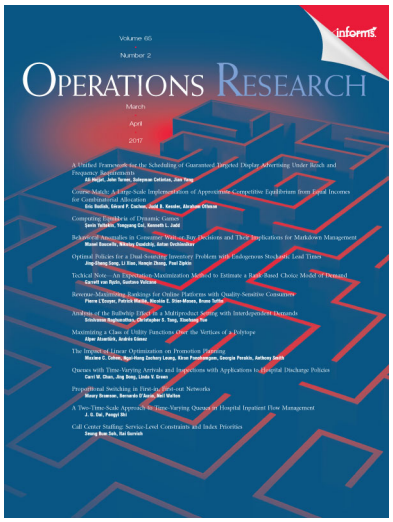

Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Branch-and-Price: Column Generation for Solving Huge Integer Programs

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, Pamela H. Vance,

To cite this article:
Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, Pamela H. Vance, (1998) Branch-and-Price: Column Generation for Solving Huge Integer Programs. Operations Research 46(3):316-329. https://doi.org/10.1287/opre.46.3.316

BRANCH-AND-PRICE: COLUMN GENERATION FOR SOLVING HUGE INTEGER PROGRAMS

CYNTHIA BARNHART*, ELLIS L. JOHNSON, GEORGE L. NEMHAUSER, MARTIN W. P. SAVELSBERGH, and PAMELA H. VANCE**

Georgia Institute of Technology, Atlanta, Georgia

(Received February 1994; revisions received May 1995, January 1996; accepted March 1996)

We discuss formulations of integer programs with a huge number of variables and their solution by column generation methods, i.e., implicit pricing of nonbasic variables to generate new columns or to prove LP optimality at a node of the branch-and-bound tree. We present classes of models for which this approach decomposes the problem, provides tighter LP relaxations, and eliminates symmetry. We then discuss computational issues and implementation of column generation, branch-and-bound algorithms, including special branching rules and efficient ways to solve the LP relaxation. We also discuss the relationship with Lagrangian duality.

The successful solution of large-scale mixed integer programming (MIP) problems requires formulations whose linear programming (LP) relaxations give a good approximation to the convex hull of feasible solutions. 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 anyway. Then, if an optimal solution to an LP relaxation is infeasible, 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, which is a generalization of branch-and-bound with LP relaxations, allows separation and cutting to be applied throughout the branch-and-bound tree.

The philosophy of branch-and-price is similar to that of branch-and-cut except that the procedure focuses on column generation rather than row generation. In fact, pricing and cutting are complementary procedures for tightening an LP relaxation. We will describe algorithms that include both, but we emphasize column generation.

In branch-and-price, sets of columns are left out of the LP relaxation 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 anyway. 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. Branching occurs when no columns price out to enter the basis and the LP solution does not satisfy the integrality conditions. Branch-and-price, which also is a generalization of branch-and-bound with LP relaxations, allows column generation to be applied throughout the branch-and-bound tree.

We have several reasons for considering formulations with a huge number of variables.

- A compact formulation of a MIP may have a weak LP relaxation. Frequently the relaxation can be tightened by a reformulation that involves a huge number of variables.
- A compact formulation of a MIP may have a symmetric structure that causes branch-and-bound to perform poorly because the problem barely changes after branching. A reformulation with a huge number of variables can eliminate this symmetry.
- Column generation provides a decomposition of the problem into master and subproblems. This decomposition may have a natural interpretation in the contextual setting allowing for the incorporation of additional important constraints.
- A formulation with a huge number of variables may be the only choice.

At first glance, it may seem that branch-and-price involves nothing more than combining well-known ideas for solving linear programs by column generation with branch-and-bound. However, as Appelgren (1969) observed 25 years ago, it is not that straightforward. There are fundamental difficulties in applying column generation techniques for linear programming in integer programming solution methods (Johnson 1989). These include:

*Currently at Massachusetts Institute of Technology, Cambridge, Massachusetts.
**Currently at Auburn University, Auburn, Alabama.
Subject classifications: Integer programming. Branch-and-bound. Decomposition algorithms.
Area of review: OPTIMIZATION.

