USING BRANCH-AND-PRICE-AND-CUT TO SOLVE ORIGIN-DESTINATION INTEGER MULTICOMMODITY FLOW PROBLEMS

CYNTHIA BARNHART

Massachusetts Institute of Technology, Center for Transportation Studies, Cambridge, Massachusetts 02139, cbarnhar@mit.edu

CHRISTOPHER A. HANE

CAPS Logistics, Atlanta, Georgia 30334, chane@baan.com

PAMELA H. VANCE

Auburn University, Industrial and Systems Engineering, Auburn, Alabama 36849, Pamela_Vance@bus.emory.edu

(Received July 1996; revisions received May 1997, February 1998; accepted July 1998)

We present a column-generation model and branch-and-price-and-cut algorithm for origin-destination integer multicommodity flow problems. The origin-destination integer multicommodity flow problem is a constrained version of the linear multicommodity flow problem in which flow of a commodity (defined in this case by an origin-destination pair) may use only one path from origin to destination. Branch-and-price-and-cut is a variant of branch-and-bound, with bounds provided by solving linear programs using column-and-cut generation at nodes of the branch-and-bound tree. Because our model contains one variable for each origin-destination path, for every commodity, the linear programming relaxations at nodes of the branch-and-bound tree are solved using column generation, i.e., implicit pricing of nonbasic variables to generate new columns or to prove LP optimality. We devise a new branching rule that allows columns to be generated efficiently at each node of the branch-and-bound tree. Then, we describe cuts (cover inequalities) that can be generated at each node of the branch-and-bound tree. These cuts help to strengthen the linear programming relaxation and to mitigate the effects of problem symmetry. We detail the implementation of our combined column-and-cut generation method and present computational results for a set of test problems arising from telecommunications applications. We illustrate the value of our branching rule when used to find a heuristic solution and compare branch-and-price and branch-and-price-and-cut methods to find optimal solutions for highly capacitated problems.

1. INTRODUCTION

Linear multicommodity flow problems are linear programs (LPs) that can be characterized by a set of commodities and an underlying network. The objective is to flow the commodities through the network at minimum cost without exceeding arc capacities. A comprehensive survey of linear multicommodity flow models and solution procedures was presented in Ahuja et al. (1993).

In this paper, we consider an origin-destination integer multicommodity flow (ODIMCF) problem, a constrained version of the linear multicommodity flow problem in which flow of a commodity (defined in this case by an origin-destination pair) may use only one path from origin to destination. ODIMCF problems are prevalent in a number of application contexts, including transportation, communication, and production. Example applications include:

1. *Bandwidth packing problems* require that bandwidth be allocated in telecommunications networks to maximize total revenue. The demands, or calls, on the networks are the commodities and the objective is to route the calls from their origin to their destination. In the case of video teleconferencing, call splitting is not allowed, and so each call must be routed on exactly one network path.
2. *Package flow problems*, such as those arising in express package delivery operations, require that shipments, each with a specific origin and destination, be routed over a transportation network. Each set of packages with a common origin-destination pair can be considered as a commodity and often must be assigned to a single network path to facilitate operations and ensure customer satisfaction.

We present a column-generation model for the class of integer multicommodity flow problems described above and a branch-and-bound solution approach involving column and row generation. Column-generation models, such as those presented in Ahuja et al. (1993), Barnhart et al. (1995a), and Jones et al. (1993), have been used extensively in modeling and solving large versions of the linear MCF problem. In column generation, sets of columns are left out of the LP because there are too many columns to handle efficiently, and most of them will have their associated variable equal to zero in an optimal solution. Then to check the optimality of an LP solution, a subproblem called the pricing problem, which is a separation problem for the dual LP, is solved to try to identify columns to enter the basis. If such columns are found, the LP is reoptimized.

Subject classifications: Networks/graphs: multicommodity. Programming, integer: branch-and-bound. Programming, linear: column generation.
Area of review: OPTIMIZATION

The ability to solve large MCF LPs allows us to consider the solution of large ODIMCF problems. We design, implement, and test a new branch-and-price-and-cut solution approach. Branch-and-price, a generalization of branch-and-bound with LP relaxations, allows column generation to be applied throughout the branch-and-bound tree. Branching occurs when no columns price out to enter the basis and the LP solution does not satisfy the integrality conditions. Branch-and-cut, another variant of branch-and-bound, allows valid inequalities—or cuts—to be added throughout the branch-and-bound tree. Our cuts are designed to eliminate problem symmetry, improving the effectiveness of the branching decisions. Problem symmetry causes branch-and-bound to perform poorly because the problem barely changes after branching (Barnhart et al. 1995, Vance et al. 1994). Branch-and-price-and-cut combines column and row generation to yield very strong LP relaxations at nodes of the branch-and-bound tree. However, synthesizing the two generation processes is nontrivial. Only a limited number of applications of combined row and column generation in solving integer programs can be found in the literature, including those presented in the survey of Desrosiers et al. (1995) and in Mehrotra (1992) and Nemhauser and Park (1991).

We implement and test our branch-and-price-and-cut (BPC) solution procedure using a set of test problems representative of problems arising in the telecommunications industry. We demonstrate that when column generation is performed only at the root node and a standard branch-and-bound approach is used, it is not possible to identify feasible IP solutions for some large problems. In contrast, we show that our solution procedure, allowing column and row generation at each node of the tree, produces near-optimal solutions to our test problems with reasonable run times on workstation class computers.

