in The Vehicle Routing Problem: Latest Advances and New Challenges

Modeling and Solving the Capacitated Vehicle Routing Problem on Trees

Bala Chandran[1] and S. Raghavan[2]

[1] Department of Industrial Engineering and Operations Research
University of California
Berkeley, CA 94720
chandran@ieor.berkeley.edu

[2] The Robert H. Smith School of Business and
Institute for Systems Research
University of Maryland
College Park, MD 20742
raghavan@umd.edu

Summary. Capacitated vehicle routing problems (CVRPs) form the core of logistics planning and are hence of great practical and theoretical interest. This chapter considers the CVRP on trees (TCVRP), a problem that often naturally arises in railway, river, and rural road networks. Our objective is to build high-quality models that exploit the tree structure of the problem that can also be easily implemented within the framework of a modeling language (a feature desired by practitioners) like AMPL, GAMS, or OPL. We present two new integer programming models for the TCVRP that explicitly take advantage of the tree structure of the graph. The two models are implemented using the AMPL model building language, and compared along several metrics—computation time, quality of the linear programming relaxation, and scalability—to examine their relative strengths.

Key words: Capacitated vehicle routing on trees; integer linear programming formulations; high-level modeling languages.

1 Introduction

The capacitated vehicle routing problem (CVRP) is a fundamental problem in combinatorial optimization with wide-ranging applications in practice. It forms the core of logistics planning and has been extensively studied by the operations research community. The last two decades have seen enormous improvements in the research community's ability to solve these problems, due to better algorithms as well as better computational capabilities. Toth and Vigo [11] provide an upto date survey of problem variants, exact solution techniques, and heuristics for the vehicle routing problem.

In this chapter, we consider a special case of the CVRP introduced by Labbé et al. [8], in which the network is a tree and routes are constrained only by vehicle capacity. The problem, referred to as the capacitated vehicle routing problem on trees (TCVRP), may be stated as follows. Given a tree $T = (N_0, E)$ with vertex set N_0 and edge set E, non-negative distances for the edges in E, a depot node in N_0 with a fleet of homogeneous vehicles of capacity C (the size of the fleet is a decision variable), and non-negative integer node demands (representing customer demands); find a collection of routes starting and ending at the depot that (i) ensures that the demand serviced by a vehicle does not exceed its capacity C, (ii) ensures that a customer's demand is serviced by exactly one vehicle, and (iii) minimizes a cost function that is a linear combination of the total distance traveled by these vehicles and the number of vehicles. Note that in the TCVRP, each vehicle route is a tree, and a vehicle may pass through nodes that are served by another vehicle.

The TCVRP arises naturally in situations where access construction costs far exceed routing costs, for example in river networks and pit mine rail networks [8]. An application arising in the flexible manufacturing environment can also be modeled as a TCVRP [2]. Also, some general CVRPs may be approximated as a TCVRP by clustering groups of customers to reduce the network to a tree, as shown in Figure 1. In particular, Basnet et al. [1] describe an application to routing milk tankers in rural New Zealand where building roads are costly due to the mountainous terrain, resulting in small hamlets of a few farms connected to each other by a sparse network.

In addition to its applications in practice, the TCVRP is compelling from a theoretical perspective. Several hard graph problems such as facility location are easy to solve on trees; therefore, an interesting question is whether the TCVRP can be solved more efficiently than the general CVRP. The TCVRP was shown to be $\mathcal{NP}$–hard by a reduction from the bin-packing problem [8]. Hence, we must look to means such as integer programming or branch-and-bound algorithms that have proven effective for solving the general CVRP, while adapting these techniques to take advantage of the tree structure of the network. Previous work on exact solutions to the capacitated vehicle routing problem on trees is sparse. Labbé et al. [8] describe a branch-and-bound approach to the problem based on bin-packing lower bounds, and a 2-approximation algorithm for the problem that takes advantage of the tree structure of the network-. Mbaraga et al. [9] extend the branch-and-bound approach to the distance-constrained vehicle routing problem on trees and the distance-constrained capacitated vehicle routing problem on trees. They introduce a mathematical programming solution approach based on a set covering formulation, which is solved using a column generation technique. They also introduce the heterogeneous capacitated vehicle routing problem on trees (all vehicles do not have the same capacity) and use the set covering approach to solve this problem.

The previous mathematical programming techniques do not explicitly take advantage of the tree structure of the underlying graph. Further, these solution

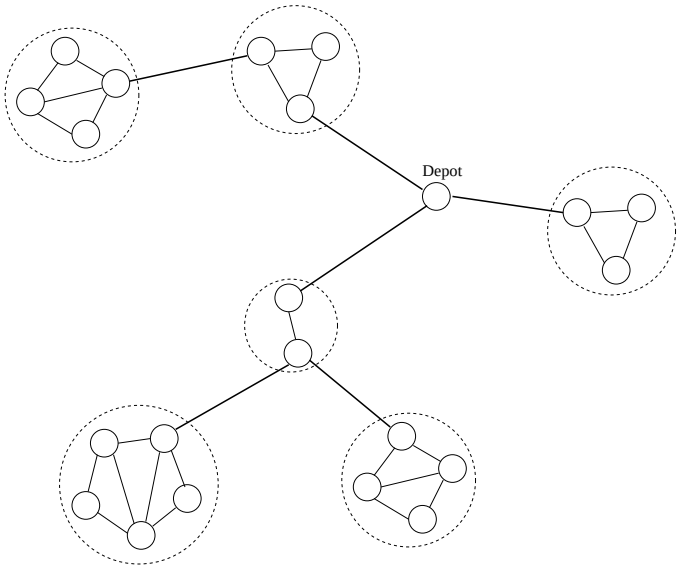

Fig. 1. General VRPs can sometimes be represented as a TCVRP by clustering certain nodes (clusters shown enclosed by dotted circles).

techniques require specialized implementation, which may be an impediment to adoption in practice. We elaborate on this point further, as it is related to one of our main motivations and contributions of this work. There have been many developments in exact optimization techniques for the CVRP (see Toth and Vigo [11]). Consequently, it is not uncommon to see CVRP problems solved to optimality using branch-and-cut or branch-and-price (i.e., column generation) techniques. In branch-and-cut techniques, the underlying formulations have an exponential number of constraints which are implemented by dynamically adding these constraints when needed (using so-called separation routines). Column generation techniques have an exponential number of variables, and are implemented by dynamically adding variables to the formulation as needed (by solving the underlying pricing problem). Both of these techniques require significant mathematical sophistication to implement. Consequently, these techniques are sometimes beyond the domain of practitioners.