0030-364X/98/4603-0316 $05.00

- Conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem.
- Solving these LPs to optimality may not be efficient, in which case different rules will apply for managing the branch-and-price tree.

Recently, several specialized branch-and-price algorithms have appeared in the literature. Our paper attempts to unify this literature by presenting a general methodology for branch-and-price. It is by no means an inclusive survey, but does develop some general ideas that have appeared only in very special contexts. Routing and scheduling has been a particularly fruitful application area of branch-and-price; see Desrosiers et al. (1995) for a survey of these results.

Section 1 presents two examples that illustrate the general concepts of a branch-and-price algorithm and, at the same time, point out some of the difficulties that may be encountered when applying it. Section 2 discusses the types of MIPs for which branch-and-price can be advantageous. Section 3 analyzes the similarities and differences between branch-and-price and Lagrangian duality. Section 4 presents the special types of branching that are required for branch-and-price to be effective. Section 5 considers the computational issues that need to be addressed when implementing a branch-and-price algorithm.

1. TWO ILLUSTRATIVE EXAMPLES

To motivate the general ideas of branch-and-price, we present two important practical examples—generalized assignment and crew scheduling—before presenting the general formulations and methodology.

1.1. Generalized Assignment

In the generalized assignment problem (GAP) the objective is to find a maximum profit assignment of m tasks to n machines such that each task is assigned to precisely one machine subject to capacity restrictions on the machines. For reasons that will become apparent later, we will consider separately the two cases of unrelated machines and identical machines.

Case 1. Unrelated Machines. The standard integer programming formulation of GAP is

$$\max \sum_{1\leqslant i\leqslant m} \sum_{1\leqslant j\leqslant n} p_{ij} z_{ij}$$

$$\sum_{1\leqslant j\leqslant n} z_{ij} = 1, \quad i = 1, \ldots, m,$$

$$\sum_{1\leqslant i\leqslant m} w_{ij} z_{ij} \leqslant d_j, \quad j = 1, \ldots, n,$$

$$z_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, j = 1, \ldots, n, \tag{1}$$

where p_{ij} is the profit associated with assigning task i to machine j, w_{ij} is the claim on the capacity of machine j by task i, d_j is the capacity of machine j, and z_{ij} is a 0-1 variable indicating whether task i is assigned to machine j.

Applying Dantzig-Wolfe decomposition to GAP with the assignment constraints defining the master problem and the machine capacity constraints defining the subproblems yields the master problem

$$\max \sum_{1\leqslant j\leqslant n} \sum_{1\leqslant k\leqslant K_j} \left(\sum_{1\leqslant i\leqslant m} p_{ij} y_{ik}^j \right) \lambda_k^j$$

$$\sum_{1\leqslant j\leqslant n} \sum_{1\leqslant k\leqslant K_j} y_{ik}^j \lambda_k^j = 1, \quad i = 1, \ldots, m,$$

$$\sum_{1\leqslant k\leqslant K_j} \lambda_k^j = 1, \quad j = 1, \ldots, n,$$

$$\lambda_k^j \in \{0, 1\}, \quad j = 1, \ldots, n, \; k = 1, \ldots, K_j, \tag{2}$$

where the first m entries of a column, given by $y_k^j = (y_{1k}^j, y_{2k}^j, \ldots, y_{mk}^j)$, form a feasible solution to the knapsack constraint

$$\sum_{1\leqslant i\leqslant m} w_{ij} y_i^j \leqslant d_j,$$

$$y_i^j \in \{0, 1\}, \quad i = 1, \ldots, m,$$

and where K_j denotes the number of feasible solutions to the above knapsack constraint. In other words, a column represents a feasible assignment of tasks to a machine.

The reason for reformulating the GAP by applying Dantzig-Wolfe decomposition is that the linear programming relaxation of the master problem (2) is tighter than the linear programming relaxation of the standard formulation since fractional solutions that are not convex combinations of 0-1 solutions to the knapsack constraints are not feasible to (2). Note that our motivation for applying decomposition is not to speed up the solution of the LP relaxation—in fact the LP relaxation of the master (2) may be harder to solve than that of (1); rather, it is to improve the LP bound.