1.1. Contributions

In our view, the contributions of this paper include:

- Development of an optimization solution strategy for an important class of problems called origin-destination integer multicommodity flow problems. The novel features of our *branch-and-price-and-cut algorithm* include:
  1. A pricing algorithm that does not change even as cuts are added, and similarly, a separation algorithm that does not change even as columns are added. Often with branch-and-price-and-cut, an objective is to maintain the form of the pricing and separation algorithms as columns and cuts are added so that algorithmic complexity and run times do not increase as the algorithm progresses because the result may cause large applications to be intractable.
  2. A more generalized branching strategy than has been presented in the literature, with benefits including:
     — Our branching rule is enforced without adding constraints to the problem, thereby maintaining the form of the pricing problem.
     — Our branching is more effective than conventional branching based on variable dichotomy because the number of potential branches is far less and the solution space is more evenly partitioned.
  3. Lifted cover inequalities that help to strengthen the LP relaxation of our model and help to overcome problem symmetry.
- Implementation and evaluation of our algorithm using test problems arising from the telecommunications industry. Our results show that with our procedure, near-optimal integer solutions can be obtained in reasonable run times on workstation class computers.

1.2. Outline

The remainder of the paper is organized as follows. In §2, we present two formulations for the ODIMCF problem. In §3, we detail the approach for obtaining ODIMCF solutions using a branch-and-price solution approach. Branching rules are introduced and details of how to generate columns satisfying the branching decisions are provided. Branch selection, node selection, and branch-and-bound termination are also discussed. Computational results to evaluate our branching rule on a static set of columns and our branch-and-price procedure are presented in §4. Next, in §5, the branch-and-price procedure is enhanced to include cut generation at nodes of the tree. Details of the cuts and their effect on column generation are included. In §6, we describe the performance of our enhanced branch-and-price-and-cut procedure. Finally, in §7 we summarize the results of our experiments.

2. ODIMCF PROBLEM FORMULATION

We consider two different formulations for the ODIMCF problem: the *node-arc* or *conventional* formulation and the *path* or *column-generation* formulation. We use the conventional formulation to design our branching strategies for the ODIMCF problem and derive our cutting planes. The column-generation formulation is used to solve the ODIMCF LP relaxation.

The origin-destination integer multicommodity flow formulation, denoted *ODIMCF*, is defined over the network G comprising node set N and arc set A. *ODIMCF* contains binary decision variables x, where x_{ij}^k equals 1 if the entire quantity (denoted q^k) of commodity k is assigned to arc ij, and equals 0 otherwise. The cost of assigning commodity k in its entirety to arc ij equals q^k times the unit flow cost for arc ij, denoted c_{ij}^k. Arc ij has capacity d_{ij}, for all $ij \in A$. Node i has supply of commodity k, denoted b_i^k, equal to 1 if i is the origin node for k, equal to -1 if i is the destination node for k, and equal to 0 otherwise.

The conventional or node-arc *ODIMCF* formulation is:

$$\min \sum_{k \in K} \sum_{ij \in A} c_{ij}^k q^k x_{ij}^k, \tag{1}$$

s.t.

$$\sum_{k \in K} q^k x_{ij}^k \leq d_{ij}, \qquad \forall\, ij \in A, \tag{2}$$

$$\sum_{ij \in A} x_{ij}^k - \sum_{ji \in A} x_{ji}^k = b_i^k, \qquad \forall\, i \in N,\ \forall\, k \in K, \tag{3}$$

$$x_{ij}^k \in \{0, 1\}, \quad \forall\, ij \in A,\ \forall\, k \in K. \tag{4}$$

Note that without restricting generality of the problem, we model the arc flow variables x as binary variables. To do this, we scale the demand for each commodity to 1 and accordingly adjust the coefficients in the objective function (1) and in constraints (2).

To contrast, the path-based or column-generation ODIMCF formulation has fewer constraints and far more variables. Again, the underlying network G is composed of node set N and arc set A, with q^k representing the quantity of commodity k. $P(k)$ represents the set of all origin-destination paths in G for k, for all $k \in K$. In the column-generation model, the binary decision variables are denoted y_p^k, where y_p^k equals 1 if all q^k units of commodity k are assigned to path $p \in P(k)$, and equals 0 otherwise. The cost of assigning commodity k in its entirety to path p equals q^k times the unit flow cost for path p, denoted c_p^k. As before, arc ij has capacity d_{ij}, for all $ij \in A$. Finally, δ_{ij}^p is equal to 1 if arc ij is contained in path $p \in P(k)$, for some $k \in K$ and is equal to 0 otherwise.

The path or column-generation ODIMCF formulation is then:

$$\min \sum_{k \in K} \sum_{p \in P(k)} c_p^k q^k y_p^k, \tag{5}$$

s.t.

$$\sum_{k \in K} \sum_{p \in P(k)} q^k y_p^k \delta_{ij}^p \leq d_{ij}, \quad \forall\, ij \in A, \tag{6}$$

$$\sum_{p \in P(k)} y_p^k = 1, \quad \forall\, k \in K, \tag{7}$$

$$y_p^k \in \{0, 1\}, \quad \forall\, p \in P(k),\ \forall\, k \in K. \tag{8}$$

For large-scale transportation, communication, and production applications, the LP relaxation of the conventional ODIMCF formulation contains a large number of constraints (equal to the number of arcs plus the product of the number of nodes and commodities) and a large number of variables (equal to the product of the number of arcs and commodities.) The column-generation LP relaxation, however, contains a moderate number of constraints (one for each commodity and one for each arc) and a huge number of variables (one for each path for each commodity.) Without decomposition, these LP relaxations may require excessive memory and/or run times to solve.

