On the Capacitated Vehicle Routing Problem*

T.K. Ralphs[†] L. Kopman[‡] W.R. Pulleyblank[§] and L.E. Trotter, Jr.[¶]

Revised December 17, 2001

Abstract

We consider the Vehicle Routing Problem, in which a fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. This difficult combinatorial problem contains both the Bin Packing Problem and the Traveling Salesman Problem (TSP) as special cases and conceptually lies at the intersection of these two well-studied problems. The capacity constraints of the integer programming formulation of this routing model provide the link between the underlying routing and packing structures. We describe a decomposition-based separation methodology for the capacity constraints that takes advantage of our ability to solve small instances of the TSP efficiently. Specifically, when standard procedures fail to separate a candidate point, we attempt to decompose it into a convex combination of TSP tours; if successful, the tours present in this decomposition are examined for violated capacity constraints; if not, the Farkas Theorem provides a hyperplane separating the point from the TSP polytope. We present some extensions of this basic concept and a general framework within which it can be applied to other combinatorial models. Computational results are given for an implementation within the parallel branch, cut, and price framework SYMPHONY.

1 Introduction

We consider the *Vehicle Routing Problem* (VRP), introduced by Dantzig and Ramser [17], in which a quantity d_i of a single commodity is to be delivered to each customer $i \in N = \{1, \cdots, n\}$ from a central depot $\{0\}$ using k independent delivery vehicles of identical capacity C. Delivery is to be accomplished at minimum total cost, with $c_{ij} \geq 0$ denoting the transit cost from i to j, for $0 \leq i, j \leq n$. The cost structure is assumed *symmetric*, i.e., $c_{ij} = c_{ji}$ and $c_{ii} = 0$.

Combinatorially, a solution for this problem consists of a partition of N into k *routes* $\{R_1, \ldots, R_k\}$, each satisfying $\sum_{j \in R_i} d_j \leq C$, and a corresponding permutation σ_i of each

*This research was partially supported by NSF Grant DMS-9527124 and Texas ATP Grant 97-3604-010

[†]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18017, tkralphs@lehigh.edu, http://www.lehigh.edu/~tkr2

[‡]School of OR&IE, Cornell University, Ithaca, NY 14853

[§]Exploratory Server Systems and The Deep Computing Institute, IBM Research, Yorktown Heights, NY 10598

[¶]School of OR&IE, Cornell University, Ithaca, NY 14853, ltrotter@cs.cornell.edu

route specifying the service ordering. This problem is naturally associated with the complete undirected graph consisting of nodes $N \cup \{0\}$, edges E, and edge-traversal costs c_{ij}, $\{i,j\} \in E$. In this graph, a solution is the union of k cycles whose only intersection is the depot node. Each cycle corresponds to the route serviced by one of the k vehicles. By associating a binary variable with each edge in the graph, we obtain the following integer programming formulation:

$$
\begin{aligned}
min \quad & \sum_{e \in E} c_e x_e & & \\
\sum_{e=\{0,j\} \in E} x_e & = 2k & & (1) \\
\sum_{e=\{i,j\} \in E} x_e & = 2 \quad \forall i \in N & & (2) \\
\sum_{\substack{e=\{i,j\} \in E \\ i \in S, j \notin S}} x_e & \geq 2b(S) \quad \forall S \subset N,\ |S| > 1 & & (3) \\
0 \leq x_e & \leq 1 \quad \forall e = \{i,j\} \in E,\ i,j \neq 0 & & (4) \\
0 \leq x_e & \leq 2 \quad \forall e = \{0,j\} \in E & & (5) \\
x_e & \quad integral \quad \forall e \in E. & & (6)
\end{aligned}
$$

For ease of computation, we define $b(S) = \left\lceil \left(\sum_{i \in S} d_i \right) / C \right\rceil$, an obvious lower bound on the number of trucks needed to service the customers in set S. Constraints (1) and (2) are the *degree constraints*. Constraints (3) can be viewed as a generalization of the subtour elimination constraints from the TSP and serve to enforce the connectivity of the solution, as well as to ensure that no route has total demand exceeding the capacity C. A (possibly) stronger inequality may be obtained by computing the solution to a Bin Packing Problem (BPP) with the customer demands in set S being packed into bins of size C. Further strengthening is also possible (see [15]). We will refer to the inequalities (3) as the *capacity constraints*.

It is clear from our description that the VRP is closely related to two difficult combinatorial problems. By setting $C = \infty$, we get an instance of the Multiple Traveling Salesman Problem (MTSP). An MTSP instance can be transformed into an equivalent TSP instance by adjoining to the graph $k-1$ additional copies of node 0 and its incident edges (there are no edges among the k depot nodes). On the other hand, the question of whether there exists a feasible solution for a given instance of the VRP is an instance of the BPP. The decision version of this problem is conceptually equivalent to a VRP model in which all edge costs are taken to be zero (so that all feasible solutions have the same cost). Hence, we can think of the first transformation as relaxing the underlying packing (BPP) structure and the second transformation as relaxing the underlying routing (TSP) structure. A feasible solution to the full problem is a TSP tour (in the expanded graph) that also satisfies the packing constraints (i.e., that the total demand along each of the k segments joining successive copies of the depot does not exceed C).

Because of the interplay between the two underlying models, instances of the Vehicle Routing Problem can be extremely difficult to solve in practice. In fact, the largest solvable instances of the VRP are two orders of magnitude smaller than those of the TSP. Exact solution of the VRP thus presents an interesting challenge to which various approaches have been proposed. The use of *1-tree* Lagrangian relaxation as a TSP solution procedure [22] was extended to the VRP using *k-trees* in [14]. Set partitioning also provides a natural approach for solving the VRP (see [10, 20, 16, 19, 2]). In this formulation, for each $S \subseteq N$ one assigns cost c_S as the TSP tour cost for S, provided $\sum_{i \in S} d_i \leq C$, and $c_S = \infty$, otherwise. Then one seeks the minimum cost k-partition of N. Finally, various implementations of branch and bound and branch and cut for the VRP have been investigated in [27, 13, 14, 3, 9, 11]. We will address branch and cut methods further in the remainder of the paper.

Intuitively, what makes this problem much more difficult than the TSP is that the cost structure is dictated purely by routing considerations and does not account for the packing structure. Hence, the routing and packing requirements are sometimes in conflict. This observation suggests the exploration of decomposition-based optimization techniques involving relaxation of one or the other of the underlying structures. In this paper, we investigate a method by which we attempt in a novel way to isolate the TSP structure from that of the BPP in order to take advantage of known techniques for optimizing over the TSP polytope. By examining the relationship between the TSP, the BPP, and the VRP, we develop a separation routine for the capacity constraints and other classes of valid inequalities. This has led us to a general approach for separation that can be applied to combinatorial optimization problems that either have difficult side constraints or are the intersection of two underlying models.

