraint region and the LP relaxation is re-solved. When the effect of the additional cuts becomes marginal the search tree is extended by branching. Any number of branching schemes are possible. The most common is to fix one of the x_{ij} variables at zero or one; a second is to fix the sum of a subset of variables to zero or one.

The central idea of branch and cut is to obtain an ever tighter representation of the convex hull of the set of feasible integer points by identifying valid inequalities and adding them to the constraint region of the original problem. By definition, these inequalities do not cut off any feasible integer points and hence are valid at every node in the search tree. Note that this is not the case with traditional Gomory fractional cuts. Ideally, we would like the new inequalities to be facets of the convex hull but sufficient theory does not exist to be able to make this determination in most instances.

The most successful applications of branch and cut have been in solving different versions of the TSP [6,13]. Other combinatorial optimization problems for which good re-

sults have been obtained include set partitioning [14] and maximum clique [15]. In the remainder of this section we describe the various components of our algorithm.

4.1. *VRPSF algorithm*

In Nemhauser and Wolsey [16], the branch-and-cut approach is called the fractional cutting-plane algorithm with strong valid inequalities. Our implementation for the VRPSF consists of the following steps:

Step 0 (*Preprocessing*) Set up directed graph and sparsify.
Step 1 (*Upper bound*) Obtain an upper bound on the MILP objective function by running a heuristic to find a feasible solution.
Step 2 (*Lower bound*). Solve the LP relaxation of the VRPSF.
Step 3 (*Optimality check*) If the optimality conditions are satisfied go to Step 8; otherwise go to Step 4.
Step 4 (*Improvement*) If a new integer solution has been obtained, use heuristic post-processor to improve the incumbent upper bound. If an improvement results, update the incumbent and go to Step 3; otherwise go to Step 5.
Step 5 (*Variable fixing*) Determine if any binary variables can be fixed at zero or one.
Step 6 (*Cut generation*) Solve the separation problem to generate valid inequalities. If no such inequalities can be identified go to Step 7; otherwise go to Step 2.
Step 7 (*Branching*) Create a new node in the search tree following the logic of branch and bound. If branching is not possible, go to Step 8; otherwise go to Step 2.
Step 8 (*Desparsification*) If no sparsified variables remain fixed at zero, stop and declare the incumbent the optimal solution to the MILP. Otherwise, introduce a subset of the sparsified variables whose reduced costs are negative back into the model and go to Step 2.

Preprocessing at Step 0 is designed to transform the original mathematical formulation into a tighter approximation and to eliminate symmetries that can slow convergence. First, we convert the symmetric VRPSF into an asymmetric approximation by adding an ε to one of the two arcs associated with each pair of nodes. (It is assumed that all distances are Euclidean and that the triangle inequality holds for all combinations of nodes in the extended graph.) For example, if the distance between node i and node j is d_{ij}, we make $d_{ji} = d_{ij} + \varepsilon$, where $\varepsilon > 0$ is arbitrarily small. This has the effect of giving preference to one direction over the other so tours visiting the same nodes but in opposite directions are no longer equivalent from a computational point of view. However, because ε is negligible, adding a value of ε to an arc length will not affect the optimal solution of the original problem.

The second stage of preprocessing involves *sparsification* of the variable set. The motivation is to eliminate, at least temporarily, those transitions that are not likely to be in an optimal solution. When the distance between node i and node j is greater than a prespecified parameter λ_j we temporarily fix the variable x_{ij} at its lower bound 0. The value of λ_j is chosen to reflect the average distance to customer j and is defined to be twice of the average distance from all other nodes to node j. That is:

$$\lambda_j = \frac{2}{\bar{n}} \sum_{\substack{i=1 \\ i \neq j}}^{\bar{n}+1} d_{ij},$$

for $j = 1, \cdots, n$, where $\bar{n} + 1$ is the total number of nodes in the extended graph. Note that we do not eliminate any transitions from customers to satellite facilities. Finally, redundant constraints are removed.

At Step 1, our VRPSF heuristic is called to provide a feasible solution, and hence an upper bound on the objective function (1). Call it $\bar{z}$. The heuristic provides good feasible solutions to the MILP which are instrumental in fathoming nodes during branch and bound.

Lower bounds on the MILP are obtained at Step 2 by solving the linear programming relaxation of the VRPSF. CPLEX is used in our implementation but any LP solver could be substituted. A check for optimality is made at Step 3. Assuming that no variables have been sparsified, if we are at the root node of the search tree and the solution to the relaxed problem is integer we have the solution to the original problem (1)–(13). Alternatively, if the solution is not integer but the corresponding objective function value $z_{LP} = \bar{z}$, then the heuristic solution is optimal and we terminate. If we are not at the root node but $z_{LP} = \bar{z}$, we can similarly conclude that the optimal solution is at hand. In all other cases, additional work is required.

When a new integer solution is obtained, we try to improve it at Step 4 with the post-processor developed for the heuristic [4]. The basic idea is to swap nodes within and among current routes in an effort to reduce the total length of all the routes and get a tighter upper bound of the problem.

At Step 5, the reduced costs obtained from the LP solution are used to try to fix binary variables at their upper or lower bound and thus shrink the size of the problem. The procedure is based on Proposition 2.1 in Nemhauser and Wolsey [16] which states in terms of a minimization problem:

Let $\bar{c}_j$ be the reduced cost of x_j. If x_j is nonbasic at its lower (upper) bound in the solution of the LP relaxation of an integer program, $x_j \in Z^1_+$, and $z_{LP} + \bar{c}_j \geq \bar{z}$ ($z_{LP} - \bar{c}_j \geq \bar{z}$), there exists an optimal solution to the integer program with x_j at its lower (upper) bound.