3. SOLUTION APPROACH

We use a branch-and-bound approach to solve the ODIMCF problems, with bounds provided by solving a LP relaxation, called the *subproblem*, at each node of the branch-and-bound tree.

3.1. LP Solution

We choose to use the column-generation solution approach to solve the LP relaxation of the path-based formulation of ODIMCF. The general idea of column generation is that optimal solutions to large LPs can be obtained without explicitly including all columns (i.e., variables) in the constraint matrix (called the *master problem* or MP). In fact, only a very small subset of all columns will be in an optimal solution, and all other (nonbasic) columns can be ignored. In a minimization problem, this implies that all columns with positive reduced cost can be ignored.

We refer to a MP with only a subset of its columns as the *restricted master problem* or RMP. The column-generation algorithm solves the LP relaxation of MP by solving the LP relaxations of several RMPs. After finding the LP solution to a RMP, we determine whether there are any columns not included in the RMP with negative reduced cost. If none can be found, the current LP solution to the RMP is optimal for MP also. If one or more such columns do exist, they are added to RMP and the process is repeated.

For any RMP, let $-\pi_{ij}$ represent the nonnegative dual variables associated with constraints (6) and σ^k represent the unrestricted dual variables associated with constraints (7). Because c_p^k can be represented as $\Sigma_{ij \in A}\, c_{ij}^k \delta_{ij}^p$, the reduced cost of column p for commodity k, denoted $\bar{c}_p^k$, is:

$$\bar{c}_p^k = \sum_{ij \in A} q^k (c_{ij}^k + \pi_{ij}) \delta_{ij}^p - \sigma^k, \ \forall\, p \in P(k),\ \forall\, k \in K. \tag{9}$$

For each RMP solution generated, the pricing problem can be solved efficiently. Columns that price out can be identified by solving one shortest path problem for each commodity $k \in K$ over a network with arc costs equal to $c_{ij}^k + \pi_{ij}$, for each $ij \in A$. Denote the cost (using arc costs $c_{ij}^k + \pi_{ij}$) of the *shortest* path $p*$ for any commodity k as c_{p*}^k. Then, if for all $k \in K$,

$$c_{p*}^k q^k - \sigma^k \geq 0,$$

the MP is solved. Otherwise, the MP is not solved, and for each $k \in K$ with

$$c_{p*}^k q^k - \sigma^k < 0,$$

path $p* \in P(k)$ is added to the RMP.

3.2. IP Solution

Because the multicommodity flow LPs are solved using column generation, if we wish to find optimal solutions, our branch-and-bound procedure must allow columns to be generated at each node of the tree. This approach is referred to as *branch-and-price*. For general expositions of branch-and-price methodology, see Barnhart et al. (1995), Vanderbeck and Wolsey (1994), and Desrosiers et al. (1995).

The key to developing a branch-and-price procedure is identifying a branching rule that eliminates the current fractional solution without compromising the tractability of the pricing problem. Barnhart et al. (1995) develop

branching rules for a number of different master problem structures. They also survey specialized algorithms that have appeared in the literature for a broad range of applications.

Parker and Ryan (1994) present a branch-and-price algorithm for the bandwidth packing problem, which is closely related to ODIMCF. The bandwidth packing problem is a version of ODIMCF where the objective is to choose which of a set of commodities to send in order to maximize revenue. They use a path-based formulation. Their branching scheme selects a fractional path and creates a number of new subproblems equal to the length of the path plus one. On one branch the path is fixed into the solution, and on each other branch one of the arcs on the path is forbidden. To limit time spent searching the tree, they use a dynamic optimality tolerance. They report the solution of 14 problems with as many as 93 commodities on networks with up to 29 nodes and 42 arcs. All but two of the instances are solved to within 95% of optimality.

Branching

Applying a standard branch-and-bound procedure to the restricted master problem with its existing columns will not guarantee an optimal (or feasible) solution. After the branching decision modifies RMP, it may be the case that there exists a column for MP that prices out favorably but is not present in RMP. Therefore, to find an optimal solution we must maintain the ability to solve the pricing problem after branching. The importance of generating columns after the initial LP has been solved is demonstrated for airline crew scheduling applications in Vance et al. (1994). They were unable to find feasible IP solutions using just the columns generated to solve the initial LP relaxation. They developed a branch-and-price approach for crew scheduling problems in which they generated additional columns whenever the LP bound at a node exceeded a preset IP target objective value.

The difficulty in incorporating column generation with branch-and-bound is that conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem. To illustrate, consider branching based on variable dichotomy in which one branch forces commodity k to be assigned to path p, i.e., $y_p^k = 1$, and the other branch does not allow commodity k to use path p, i.e., $y_p^k = 0$. The first branch is easy to enforce because no additional paths need to be generated once k is assigned to path p. The latter branch, however, cannot be enforced if the pricing problem is solved as a shortest path problem. There is no guarantee that the solution to the shortest path problem is not path p. In fact, it is likely that the shortest path for k is indeed path p. As a result, to enforce a branching decision, the pricing problem solution must be achieved using a *next shortest path procedure.* In general, for a subproblem involving a set of a branching decisions, the pricing problem solution must be achieved using a kth *shortest path procedure*.