In what follows, we will discuss both theoretical and implementational aspects of this and other procedures and present computational results with the SYMPHONY framework for parallel branch, cut, and price. We will not discuss the details of the underlying framework. For more information on SYMPHONY and the branch and cut algorithm itself, the reader is referred to the *SYMPHONY User's Guide* [32] and [25]. The computational results reported here are still preliminary. However, the theoretical framework is well-developed and has wide applicability.

2 Separation of the Capacity Constraints

The most important and challenging aspect of any branch and cut algorithm is designing subroutines that effectively separate a given fractional point from the convex hull of integer solutions. This has been, and still remains, a very challenging aspect of applying branch and cut techniques to the VRP. Many classes of valid inequalities for the VRP polytope have been reported in the literature (see [26, 27, 1, 12, 15, 28, 9, 8, 11]), but the separation problem remains difficult to solve for most known classes. Classes that can be effectively separated tend to be ineffective in the context of branch and cut.

For the remainder of the paper, we will focus primarily on the problem of separating an arbitrary fractional point from the VRP polytope using the capacity constraints (3). In the IP formulation described by (1)-(6), the capacity constraints provide the link between the packing and routing structure and are hence understandably important in describing the polyhedral structure of the VRP polytope. However, the separation problem for these constraints was shown to be $\mathcal{NP}$-complete by Harche and Rinaldi (see [9]) even if $b(S)$ is taken to be $\lceil (\sum_{i \in S} d_i)/C \rceil$. This means that solving the complete LP relaxation of our formulation is, unfortunately, an $\mathcal{NP}$-complete problem.

Because of the apparent intractability of solving the separation problem, a good deal of effort has been devoted in the literature to developing effective separation heuristics for these constraints. However, until the paper by Augerat, et al. [9], most known algorithms were not very effective at locating violated capacity constraints. In [8], it is shown that the fractional version of these constraints (i.e., with $b(S)$ replaced by $\left(\sum_{i \in S} d_i\right)/C$) is polynomially separable. This method can be used as a heuristic for finding violated constraints of the form (3).

Let us return to the model suggested in Sect. 1 in which the TSP is viewed as a relaxation of the VRP on a slightly expanded graph. As indicated, a TSP tour provides a feasible VRP solution when it also yields a feasible solution to the BPP. We denote by $\mathcal{T}$ the TSP polytope, i.e., the convex hull of incidence vectors of all TSP tours, and by $\mathcal{R}$ the polytope generated by the incidence vectors of tours corresponding to feasible VRP solutions. Thus, $\mathcal{R} \subseteq \mathcal{T}$ and the extremal elements of $\mathcal{R}$ are among those of $\mathcal{T}$.

Suppose that at some node in the branch and cut search tree, the LP solver has returned an optimal solution $\hat{x}$ to the current LP relaxation. We define the *support graph* or *fractional graph* corresponding to $\hat{x}$ to be $\hat{G} = (N \cup \{0\}, \hat{E})$ where $\hat{E} = \{e : \hat{x}_e > 0\}$. Suppose that $\hat{x}$ is integral. If $\hat{x} \notin \mathcal{T}$, then $\hat{G}$ must be disconnected since the degree constraints are included explicitly in each LP relaxation. In this case, the set of nodes in any connected component of $\hat{G}$ that does not include the depot induces a violated capacity constraint, so we *CUT*; that is, we add this inequality to the LP and re-optimize. On the other hand, when $\hat{x} \in \mathcal{T}$, we consider whether $\hat{x} \in \mathcal{R}$. If not, then $\hat{x}$ must again violate a capacity constraint induced by the nodes of some depot-to-depot segment of the tour corresponding to $\hat{x}$, so we CUT. Finally, when $\hat{x} \in \mathcal{R}$, then $\hat{x}$ provides a feasible solution to the VRP and investigation of the current search node terminates. Thus we assume henceforth that $\hat{x}$ is not integer-valued.

2.1 Heuristics

Because of the difficulty of this separation problem, we first apply several simple heuristics in an attempt to determine a capacity constraint violated by $\hat{x}$. These heuristics work within the support graph $\hat{G}$ described above and first appeared in [31], though some were also discovered independently by other authors (see [8]). As discussed above, we assume in what follows that this graph is connected, and we associate with it a vector $\omega \in \mathbb{R}^{\hat{E}}$ of edge weights whose components are defined by $\omega_e = \hat{x}_e$. In what follows, we will also use the notation

$$\omega(F) = \sum_{e \in F} \omega_e, \ F \subseteq \hat{E}, \text{ and} \tag{7}$$

$$\delta(S) = \{\{i,j\} \in \hat{E} : i \in S, j \notin S\}, \ S \subseteq N \cup \{0\}. \tag{8}$$

The *connected components heuristic* considers one-by-one the connected components of the support graph after removing the depot. Suppose S is the node set of such a component. If $\hat{x}$ violates the capacity restriction determined by S, we CUT. If not, we replace $S \leftarrow S \backslash \{v\}$, with $v \in S$ chosen so that violation becomes more likely after its removal, and

Connected Components Heuristic
Input: The support graph $\hat{G}$ of $\hat{x} \in \mathbb{R}^E$.
Output: A set $\mathcal{C}$ of capacity constraints violated by $\hat{x}$.
Step 1. Construct the node sets $C_1, \ldots, C_p$ of the maximal connected components of $\hat{G} \setminus \{0\}$. Set $i \leftarrow 0$.
Step 2. Set $i \leftarrow i+1$. If $\omega(\delta(C_i)) < 2b(C_i)$, then add this violated constraint to $\mathcal{C}$.
Step 3. Otherwise, attempt to locate a subset of C_i that induces a violated constraint as follows. Determine a node $v \in C_i$ such that $b(C_i \setminus \{v\}) = b(C_i)$ and $\omega(\delta(C_i \setminus \{v\})) - 2b(C_i \setminus \{v\}) < \omega(\delta(C_i)) - 2b(C_i)$. If no such node exists, go to Step 2.
Step 4. Otherwise, if $C_i \setminus \{v\}$ induces a violated constraint, add this constraint to $\mathcal{C}$ and continue with Step 2 as long as $i < p$.
Step 5. If $C_i \setminus \{v\}$ is nonempty, set $C_i \leftarrow C_i \setminus \{v\}$ and repeat Step 3. Otherwise, continue with Step 2 as long as $i < p$.

Figure 1: The connected components heuristic

the procedure iterates until no such node can be found. If the process discovers no violated inequality, we move on to consider the next component of the support graph. A variant, called the *2-connected components heuristic*, which begins with the 2-edge connected components (those for which the removal of at least two edges is required to disconnect the component) of the support graph after the depot's removal, was also used. The first of these algorithms is presented in Fig. 1.