Having perhaps reduced the number of variables in the model, the next step is to identify one or more valid inequalities that cut off the current fractional solution. This is done at Step 6 where we solve a separation problem. In particular, we try to identify several of the more common cuts associated with the TSP formulation (recall that all cuts which are valid for the TSP are valid for the VRP). The form of these cuts is given in Section 4.2; in Section 4.3 we present the algorithmic procedures used to generate them.

If we arrive at Step 7, it means that a gap exists between the upper bound $\bar{z}$ and the lower bound z_{LP}, and it is either not possible or not practical to generate additional cuts at the current node. What usually happens at Step 6 is that after several iterations of running the separation routines, diminishing returns set in. That is, the cuts generated have little or no effect in raising the lower bound. Although the current point is removed, the objective function value remains virtually the same as the computations continue. When this condition occurs it is best to begin branch and bound. Because all cuts previously generated remain valid at each node of the search tree, they are simply carried forward and the new LP is optimized.

After the branch and bound component of the algorithm converges, a question remains as to whether the incumbent solution could be improved by introducing any of the variables set to zero during sparsification at Step 0. If the reduced cost of any of those variables is negative, raising their upper bound to 1 would lead to a reduction in the current objective function value implying that the incumbent is not optimal for the original MILP. At Step 8, then, we release all of the sparsified variables whose reduced costs are negative and then re-solve the entire problem.

The algorithm itself has been implemented in C using MINTO (Mixed INTeger Optimizer), a tool developed at Georgia Tech for aiding the solution of structured MIPs [17]. MINTO provides a framework for organizing data, introducing cuts or columns into the formulation, and performing implicit enumeration. It embodies a number of preprocessing routines and can call either CPLEX or OSL to solve the underlying linear programs.

4.2. *Valid inequalities*

In order to identify valid inequalities at Step 6, we need to construct what is known as the support graph. This is done by transferring the LP solution to a directed graph $\vec{G}_0(V,A)$, where $V = I_0 \cup F_0 \cup \cdots \cup F_s$ and $A = \{(i,j) : i,j \in I_0 \cup F_0 \cup \cdots \cup F_s \text{ and } x_{ij} > 0\}$. Each arc (i,j) has an associated weight $w_{ij} = x_{ij}$, implying that there is no arc (i,j) in $\vec{G}_0$ if $x_{ij} = 0$. Therefore, when the solution to the LP relaxation is feasible to the original problem, the support graph comprises a set of feasible tours with each arc weight being 1.

Figure 1 is a support graph obtained from the LP relaxation of a 15-customer, 1-satellite facility problem. Assume all vehicles have a capacity of 200 units. In the graph, 0 represents the depot, s_1 is a satellite facility, and s_2 is the depot being used for replenishment purposes. Thus at most one visit is permitted to the depot and to the satellite facility for reloading. The remaining nodes correspond to customers 1 to 15. The numbers alongside each arc are the values of x_{ij}. As mentioned, if there is no arc between a certain pair of nodes, say i and j, then $x_{ij} = 0$ in the LP solution. Customer demand is given in brackets next to each customer node.

In the remainder of this section, we discuss the two cuts that were implemented.

1. *Subtour elimination constraints*: A feasible solution for the VRPSF consists of a set of disjoint directed cycles each containing the depot and any number of satellite facilities. All other cycles in the graph indicate infeasibility and should be eliminated. Let θ_S be the number of vehicles needed to service the customers in the set S, and let $\bar{S} = V \backslash S$ be the remaining nodes in the support graph including the depot and the satellite facilities. The most common constraint used to eliminate subtours is:

$$\sum_{i \in S} \sum_{\substack{j \in S \\ j \neq i}} x_{ij} \leq |S| - \theta_S, \tag{14}$$

which states that the number of transitions in the set S must be less than or equal to the number of nodes in S minus the number of vehicles needed to service the customers in S. For the TSP, $\theta_S = 1$. What makes the VRPSF more difficult than the traditional VRP is the fact that S is not permitted to contain satellite facilities or the

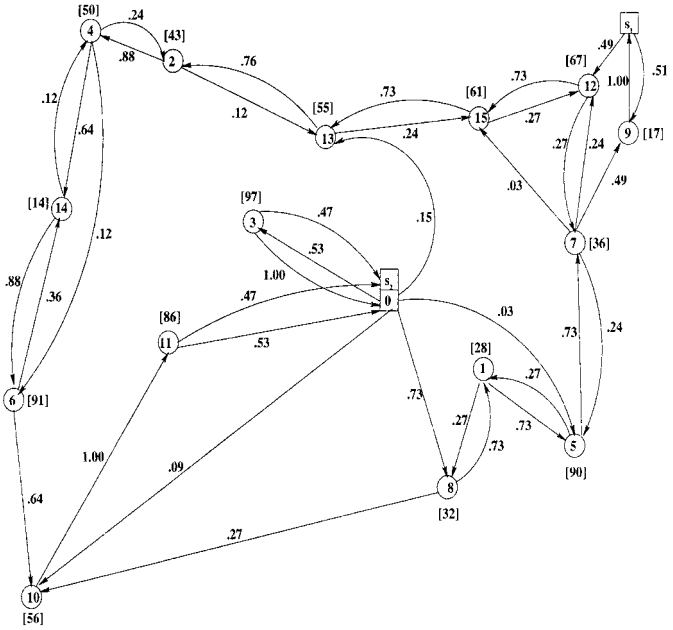

Fig. 1. Example of support graph obtained from LP solution.

depot. For the traditional VRP, if S contains the depot then $\bar{S}$ doesn't, so it is relatively easy, as explained in the next section, to identify good candidates for violation of (14). For the VRPSF, though, the situation is more complicated because it is first necessary to identify a candidate set S without regard to satellite facilities, and if one or more facilities are present then move them to $\bar{S}$. After this rearrangement (14) might no longer be violated.