For the multicommodity flow application, our objective is to ensure that the pricing problem for the LP with the branching decisions included can be solved efficiently with a shortest path procedure. That is, our objective is to design a branching rule that does not destroy the structure of the pricing problem. In general, this can be achieved by basing our branching rules on variables in the original formulation and not on variables in the column-generation formulation (Barnhart et al. 1995, Desrosiers et al. 1995). This means that our branching rules should be based on the arc flow variables x_{ij}^k.

Consider then branching based on variable dichotomy in the original variables. On one branch, we would force flow of commodity k to use arc ij, i.e., $x_{ij}^k = 1$; and on the other branch, we wouldn't allow commodity k to use arc ij, i.e., $x_{ij}^k = 0$. This time, the second branch is easy to enforce in the pricing problem by setting the cost of arc ij for k to a very large value. Enforcing the first branching decision, however, makes the pricing problem intractable. Whereas a shortest path containing an arc ij can be found by solving two shortest paths procedures, one from node j and one from the origin node of k to node i, it is not possible to find efficiently the shortest path containing a *set* of arcs, as required in subproblems at depths of two or more in the tree.

We propose a new branching strategy that:

(1) is based on the arc flow variables in the original problem formulation; and

(2) is compatible with the pricing problem solution procedure; that is, can be enforced without destroying the structure of the pricing problem.

We derive our branching rule by observing that if commodity k is assigned to more than one path, say for the purposes of this discussion, to two paths, then the two paths differ by at least one arc and, further, that the two paths have at least two nodes in common (i.e., the origin and destination nodes are contained in both paths.) We define the first node at which the two paths split as the *divergence node*. Given any two distinct paths $p1$ and $p2$ for k, we can find their divergence node by tracing each path, beginning with the origin node of k, one arc at a time until two different arcs called $a1$ and $a2$ are identified for each path. The from node of these arcs is the divergence node, denoted d. We denote the set of arcs originating at d as $A(d)$ and let $A(d, a1)$ and $A(d, a2)$ represent some partition of $A(d)$ such that the subset $A(d, a1)$ contains $a1$ and the subset $A(d, a2)$ contains $a2$. We branch creating two subproblems. For the first we require

$$\sum_{p \cap A(d,a1) \neq 0} y_p^k = 0,$$

and for the second we require

$$\sum_{p \cap A(d,a2) \neq 0} y_p^k = 0.$$

On the first branch we do not allow k to use any of the arcs in $A(d, a1)$, and similarly on the second branch we do not allow k to use any of the arcs in $A(d, a2)$. Note that these decisions do *not* require that k use *any* of the arcs in $A(d)$; that is, a path for k *not* containing node d is feasible for both of the subproblems.

The resulting division of the problem is valid because:

(1) if the LP solution is fractional, we can always find a divergence node d and a partition of $A(d)$ that will eliminate the fractional solution; and

(2) there are a finite number of branches because there are a finite number of arcs and commodities.

A major benefit of our branching rule is that it more evenly divides the problem because we branch on forbidding a *set* of arcs rather than a *single* arc. Forbidding a set of arcs may achieve faster convergence than forbidding a single arc because the exclusion of a single arc may not have much impact. Note that forbidding a single arc is a special case of our strategy where $|A(d, a1)| = 1$ or $|A(d, a2)| = 1$.

This new branching rule is a generalization of the Ryan and Foster (1981) rule for master problems with set partitioning structure. In their rule, they require two rows of the master problems to be covered by the same column on one branch and covered by different columns on the other. This rule is analogous to requiring/forbidding arcs for a commodity in ODIMCF. As we have pointed out, this approach does not allow us to maintain the shortest path structure of the pricing problem; however, our generalization in which we forbid subsets of arcs and never require the use of any arc does maintain the pricing problem structure.

Subproblem Solution

At each node of the branch-and-bound tree, a restricted multicommodity flow LP, called a subproblem, must be solved. Because the subproblem solution must satisfy the set of branching decisions made along its predecessor path in the tree, it is necessary to restrict the column-generation algorithm so that variables violating these rules are not generated in solving the pricing problem. The challenge is to ensure this without increasing the complexity of the pricing problem solution algorithm. To achieve this, observe that *every* branch forbids the assignment of flow of some commodity to one or more arcs. That means at *any* node in the tree, it is possible to satisfy the branching decisions by restricting flow of possibly several commodities, where the flow restriction for a single commodity is to forbid use of a (possibly large) set of arcs. By setting the commodity's cost on each forbidden arc to a very high value, the pricing problem can still be solved using a shortest path algorithm. As long as a feasible solution exists for that commodity, the shortest path generated will not violate the branching decisions. Then, all the paths generated for a subproblem will satisfy all the imposed restrictions.

Branch Selection. Given a fractional LP solution, we select the next branch as follows:

Step 1. Among the commodities whose flow is split, identify the commodity k with the greatest flow, denoted q^k.

Step 2. Identify the two paths p and p' with the greatest fractions y_p^k and $y_{p'}^k$ of the flow of commodity k. Without loss of generality, let path p be shorter than p'.