On the other hand, over the past 10 years, there has been an increased use of high-level modeling languages in practice (see, for example, Kallrath [7]). Modeling languages make optimization packages like CPLEX, OSL, and XPRESS (that solve linear, integer, and mixed-integer programs) quite easily accessible to practitioners. They do not require the mathematical and programming sophistication that is required to implement specialized techniques in the optimization language. Of course, the downside is that solution techniques developed in high-level modeling languages are likely to be significantly

slower than those developed in the native code of the optimization software. However, the fact remains that models and methodologies that can be easily implemented in a high-level modeling language are more likely to be implemented in practice, than specialized algorithms, that require sophisticated implementation.

Consequently, our study should not be viewed as an attempt to solve the problem in the shortest amount of time; rather, it is an attempt to solve the problem as efficiently as possible *while working within the degrees of freedom offered by general purpose commercial solvers and modeling languages*. To this end we propose two integer programming formulations for the TCVRP that exploit the tree structure of the underlying network and are easy to build in a high-level modeling language. Our main contribution is that we develop models that take advantage of the tree structure of the problem with a performance that is competitive with solution methods that use special purpose code.

The rest of the chapter is organized as follows. In Section 2 we introduce notation and state our assumptions. In Section 3, we present our first formulation that is based on a depth first indexing of nodes in the tree. In Section 4, we present our second integer program that is based on the observation that all vehicle routes for the TCVRP may be modeled as trees. In Section 5, we describe an approximation algorithm for the problem that combines the approaches of Labbé et al. [8] and Basnet et al. [1]. In Section 6, we discuss our computational experience with these formulations. In Section 7, we present our conclusions.

2 Preliminaries

Given a rooted tree $T = (N_0, E)$, the root of T (representing the depot) is a unique node in N_0. The set of nodes other than the depot is denoted by N. If $[i, j] \in E$ and i is closer to the root than j, then i is the parent of j and j is a child of i. Node i is an ancestor of node j (and j is a descendant of node i) if i lies on the unique simple path from the root to j. We will use the convention that the tree is represented topologically "downward". Therefore, a node is "below" its ancestors and "above" its descendants. A leaf node of the graph is a node that does not have children. A sub-tree S is a connected sub-graph of T. The root of a sub-tree S, denoted by r_S, is the node in S that is closest to the depot. Given a node i, the tree "below" node i denoted by $T(i)$ is the sub-tree induced by node i and its descendants. The *weight* of a sub-tree is the sum of weights of the edges in the sub-tree.

Given a subset of nodes $L \subseteq N_0$, the minimal covering sub-tree (henceforth referred to as the covering sub-tree) is the union of all (unique) paths from each node in L to the depot. Observe that the covering sub-tree is rooted at the depot. The covering sub-tree is the minimum set of edges that need to be traversed in order to visit each node in L.

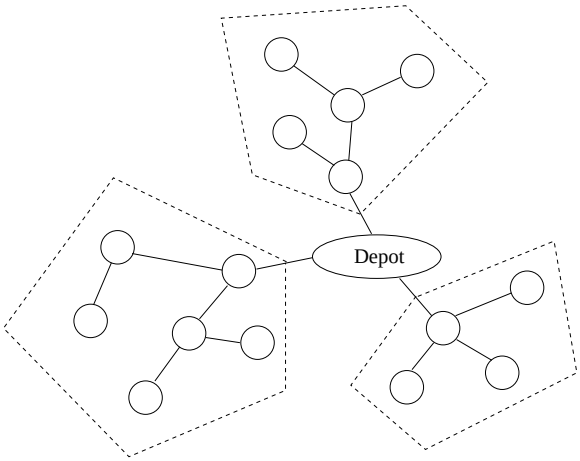

Fig. 2. A tree in which the degree of the depot is greater than 1 can be split into multiple sub-trees.

Without loss of generality, we assume that the degree of the depot is 1. First, consider the situation where there is no fixed cost associated with the use of a vehicle. Observe that a route that includes the depot as a non-terminal node can be broken up into multiple routes—each route originating and terminating at the depot—with no change in the objective value. Thus, in this situation, the problem may simply be decomposed into multiple smaller problems—one for each subtree incident to the depot node. Figure 2 illustrates this situation. On the other hand, when there is a fixed cost associated with a route (such as a loading/unloading cost), we add a new node and connect it to the depot by an edge whose cost equals the fixed cost. We make this new node the depot (observe that this node has degree 1). Finally, we set the demand of the old depot node to zero and solve the TCVRP on this network (where the depot has degree 1).

3 Depth First Ordered Formulation

Our first formulation exploits Lemma 1, which is illustrated in Figure 3.

Lemma 1. *Suppose the nodes in a tree are indexed in depth-first order. Then, given a set of nodes in a vehicle's route, a minimum cost route is obtained by visiting the nodes in increasing order of index.*

Proof. Given a set of nodes and the corresponding covering sub-tree, the vehicle must traverse each edge in the sub-tree *at least* twice — once going away

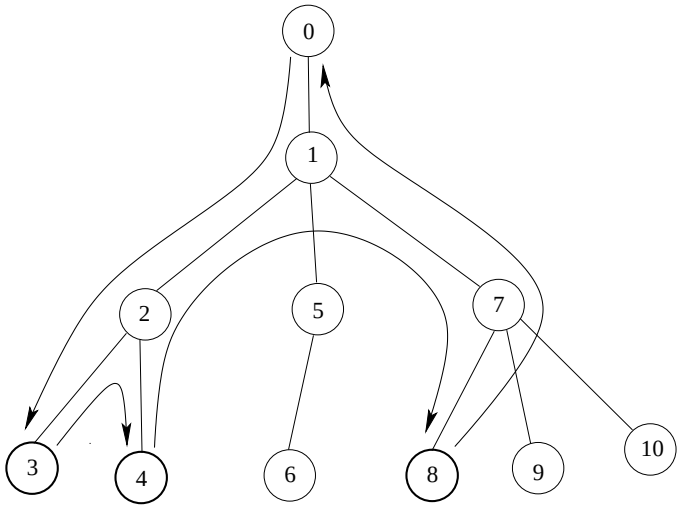

Fig. 3. Idea behind the DFS formulation. Given a set of nodes serviced by a vehicle (3, 4, and 8), the distance-minimizing tour for the vehicle is obtained by visiting nodes in order of increasing index when nodes are indexed by a depth-first ordering.

from the depot and once towards. Therefore, a lower bound on the tour length is twice the weight of the covering sub-tree.