2. *Lifted $\vec{D}_k$ and $\overleftarrow{D}_k$ inequalities*: Assume we have a directed cycle in the support graph and consider the TSP version of (14); i.e., where the lower bound on the number of vehicles is not taken into account. Grötschel and Padberg [5] have given a sequential lifting procedure that can be used to raise some of the left-hand-side coefficients for cycles of arbitrary length. The resultant constraints are termed $\vec{D}_k$ and $\overleftarrow{D}_k$ inequalities. Lifting is extremely beneficial because it tightens a constraint. Fischetti and Toth [13] have discussed implementation issues and present test results for the TSP.

Figure 2 gives a simple example of how lifting works. In (a) the valid inequality is $x_{12} + x_{23} + x_{31} \leq 2$. After applying the procedure, we get the inequality shown in (b) which can be written as $x_{12} + x_{23} + x_{31} + 2x_{21} \leq 2$. The double arc going from node 2 to node 1 signifies that the coefficient associated with the variables x_{21} is 2. For each cycle of length three, there are three possibilities. For cycles of length four, there are over 40.

In general, a set of k nodes $\{i_1, i_2, \ldots, i_k\} \subset V$, $3 \leq k \leq n-1$, gives rise to the following valid constraints [5]:

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=2}^{k-1} x_{i_j i_1} + \sum_{j=3}^{k-1}\sum_{h=2}^{j-1} x_{i_j i_h} \leq k-1, \quad (15)$$

called $\overleftarrow{D}_k$-inequalities and

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=3}^{k} x_{i_1 i_j} + \sum_{j=4}^{k}\sum_{h=3}^{j-1} x_{i_j i_h} \leq k-1, \quad (16)$$

called $\vec{D}_k$-inequalities. Note that $\overleftarrow{D}_3$ and $\vec{D}_3$ are equivalent.

An example of a $\overleftarrow{D}_k$-inequality can be found in Fig. 1. Consider the node sequence $15 \rightarrow 12 \rightarrow 7$ with arc (12,15) lifted. This gives:

$$\begin{aligned} x_{15,12} + x_{12,7} + x_{7,15} + 2x_{12,15} &= 0.27 + 0.27 + 0.03 + 2(0.73), \\ &= 2.03, \\ &> 3 - 1 = 2, \end{aligned}$$

which is a violation of (15) for $k = 3$. An example of a $\vec{D}_k$-inequality is the sequence $9 \rightarrow 7 \rightarrow 12 \rightarrow s_1$ with arcs (9,12) and (9,s_1) lifted. This gives:

$$\begin{aligned} x_{9,7} + x_{7,12} + x_{12,s_1} + x_{s_1,9} + 2(x_{9,12} + x_{9,s_1}) + x_{s_1,12} &= 0 + 0.24 + 0 + 0.51 + 2(0+1) + 0.49, \\ &= 3.24, \\ &> 4 - 1 = 3. \end{aligned}$$

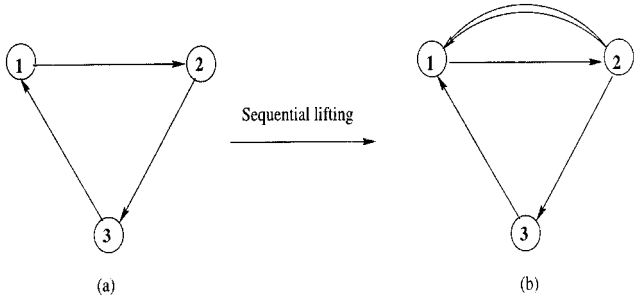

Fig. 2. Example of sequential lifting.

4.3. *Heuristics for the separation problem*

At Step 6 of the branch and cut algorithm an attempt is made to remove fractional LP solutions by solving a separation problem. Because this problem is NP-hard regardless of which of the above violations we are searching for, we resort to heuristics. The first step is to construct the corresponding support graph. In all cases, whether we are trying to identify subtour elimination constraints or $\vec{D}_k$ or $\overleftarrow{D}_k$ inequalities, we start with the undirected version of the graph denoted by $G_0(V, E)$, where $V = I_0 \cup F_0 \cup \cdots \cup F_s$ and $E = \{(i,j) : i,j \in I_0 \cup F_0 \cup \cdots \cup F_s \text{ and } x_{ij} + x_{ji} > 0\}$. In G_0, each edge (i,j) has an associated weight $w_{ij} = x_{ij} + x_{ji}$.