Step 3. Locate the divergence node d on path p for commodity k. Let arcs $a1$ and $a2$ be incident to d and in paths p and p', respectively.

Step 4. By dividing the set of arcs incident to node d, construct set $A(d, a1)$ containing arc $a1$ and set $A(d, a2)$ with arc $a2$. Let the size of the two sets be roughly equal.

Step 5. Create two new nodes, one where the arcs in $A(d, a1)$ are forbidden for commodity k and one where the arcs in $A(d, a2)$ are forbidden for commodity k.

Node Selection. A depth-first search of the tree is used throughout the algorithm. We choose to search the side of the tree where the shorter path p is still allowed (i.e., we choose the side where the arcs in $A(d, a2)$ are forbidden.) In many integer programming algorithms, the nodes are selected in the order of the best LP bound once a feasible solution has been found. In our experience, changing to a best-bound search gave an improvement in the number of nodes searched for the problems we were able to solve optimally, but it resulted in a degradation in solution quality for those that were stopped (by the time limit) before proving optimality. We believe there were two main reasons for this degradation. First, a best-bound search will generally be able to search many fewer nodes within a given amount of time than a depth-first search. This is largely because in a depth-first search, information from the LP solution of the parent node is readily available to speed up the solution of the LP relaxation at each node. Second, best-bound tends to search nodes that are relatively high in the tree, but for this application it was necessary to go deep in the tree to find integer solutions.

Branch-and-Price Termination. Our branch-and-price solution procedure is terminated when either a provably optimal integer solution is found or the run time exceeds one hour on a workstation class computer.

3.3. Computational Results: Branch-and-Price

We ran several computational trials on the set of 15 test problems, whose characteristics (commodities, nodes, arcs, saturated arcs) are given in Table 1. In the first 14 problems, all the arcs are capacitated, whereas in the last problem, 96 of the 130 arcs are capacitated. The column "saturated arcs" gives the number of arcs that were filled to capacity in the LP solution. The larger the number of saturated arcs, the more difficult we would expect the instance to be in general because more commodities will be likely to have their flow split. The first 14 problems are the same test problems used by Parker and Ryan (1994). They

Table 1. Problem characteristics.

Problem	Commodities	Nodes	Arcs	Saturated Arcs
1	35	14	16	8
2	68	24	24	7
3	70	29	61	31
4	58	18	29	16
5	47	19	25	6
6	93	27	37	27
7	93	23	29	18
8	41	28	31	8
9	87	24	42	24
10	41	19	19	8
11	23	14	16	5
12	81	26	36	10
13	52	29	31	8
14	46	20	23	8
15	585	50	130	28

were generated to be representative of bandwidth packing problems arising from a teleconferencing application. We converted these problems into ODIMCF problems by adding an artificial arc for each commodity with cost equal to the revenue associated with the commodity; all original arcs had cost zero. Thus the objective was to minimize the value of the calls that were not sent (i.e., those that had to use the artificial arcs). The final test problem is much larger. It came from a message routing problem arising in telecommunications. The commodities were sets of messages that share a common origin and destination. In this application, commodity splitting is permissible. We treated this problem as an ODIMCF to evaluate the performance of our approach on a larger problem.

First, to evaluate our branching rule, we compared a branch-and-bound algorithm using our branching rule to a standard branch-and-bound procedure. Specifically, the standard algorithm was the default branching strategy used by MINTO. In both algorithms, columns were generated to solve the LP at the root node only and branch-and-bound was applied to the resulting IP. Column generation at the root node only is often used as a heuristic approach, so we believed it would be worthwhile to determine whether our branching ideas would be valuable to these implementations. The results for branch-and-bound with our branching rule and the default branching rule are given in Table 2. The table displays the number of branch-and-bound nodes searched, the gap between the optimal integer solution and the solution obtained (entries reported with an * are gaps between the optimal LP solution and the solution obtained because the optimal IP solution is unknown for these problems), and the CPU time in seconds on an IBM RS6000/590 using MINTO 2.1 and CPLEX 3.0. Solution times of less than one hour indicate that optimal solutions are obtained for the resulting IP composed only of the columns generated in solving the root node LP. In general, compared to the customized branch-and-bound, the standard branch-and-bound required more time to generate poorer quality solutions. A case in point is problem 15, in which standard branch-and-bound was unable to find a feasible solution within the one hour allotted (after searching over a quarter of a million nodes!), but the customized rule found a solution within 5% of the LP bound.

Table 3 gives computational results for our branch-and-price algorithm. The number of columns generated, number of nodes searched, LP-IP gap, and CPU time on an IBM RS6000/590 are given. We are able to prove optimality for 11 of the 15 test problems within one hour of CPU time. It is interesting to note that for problems 6, 9, and 15 branch-and-bound was able to find better feasible solutions in the time allowed. This is partly because of the computational demands of the branch-and-price algorithm, which requires a great deal of computational effort at each node and is therefore generally able to search many fewer nodes than the branch-and-bound approach within the time limit. Of the bandwidth packing problems we observe that 3, 6, and 9 are difficult instances. In fact, these are the same instances that Parker and Ryan (1994) also found to be difficult. They found solutions within

Table 2. Branch-and-bound: standard vs. custom branching.