Suppose we construct a tour by visiting nodes in order of depth first order. Given a covering sub-tree S, let $T(i)$ be a sub-tree of S rooted at node i. For any i in S, the lowest indexed node in $T(i)$ is i (due to depth-first ordering of nodes).

For every i in S, the vehicle enters the sub-tree $T(i)$, and services all nodes in $T(i)$. The key observation is that once the vehicle leaves $T(i)$ it never returns since nodes are being serviced in increasing order of index. Therefore, for all i in S, the edge from i to its parent is traversed *at most* twice — once into the sub-tree and once out of it. This yields a feasible tour of length twice the weight of the covering sub-tree.

Since the feasible tour length is equal to the lower bound, it must be optimal. □

A tour for each vehicle consists of a path from the depot to the first (lowest indexed) node in the vehicle tour, a set of arcs connecting nodes in the vehicle tour in increasing order of node index, and a path from the last (highest indexed) node in the tour back to the depot.

We introduce the following four sets of binary variables in our formulation.

Decision Variables

$x_{ij} = \begin{cases} 1 \text{ if node } i \text{ immediately precedes node } j \text{ in the vehicle tour;} \\ 0 \text{ otherwise.} \end{cases}$

$y_{ij} = \begin{cases} 1 \text{ if nodes } i \text{ and } j \text{ are visited by the same vehicle;} \\ 0 \text{ otherwise.} \end{cases}$

$w_i = \begin{cases} 1 \text{ if node } i \text{ is the first node visited in the vehicle tour;} \\ 0 \text{ otherwise.} \end{cases}$

$z_i = \begin{cases} 1 \text{ if node } i \text{ is the last node visited in the vehicle tour;} \\ 0 \text{ otherwise.} \end{cases}$

Our model uses the following data that is available as input.

Input Parameters

D_i Demand at node i
L_{ij} Shortest path distance between nodes i and j
S_i Shortest path distance between the node i and the depot
N Set of all nodes other than the depot; $N = \{1, .., n\}$
C Capacity of each vehicle
$V_k^{\min}$ A lower bound on the number of vehicles required to service nodes $\{k, \ldots, n\}$

We may now state our formulation as follows, assuming that nodes are indexed in depth first order.

$$\text{Minimize} \quad \sum_{i=1}^{n} S_i(w_i + z_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} L_{ij} x_{ij} \tag{1}$$

$$\text{Subject to} \quad D_i + \sum_{j=i+1}^{n} D_j y_{ij} \leq C \qquad \forall\, i \in N \tag{2}$$

$$y_{ij} + y_{jk} - y_{ik} \leq 1 \qquad \forall\, i, j, k \in N \;:\; i < j < k \tag{3}$$

$$x_{ij} - y_{ij} \leq 0 \qquad \forall\, i, j \in N \tag{4}$$

$$w_j + \sum_{i=1}^{j-1} x_{ij} = 1 \qquad \forall\, j \in N \tag{5}$$

$$z_i + \sum_{j=i+1}^{n} x_{ij} = 1 \qquad \forall\, i \in N \tag{6}$$

$$w_i,\ x_{ij},\ y_{ij},\ z_i \in \{0, 1\}\ \forall\, i, j \in N :\ i < j \tag{7}$$

The objective function is to minimize the distance of all traversed arcs (leaving and entering the depot and those in-between). Constraint (2) is the vehicle capacity constraint. Although there is no explicit concept of a vehicle in this formulation, capacity constraints are captured by summing demand over nodes that are in the same vehicle. Constraint (3) creates a clique among nodes in the same vehicle—if nodes i and j are in the same vehicle, and nodes

j and k are in the same vehicle, then nodes i and k must be in the same vehicle. Constraint (4) enforces that node i cannot precede node j unless they are both in the same vehicle. Constraints (5) and (6) force the demand at a node to be served by exactly one vehicle. Constraint (7) enforces that all variables are binary. An interesting observation is that the integrality of x, w, and z variables can be relaxed (i.e., $0 \leq x, w, z \leq 1$) without affecting the integrality of the solution. However, we observed that in practice, the problem was solved faster if all variables were specified to be binary. Note that the number of integer variables in the DFS formulation is $O(|N|^2)$ and the number of constraints is $O(|N|^3)$.

3.1 Valid Inequalities

We now describe some valid inequalities to improve the strength of our formulation.

The number of vehicles that service the set of nodes $\{k, \ldots, n\}$ is the number of vehicles that enter this set directly from the depot plus the number of vehicles that enter the set from a lower indexed node in $\{1, \ldots, k-1\}$. Our first valid inequality states that the number of vehicles servicing nodes $\{k, \ldots, n\}$ is at least some lower bound $V_k^{\min}$.

$$\sum_{j=k}^{n} \left(w_j + \sum_{i=1}^{k-1} x_{ij} \right) \geq V_k^{\min} \qquad \forall\ k \in \{1, \ldots, n\} \tag{8}$$

The lower bound may be obtained, for example, by solving a bin-packing problem with the node demands. Since the bin-packing problem is itself not trivially solved and our objective is ease and speed of implementation, we use an $O(|N|^2)$ procedure, developed by Labbé et al. [8], to compute a lower bound on the bin-packing solution.[3]

The next set of valid inequalities are the following: two vehicles whose cumulative demand exceeds the vehicle capacity cannot be in the same vehicle. This can be stated as follows.

$$y_{ij} = 0 \qquad \forall\ i, j \in N : i < j,\ D_i + D_j > C \tag{9}$$

[3] The bound relies on the fact that at most one node with demand greater than $C/2$ and at most two nodes with demand in $(C/3, C/2]$ can belong to the same vehicle. First, nodes with demand $> C/2$ are assigned to separate vehicles. An attempt is then made to assign the nodes with demand in $(C/3, C/2]$ to these vehicles according to a first-fit procedure [4], i.e., by assigning the remaining node with smallest demand to the vehicle with the least remaining capacity (ties are broken arbitrarily) At most one node with demand in $(C/3, C/2]$ can fit in each of these bins. Let K be the number of nodes with demand in $(C/3, C/2]$ that cannot be assigned to a vehicle through this process. Then, at least $\lceil K/2 \rceil$ vehicles are needed. A lower bound on the packing problem is then generated using the vehicles assigned so far, $\lceil K/2 \rceil$, and remaining nodes with demand $\leq C/3$.