To solve the LP relaxation of (2), pricing or column generation is done by solving n knapsack problems. Several computational or implementation issues arise in solving these LPs including whether the knapsacks should be solved exactly or approximately, whether nonbasic columns should be kept or regenerated, whether the master LP should be solved to optimality, etc. These issues will be addressed in more detail in Section 5.

The LP relaxation of the master problem solved by column generation may not have an integral optimal solution and applying a standard branch-and-bound procedure to the master problem over the existing columns is unlikely to find an optimal, or even good, or even feasible, solution to the original problem. Therefore it may be necessary to generate additional columns in order to solve the linear programming relaxations of the master problem at nonroot nodes of the search tree.

Standard branching on the λ-variables creates a problem along a branch where a variable has been set to zero. Recall that y_k^j represents a particular solution to the jth knapsack problem. Thus $\lambda_k^j = 0$ means that this solution is excluded. However, it is possible (and quite likely) that the next time the jth knapsack problem is solved the optimal knapsack solution is precisely the one represented by y_k^j. In

that case, it would be necessary to find the second best solution to the knapsack problem. At depth l in the branch-and-bound tree we may need to find the lth best solution. Fortunately, there is a simple remedy to this difficulty. Instead of branching on the λ's in the master problem, we use a branching rule that corresponds to branching on the original variables z_{ij}. When $z_{ij} = 1$, all existing columns in the master that don't assign task i to machine j are deleted and task i is permanently assigned to machine j, i.e., variable y_i^j is fixed to 1 in the jth knapsack. When $z_{ij} = 0$, all existing columns in the master that assign job i to machine j are deleted and task i cannot be assigned to machine j, i.e., variable y_i^j is removed from the jth knapsack. Note that each of the knapsack problems contains one fewer variable after the branching has been done. Observe that since we know how to fix a single original variable z_{ij} to 0, we can also perform GUB branching on the assignment constraints.

Case 2. Identical Machines. This is a special case of the problem with unrelated machines and therefore the methodology described above applies. However, we need only one subproblem since all of the machines are identical, which implies that the λ_k^j can be aggregated by defining $\lambda_k = \Sigma_j \lambda_k^j$ and that the convexity constraints can be combined into a single constraint $\Sigma_{1 \leq k \leq K} \lambda_k = n$ where λ_k is restricted to be integer. In some cases the aggregated constraint will become redundant and can be deleted altogether. An example of this is when the objective is to minimize $\Sigma \lambda_k$, i.e., the number of machines needed to process all the tasks. Note that this special case of GAP is equivalent to a 0-1 cutting stock problem.

A much more important issue here concerns symmetry, which causes branching on the original variables to perform very poorly. With identical machines, there are an exponential number of solutions that differ only by the names of the machines, i.e., by swapping the assignments of two machines we get two solutions that are the same but have different values for the variables. This statement is true for fractional as well as 0-1 solutions. The implication is that when a fractional solution is excluded at some node of the tree, it pops up again with different variable values somewhere else in the tree. In addition, the large number of alternate optima dispersed throughout the tree renders pruning by bounds nearly useless.

The remedy here is a different branching scheme, proposed by Ryan and Foster (1981) for crew scheduling, that works directly on the master problem but focuses on pairs of tasks. In particular, we consider rows of the master with respect to tasks r and s. Branching is done by dividing the solution space into one set in which r and s appear together, in which case they can be combined into one task when solving the knapsack, and into another set in which they must appear separately, in which case a simple two-variable GUB constraint is added to the knapsack. Note that the structure of the subproblems is no longer the same on the different branches.

1.2. Crew Scheduling

In a crew scheduling or pairing problem, sequences of flights, called pairings, are assigned to crews so that each flight segment for a specific fleet of airplanes is assigned to exactly one crew. The first segment in a pairing must depart from the crew's base, each subsequent segment departs from the station where the previous one arrived, and the last segment must return to the base. A sequence can represent several days of flying, typically three–five days for a domestic problem of a major U.S. carrier.