		Custom Branching			Standard Branching		
Problem	Columns	Nodes	Gap	Time	Nodes	Gap	Time
1	180	41	0.00%	0.69	47	0.00%	0.62
2	295	5	0.00%	0.58	9	0.00%	0.68
3	489	57404	10.70%*	2989.80	233571	12.34%*	3600.00
4	347	2712	0.75%	29.69	5412	0.75%	55.72
5	232	21	0.23%	0.63	93	0.23%	1.19
6	463	3462	1.45%*	55.74	256333	3.56%*	3600.00
7	439	7813	0.17%	101.73	69156	0.17%	854.84
8	195	7	0.00%	0.45	5	0.00%	0.42
9	464	20147	2.86%*	318.29	292445	3.16%*	3600.00
10	182	47	0.00%	0.67	312	0.00%	2.33
11	106	21	0.00%	0.36	18	0.00%	0.34
12	420	663	0.27%	10.80	3201	0.27%	46.39
13	253	125	0.00%	1.55	107	0.00%	1.46
14	225	82	0.22%	1.20	148	0.22%	1.62
15	2142	78577	4.76%*	3600.00	252648	—	3600.00

Table 3. Branch-and-price.

Problem	Columns	Nodes	Gap	Time
1	195	45	0.00%	1.61
2	295	5	0.00%	0.88
3	9088	7618	10.3%	3600.00
4	1936	1252	0.00%	115.00
5	261	19	0.00%	1.40
6	12353	13274	2.79%	3600.00
7	9188	12808	0.00%	2637.40
8	195	7	0.00%	0.68
9	19184	8512	6.0%	3600.00
10	185	51	0.00%	2.10
11	109	25	0.00%	0.61
12	5939	6405	0.00%	1145.70
13	287	133	0.00%	7.20
14	461	156	0.00%	8.76
15	5156	3194	26.94%	3600.00

11.5%, 7.1%, and 4.7% of optimality for these problems, respectively. However, a direct comparison with their results is not appropriate because their algorithm did not attempt to prove optimality. These are the three largest problems in terms of number of arcs in the network and are three of the five largest problems in terms of the number of commodities.

Symmetry Effects

We observed that when we disallowed one commodity from a subset of arcs, the values of the LP solutions for the two subproblems showed little or no change in objective function value. We also found that while the split commodities were changing, the same arcs were showing up repeatedly in the branching decisions. An explanation for this can be seen by examining two subpaths s_1 and s_2, both beginning with some node o and ending with some node d. It is possible that both s_1 and s_2 are contained in origin-destination paths for more than one, and maybe several, commodities. Denote this set of commodities as K'. Assume without loss of generality that s_1 has cost not greater than that of s_2. If in an LP solution, one or more arcs in s_1 are saturated, then it is possible that some of the commodities in K' are assigned to s_2. In this scenario, it is likely that one of the commodities, call it k^*, will be assigned to both s_1 and s_2. When the branching decision forces k^* off subpath s_1 (s_2), the result is a solution with the same total amount of flow assigned to subpaths s_1 and s_2, with the only difference being that some other commodity $k' \in (K'\backslash\{k^*\})$ has its flow split between the two subpaths. As long as arc cost is not differentiated by commodity, the costs of the solutions before and after branching will be the same.

This ineffectiveness of the branching strategy results from what is referred to as *problem symmetry*. Our remedy is to generate cuts to combat this symmetry before branching so that meaningful progress can be made in improving the LP bound as we go deeper in the tree.

4. BRANCH-AND-PRICE-AND-CUT

In the last decade, a great deal of attention has been given to the "branch-and-cut" approach to solving MIPs. Hoffman and Padberg (1985) and Nemhauser and Wolsey (1988) give general expositions of this methodology. The basic idea of branch-and-cut is simple. Classes of valid inequalities, preferably facets of the convex hull of feasible solutions, are left out of the LP relaxation because there are too many constraints to handle efficiently, and most of them will not be binding in an optimal solution. Then, if an optimal solution to an LP relaxation is infeasible to the IP, a subproblem called the separation problem is solved to try to identify violated inequalities in a class. If one or more violated inequalities are found, some are added to the LP to cut off the infeasible solution. Then the LP is reoptimized. Branching occurs when no violated inequalities are found to cut off an infeasible solution. Branch-and-cut allows separation and cutting to be applied throughout the branch-and-bound tree.

4.1. Lifted Cover Inequalities

Notice that the arc capacity constraints (2)

$$\sum_{k \in K} q^k x_{ij}^k \leq d_{ij}, \quad \forall\, ij \in A$$

in the node-arc formulation of ODIMCF are simply 0–1 knapsack inequalities. Thus, it is possible to use valid inequalities for the knapsack problem to strengthen the node-arc formulation of ODIMCF. One such class of valid inequalities are *lifted cover inequalities*. For a given arc ij, the set $C \subseteq K$ is called a *cover* if $\Sigma_{k \in C} q^k > d_{ij}$. The cover C is *minimal* if for each $l \in C$, $\Sigma_{k \in C} q^k - q^l \leq d_{ij}$. Covers give rise to a class of valid inequalities of the form

$$\sum_{k \in C} x_{ij}^k \leq |C| - 1,$$

called *cover inequalities*.