If a valid inequality is found, there still remains a question as to whether it should be added to the formulation. In practice, many negligible violations may be identified at each iteration. Adding each to the LP will greatly increase its size without necessarily tightening the approximation to *conv*(S). For purposes of deciding when to add a new constraint, we consider the *degree of violation*. Specifically, if a valid inequality is of the form:

$$\sum_i \sum_j a_{ij} x_{ij} \leq b,$$

we define the degree of violation δ as:

$$\delta = \frac{\sum_i \sum_j a_{ij} w_{ij} - b}{\sum_i \sum_j a_{ij} w_{ij}},$$

where w_{ij} is the actual value of x_{ij} obtained from the LP solution. Letting $\bar{\delta}$ be a threshold parameter, we add a cut whenever $\delta > \bar{\delta}$. In the code, $\bar{\delta} = 0.05$.

Algorithm for finding subtour elimination inequalities

We begin by searching for violations of Equation (14). A potential violation can be found by identifying the minimum cut on the undirected support graph. Recently, Nagamochi and Ibaraki [18] proposed a simple algorithm

for this purpose which can be shown to be valid for all weighted undirected and directed graphs as well. We use a modified version due to Frank [19] which identifies potential cuts at each iteration. With respect to (14), if the degree of violation of any of these cuts is greater than 0.05, they are added to the LP formulation.

Min-cut algorithm for undirected graph

Step 1 Select any node and construct the *legal ordering* of all remaining nodes as follows: Let S be the set of nodes already ordered. From those nodes not in S, select the one with the maximum weight (largest number of edges for the unweighted case) connected to S. Call it p. Now form a composite node $S \cup \{p\}$ by removing the edges between p and S in the support graph and modifying the edge weights of those nodes connected to $S \cup \{p\}$. For node j not in $S \cup \{p\}$, for example, this modification is done by summing the weights on arcs (j, S) and (j, p). Continue until all nodes are ordered. Break ties arbitrarily.

Step 2 Record the weight of the cut between the last two nodes in the legal ordering. Save as the incumbent if this was the first time, otherwise update the min-cut. If only two nodes remain, stop and reconstruct the solution from the incumbent. Otherwise, contract the last two nodes in the legal ordering (call them p and q) by removing their edges in the support graph and modifying the edge weights of those arcs that have one end in both p and q. That is, merge these edges and increase their weights accordingly.

Step 3 Perform the legal ordering again on the reduced graph always starting with the same node. Go to Step 2.

If we let n be the number of nodes in the support graph and m the number of edges, the above algorithm has complexity $O(nm)$ assuming that the legal ordering at Step 1 is done in $O(m)$ time which is possible. At Step 2 of each iteration, a new subset S is identified that potentially violates (14). The following procedure performs a check.

Violation check

Step 1 If the depot is not in S go to Step 2; otherwise, redefine S as $\bar{S}$ (and *vice versa*).

Step 2 Move all satellite facilities to $\bar{S}$.

Step 3 Calculate δ. If δ is greater than 0.05, add the corresponding inequality to the LP model.

We note that the min-cut algorithm, while exact for identifying violated subtours eliminate constraints for the TSP, 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 (14) although a violation might exist somewhere else in the graph, say for a different set S with a larger value of θ_S. Because θ_S always equals 1 for the TSP, if a violation exists, the min-cut algorithm will always find it.

Algorithm for finding $\bar{D}_k$ and $\vec{D}_k$ inequalities

To search for violations of the $\bar{D}_k$ and $\vec{D}_k$ inequalities, we similarly start with the undirected support graph G_0 and try to identify cycles. This is done with a depth-first search starting at each customer node. At this stage, we are simply looking for an ordered set of nodes $i_1, i_2, \ldots, i_k$ $(3 \leq k \leq n-1)$ without regard to arc direction. For each cycle C found, we see if there is a violation of either the $\bar{D}_k$ and $\vec{D}_k$ inequalities by evaluating Equations (15) and (16), respectively. This step requires a decomposition of the weights w_{ij} into their individual components x_{ij} and x_{ji}. Because the starting point of a cycle is arbitrary and the results depend on the starting point, it is necessary to evaluate the k forward strings $i_1, \ldots, i_k$ through $i_k, i_1, \ldots, i_{k-1}$ for (15) and the k backward strings $i_k, i_{k-1}, \ldots, i_1$ through $i_1, i_k, \ldots, i_2$ for (16). If the degree of violation is greater than the threshold value $\bar{\delta}$, the constraint is added to the model.

Cycle detection

Step 1 Start with all nodes unmarked in G_0. Set $i = 1$.

Step 2 Beginning with node i perform an exhaustive depth-first search and record each cycle found. Go to Step 3.

Step 3 If the cycle has at least three nodes and at least one of the nodes is unmarked, compute the left-hand sides of Equations (15) and (16). Label all nodes in the cycle as marked.

Step 4 If $i < n$, put $i \leftarrow i + 1$ and go to Step 2; else stop.

The application of the above procedure identifies a series of cycles in the undirected support graph. Step 2 is exhaustive in the sense that all possibilities are explored. That is, we try to identify all cycles of length 3 or more. At Step 3, we want to make sure a new cycle results rather than a variation of an existing one. If a cycle does not include the depot, the following procedure is called. Otherwise the cycle is ignored.

Violation check

Step 1 For a given cycle of length k, start with the sequence $(i_1, i_2, \cdots, i_k)$.

Step 2 Using the solution x_{ij} obtained from the current LP, evaluate the left-hand sides of (15) and (16) and save all values greater than $k - 1$.

Step 3 If all $2k$ cases have not been examined index the sequence by one or reverse it, depending on the stage of the calculations, and go to Step 2.