The *shrinking heuristic* begins by considering the edges of the support graph. If the two end nodes of any edge not adjacent to the depot determine a capacity constraint violated by $\hat{x}$ (i.e., the set $S = \{i, j\}$ induces a violated capacity constraint), we CUT. If not, suppose and edge $e = \{i, j\}$ not adjacent to the depot satisfies $\omega_e \geq 1$. In this case, we *shrink*, or *contract*, edge e in the usual manner, identifying its end nodes and summing components of ω for edges which are identified in consequence. In the resulting *contracted graph*, the two endpoints that are identified form a *supernode*, which is associated with a subset of N. We say that the supernode *contains* the nodes in its associated subset. The demand associated with this new supernode is the sum of the demands of the nodes it contains. When two supernodes i and j are identified, the resulting supernode contains the union of the subsets associated with i and j. Hence, the nodes of a contracted graph define a partition of the original node set with each node in the contracted graph associated with one member of this partition.

It is not difficult to see that this shrinking process does not interfere with violated capacity constraints—if S induces a violated capacity constraint, then there must exist a set S', with either $i, j \in S'$ or $i, j \notin S'$, which also induces a violated capacity constraint. Thus the heuristic proceeds iteratively, alternately shrinking an edge e for which $\omega_e \geq 1$ and checking whether any pair of end nodes determines a violated capacity constraint. If a violated constraint is produced, we CUT; if not, the procedure iterates until every edge

Shrinking Heuristic
Input: The support graph $\hat{G}$ of $\hat{x} \in \mathbb{R}^E$.
Output: A set $\mathcal{C}$ of capacity constraints violated by $\hat{x}$.
Step 1. If $\exists e = \{i,j\} \in \hat{E}$ such that e is not adjacent to the depot and $\omega_e > 2 - b(\{i,j\})$, then $S = \{i,j\}$ induces a violated capacity constraint. Add this constraint to $\mathcal{C}$. If $\omega_e < 1 \; \forall e \in \hat{E}$ or every remaining edge is adjacent to the depot, then STOP.
Step 2. Select $e \in \hat{E}$ with $\omega_e \geq 1$ such that e is not adjacent to the depot, and shrink e to produce $\hat{G}_e$. Set $\hat{G} \leftarrow \hat{G}_e$ and update ω appropriately. Go to Step 1.

Figure 2: The shrinking heuristic

e is either adjacent to the depot or satisfies $\omega_e < 1$. Note that in a contracted graph, it is entirely possible that $\omega_e > 1$ for some edge e. The algorithm is described in Fig. 2.

The *extended shrinking heuristic* is based on the minimum cut algorithm suggested by Nagamochi and Ibaraki in [30] and is similar in spirit to the shrinking heuristic. In this extension of the basic algorithm, shrinking continues with edges of weight less than one in the order prescribed by the algorithm of [30]. As in the shrinking heuristic, each edge is checked before contraction to determine if it induces a violated capacity constraint. Because the sequence of edges is chosen in such a way that the weight of each cut examined is "small," this algorithm may lead to the discovery of violated capacity constraints not discovered by other heuristics.

If the above heuristics fail to locate a violated capacity constraint, we apply the *greedy shrinking heuristic* first described in [9]. In this heuristic, we begin with a small set of nodes S, selected either at random or by one of several heuristic rules. In each iteration, we try to grow the set S in a greedy way by adding a node $j \notin S$ such that $\sum_{e \in \{\{i,j\} \in E : i \in S\}} x_e$ is maximized (and hence $\sum_{e \in \delta(S)} x_e$ is minimized). Notice the contrast between this strategy and that taken by the *connected components heuristic*, where a large initial set is shrunk using a similar rule.

2.2 The Decomposition Algorithm

If the heuristics described above have failed to produce a violated capacity constraint, then we resort to a *decomposition algorithm*, which originally appeared in [31]. At a high level, the algorithm can be described in the following terms. Given a fractional solution $\hat{x}$ to some LP relaxation, we attempt to determine whether $\hat{x}$ lies within $\mathcal{T}$ by expressing $\hat{x}$ as a convex combination of incidence vectors of tours, i.e., of extreme points of $\mathcal{T}$. More precisely, where T denotes the matrix whose columns are the extreme points of $\mathcal{T}$, we are asking whether the linear program

$$\max\{\mathbf{0}^\top \lambda : T\lambda = \hat{x},\ \mathbf{1}^\top \lambda = 1,\ \lambda \geq 0\}, \tag{9}$$

has a finite optimum. If such a decomposition of $\hat{x}$ into a convex combination of tour vectors is possible, Carathéodory's Theorem assures only a modest number of columns of T will be required. Generating the matrix T is, needless to say, difficult at best, and the reader should for the moment put aside consideration of exactly *how* this is to be accomplished.

Note that when $\hat{x}$ violates a capacity constraint and $\hat{x}$ is a convex combination of tour vectors, then some member of the decomposition must also violate that capacity restriction. Thus, given such a convex representation for $\hat{x}$, we examine the depot-to-depot segments of its tours in an attempt to find a violated capacity restriction. Sect. 2.6 describes in more detail how this is done (see also Fig. 4). If successful, the violated inequality is returned and we CUT. If not, i.e., when there is a convex decomposition of $\hat{x}$ (hence $\hat{x} \in \mathcal{T}$) and we find no evident capacity violation, then separation fails and we must BRANCH. An example of a decomposition is shown in Fig. 3. The fractional graph in this figure is composed of 1/3 of each of the three tours shown below it. The fractional edges are dashed and have their values listed next to them. The values listed next to the nodes are their associated demands. In this case, the capacity C is 6000, so the cut indicated in the fractional graph induces a violated capacity constraint. This violated constraint is found by examining the top route of the middle tour in the figure.

When no convex decomposition of $\hat{x}$ exists, then $\hat{x} \notin \mathcal{T}$ and the Farkas Theorem provides a hyperplane separating $\hat{x}$ from $\mathcal{T}$. Specifically, in this case there must exist a vector a and scalar α for which $at \geq \alpha$ for each column t of T, yet $a\hat{x} < \alpha$. This inequality corresponds to a row of the current basis inverse when solving the LP defined in (9) and can be readily obtained. The inequality $ax \geq \alpha$ is returned to CUT, and the process iterates.