Now consider a LP solution to MCF. If the solution is fractional, at least one commodity k has flow assigned to more than one path. Consider one of these commodities, denoted k^*, and let path p^* represent the shortest *saturated* path to which k^* is assigned. (A path is saturated if one or more arcs contained in the path has flow at capacity.) Such a path will always exist because k^* is assigned to more than one path and each subproblem solution minimizes costs. Let a represent a saturated arc on path p^*. If k^* is the only split commodity assigned to a and C_a is the set of all commodities using a, then C_a is a cover and the corresponding cover inequality is violated by the current LP solution.

However, in general there may be more than one split commodity assigned to the saturated arcs in a LP solution to MCF so that no violated cover inequalities exist even though the solution is nonintegral. This follows from the fact that the cover inequalities do not in general define facets or even high dimensional faces of the convex hull of

feasible solutions to the knapsack problem. Cover inequalities need to be *lifted* in order to obtain facets of the knapsack polytope associated with an arc capacity constraint. A *lifted cover inequality* (LCI) is an inequality of the form

$$\sum_{k\in C} x_{ij}^k + \sum_{k\in \bar{C}} \alpha_k x_{ij}^k \leq |C| - 1,$$

where C is a minimal cover and $\bar{C} = K\backslash C$. The *lifting coefficients*, α_k, are nonnegative integers that are determined by solving a series of knapsack problems (one for each member of $\bar{C}$). In general, a single knapsack constraint may yield several minimal covers, and a single minimal cover may yield several LCIs, depending on the order in which the lifting coefficients are calculated. Because ODIMCF instances have a knapsack constraint for each arc, there are too many LCIs to include explicitly in the formulation.

Given a fractional solution to the knapsack problem, Gu et al. (1995b) showed that finding a most violated LCI is NP-hard. Therefore, we do not solve the separation problem for a fractional solution to ODIMCF exactly. Instead we use a fast heuristic proposed in Gu et al. (1995a). We attempt to identify one violated LCI for each saturated arc in the LP solution.

4.2. Translating Inequalities to the Path-Based Formulation

We can translate the lifted cover inequalities that we derive for the node-arc formulation to valid inequalities for the path-based formulation using the relationship between the arc-flow (x_{ij}^k) and path-flow (y_k^p) variables. Note that

$$x_{ij}^k = \sum_{p\in P(k)} y_p^k \delta_{ij}^p.$$

In terms of the path-flow variables, a lifted cover inequality can be written as

$$\sum_{k\in C} \sum_{p\in P(k)} y_p^k \delta_{ij}^p + \sum_{k\in \bar{C}} \alpha_k \sum_{p\in P(k)} y_p^k \delta_{ij}^p \leq |C| - 1.$$

LCIs are a well-known class of valid inequalities. However, to our knowledge this is the first time they have been applied in the context of the path-based formulation of ODIMCF. In general, as we pointed out earlier, very little work has been done that incorporates both cutting and column generation.

LCIs help to overcome problem symmetry by letting the LP select which of the commodities should remain and which should be removed from saturated arcs. This is more effective than using branching to specify (combinatorially) the set of commodities that will remain or be removed.

4.3. The Algorithm

At any node in the branch-and-bound tree, if no more columns price out and the LP solution is fractional, we attempt to identify a violated LCI for each saturated arc. If any inequalities are found, we add them to the MCF subproblem (thereby cutting off the current solution). Then, we reoptimize the LP. Observe that reoptimizing the LP will require solving the pricing problem.

Consider a single lifted cover inequality. It has a coefficient of α_{lm}^k (where $\alpha_{lm}^k = 1$ if $k \in C$) for every path containing a specific arc, say lm, for commodity k. This means that for each commodity k, the reduced cost computation for *every* path containing lm must be adjusted by the dual variable, call it $-\gamma_{lm}$, associated with the cover row multiplied by α_{lm}^k. The pricing problem to determine the minimum reduced cost path for each k can be solved efficiently by executing a shortest path algorithm on the network with the modified cost of arc lm for commodity k, denoted c'^k_{lm} as:

$$c'^k_{lm} = c^{lm}_k + \pi_{lm} + \alpha^k_{lm}\gamma_{lm} \qquad \forall\, k \in K.$$

This process is extended for several lifted cover inequalities by similarly modifying the cost of *one* arc for each inequality.

This process of finding violated cover inequalities and resolving the subproblem LP is repeated until no violated inequalities are found. At that point, branching occurs.

Table 4. Branch-and-price-and-cut.

Problem	Columns	Rows	Nodes	Gap	Time
1	180	7	131	0.00%	0.50
2	295	3	1	0.00%	0.88
3	7378	835	1564	3.63%	3600.00
4	418	88	17	0.00%	7.41
5	232	9	1	0.00%	0.63
6	4345	728	5174	2.16%	3600.00
7	493	154	28	0.00%	23.11
8	206	8	3	0.00%	0.84
9	8186	751	2084	1.16%	3600.00
10	190	23	7	0.00%	1.11
11	106	7	1	0.00%	0.31
12	430	54	4	0.00%	4.25
13	253	7	1	0.00%	0.74
14	248	37	4	0.00%	2.36
15	5836	69	2396	—	3600.00

5. COMPUTATIONAL RESULTS: BRANCH-AND-PRICE-AND-CUT

Table 4 gives computational results for our branch-and-price-and-cut algorithm. Again the number of columns generated, number of nodes searched, LP-IP gap, and CPU time on an IBM RS6000/590 are given. We observe the following:

1. We are able to prove optimality for 4 of the 15 problems at the root node vs. 0 of 15 for branch-and-price.

2. Compared to branch-and-price, node counts generally decrease dramatically. For the 11 problems solved to optimality, the average number of nodes explored decreases from 1,900 to 18.