Step 4 Calculate a δ for all inequalities (15) and (16) for which a violation was detected. If $\delta > 0.05$ add the corresponding constraint to the LP model.

After the separation problem is solved and new constraints are added, MINTO checks the activity of the current constraints. If a particular constraint has not been active for 10 iterations, it is removed from the model. The LP is then re-optimized (with CPLEX in our case) and if the solution is still fractional, the separation subroutine is called again. This process continues until no violations are identified that exceed the threshold $\bar{\delta}$.

5. Computational results

To evaluate the branch and cut methodology, three different classes of problems were randomly generated and solved. The classes were defined by 15, 18 and 20 customers, respectively, each with 0, 1 and 2 satellite facilities. To generate data sets, we start by assuming that all customers are uniformly located on a 100×100 grid. The depot is randomly placed in a 40×40 center square of the grid and the satellite facilities are located outside this square in four possible positions. Depending on the number of satellite facilities, each is randomly placed in a 10×10 square at the various corners of the grid. This procedure was designed to reflect the service area of a propane distribution company. For a complete description, see Bard *et al.* [4].

The daily demand for each customer is specified to be uniform between 0 and 1/2 of the truck capacity. The fleet is assumed to be homogeneous with each vehicle having a capacity of 200 units. During a shift, each satellite facility and the depot may be visited two times for reloading. This value was chosen primarily to limit the size of the formulation. For each possible visit to a satellite facility, n binary variables and $O(n^2)$ constraints must be added to the model. Recall that a linear increase in the number of binary variables implies an exponental increase in the computational burden.

Table 1 displays the computational results for the 15-customer problems. The dimensions of each instance are given by n and s, where n is the number of customers and s is the number of satellite facilities. The column labeled *UB* reports the upper bound obtained from our revised Clarke–Wright heuristic. With regard to lower bounds, *LP_obj1* is the relaxed LP solution to the original MILP (1)–(13) while *LP_obj2* is the LP solution obtained just prior to branching. The difference between the two indicates the improvement in the lower bound realized at the first node of the search tree due strictly to the addition of valid inequalities. The best integer solution uncovered is denoted by *Best* and *Gap* gives the percentage difference between the upper and lower bounds at termination. *CPU* (*sec*) gives the total CPU time spent working on all aspects of the problem. All codes were written in C and run on a SUN Sparcstation 10, model 20.

A limit of 3600 CPU seconds was placed on each of the 15 problem instances listed in Table 1. The optimal solution was found in nine of the 15 instances, and in six of those it was found by the heuristic. In the nine other instances, the heuristic was often less than a single percentage point away from the best integer solution found. The largest gap between the upper bound and lower bound is seen to be 4.64% for problem set no. 2, $s = 2$. On average, the conditional gap is 2.3%. When there are no satellite facilities optimality is achieved in all five instances. As expected, the degree of difficulty increases with s. Curiously, though, no correlation was observed between the computational effort and the gap between the upper bound *UB* found by the heuristic and the value of the first LP solution, *LP_obj1*. For problem set no. 4, for example, this gap is essentially the same when $s = 0$ and

Table 1. Test results for the 15-customer problems

Number	*Dimensions*		*B&C*					
	n	*s*	*UB*	*LP_obj1*	*LP_obj2*	*Best*	*Gap (%)*	*CPU (sec)*
1	15	0	251	130.15	177.02	251	0	27
	15	1	251	130.13	163.42	250	0	3119
	15	2	241	130.13	137.01	240	1.04	3600
2	15	0	252	128.39	221.00	252	0	671
	15	1	239	128.18	187.64	233	1.23	3600
	15	2	239	128.18	173.00	237	4.64	3600
3	15	0	288	174.50	250.51	288	0	1058
	15	1	281	174.50	212.00	267	0	70
	15	2	271	174.23	193.60	263	2.63	3600
4	15	0	307	158.62	167.78	307	0	13
	15	1	307	157.58	251.44	307	0	3141
	15	2	290	156.42	262.50	287	0	3073
5	15	0	269	163.90	215.00	269	0	6
	15	1	269	163.90	215.00	268	0.50	3600
	15	2	269	156.38	210.50	256	3.73	3600

$s = 1$; however, the first instance solved in 13 seconds while the second ran 52 minutes before converging.

From a systems point of view, it is important to assess the costs and benefits associated with additional satellite facilities. For problem set no. 5, for example, we see that virtually no advantage is realized when one facility is added but adding two produces a 4.8% drop in total distance traveled. On average, the drop across all problem sets is 5.9%. For a particular application, the system designer would have to determine if the expected incremental benefits, including improved service, resulting from the addition of one or more facilities are sufficient to offset their life-cycle costs.

To provide an indirect assessment of the methodology, we tried to solve several of the problems with MINTO using various combinations of its built-in options. For example, for problem no. 1 (no satellite facilities) with all features activated, MINTO ran for the full 3600 second limit, terminating with a gap of 35%. From Table 1 we see that the branch and cut algorithm found the optimal solution in 27 seconds. This difference is typical of what was observed for other cases and can be partially explained by the fact that the relaxed version of the VRPSF does not produce violations of the types of valid inequalities MINTO is capable of identifying; i.e., clique inequalities, knapsack cover inequalities, and flow cover inequalities. As such, MINTO reduces to a branch and bound algorithm with intelligent preprocessing.