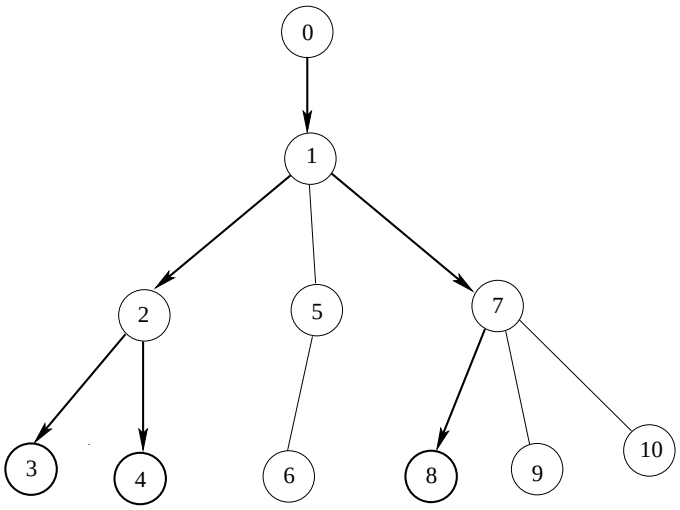

Fig. 4. Idea behind the treeroute formulation. Given a set of nodes serviced by a vehicle (3, 4, and 8), the distance-minimizing tour for the vehicle is a tree.

Note that this inequality is a restricted case of cover inequalities (see, for example, Nemhauser and Wolsey [10]). We do not explicitly add any more cover inequalities since there are an exponentially large number of them, and integer programming software systems have built-in algorithms for adding these cuts efficiently.

4 Treeroute Formulation

In this formulation, we exploit the fact that the route of a vehicle is a tree. We replace the edges in the tree with arcs pointing "downwards" to eliminate ambiguity in the parent-child relationship of the nodes in an edge—given an arc (i, j), i is the parent and j is the child. The route is constructed for each vehicle by "building upwards" from every node serviced by the vehicle towards the depot. This generates the covering sub-tree of the nodes, and will henceforth be referred to as the route-tree of that vehicle. This is shown in Figure 4.

We introduce the following binary variables in our model.

Decision Variables

$$x_{ijv} = \begin{cases} 1 \text{ if arc } (i,j) \text{ is in the route of vehicle } v; \\ 0 \text{ otherwise.} \end{cases}$$

$$y_{iv} = \begin{cases} 1 \text{ if vehicle } v \text{ services node } i; \\ 0 \text{ otherwise.} \end{cases}$$

Our model uses the following data that is available as input.

Input Parameters

L_{ij} Distance from node i to node j
D_i The demand at node i
P_i The parent node of i
V The set of vehicles
N The set of nodes other than the depot
C Capacity of each vehicle
$c(i)$ Set of children of i
$T(i)$ Sub-tree rooted at node i
A The set of all arcs (directed downwards) in the tree
$V_i^{\min}$ A lower bound on the number of vehicles required to service all nodes in $T(i)$.

We may now formulate our model as follows.

$$\text{Minimize} \quad 2 \sum_{(i,j)\in A} L_{ij} \sum_{v\in V} x_{ijv} \tag{10}$$

$$\text{Subject to} \quad x_{P_i iv} \geq x_{ijv} \quad \forall\, v \in V, i \in N : c(i) \neq \phi, j \in c(i) \tag{11}$$

$$x_{P_i iv} \geq y_{iv} \quad \forall\, i \in N, v \in V \tag{12}$$

$$\sum_{i\in N} y_{iv} D_i \leq C \quad \forall\, v \in V \tag{13}$$

$$\sum_{v\in V} y_{iv} = 1 \quad \forall\, i \in N \tag{14}$$

$$y_{jv},\ x_{ijv} \in \{0,1\}\ \forall\ (i,j) \in A, v \in V \tag{15}$$

The objective is to minimize the weighted sum over all arcs of the number of vehicles that use each arc. Constraint (11) ensures that if an arc (i,j) is in the vehicle route tree, then the unique preceding arc (P_i, i) must also exist in the route tree. Constraint (12) ensures that, in order to service a particular node, a vehicle must travel along the unique arc leading to that node. Constraint (13) is the vehicle capacity constraint. Constraint (14) enforces that the demand at every node is completely satisfied by exactly one vehicle.

We note that the x variables may be relaxed without affecting integrality of the solution; however, our initial computations indicated that the problem was solved faster if all variables were defined to be binary. Also, observe that the number of integer variables in the treeroute formulation is $O(|N||V|)$, and the number of constraints is $O(|N||V|)$.

4.1 Valid Inequalities

We now discuss valid inequalities for the treeroute formulation.

As the model stands it suffers from symmetry, i.e., exchanging the set of customers between any two vehicles produces an alternative optimal solution.

Table 1. Two solutions that are feasible to the first set of symmetry cuts. Only solution 2 is feasible to the second set.

Solution 1		Solution 2	
Vehicle	**Customers**	**Vehicle**	**Customers**
2	5, 7, 9	2	4, 6, 8
3	4, 6, 8	3	5, 7, 9

This phenomenon increases run-time by expanding the search space of solutions. To get rid of the symmetry in the problem we introduce the following two sets of valid inequalities.

The first set of symmetry cuts forces each node to be serviced by a vehicle whose index is not greater than the index of the node. Node 1 is in vehicle 1, node 2 is in vehicles 1 or 2, node 3 is in vehicles 1, 2, or 3, and so on. This can be stated as follows.

$$\sum_{j=1}^{i} y_{ij} = 1 \qquad \forall\ i \in \{1, \ldots, |V|\} \tag{16}$$

The second set of symmetry cuts force the lowest indexed node in each vehicle to increase with vehicle index. This is stated as follows.

$$y_{i,i-k+3} \leq \sum_{j=1}^{k-2} y_{i-j,i-k+2} \qquad \forall\ i \in \{k, .., |V|\}, k \in \{3, .., |V|\} \tag{17}$$

Consider the solutions in Table 1 that are feasible to constraint (16). In the first solution, the lowest indexed node of vehicle 2 is greater than the lowest indexed node of vehicle 3. Thus, the first solution violates constraint (17) while the second solution is feasible.

The next set of inequalities stipulates that two nodes whose cumulative demand is greater than the capacity cannot be in the same vehicle.

$$y_{iv} + y_{jv} \leq 1 \qquad \forall\ i, j \in N : D_i + D_j > C, v \in V \tag{18}$$

In our computational experiments we found that the above valid inequalities improve the LP relaxation when the node demands are large (the mean node demand is comparable to half the vehicle capacity). To address the situation when the node demands are low we develop additional valid inequalities.