We suggest two means for dealing with the intractability of matrix T. One approach is to restrict the column set of the matrix T. Note that any tour t present in a convex decomposition of $\hat{x}$ must *conform* to $\hat{x}$; i.e., t must satisfy

$$t_e = 1 \text{ for each } e \text{ such that } \hat{x}_e = 1; \tag{10}$$

$$t_e = 0 \text{ for each } e \text{ such that } \hat{x}_e = 0. \tag{11}$$

Thus one obvious means of restriction is to simply require that (10) and (11) hold for all columns of T. When the *fractional support* of $\hat{x}$, the set of edges e for which $0 < \hat{x}_e < 1$, is of small cardinality, we can simply enumerate all such tours using depth-first search and thereby create T *explicitly*. The disadvantage of this approach is that any resulting Farkas inequalities must be lifted in order to be made valid, another computationally intensive task. The details of how this is done are given in Sect. 2.4.

A second approach is to use column generation to handle T *implicitly*. Here T is initially comprised of only a partial collection of tours. The algorithm proceeds as before, asking whether $\hat{x}$ is a convex combination of the columns of T. When a convex representation is found among the columns of T, we again check whether any of its tours determines a violated capacity constraint. If so, we CUT. As before, when there is no convex decomposition of $\hat{x}$ using the known tours, the Farkas Theorem provides (a, α) for which $at \geq \alpha$ for each column t of T, but $a\hat{x} < \alpha$. Now we minimize over $\mathcal{T}$ with cost vector a using TSP optimization techniques. Suppose t^* is the incidence vector of the optimal tour which results. If $at^* \geq \alpha$, then the minimization guarantees that $ax \geq \alpha$ separates $\hat{x}$ from $\mathcal{T}$, so this inequality is passed to CUT. If $at^* < \alpha$, then t^* is not among the columns of T, so t^* is appended to T and we ask again whether a convex decomposition of $\hat{x}$ can be obtained from the columns

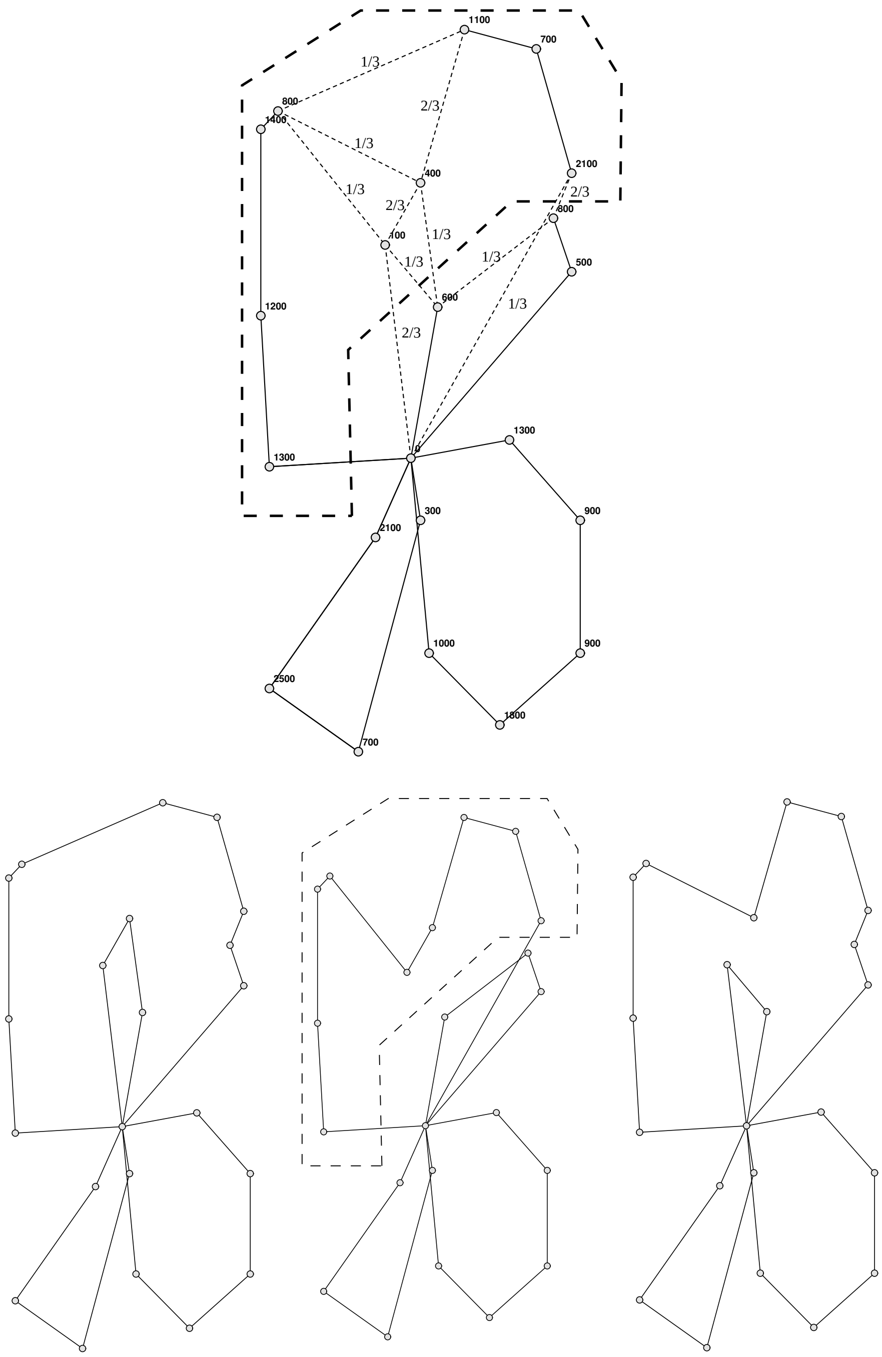

Figure 3: The decomposition of a fractional graph

Decomposition Algorithm (High Level Description)
Input: $\hat{x} \in \mathbb{R}^E$
Output: A valid inequality for $\mathcal{P}'$ which is violated by $\hat{x}$, if one is found.
Step 0. Apply separation algorithms and heuristics for $\mathcal{P}$ and $\mathcal{P}'$. If one of these returns a violated inequality, then STOP and output the violated inequality.
Step 1. Otherwise, solve the linear program

$$\max\{\mathbf{0}^\top \lambda : T\lambda = \hat{x},\ \mathbf{1}^\top \lambda = 1,\ \lambda \geq 0\}, \tag{12}$$

where T is a matrix whose columns are the incidence vectors of members of $\mathcal{F}$.
Step 2. The result of Step 1 will be either (1) a subset D of the columns of T participating in a convex combination of $\hat{x}$; or (2) a valid inequality (a, β) for $\mathcal{P}$ that is violated by $\hat{x}$. In the first case, go to Step 3. In the second case, STOP and output the violated inequality.
Step 3. Scan the members of $\mathcal{F}$ corresponding to the columns in D. For each inequality in $\mathcal{L}$ violated by a column of D, check whether it is also violated by $\hat{x}$. If a constraint violated by $\hat{x}$ is encountered, STOP and output it. Otherwise, $\hat{x}$ must satisfy all the constraints in $\mathcal{L}$, provided that all constraints in $\mathcal{L}$ violated by some member of D were enumerated.