Additional computational measures accompanying the branch and cut algorithm are presented in Table 2. *Dimensions* provides the size of the test case as in Table 1. The column labeled *Nodes* gives the number of nodes created and evaluated in the development of the search tree. *Subtour* indicates the number of subtour elimination constraints added and *Lifted cycles* indicates the number of $\bar{D}_k$ and $\bar{D}_k$ cuts added. The total number of linear programs solved is specified in the column *LPs solved*. The final column ΔLP gives the percentage improvement in the lower bound from the first LP to the last LP solved just prior to termination, and is defined as $([LP\_final - LP\_obj1]/[LP\_final]) \times 100\%$.

As can be seen from the data in Table 2, several thousand nodes were generated for most problem instances. The numbers ranged from 11 for the instance in problem set no. 5 with $s = 0$ to 5727 for the instance in problem set no. 4, $s = 1$. Also, far more subtour elimination constraints were identified than $\bar{D}_k$ and $\bar{D}_k$ inequalities. The number of LPs solved was expectedly proportional to the number of nodes examined (roughly 2 per node). The final column ΔLP gives a good indication of the effectiveness of the cuts. On average, they raised the lower LP bound by 25.2%.

The results for the 18-customer and 20-customer data sets are reported in Tables 3–6. Once again, for each problem class, 15 instances were attempted. A limit of 7200 CPU seconds was placed on the individual runs whether there were zero, one or two satellite facilities in the data set. As seen in Table 3, six out of the 15 instances or 40% were solved to optimality. The largest gap was 3.9% and the average gap for those that didn't converge was 2.71%. Also notice that there is a significant difference between the solution to the first LP (*LP_obj1*) and the final LP at the root node of the search tree (*LP_obj2*). A significant difference is similarly observed between the best integer solution found and (*LP_obj2*). Because the gaps are relatively small or zero, this implies that some branching was necessary to obtain convergence.

Table 2. Characteristics of branch and cut algorithm for the 15-customer case

Number	*Dimensions*		*B&C*				
	n	*s*	*Nodes*	*Subtour*	*Lifted cycles*	*LPs solved*	ΔLP *(%)*
1	15	0	100	208	60	180	48.15
	15	1	5685	5872	656	9104	34.58
	15	2	3906	7086	500	7583	42.31
2	15	0	1697	1593	130	2692	12.30
	15	1	2522	8480	45	5681	18.46
	15	2	2452	8032	58	5661	23.45
3	15	0	2233	2119	180	3622	13.02
	15	1	117	312	51	263	34.64
	15	2	2822	7463	55	5980	24.40
4	15	0	33	79	0	63	45.35
	15	1	5727	7137	302	9297	18.09
	15	2	3461	6128	996	6275	8.54
5	15	0	11	53	13	24	20.07
	15	1	2660	6715	704	5223	19.38
	15	2	1273	6242	868	3368	14.59

Table 3. Test results for the 18-customer problems

Number	*Dimensions*		*B&C*					
	n	*s*	*UB*	*LP_obj1*	*LP_obj2*	*Best*	*Gap (%)*	*CPU (sec)*
1	18	0	288	160.38	244.03	288	0	336
	18	1	282	157.08	216.66	259	0	201
	18	2	282	157.08	238.00	259	0	73
2	18	0	249	153.97	180.89	249	0	10
	18	1	249	151.95	215.60	246	2.10	7200
	18	2	249	151.95	202.00	249	3.71	7200
3	18	0	310	138.99	169.47	310	0	48
	18	1	275	137.94	225.92	275	1.26	7200
	18	2	275	137.94	184.76	275	2.55	7200
4	18	0	263	145.66	198.20	263	3.90	7200
	18	1	251	145.66	205.11	251	3.52	7200
	18	2	251	145.66	190.42	251	3.81	7200
5	18	0	263	167.35	217.28	263	0	137
	18	1	263	166.02	228.39	263	2.50	7200
	18	2	250	166.02	232.00	250	1.10	7200

These observations apply equally to the results in Table 5 where three of the 15 instances for the 20-customer data sets were solved to optimality. Evidently, the more customers in a data set, the harder the problem. The largest gap between the upper and lower bounds was 6.77% and the average gap for the remaining 12 instances that didn't converge was 3.46%.

Tables 4 and 6 summarize the characteristics of branch and cut algorithm for the 18 and five-customer cases. Once again, many more subtour elimination constraints than lifted cycles were identified, and approximately two LPs were solved at each node of the search tree. The average improvement realized from the first LP to the last LP is 17.27 and 18.19%, respectively.

6. Discussion

Our branch and cut algorithm for the VRPSF has four major components: preprocessing, upper bounding, cut identification, and implicit enumeration. The original problem is simplified considerably with preprocessing and sparsification. Because most of the computational effort involves solving linear programs, a tight formulation is essential. A second essential ingredient is good feasible solutions prior to starting the branch and cut algorithm. The test results show that the revised CW heuristic provides optimal or near-optimal solutions in almost every case examined. This is quite common in general for optimal-seeking algorithms, where the majority of the

Table 4. Characteristics of branch and cut algorithm for the 18-customer case

Number	*Dimensions*		*B&C*				
	n	*s*	*Nodes*	*Subtour*	*Lifted cycles*	*LPs solved*	*ΔLP (%)*
1	18	0	565	496	66	877	15.26
	18	1	209	460	36	431	16.34
	18	2	86	298	16	194	8.11
2	18	0	12	79	48	30	27.35
	18	1	4749	17 144	78	9812	10.47
	18	2	2132	9220	156	5221	15.77
3	18	0	81	205	78	173	45.33
	18	1	4537	7936	80	9367	16.80
	18	2	4015	10 102	104	9290	31.06
4	18	0	4206	7457	112	8932	21.58
	18	1	3148	8184	126	7226	15.30
	18	2	2886	7532	150	6834	1.30
5	18	0	237	335	76	405	17.38
	18	1	3065	9055	104	6989	10.93
	18	2	2577	10 394	88	5948	6.17