3. Except for problem 15, we close the optimality gap significantly for the problems we were unable to solve optimally. Problem 15 seems to be considerably more difficult than the others. It has many more commodities and the

arc capacities are relatively large compared to the commodity flow values. For problems 1 through 14, most arcs can carry fewer than 5 commodities, whereas for problem 15, an arc can carry more than 20 commodities. This prevents the lifted cover inequalities from being effective.

6. SUMMARY

In this paper we presented new algorithms for origin-destination integer multicommodity flow problems. We presented a new branching rule, developed a branch-and-price algorithm, and enhanced the branch-and-price algorithm by adding cuts. We showed that if columns are generated only at the root node, our branching rule is more effective at finding the optimal solution over the resulting set of columns than a standard rule. This result is important for practitioners who often use column generation at the root node only as a heuristic approach. We then presented results for our branch-and-price algorithm. The approach was able to solve most of the problems to provable optimality; however, it required searching a large number of nodes in many cases. We attributed this computational difficulty to problem symmetry. To combat this symmetry, we added cuts to the problem and devised a branch-and-price-and-cut algorithm that allows column generation and cutting to be applied throughout the tree. We found that the addition of cuts dramatically decreased the number of nodes searched in the instances for which the optimal solution was found and decreased the optimality gap for most of the other instances. The only exception was the final problem. We speculate that the cuts were ineffective in this case because the commodity volumes were small relative to the arc capacities thus weakening the cutting planes.

ACKNOWLEDGMENT

This research has been supported by the following grants and contracts: NSF DDM-9058074, NSF DMI-95-2502.

REFERENCES

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ.

Appelgren, L. H. 1969. A column generation algorithm for a ship scheduling problem. *Transp. Sci.* **3** 53–68.

Barnhart, C., C. A. Hane, E. L. Johnson, G. Sigismondi. 1995a. A column generation and partitioning approach for multi-commodity flow problems. *Telecommunication Systems* **3** 239–258.

——, E. L. Johnson, G. L. Nemhauser, G. L. Savelsbergh, P. H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*. **46**(3) 316–329.

CPLEX Optimization, Inc. 1990. *Using the* CPLEX™ *Linear Optimizer*, Indine Village, NV.

Dantzig, G. B., R. M. Van Slyke. 1967. Generalized upper bounding techniques. *J. Comput. System Sci.* **1** 213–226.

——, P. Wolfe. 1960. Decomposition principle for linear programs. *Oper. Res.* **8** 108–111.

Desrosiers, J., Y. Dumas, M. M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. In *Handbooks in Operations Research and Management Science*. M. E. Ball, T. L. Magnanti, C. Monma, G. L. Nemhauser (eds.). Elsevier, Amsterdam.

Gu, W., G. L. Nemhauser, M. W. P. Savelsbergh. 1995. Lifted cover inequalities for 0–1 integer programs I: computation. Report COC-95-02, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

——, ——, ——. 1995. Lifted cover inequalities for 0–1 integer programs II: complexity. Report COC-95-03, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

Hoffman, K., M. Padberg. 1985. LP-based combinatorial problem solving. *Ann. Oper. Res.* **4** 145–194.

IBM Corporation. 1992. *Optimization Subroutine Library, Guide and Reference*, IBM Systems J. **31** (1) SC23-0519.

Jones, K. L., I. J. Lustig, J. M. Farvolden, W. B. Powell. 1993. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming* **62** 95–117.

Mehrotra, A. 1992. Constrained Graph Partitioning: Decomposition, Polyhedral Structure and Algorithms. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA.

Moore, E. 1957. The shortest path through a maze. *Proc. International Symposium on the Theory of Switching.* Harvard University Press, Cambridge, MA.

Nemhauser, G. L., S. Park. 1991. A polyhedral approach to edge coloring. *Oper. Res. Lett.* **10** 315–322.

——, M. W. P. Savelsbergh, G. C. Sigismondi. 1994. MINTO, a Mixed INTeger Optimizer. *Oper. Res. Lett*. **15**.

——, L. A. Wolsey. 1988. *Integer and Combinatorial Optimization*. Wiley, Chichester, UK.

Pape, U. 1974. Implementation and efficiency of Moore algorithms for the shortest route problem. *Mathematical Programming* **7** 212–222.

Parker, M., J. A. Ryan. 1994. A column generation algorithm for bandwidth packing. *Telecommunications Systems* **2** 185–195.

Rosen, J. B. 1964. Primal partition programming for block diagonal matrices. *Numerische Mathematik* **6** 250–260.

Ryan, D. M., B. A. Foster. 1981. An integer programming approach to scheduling. In *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. A. Wren (ed.). North-Holland, Amsterdam.

Vance, P. H., C. Barnhart, E. L. Johnson, G. L. Nemhauser. 1997. Airline crew scheduling: A new formulation and decomposition algorithm. *Oper. Res.* **45**(2) 188–200.

——, ——, ——, ——, D. Mahidara, A. Krishna, R. Rebello. 1994. *Exceptions in Crew Planning*. ORSA/TIMS Detroit, MI.

Vanderbeck, F., L. A. Wolsey. 1996. An exact algorithm for IP column generation. *Oper. Res. Letters* **19** 151–159.