Figure 4: The decomposition algorithm for separating side constraints

of T. The process iterates until either a decomposition is found or it is proven that $\hat{x} \notin \mathcal{T}$. This algorithm is described in a more general setting in the next section.

2.3 A General Framework

The decomposition algorithm just described can be applied in a straightforward manner to other combinatorial optimization problems. In general, consider a combinatorial optimization problem $CP = (E, \mathcal{F})$, defined by a *ground set* E and a set $\mathcal{F}$ of *feasible* subsets of E. Let $\mathcal{P} = conv\left(\{x^S : S \in \mathcal{F}\}\right)$ and suppose that we have effective separation algorithms and heuristics for $\mathcal{P}$, so that we can solve CP efficiently. Suppose we would now like to solve a *restriction* $CP' = (E, \mathcal{H})$ where $\mathcal{H} \subset \mathcal{F}$. Let $\mathcal{P}' = conv\left(\{x^S : S \in \mathcal{H}\}\right)$. The inequality $ax \geq \alpha$, denoted (a, α), is a *side constraint* for CP if it is a valid inequality for $\mathcal{P}'$ but not for $\mathcal{P}$.

Suppose we want to separate over some class of side constraints $\mathcal{L}$. Suppose also that given $s \in \mathcal{F}$, we can determine efficiently if $s \in \mathcal{H}$ and if not, generate a constraint $ax \geq \alpha$ in the class $\mathcal{L}$ violated by s. Applying the previously described methodology, we immediately arrive at a separation algorithm for CP', shown in Fig. 4. The column generation algorithm (Step 1) is shown in Fig. 5.

Column Generation Algorithm for Step 1
Input: $\hat{x} \in \mathbb{R}^E$
Output: Either (1) a valid inequality for $\mathcal{P}$ violated by $\hat{x}$; or (2) a subset D of the columns of T and a vector $\lambda \geq 0$ such that $D\lambda = \hat{x}$ and $\mathbf{1}^T\lambda = 1$.
Step 1.0. Generate a matrix T' containing a small subset of promising columns from T.
Step 1.1. Solve (12) using the dual simplex algorithm replacing T by T'. If this LP is feasible, then STOP. The columns corresponding to the nonzero components of $\hat{\lambda}$, the current solution, comprise the set D.
Step 1.2. Otherwise, let r be the row in which the dual unboundedness condition was discovered, and let $(a, -\beta)$ be the r^{th} row of B^{-1}. Solve CP with cost vector c defined by

$$c_e = \begin{cases} M & \text{if } \hat{x}_e = 0; \\ -M & \text{if } \hat{x}_e = 1; \\ a_e & \text{otherwise} \end{cases} \tag{13}$$

$\forall e \in E$. M is chosen large enough to ensure conditions (10) and (11) are met. Let t be the incidence vector of the result. If $at < \beta$, then t is a column eligible to enter the basis. Add t to T' and go to 1.1. Otherwise, impose the appropriate Farkas inequality (see Sect. 2.4).

Figure 5: Column generation for the decomposition algorithm

2.4 Lifting the Farkas inequalities

As discussed earlier, if the matrix T is restricted, the resulting Farkas inequalities must be lifted in order to be valid. Several standard techniques can be used to lift these inequalities back into the original solution space. The easiest and most straightforward approach, which we will call the *big M method*, is simply to apply an appropriately large coefficient to each one of the fixed edges to ensure that the inequality remains valid when lifted. In other words, define the vector a' to be

$$a_e{}' = \begin{cases} M & \text{if } e \in E_0; \\ -M & \text{if } e \in E_1; \\ a_e & \text{otherwise} \end{cases} \tag{14}$$

where (a, β) is the original inequality and

$$E_0 = \{e : \hat{x}_e = 0\}, \tag{15}$$
$$E_1 = \{e : \hat{x}_e = 1\}. \tag{16}$$

Then the inequality $a'x \geq \beta - M|E_1|$ is valid for $\mathcal{P}'$ and is violated by $\hat{x}$ as long as $M \geq max\{\beta - ax : x \in \mathcal{P}'\}$. Any lower bound for CP' obtained with cost vector a can hence be used to derive a suitable constant.

To get stronger coefficients, one can use a sequential lifting procedure based on the same idea. We assume without loss of generality that $a_e = 0 \ \ \forall e \in E_1 \cup E_0$. The coefficients for

Procedure for Determining Lifting Coefficients for E_1
Input: E_1, E_0, and (a, β) valid for $\mathcal{P}_0^1 = \{x \in \mathcal{P}' : x_e = 1 \ \ \forall e \in E_1, x_e = 0 \ \ \forall e \in E_0\}$.
Output: An inequality (a', β') valid for $\mathcal{P}_0 = \{x \in \mathcal{P}' : x_e = 0 \ \ \forall e \in E_0\}$.
Step 1. Set $E_1^0 \leftarrow E_1$, $a^0 \leftarrow a$, $\beta^0 \leftarrow \beta$, and $i \leftarrow 0$.
Step 2. Choose $e^i \in E_1^i$ and compute $l(e^i)$, a lower bound on the optimal value of CP' computed with cost vector a^i and the constraint that $x_e = 1 \ \ \forall e \in E_1^i \setminus \{e^i\}$, $x_e = 0 \ \ \forall e \in E_0$.
Step 3. Set

$$a_e^{i+1} = \begin{cases} \beta^i - l(e^i) & \text{if } e = e^i, \\ a^i & \text{otherwise,} \end{cases} \tag{17}$$

Step 4. Set $E_1^{i+1} \leftarrow E_1^i \setminus \{e^i\}$, $\beta^{i+1} \leftarrow l(e^i)$, and $i \leftarrow i + 1$.
Step 5. If $E_1^i \neq \emptyset$, go to Step 2. Otherwise, output (a^i, β^i).

Figure 6: Lifting Procedure for the Farkas Inequalities

the members of E_1 are computed first using the procedure outlined in Fig. 6. An analogous procedure can be used to determine the coefficients for the edges in E_0.

2.5 Extensions to the Basic Approach

Here we indicate several means to improve the efficiency of the decomposition algorithm, as reported in [23]. We have already observed that T can be restricted to only those columns that *conform* to the current fractional solution. Consider the effect of further restricting the columns of T to only extreme points of $\mathcal{P}'$. At first, this may seem to defeat our purpose since in this case, we cannot possibly succeed in finding a decomposition. However, the Farkas cut generated when we fail to find a decomposition separates $\hat{x}$ from $\mathcal{P}'$ and hence can still be used to CUT. This basic approach was later used to generate valid inequalities for the TSP in [6].