Table 5. Test results for the 20-customer problems

Number	*Dimensions*		*B&C*					
	n	*s*	*UB*	*LP_obj1*	*LP_obj2*	*Best*	*Gap (%)*	*CPU (sec)*
1	20	0	345	202.98	265.38	345	0	416
	20	1	345	202.98	289.00	340	0	4635
	20	2	345	202.98	294.16	317	0	3486
2	20	0	355	168.97	286.00	335	0.14	7200
	20	1	338	168.97	297.00	324	0.77	7200
	20	2	333	166.06	277.60	333	6.77	7200
3	20	0	269	157.81	225.48	269	1.59	7200
	20	1	269	157.81	183.82	269	3.60	7200
	20	2	237	157.81	170.15	237	1.86	7200
4	20	0	323	149.88	275.00	323	4.34	7200
	20	1	304	149.88	257.38	304	2.97	7200
	20	2	290	149.88	237.11	290	5.52	7200
5	20	0	478	183.55	301.26	478	6.05	7200
	20	1	459	181.62	289.89	459	2.70	7200
	20	2	403	180.92	273.93	402	5.22	7200

computational effort is spent in validating solutions obtained with complementary heuristics.

Finding valid inequalities is at the center of the branch and cut algorithm. In our case, we tried to identify violated subtour elimination constraints and lifted cycles $\bar{D}_k$ and $\vec{D}_k$ to enhance the LP formulation. These inequalities cut off fractional solutions and provide an ever tighter polyhedral representation of the feasible region. When no new cuts can be found or diminishing returns sets in, branch and bound is used to fix variables. The current algorithm solved most 15-customer instances to optimality well within the 3600 second limit imposed, but only 40 and 20% of the 18- and 20-customer instances within 7200 seconds. Nevertheless, for those problems not converging, the gap between the upper and lower bound was always less than 7% and on average less than 3%.

The question remains as to how much additional improvement is possible and at what cost. The place to start might be with the identification of other types of valid inequalities. Combs are an attractive option because they reportedly work well for the TSP; inequalities resulting from paths containing nodes that cannot be on the same route in a feasible solution are another. The latter are quite powerful when there are many incompatible customers in the data set. This may occur, for example, when customers have tight time windows or large individual demands. In our data sets, the absence of time windows and the use of satellite facilities argue against this type of cut.

Table 6. Characteristics of branch and cut algorithm for the 20-customer case

Number	*Dimensions*		*B&C*				
	n	*s*	*Nodes*	*Subtour*	*Lifted cycles*	*LPs solved*	ΔLP *(%)*
1	20	0	705	699	60	1142	23.08
	20	1	3558	9066	142	7397	15.00
	20	2	2135	4907	138	4025	7.20
2	20	0	5652	4504	58	9707	14.63
	20	1	3745	5973	48	8186	7.62
	20	2	2631	5980	28	6096	10.58
3	20	0	3141	3860	112	6262	14.83
	20	1	3945	5531	108	8070	29.11
	20	2	3868	7765	110	8407	26.84
4	20	0	4506	6268	130	9070	1.78
	20	1	3276	6891	114	7099	12.74
	20	2	2620	7182	118	6170	13.46
5	20	0	3545	4067	86	6901	32.91
	20	1	3591	4932	44	7394	35.09
	20	2	2157	3752	70	5317	28.10

The investigation of comb inequalities is left for future research. In practice, they arise quite frequently but may require considerable work to uncover. The set {1, 5, 7, 9, 12, s_1} in Fig. 1, for example, is a violated comb inequality with handle {5, 7, 9} and teeth {1, 5}, {7, 12} and {9, s_1}.

Acknowledgements

This work was partially supported by the Texas Higher Education Coordinating Board under the Advanced Research Program, grant ARP-003.