Our next set of valid inequalities places a lower bound on the number of vehicles that use an arc. For each arc, such a bound could be obtained by solving a bin-packing problem on the nodes of the sub-tree below that arc. The constraint, illustrated in Figure 5, is stated as follows.

$$\sum_{v \in V} y_{P_i iv} \geq V_i^{\min} \qquad \forall i \in N \tag{19}$$

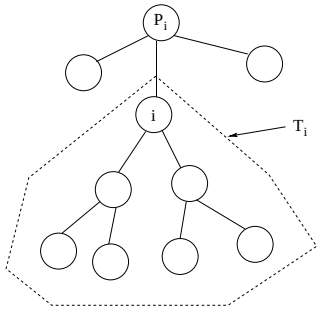

Fig. 5. The solution to the bin-packing problem on demands in $T(i)$ is a lower bound on the number of vehicles that travel along arc (P_i, i).

The lower bound we use is the bin-packing lower bound developed by Labbé et al. [8].

The final valid inequality stipulates that if a vehicle travels down an arc but does not service the node at the end of that arc, it *must* service some node below that arc in the tree. This constraint is illustrated in Figure 6.

$$\sum_{j \in c(i)} x_{ijv} \geq x_{P_i iv} - y_{iv} \qquad \forall\, v \in V, i \in N, c(i) \neq \phi \tag{20}$$

5 Approximation Algorithm

In this section we describe an approximation algorithm that is a combination of the algorithms of Labbé et al. [8] and Basnet et al. [1]. We will later use the upper bound distance from this algorithm to obtain an upper bound on the fleet size.

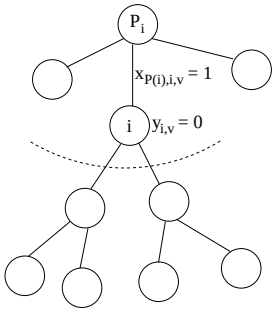

Fig. 6. If a vehicle travels along arc (P_i, i) but does not service node i, then it must service some node below i.

The heuristic of Labbé et al. [8] proceeds as follows. At each iteration, an arbitrary leaf node is chosen, and an attempt is made to merge the node with its parent. If the cumulative demand of the leaf node and its parent is less than or equal to the vehicle capacity, the demand of the leaf node is added to that of its parent, and the leaf node is removed from the tree. When the merger is not possible due to the cumulative demand exceeding vehicle capacity, there are two cases. If the leaf node has greater demand than that of its parent, a new vehicle route is created to the leaf node, and the leaf node is eliminated from the tree. Otherwise, a new route is created to the parent, the demand at the parent is replaced by that of its child, and the leaf node is eliminated. The process terminates since a leaf node is eliminated from the tree at each iteration.

The heuristic of Basnet et al. [1] could be viewed as a savings heuristic (Clarke and Wright [3]) specifically adapted to trees. Their heuristic proceeds as follows (the description provided here is different from the one in their paper). Define a *non-grandparent* node to be one that has only leaf nodes as children. At each iteration, the heuristic picks a non-grandparent node i that is farthest from the root. It then picks an arbitrary child j of this node i, and attempts to merge it with another child of node i. If i has multiple children, then j is merged with another child of i, say k, with the greatest demand such that $D_j + D_k \leq C$. While Basnet et al.[1] did not provide any worst case analysis, they showed that, in practice, their algorithm was better than that of Labbé et al. [8] over a wide range of problem instances.

Our algorithm proceeds as follows. At each stage, we pick a non-grandparent node i. We then pack the sub-tree $T(i) = c(i) \cup \{i\}$ into a minimal set of bins using a bin-packing heuristic. This creates possibly several bins, each with some total demand not exceeding the vehicle capacity. The nodes in the subtree $T(i)$ are then removed from the graph and are replaced by nodes representing the packed bins (one node for each packed bin), which are added to the tree as the children of P_i (parent of node i). This procedure continues until the depot is reached.

Our heuristic could be viewed as a combination of the heuristics of Labbé et al. [8] and Basnet et al. [1] in the following sense. While the former heuristic attempts to pack the demand of an arbitrary leaf node and its parent, and the latter attempts to pack the demand of all children of the farthest non-grandparent node from the depot, our algorithm picks an arbitrary non-grandparent node and attempts to pack its demand and those of its children. What sets our heuristic apart is that it simultaneously considers several nodes for merging, instead of merging nodes in a pair-wise fashion.

Figure 7 illustrates the algorithm. The sample problem is shown in Figure 7(a). In the first iteration, the nodes with demands 40, 50 and 70 are packed. This results in two nodes—one with demand 90 and one with demand 70. In the next iteration (see Figure 7(b)), nodes with demand 5, 10, 20, 70 and 90 are packed to give two nodes—one with demand 100 and one with demand 95. Since the depot has been reached (in Figure 7(c)), the algorithm stops. The

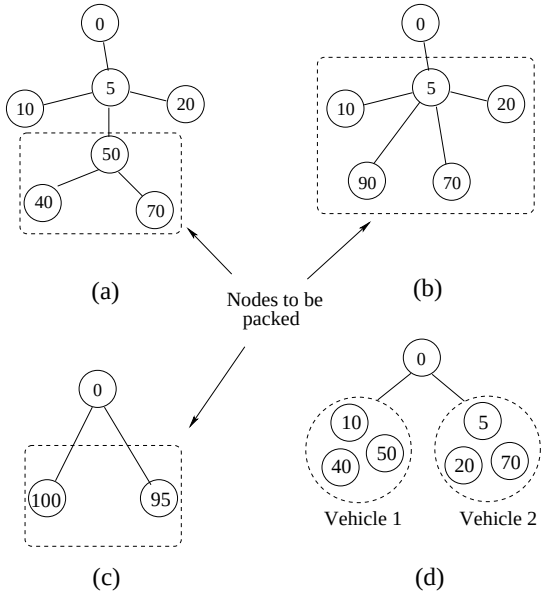

Fig. 7. (a) Initial tree, (b) and (c) Tree after packing operations, (d) Feasible solution obtained from the approximation algorithm.

solution is shown in Figure 7(d). The pseudocode for the algorithm is given in Figure 8.