Next, consider the polytope $\mathcal{P}''$ defined as the convex hull of incidence vectors of solutions whose cost is less than or equal to the current upper bound. It is immediate that $\mathcal{P}'' \subseteq \mathcal{P}'$ and furthermore, it is easily seen that $\min\{cx : x \in \mathcal{P}'\} = \min\{cx : x \in \mathcal{P}''\}$. We can therefore further limit enumeration to only those columns whose cost is less than the current upper bound and still generate a valid Farkas inequality.

This observation suggests considering what happens when we limit the enumeration to the point where T becomes empty. In this case, we have proven that there does not exist a feasible solution whose cost is below the current upper bound and whose structure conforms to that of the current fractional solution. Hence, we can impose the following cut, which we term a *no-columns cut*:

$$\sum_{e \in E_1} x_e - \sum_{e \in E_0} x_e \leq |E_1| - 1, \tag{18}$$

where E_0 and E_1 are defined as in (15) and (16). These cuts are a special case of *hypo-tours* introduced in [9]. Notice that if we limit column generation by both feasibility and cost, as suggested, we can *always* generate one of these inequalities. To see this, suppose T is composed only of extreme points of $\mathcal{P}''$. Then each of these columns generates a possible new upper bound and can hence be removed from T, leaving T empty.

Further modifications can be implemented to ensure that the maximum number of possible cuts is generated. In most cases, although only one row of the basis inverse is required to prove the infeasibility of the decomposition LP, multiple rows may provide this proof and all can be used to generate Farkas inequalities. Furthermore, even if we do not wish to limit column generation as described above, we can still easily generate additional inequalities by eliminating columns of T that are either not feasible, or have cost above the current upper bound, and then imposing a new group of cuts based on the reduced matrix.

2.6 Implementation for the VRP

When applying the decomposition algorithm to the VRP, we employ some additional techniques. Our method for generating the columns of T is a combination of the enumerative search and dynamic column generation methods described earlier. The enumerative generator performs a depth-first search of the fractional graph until a specified number of TSP tours are found. If the specified limit is reached before all possible tours are enumerated, then the column generation algorithm is invoked. The column generator is a simplified version of the CONCORDE TSP solver [4] called TINYTSP.

We tried a method described in [23] in which we first enumerated the connected components of the support graph $\hat{G}$ after removing the depot, as in the *connected components heuristic*. We then considered the subgraphs induced by each of these components plus the depot. It is easy to show that if the current fractional solution violates some capacity constraint, then there must also be a violation exhibited in at least one of the described subgraphs (see [23]). This component-wise approach can lead to a dramatic time savings for the column enumeration method of generating T, but leads to unnecessary complication and yields little advantage for the dynamic column generation algorithm. It was therefore abandoned.

Because the algorithm tends to be inefficient for dense graphs, we apply it only to fractional graphs whose density is below a specified threshold. To avoid tailing off of the objective function value, we also only rerun the algorithm in a particular search node if the gap decreased by at least a certain specified fraction since the last call to the algorithm. Finally, we place a time limit on the column generation process in order to avoid excessive computational effort in this phase of the algorithm.

To allow the algorithm to execute as quickly as possible, we do not lift the coefficients corresponding to variables that have been fixed by reduced cost or by branching. This means that the Farkas inequalities that are derived are only locally valid and do not get used in other parts of the branch and cut tree. To lift the remaining coefficients, we tried three different methods. The first method was simply to avoid the necessity of lifting by not restricting the matrix T as described in Sect. 2.2. The second method we implemented was the *big M method* with coefficients calculated using TINYTSP (see Sect. 2.4). Finally, we also implemented the sequential lifting procedure described in Fig. 6.

We conclude this section by discussing the method by which we discover violated capacity constraints once we have successfully obtained a decomposition of the current fractional solution. This is a critical step, as it determines the ultimate success or failure of the algorithm. It is important to understand that the implementation of this step determines whether or not the algorithm will perform *exact* separation (i.e., be guaranteed to find a violated inequality if one exists). In order to perform exact separation, we need to be able to determine, given an extreme point t of $\mathcal{T}$ and a fractional point $\hat{x}$, the *full set* of inequalities violated by t and we must check each one for violation by $\hat{x}$. In the VRP, it is easy to determine whether there exists *some* inequality violated by a given $t \in \mathcal{T}$. However, it is another matter to enumerate *the full set* of such violated inequalities, since in general there can be exponentially many of them. We implemented an exact separation version of this algorithm, but preliminary experiments indicated it was much too inefficient and did not generate many additional inequalities. Instead, we used a few simple heuristics to generate potential violated inequalities.

3 Computational Results

The separation routines were embedded in the generic, parallel branch and cut framework SYMPHONY developed by Ralphs [31] and Ladányi [24]. The implementation of this shell is reviewed in [25]. Our test set consisted of medium-sized VRP instances taken from the TSPLIB [35] repository and from that maintained by Augerat [7]. This test set includes the standard problems from the literature, but is larger and more varied than that used in previous papers. The full test set and source code used in this paper are available at `http://www.branchandcut.org/VRP`.

Ten of the problems in the set (those whose names are preceded by an 'E') are derived from that used by Christofides and Eilon in [13], which is included among the VRP instances at TSPLIB. Three of those (E-n76-k7, E-n76-k8, and E-n101-k8) were solved for the first time during this work using the parallel version of SYMPHONY on an IBM SP2 parallel computer with up to 80 processors. Recently, Blasum and Hochstättler solved E-n76-k7 and E-n76-k8 on a single processor using additional cutting planes [11]. We have also since managed to solve E-n76-k7 sequentially, but required a larger tree. Many larger problems from the test set were solved to optimality during this work (some presumably for the first time), but are not reported here. We plan to release a supplementary report detailing our experience with solving those instances. It is worth noting that the smallest instance we are aware of that has not been solved to optimality is B-n50-k8. We managed to solve a version with the truck capacity increased to 150, but could not solve the original instance.

We performed several experiments to test the effectiveness of various branching rules and separation procedures. The results are shown in Table 1. These results were obtained using the sequential version of SYMPHONY 2.7 with CPLEX 6.6 as the LP engine and were run on a 700MHz Intel Pentium III Xeon platform under Redhat Linux 6.2. The previously unsolved problems are included in the table for informational purposes, but are not included in the totals for obvious reasons. Comparison with previous results is difficult at best, not only because of hardware differences, but because the test sets used in previous papers were not large or sufficiently varied enough to permit a fair assessment.