References

[1] Beltrami, E. and Bodin, L. (1974) Networks and vehicle routing for municipal waste collection. *Networks*, **4**, 65–94.
[2] Dror, M., Ball, M. and Golden, B. (1985) A computational comparison of algorithms for the inventory routing problem. *Annals of Operations Research*, **4**, 3–23.
[3] Jaillet, P., Huang, L., Bard, J.F. and Dror, M. (1998) A rolling horizon framework for the inventory routing problem. Working paper, Department of Management Science and Information systems, University of Texas, Austin, TX.
[4] Bard, J.F., Huang, L., Jaillet, P. and Dror, M. (1998) A decomposition approach to the inventory routing problem with satellite facilities. To appear in *Transportation Science*.
[5] Grötschel, M. and Padberg M. (1985) Polyhedral theory. *The Travelling Salesman Problem*, Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (eds), John Wiley & Sons, Chichester, England, pp. 251–306.
[6] Padberg, M. and Rinaldi, G. (1993) A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, **33**, 60–100.
[7] Laporte, G. (1992) The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, **59**, 345–358.
[8] Laporte, G., Mercure, H. and Nobert, Y. (1986) Exact algorithms for the asymmetrical capacitated vehicle routing problem. *Networks*, **16**, 33–46.
[9] Christofides, N., Mingozzi, A. and Toth, P. (1981) Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming* **20**, 255–282.
[10] Desrochers, M., Desrosiers, J. and Solomon, M.M. (1992) A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, **40**, 342–355.
[11] Fisher, M. and Jaikumar, R. (1981) A generalized assignment heuristic for vehicle routing. *Networks*, **11**, 109–124.
[12] Laporte, G., Norbert, Y. and Desrochers, M. (1985) Optimal routing under capacity and distance restrictions. *Operations Research*, **33**, 1050–1073.
[13] Fischetti, M. and Toth, P. (1997) A polyhedral approach for the exact solution of hard ATSP instances. *Management Science*, **43**(11), 1520–1536.
[14] Hoffman, K.L. and Padberg, M. (1993) Solving airline crew scheduling problem by branch-and-cut. *Management Science*, **39**(6), 657–682.
[15] Balas, E., Ceria, S., Cornuéjols, G. and Pataki, G. (1996) Polyhedral methods for the maximum clique problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **26**, 11–27.
[16] Nemhauser, G.L. and Wolsey, L.A. (1988) *Integer and Combinatorial Optimization*. John Wiley & Sons, New York.
[17] Savelsbergh, M.W.P. and Nemhauser, G.L. (1995) Functional description of MINTO, a Mixed INTeger Optimizer. Georgia Institute of Technology, School of Industrial and Systems Engineering, Altanta, GA.
[18] Nagmochi, H. and Ibaraki, T. (1992) Computing edge-connectivity in multigraphs and capaciated graphs. *SIAM Journal of Descrete Mathematics*, **5**(1), 54–66.
[19] Frank, A. (1994) On the edge-connectivity algorithm of Nagamochi and Ibarkai. Working paper, ARTEMIS-IMAG Universite de Grenoble, Grenoble, France.

Biographies

Jonathan F. Bard is a Professor of Operations Research & Industrial Engineering in the Mechanical Engineering Department at the University of Texas at Austin. He holds the Industrial Properties Corporation Endowed Faculty Fellowship and serves as the graduate advisor for the Manufacturing Systems Engineering Program. He received a D.Sc. in Operations Research from The George Washington University. He has previously taught at the University of California – Berkeley and Northeastern University. Dr. Bard's research interests are in airline operations, the design and analysis of manufacturing systems, R&D project management, and vehicle routing. Prior to beginning his academic career, he worked as a program manager for the Aerospace Corporation, and as a systems engineer for Booz, Allen &Hamilton. He is currently the Editor of *IIE Transactions on Operations Engineering*, an Associate Editor for *Management Science*, and serves on the editorial board of several other journals. He is a Fellow of IIE and a senior member of IEEE and INFORMS. In the past, he has held a number of offices in each of these organizations. His research has been published in a wide variety of technical journals.

Liu (Lucy) Huang received a Ph.D. in Management Science and Information Systems from The University of Texas at Austin. She is currently a software engineer in the Wireless Intelligent Network Division at DSC Communications Corporation, and was formerly a software engineer working on computer-aided dispatch systems at Arrowsmith Technologies, Inc. Prior to that she taught at The University of Electric Science and Technology of China. Her research interests are in network operations, telecommunications, and vehicle routing.

Moshe Dror is a Professor in the MIS Department in the College of Business and Public Administration, University of Arizona. He received an M.Sc. in Mathematical Methods in Engineering and an I.E. (Professional Industrial Engineering degree) from Columbia University, and a Ph.D. in Management Science from the University of Maryland at College Park, with minors in Transportation and Computer Science. He currently serves as *IIE Transactions on Operations Engineering* Department Editor for Applied Optimization and is on the editorial board of *Computers & Operations Research*, and *Foundations of Computing & Decision Sciences*. He co-edited a special issue of *Operations Research* on "Stochastic and Dynamic Models in Transportation" with Warren Powell, and a special issue of the *European Journal of Operational Research* on "Large Scale, High Priced Applications". His current research ranges from combinatorial problems in routing and machine scheduling to problems of cost allocation in logistics, and interactive multiobjective linear programming systems. He is also working on arc routing and inventory distribution-related research in industry.

Patrick Jaillet is the Director of the Institute for Computational Finance, and the Chairman of the Department of Management Science and Information Systems. He is the Fayez Sarofim & Co. Centennial Professor in Business at UT Austin and a Professor in the Department of Civil Engineering at The University of Texas at Austin. He is also

Professor (on leave) in the Mathematics Department at the Ecole Nationale des Ponts et Chaussée, Paris, France. After his undergraduate studies in France (Mathematics and Computer Science, 1981), he received an M.S. (1982) and a Ph.D. (1985) in Applied Mathematics/Operations Research from MIT. Dr. Jaillet's research includes mathematical and optimization models in finance, combinatorial optimization, and real-time models in transportation. His research has been funded by various governmental agencies in both the U.S. and France. He has been a Visiting Scientist at many universities, including MIT and Berkeley, and was a Fulbright Scholar in 1990. Dr. Jaillet is a member of the Institute for Operations Research and the Management Sciences (INFORMS), the Mathematical Programming Society (MPS), and the Society of Industrial and Applied Mathematics (SIAM). He is an Associate Editor for *Operations Research*, an Appointed Reviewer for *Mathematics Abstracts*, and serves on the Canadian Natural Sciences and Engineering Research Council.

Contributed by the Applied Optimization Department