Lemma 2. Approx-TCVRP*, described in Figure 8, is a 2-approximation algorithm.*

Proof. Our proof is similar to that of Labbét al. [8]. At each iteration, a node i is chosen and the set of nodes $c(i) \cup \{i\}$ is packed. The result of this packing generates a set of bins such that at most one bin carries demand not exceeding half the vehicle capacity (if two bins each carried demand less than or equal to half the vehicle capacity, they would be merged into one vehicle). Thus, the number of vehicles that travel down the arc (P_i, i) is at most $\lceil (\sum_{j \in T(i)} D_i)/(C/2) \rceil$. The number of vehicles required to service $T(i)$ is at least $\lceil (\sum_{j \in T(i)} D_i)/C \rceil$. Thus, the number of vehicles that travel on arc (P_i, i) from the algorithm is at most twice the lower bound. Summing over all arcs in the tree, the total distance traveled by all vehicles in the algorithm is at most twice that of the lower bound, and thus at most twice the optimal distance. □

The algorithm has as many iterations as there are non-leaf nodes. Hence, the complexity of the algorithm is essentially the complexity of the packing function times the number of non-leaf nodes. If we choose a "simple" pack-

/ This function computes an upper bound on the optimal TCVRP distance.*/*

Notation:

N	Set of nodes in the tree.
A	Set of arcs in the tree.
P_i	Parent of node i.
$c(i)$	Set of children of node i.
C	Vehicle capacity.
z	The total distance of the feasible solution.
`BinPack`	A function that takes as input a set of demand nodes, packs them into bins of capacity C, and returns the set of packed bins.

function approx-TCVRP:
begin
$z \leftarrow 2\sum_{i \in N} S_i$;
while $\exists\ i \in N:\ (i \neq \text{Depot}) \wedge (c(i) \neq \phi) \wedge (c(j) = \phi\ \forall j \in c(i))$ **do**
$R \leftarrow c(i) \cup i$;
$Q \leftarrow$ `BinPack`(R);
$z \leftarrow z - S_i\,(|R| - |Q|)$;
$P_j \leftarrow P_i\ \forall\ i \in Q$;
$N \leftarrow N \cup Q \setminus R$;
$A \leftarrow A \cup \{(P_i, i)\ \forall j \in Q\} \setminus \{(i,k)\ \forall k \in c(i)\} \setminus \{(P_i, i)\}$;
end

Fig. 8. The approx-TCVRP algorithm.

ing heuristic such as the first fit decreasing heuristic which has complexity $O(|N|^2)$, the overall complexity of the approximation algorithm is $O(|N|^3)$.

6 Computational Experiments

6.1 Enhancements to the Treeroute Formulation

We now discuss some details of the implementation of the treeroute formulation that significantly affect its performance.

Upper bound on the fleet size

The treeroute formulation requires the maximum number of vehicles to be specified. An upper bound on the number of vehicles (see Section 5) is given by

$$|V| \leq \left\lceil \frac{2\sum_{i \in N} D_i}{C} \right\rceil$$

However, we would like to have a tighter upper bound on the number of vehicles to reduce the number of variables in the formulation. Let $\hat{z}$ be a heuristic upper bound on the optimal distance. Consider the optimization problem obtained by taking the constraints in the treeroute formulation, placing a bound on the objective function, and maximizing the number of vehicles leaving the depot

$$\text{Maximize} \quad \sum_{v \in V} x_{0c(0)v} \tag{21}$$

$$\text{Subject to} \quad 2 \sum_{(i,j) \in A} \sum_{v \in V} L_{ij} x_{ijv} \leq \hat{z} \tag{22}$$

$$\text{and} \qquad (11) - (20)$$

The objective function maximizes the number of vehicles leaving the depot. Clearly, an optimal solution to this optimization problem will be an upper bound on the number of vehicles in the optimal solution to the TCVRP. However, since this integer program is as hard to solve as the original TCVRP, we solve the linear programming relaxation of this optimization problem and round down the optimal objective function value to obtain an upper bound on the fleet size. The time taken to solve this LP is negligible, and in our experiments gave very tight upper bounds (within 5-10% of optimal) on the fleet size.

Node ordering

In its basic form, the LP relaxation is quite poor. This is because the LP relaxation tends to split demands between vehicles. In initial experiments, we found that the ordering of nodes (order in which nodes are indexed) affects performance. The node indexing scheme that we found to improve performance the greatest was to order nodes in order of decreasing demand. This makes use of the symmetry cuts early on to "fill up" lower indexed vehicles (which leads to less unused capacity). There is less opportunity to split the demand of a high-indexed node among the low indexed vehicles.

Variable branching order

Finally, we consider the order of branching of variables in the branch-and-cut procedure. Most software packages for solving integer linear programs allow the user flexibility in specifying the branching order of variables. Since the lower indexed nodes have fewer vehicles available to them, we branch on these variables first (to prevent splitting early on in the branch-and-cut tree). Thus, y_{iv} is branched on before y_{jv} for all $i < j$. Given a node, the order of branching among the different vehicles is left to the default strategy of the solver.

6.2 Experimental Setup

We tested the formulations on networks with 20, 40, 60, 80, and 100 nodes. We forced the degree of the depot to 1 in all our test instances. In reality, since the degree of the depot need not be 1, it is conceivable that substantially larger problems than the ones listed in this section can be solved (recall problems with depot degree k can be decomposed into k separate problems).

We generated test problems as in Labbé et al. [8] and Mbaraga et al. [9]. The arc distances were integers uniformly distributed in [1,100]. The vehicle capacity for all vehicles was set to be 100 units. Each node in the tree (apart from the depot) had between 1 and 5 children (distributed uniformly). Node demands were integers uniformly distributed between a lower and upper bound, which were varied to create 10 instance classes with the following demand bounds: [1,100], [10,90], [20,80], [30,70], [1,50], [1,30], [1,10], [30,30], [20,20], and [10,10].

The times reported in this section are CPU times. The formulations were coded in AMPL [5] and solved using CPLEX 9.0 [6] on a PC with 3.06 GHz processor and 512 MB of RAM. The data for all the problems requires some pre-processing (node ordering, finding lower and upper bounds on the number of vehicles, generating arc minimum-traversal constraints) but the time required to perform this computation is insignificant (of the order of seconds) compared to the time to solve the IP, and is not included under the solving time listed.

6.3 Experimental Results

Computational results are presented in Tables 2 through 6, which contain the following information/notation.