	With Decomposition			Without Decomposition		
problem	Tree Size	Tree Depth	Wallclock	Tree Size	Tree Depth	Wallclock
E − n30 − k3	123	17	4.01	206	25	3.53
E − n31 − k7	7	3	1.08	36	31	3.32
E − n33 − k4	8	4	1.67	5	3	0.35
E − n51 − k5	24	7	37.74	42	2	22.86
gr − n21 − k3	1	0	0.05	4	3	0.14
gr − n24 − k4	4	3	1.06	4	7	0.32
fri − n26 − k3	23	9	0.95	16	10	.35
swiss − n42 − k5	10	4	4.69	23	21	1.52
att − n48 − k4	346	24	112.85	440	9	30.58
gr − n48 − k3	31	11	5.86	36	12	2.92
hk − n48 − k4	115	11	87.28	122	7	17.02
A − n32 − k5	8	3	0.79	12	6	0.70
A − n33 − k5	13	4	4.53	9	5	0.78
A − n33 − k6	17	6	8.84	36	7	2.41
A − n34 − k5	15	6	4.80	15	7	1.51
A − n36 − k5	45	10	14.83	91	10	7.92
A − n37 − k5	12	6	6.40	18	5	2.31
A − n38 − k5	195	16	46.45	351	20	21.55
A − n45 − k6	315	18	253.28	422	20	120.59
A − n46 − k7	6	3	6.27	8	3	5.83
A − n44 − k6	1403	23	935.67	1966	24	621.22
A − n53 − k7	788	19	1379.20	1222	18	733.20
B − n31 − k5	8	7	0.80	7	3	0.20
B − n34 − k5	562	16	27.34	1120	21	17.76
B − n38 − k6	28	7	13.25	88	12	6.43
B − n39 − k5	10	7	0.61	5	4	0.20
B − n41 − k6	66	11	32.51	43	9	3.40
B − n43 − k6	551	23	341.53	509	22	35.09
B − n44 − k7	1	0	0.88	4	3	0.55
B − n45 − k5	58	12	31.04	52	8	5.04
B − n50 − k7	3	3	0.50	8	5	0.52
B − n51 − k7	294	41	102.94	340	36	23.44
B − n52 − k7	10	8	2.59	21	11	0.85
B − n56 − k7	97	15	27.41	91	13	8.63
B − n64 − k9	18	7	24.06	18	7	8.51
Total	5215	–	3523.76	7390	–	1711.55
E − n76 − k7	121,811	–	288,640	115,991	–	278,613
E − n76 − k8	484,245	–	1,927,422	–	–	–
E − n101 − k8	244,968	–	1,900,671	–	–	–

Table 1: Results of computational experiments

3.1 Branching

We experimented with branching strategies involving branching on cuts and variables. To branch on cuts, we used a straightforward method that consisted of maintaining a small pool of candidate cuts that had been removed from the current relaxation because of slackness. Branching on variables was done in the usual way. The overall strategy was to first develop a pool of candidates consisting of those cuts and variables whose current values maximized the distance from each side of the corresponding disjunction (i.e., the zero-one variables closest to value one-half, etc.). We then applied strong branching to this list of candidates by pre-solving the two potential children of the current search node for each candidate. Any candidate spawning a child that could be pruned in the pre-solve was immediately chosen for branching. Otherwise, we branched on the candidate that maximized the minimum objective function value in each of its two children.

Overall, it was found that the most effective strategy was to branch only on variables. Our method of branching on cuts was generally ineffective, usually leading to increased running times. However, Augerat, et al., had success with a slightly different implementation in [9]. Although we tried several alternative methods of selecting the candidate variables for strong branching, the most effective method proved to be the classical method of choosing the variables farthest from being integer-valued. Strong branching was found to be an extremely effective tool for reducing the size of the tree. We used a *graduated strong branching* technique by which the number of strong branching candidates evaluated near the root of the tree is gradually reduced at lower levels. This method proved effective at reducing the expense of strong branching without increasing the size of the tree.

3.2 Separation

We performed several experiments to test the effectiveness of both the separation heuristics and the decomposition algorithm. When using decomposition, we observed a significant reduction in the number of search tree nodes examined, but because of the expense of generating the matrix T and lifting the Farkas inequalities, running times were usually increased. In trying to strengthen the Farkas inequalities, we experimented with various methods of sequential lifting, various ways of restricting T, and various ways of limiting column generation. The Farkas cuts were most effective when column generation was not limited except by enforcing conditions (10) and (11). This method also allowed the greatest opportunity for finding a decomposition and hence locating additional violated capacity constraints.

The Farkas cuts were ineffective in some cases. Because of the restriction of T, the unlifted Farkas inequalities were typically extremely sparse and localized, often having only one or two nonzero coefficients. We conjecture that lifting these cuts may cause them to lose their relevance once the solution is perturbed slightly. However, the structure of these cuts and the effect of lifting them needs to be further examined.

Perhaps the most intriguing aspect of this work has been the chance to develop some insight into the polyhedral structure of the VRP polytope. The decomposition algorithm allows us to answer some interesting questions about the behavior of branch and cut algorithms for this problem and about the performance of our separation heuristics. In particular, we can for the first time assess exactly how good our heuristics are at finding existing violated capacity constraints. With respect to this question, the answer is strongly positive. The mix of separation algorithms we describe in this paper seems to be highly effective. The decomposition algorithm was only able to find additional violated inequalities about 20% of the time when the current fractional solution was actually inside the VRP polytope. Furthermore, the additional cuts that were found did not seem to be as effective as the ones that had been found by the heuristics.

4 Conclusions and Future Work

Overall, our branch and cut algorithm has demonstrated promising performance. During this work, we were able to solve several previously unsolved problems and were also able to significantly improve on previously reported solution times through application of the ideas in this paper and through advances in SYMPHONY, our underlying branch and cut framework. Many of these advances are detailed in [25]. Still further progress should come from additional separation over a wider class of inequalities. This has been demonstrated in recent work by Blasum and Hochstättler [11] in which they further improve on our solution times by implementing a much wider variety of cutting plane routines.

By analyzing the results of applying the decomposition algorithm, we were able to discern that after our separation heuristics failed, the fractional solution to the LP relaxation was still outside the TSP polytope more than 20% of the time. Furthermore, this varied dramatically by problem instance. In certain instances, the fractional solutions were virtually *always* outside the TSP polytope. In others, they never were. It could therefore be fruitful to simply apply TSP separation algorithms to the VRP directly in cases where the tendency is to be outside the polytope. This is an example of one way in which we could further leverage the tremendous amount of information gathered in the process of attempting to find a decomposition.

The decomposition algorithm proved to be an effective tool when measured with respect to its ability to find violated inequalities and reduce the size of the search tree, but improvements must be made in its implementation in order for it to effect a decrease in running times. Changes in both the relaxation and the restriction of the matrix T could help. Obvious candidates for further relaxation are the Assignment Problem or the the Minimum k-tree Problem. These relaxations could be solved more quickly than the TSP and would still yield a method for locating violated capacity constraints.

It should be noted that this separation algorithm has interesting theoretical connections to other decomposition-based algorithms, such as Lagrangian relaxation and Dantzig-Wolfe Decomposition. These connections will be explored in a forthcoming paper [34]. Because of its generality, this separation algorithm holds the potential for application to new problem settings, including more complex VRP models. Because heuristic separation for capacity cuts has improved significantly in recent years, the standard capacitated VRP model may no longer be the ideal platform for development of decomposition. However, the algorithm can be easily modified to handle other types of side constraints, including time window and distance constraints, that are common in practical applications.

References

[1] J.R. Araque, L. Hall, and T. Magnanti (1990): Capacitated Trees, Capacitated Routing and Associated Polyhedra. Discussion paper 9061, CORE, Louvain La Nueve

[2] Y. Agarwal, K. Mathur, and H.M. Salkin (1989): Set Partitioning Approach to Vehicle Routing. Networks **7**, 731–749

[3] J.R. Araque, G. Kudva, T.L. Morin, and J.F. Pekny (1994): A Branch-and-Cut Algorithm for Vehicle Routing Problems. Annals of Operations Research **50**, 37–59

[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook (2001): CONCORDE TSP Solver. Available at `www.math.princeton.edu/tsp/concorde.html`

[5] D. Applegate, R. Bixby, V. Chvátal, and W. Cook (1995): Finding Cuts in the TSP (A Preliminary Report). DIMACS Technical Report 95-05

[6] D. Applegate, R. Bixby, V. Chvátal, and W. Cook (2001): TSP Cuts Which Do Not Conform to the Template Paradigm. *Computational Combinatorial Optimization*, D. Naddef and M. Jünger, eds., Springer, Berlin, 261–303

[7] P. Augerat (1995): VRP problem instances. Available at `http://www.branchandcut.org/VRP/data/`

[8] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef (1998): Separating Capacity Constraints in the CVRP Using Tabu Search. European Journal of Operations Research, **106**, 546–557

[9] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, G. Rinaldi (1995): Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem. Research Report 949-M, Université Joseph Fourier, Grenoble, France

[10] M.L. Balinski and R.E. Quandt (1964): On an Integer Program for a Delivery Problem. Operations Research **12**, 300–304

[11] U. Blasum and W. Hochstättler (2000): Application of the Branch and Cut Method to the Vehicle Routing Problem. Zentrum für Angewandte Informatik, Köln, Technical Report zpr2000-386

[12] V. Campos, A. Corberán, and E. Mota (1991): Polyhedral Results for a Vehicle Routing Problem. European Journal of Operations Research **52**, 75–85

[13] N. Christofides and S. Eilon (1969): An Algorithm for the Vehicle Dispatching Problem. Operational Research Quarterly **20**, 309–318

[14] N. Christofides, A. Mingozzi and P. Toth (1981): Exact Algorithms for Solving the Vehicle Routing Problem Based on Spanning Trees and Shortest Path Relaxations. Mathematical Programming **20**, 255–282

[15] G. Cornuéjols and F. Harche (1993): Polyhedral Study of the Capacitated Vehicle Routing Problem. Mathematical Programming **60**, 21–52

[16] F.H. Cullen, J.J. Jarvis and H.D. Ratliff (1981): Set Partitioning Based Heuristic for Interactive Routing. Networks **11**, 125–144

[17] G.B. Dantzig and R.H. Ramser (1959): The Truck Dispatching Problem. Management Science **6**, 80–91

[18] M.L. Fisher (1988): Optimal Solution of Vehicle Routine Problems Using Minimum k-Trees. Operations Research **42**, 626–642

[19] M.L. Fisher and R. Jaikumar (1981): A Generalized Assignment Heuristic for Solving the VRP. Networks **11**, 109–124

[20] B.A. Foster and D.M. Ryan (1976): An Integer Programming Approach to the Vehicle Scheduling Problem. Operational Research Quarterly **27**, 367–384

[21] M.R. Garey and D.S. Johnson (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., San Francisco

[22] M. Held and R.M. Karp (1969): The Traveling Salesman Problem and Minimal Spanning Trees. Operations Research **18**, 1138–1162

[23] L. Kopman (1999): A New Generic Separation Routine and Its Application in a Branch and Cut Algorithm for the Vehicle Routing Problem. Ph.D. Dissertation, Field of Operations Research, Cornell University, Ithaca, NY, USA

[24] L. Ladányi (1996): Parallel Branch and Cut and Its Application to the Traveling Salesman Problem. Ph.D. Dissertation, Field of Operations Research, Cornell University, Ithaca, NY, USA

[25] L. Ladányi, T.K. Ralphs, and L.E. Trotter (2001): Branch, Cut, and Price: Sequential and Parallel. *Computational Combinatorial Optimization*, D. Naddef and M. Jünger, eds., Springer, Berlin, 223–260

[26] G. Laporte and Y. Nobert (1981): Comb Inequalities for the Vehicle Routing Problem. Methods of Operations Research **51**, 271–276

[27] G. Laporte, Y. Nobert and M. Desrochers (1985): Optimal Routing with Capacity and Distance Restrictions. Operations Research **33**, 1050–1073

[28] A.N. Letchford, R.W. Eglese, and J. Lysgaard (2001): Multistars, Partial Multistars and the Capacitated Vehicle Routing Problem. Technical Report available at `http://www.lancs.ac.uk/staff/letchfoa/pubs.htm`

[29] D. Naddef and G. Rinaldi (2000): Branch and Cut. To appear in P. Toth and D. Vigo, eds., *Vehicle Routing*, SIAM.

[30] H. Nagamochi and T. Ibaraki (1992): Computing Edge Connectivity in Multigraphs and Capacitated Graphs. SIAM Journal of Discrete Mathematics **5**, 54–66

[31] T.K. Ralphs (1995): Parallel Branch and Cut for Vehicle Routing. Ph.D. Dissertation, Field of Operations Research, Cornell University, Ithaca, NY, USA

[32] T.K. Ralphs (2001): SYMPHONY Version 2.8 User's Guide. Available at `www.branchandcut.org/SYMPHONY`

[33] T.K. Ralphs and L. Ladányi (1999): SYMPHONY: A Parallel Framework for Branch and Cut. White paper, Rice University

[34] T.K. Ralphs (2002): Decomposition-based Algorithms for Large-scale Discrete Optimization. In preparation.

[35] G. Reinelt (1991): TSPLIB—A traveling salesman problem library. ORSA Journal on Computing **3**, 376–384. Update available at `http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/`