1. Solved — Number of instances (out of 10) that were solved to optimality within 3600 CPU seconds.
2. Time — The CPU time averaged only over solved instances.
3. LP/IP — Ratio of the linear programming relaxation to the optimal distance, averaged only over solved instances.
4. LP*/IP — Ratio of linear programming relaxation *without valid inequalities* (constraints (2)–(6) for the DFS formulation and constraints(11)–(14) for the treeroute formulation) to the optimal distance, reported only for the 20-node instances.
5. Gap — This is defined as $(U - L)/L$, where U is the best solution found at termination, and L is the bound at termination (3600 seconds). This is averaged only over unsolved instances.

We observe that for the 20 node instances, the treeroute formulation performs extremely well, while the DFS formulation performs well on all but three instance classes, i.e., when the demands are in $[1, 50]$, $[1, 30]$, and $[30, 30]$. This

Table 2. Results for 20 Node Problems.

DFS FORMULATION								
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	LP*/IP	Gap
[1,100]	1310	380	10	0.26	185	0.980	0.690	-
[10,90]	1310	380	10	0.37	274	0.977	0.662	-
[20,80]	1304	380	10	0.17	10	0.973	0.652	-
[30,70]	1297	380	10	0.07	3	0.980	0.602	-
[1,50]	1216	380	9	267.45	66492	0.983	0.738	0.012
[1,30]	1216	380	10	50.97	2568	0.990	0.839	-
[1,10]	1216	380	10	0.06	0	1.000	0.932	-
[30,30]	1216	380	5	1678.45	524860	0.965	0.667	0.029
[20,20]	1216	380	10	1.33	30	1.000	0.738	-
[10,10]	1216	380	10	2.92	16	1.000	0.953	-
TREEROUTE FORMULATION								
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	LP*/IP	Gap
[1,100]	655	456	10	0.03	10	0.979	0.447	-
[10,90]	644	456	10	0.06	44	0.977	0.481	-
[20,80]	668	475	10	0.07	21	0.972	0.491	-
[30,70]	631	448	10	0.08	54	0.980	0.439	-
[1,50]	319	224	10	1.36	1251	0.982	0.700	-
[1,30]	214	144	10	0.06	49	0.990	0.818	-
[1,10]	129	76	10	0.00	0	1.000	0.930	-
[30,30]	368	266	10	18.26	19791	0.960	0.633	-
[20,20]	220	152	10	0.02	1	1.000	0.710	-
[10,10]	129	76	10	0.01	0	1.000	0.943	-

difference in performance occurs in spite of the LP relaxations being comparably strong. Note that the valid inequalities significantly improve the quality of the LP-relaxation for both formulations.

As the problem sizes increase, the DFS formulation starts to perform very poorly except on the $[20, 80]$ and $[30, 70]$ instances. Interestingly, although the time taken by the treeroute formulation on these instances is less than the DFS formulation, the number of branch-and-bound nodes explored by the DFS formulation is much smaller. This suggests that the DFS formulation performs a lot of work solving the successive linear programs, which is not surprising given the huge number of constraints.

In all of the instances, the termination gap for the treeroute formulation is very small (almost always less than 2%, and often less than 1%), which is a sufficiently high accuracy in most practical situations.

Table 3. Results for 40 Node Problems.

DFS FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	10433	1560	5	90.02	4859	0.993	0.016
[10,90]	10375	1560	6	944.99	29622	0.972	0.013
[20,80]	10390	1560	8	38.26	648	0.960	0.015
[30,70]	10369	1560	10	5.63	82	0.968	-
[1,50]	10036	1560	0	-	-	-	0.023
[1,30]	10036	1560	2	61.04	16	1.000	0.011
[1,10]	10036	1560	10	283.61	86	0.998	-
[30,30]	10036	1560	0	-	-	-	0.024
[20,20]	10036	1560	4	932.21	1920	1.000	0.000
[10,10]	10036	1560	4	1251.99	408	1.000	0.001
TREEROUTE FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	2544	1864	10	1.65	319	0.979	-
[10,90]	2447	1794	10	14.31	2481	0.968	-
[20,80]	2594	1888	10	135.30	30969	0.959	-
[30,70]	2516	1833	9	70.75	13905	0.969	0.003
[1,50]	1205	913	5	356.74	38562	0.998	0.023
[1,30]	743	554	10	113.93	43781	0.996	-
[1,10]	346	226	10	0.04	6	0.998	-
[30,30]	1389	1061	4	321.46	13011	0.982	0.012
[20,20]	836	624	10	18.35	1187	1.000	-
[10,10]	459	320	10	0.96	176	0.999	-

The DFS formulation appears to perform best on instances where the demand is in $[30, 70]$. This is because the size of the cliques created are relatively small. The DFS formulation does extremely poorly when the demand is small (packing problem is more complex). The treeroute formulation, on the other hand, does poorly on the $[30, 70]$ and $[20, 80]$ instances and does well when the demands are relatively small. Although the LP relaxation is very strong, the treeroute formulation struggles on some instances due to problem symmetry.

7 Conclusions

In this chapter we presented two integer linear programming formulations and valid inequalities for the capacitated vehicle routing problem on trees.

Table 4. Results for 60 Node Problems.

DFS FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	35327	3540	3	811.53	9978	0.977	0.022
[10,90]	35373	3540	3	710.26	10341	0.981	0.023
[20,80]	35306	3540	7	605.51	2928	0.973	0.006
[30,70]	35260	3540	9	316.52	2	0.971	0.002
[1,50]	34456	3540	0	-	-	-	0.030
[1,30]	34456	3540	0	-	-	-	0.018
[1,10]	34456	3540	7	399.91	0	1.000	0.024
[30,30]	34456	3540	0	-	-	-	0.039
[20,20]	34456	3540	0	-	-	-	0.000
[10,10]	34456	3540	0	-	-	-	0.012
TREEROUTE FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	5744	4283	9	100.97	7757	0.971	0.010
[10,90]	5965	4437	7	34.52	2939	0.977	0.005
[20,80]	5843	4342	7	736.61	58913	0.975	0.003
[30,70]	5787	4295	9	230.44	20653	0.974	0.004
[1,50]	2829	2171	0	-	-	-	0.026
[1,30]	1630	1251	4	382.15	11975	0.998	0.004
[1,10]	707	496	10	0.34	18	0.999	-
[30,30]	3127	2407	0	-	-	-	0.015
[20,20]	1831	1416	5	474.45	5578	1.000	0.000
[10,10]	1063	791	6	444.46	28984	1.000	0.004

The formulations proposed exploit the tree structure of the graph to achieve high quality solutions for the TCVRP.

While it is difficult to compare the results of Mbaraga et al. [9] to ours due to differences in testing methodology (restricting the degree of the depot, different cutoff time limits, different computing resources), we were able to solve problems of comparable size to those in their study (they tested and were able to solve problems of up to 140 nodes). Our key contribution is that the performance of these formulations is competitive with other special purpose code developed for the problem, and can be implemented easily using off-the-shelf software for modeling and solving integer programs. This is a significant benefit to practitioners, who may not have the mathematical and programming sophistication to implement branch-and-cut, branch-and-price,

Table 5. Results for 80 Node Problems.

DFS FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	83897	6320	0	-	-	-	0.042
[10,90]	83956	6320	0	-	-	-	0.037
[20,80]	83993	6320	3	1622.71	326	0.974	0.024
[30,70]	84062	6320	6	2016.72	1	0.976	0.028
[1,50]	82476	6320	0	-	-	-	0.023
[1,30]	82476	6320	0	-	-	-	0.018
[1,10]	82476	6320	0	-	-	-	0.003
[30,30]	82476	6320	0	-	-	-	0.047
[20,20]	82476	6320	0	-	-	-	0.002
[10,10]	82476	6320	0	-	-	-	0.008
TREEROUTE FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	9670	7173	4	896.29	30387	0.975	0.009
[10,90]	10167	7616	4	1526.81	59840	0.969	0.009
[20,80]	10671	7853	4	719.30	17443	0.968	0.011
[30,70]	10941	8105	7	167.33	7033	0.977	0.004
[1,50]	4472	3460	0	-	-	-	0.023
[1,30]	2880	2244	1	3489.74	14669	1.000	0.014
[1,10]	1110	806	10	4.03	183	1.000	-
[30,30]	5715	4440	0	-	-	-	0.024
[20,20]	3280	2544	1	245.37	876	1.000	0.002
[10,10]	1767	1359	3	899.29	12397	1.000	0.005

or even native optimization code (i.e., code directly using the callable library of the optimization package).

The formulations were tested on a large set of instances with varying number of nodes and demand patterns. As shown by the results in Table 2, the proposed valid inequalities significantly improve the strength of both formulations. It is found that the DFS formulation performs well on small problems where the demands exhibit low variance and average demand is large. The DFS formulation performs poorly when the packing problem is complex (if there exist demands that are much smaller than the vehicle capacity). The treeroute formulation performs well on instances where the average demand is large and the demands show considerable variance, due to a lack of symmetry in these problems. In general, both formulations have tight LP relaxations,

Table 6. Results for 100 Node Problems.

DFS FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	164495	9900	0	-	-	-	0.043
[10,90]	164450	9900	0	-	-	-	0.044
[20,80]	164540	9900	0	-	-	-	0.039
[30,70]	164491	9900	1	3480.01	0	0.972	0.035
[1,50]	162096	9900	0	-	-	-	0.041
[1,30]	162096	9900	0	-	-	-	0.020
[1,10]	162096	9900	0	-	-	-	0.005
[30,30]	162096	9900	0	-	-	-	0.048
[20,20]	162096	9900	0	-	-	-	0.003
[10,10]	162096	9900	0	-	-	-	0.005
TREEROUTE FORMULATION							
Demand	Cons.	Vars.	Solved	Time (sec.)	Nodes	LP/IP	Gap
[1,100]	15357	11484	1	128.74	5743	0.982	0.007
[10,90]	16022	12019	1	16.74	54	0.971	0.008
[20,80]	16530	12296	2	475.99	30570	0.981	0.009
[30,70]	16739	12474	2	30.87	672	0.975	0.010
[1,50]	7882	6158	0	-	-	-	0.041
[1,30]	4457	3524	0	-	-	-	0.015
[1,10]	1717	1287	8	53.56	867	0.999	0.001
[30,30]	9088	7088	0	-	-	-	0.026
[20,20]	5080	4019	0	-	-	-	0.003
[10,10]	2728	2119	1	57.79	285	1.000	0.003

but the treeroute formulation often out-performs the DFS formulation and should be the model of choice.

Recall, the number of integer variables in the DFS formulation is $O(|N|^2)$ and for the treeroute formulation is $O(|N||V|)$. Further, the number of constraints in the DFS model is $O(|N|^3)$, while that in the treeroute formulation is $O(|N||V|)$. Hence, the treeroute formulation is more scalable than the DFS formulation, which is one reason for its better performance. We should also note that the 2-approximation algorithm described in Section 5 gives very tight upper bounds (usually within 2% of optimal).

The treeroute formulation can be extended to the heterogeneous vehicle routing problem introduced by Mbaraga et al. [9]. This is done by eliminating the symmetry cuts and modifying the capacity constraint (13) to

$$\sum_{i \in N} y_{iv} D_i \leq C_v \qquad \forall\ v \in V, \tag{23}$$

where C_v is the capacity of vehicle v.

In summary, we have developed strong models that exploit the structure of the underlying graph, and a heuristic procedure, for the TCVRP problem. Our motivation, was to develop models within the framework afforded by high-level optimization modeling languages like AMPL, GAMS, and OPL, with a view to easy implementation for the practitioner. To that end, we believe we have successfully demonstrated that by cleverly exploiting the structure of the underlying graph, it is possible to solve large-scale versions of the TCVRP even while using a high-level modeling language. We hope this research will spur further discussion of specialized models for VRP (and other network problems), with a greater focus on the use of the models within a high-level modeling language (as is often the case in practice).

References

1. C. Basnet, L.R. Foulds, and J.M. Wilson. Heuristics for vehicle routing on tree-like networks. *Journal of the Operational Research Society*, 50:627–635, 1999.
2. I. Berger, J.-M. Bourjolly, and G. Laporte. Branch-and-bound algorithms for the multi-product assembly line balancing problem. *European Journal of Operational Research*, 58:215–222, 1992.
3. G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.
4. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing – an updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithms design for computer system design*, pages 49–106. Springer-Verlag, New York, 1984.
5. R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, 1993.
6. ILOG. Cplex 9.0 reference manual, 2003.
7. J. Kallrath, editor. *Modeling Languages in Mathematical Optimization*, volume 88 of *Applied Optimization*. Springer Publishing Company, 2004.
8. M. Labbé, G. Laporte, and H. Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39:616–622, 1991.
9. P. Mbaraga, A. Langevin, and G. Laporte. Two exact algorithms for the vehicle routing problem on trees. *Naval Research Logistics*, 46:75–89, 1999.
10. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.
11. P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, Philadelphia, PA, 2002.