Pairings are subject to a number of constraints resulting from safety regulations and contract terms. These constraints dictate restrictions such as the maximum number of hours a pilot can fly in a day, the maximum number of days before returning to the base and minimum overnight rest times. In addition, the cost of a pairing is a messy function of several attributes of the pairing.

For these reasons, it is not desirable to formulate a crew scheduling problem with variables z_{ij} where $z_{ij} = 1$ if crew i is assigned to segment j since the constraints on z_{ij} and the cost are highly nonlinear and difficult to express. The alternative approach is to implicitly or explicitly enumerate feasible pairings and then to formulate a set partitioning problem in which each column or variable corresponds to a pairing and the objective is to partition all of the segments into a set of minimum cost pairings.

Although enumerating feasible pairings is complex because of all of the rules that must be observed, it can be accomplished by first enumerating all feasible possibilities for one day of flying and then combining the one-day schedules to form pairings. The major drawback is the total number of pairings, which grows exponentially with the number of flights. For example, Vance (1993) found more than five million pairings in a daily problem with 253 flights. Problems with 1,000 flights are likely to have billions of pairings.

The last generation's methodology for a 1,000 flight problem, which is the typical size for a U.S. domestic carrier, would have enumerated in advance about 100,000 low cost pairings, a very small fraction of the total. Then, with this fixed set of columns, attempted to solve, or get a good feasible solution to, the set partitioning problem (0-1 integer program) defined by these 100,000 pairings, which itself is a very difficult task. Since only a tiny fraction of all of the pairings are available to the integer program or its linear programming relaxation, this methodology may not produce a solution that is close to being optimal to the problem in which all the pairings are considered. There is now considerable empirical evidence to support this fact.

Branch-and-price can implicitly consider all of the pairings. It is possible to represent pairings as suitably constrained paths in a network, and then to evaluate their costs, i.e., price out nonbasic columns in a simplex algorithm, using a multilabel shortest path or multistate dynamic programming algorithm, see Desrochers and Soumis (1989), Barnhart et al. (1995), and Vance (1993).

It is not sufficient, however, to use only those columns that have been generated in the solution of the initial LP. It is essential to generate columns during the solution of LPs throughout the tree. For example, in a small 400 flight crew pairing instance, 9400 pairings were generated to solve the initial LP relaxation. The LP bound was 1600. The only IP solution that we could find using this small set of candidate pairings had cost 13,000. (We quit after two days of CPU time using OSL on an RS 6000 model 550.) By generating columns in the tree, we got an IP solution with cost 800! This cost is less than the cost of the LP relaxation at the root node because of approximations in the pricing algorithm that prevented fully optimizing the LP relaxation. Another example of the importance of generating columns after the initial LP has been solved can be found in the work of Marsten (1994), who sets a target value for the IP and generates additional columns whenever the LP bound at a node exceeds that target value.

It is very important to understand that it is not necessary to solve an LP to optimality when processing a node. The primary reason for solving the LP to optimality is the possibility of pruning the node by bounds. By not solving the LP to optimality, this option may be precluded and branching may be required. However, pruning may still be possible without solving the LP to optimality if we can compute a bound on the value of the LP that is strong enough to prune the node. See Section 5 for further discussion of this idea.

2. SUITABLE MODELS FOR COLUMN GENERATION

2.1. General Models

The general problem P we consider is of the form

$$\begin{aligned} &\max\ c(x)\\ &Ax \leqslant b,\\ &x \in S,\\ &x \quad \text{integer}. \end{aligned} \tag{3}$$

The function $c(x)$ is not required to be linear although, for computational reasons, we need that the relaxed problem with $Ax \leqslant b$ omitted is relatively easy to solve.

The fundamental idea of column generation is that the set

$$S^* = \{x \in S : x \text{ integer}\},$$

is represented by a finite set of vectors. Specifically, if S is bounded then S^* is just a finite set of points, say $S^* = \{y_1, \ldots, y_p\}$. Moreover if x is binary, then S^* coincides with the extreme points of its convex hull, denoted by $conv(S^*)$. Since representing a bounded polyhedron by its extreme points is the basic construct of Dantzig-Wolfe decomposition and generalized linear programming (Dantzig and Wolfe 1960), our column generation approach to integer programming is closely related to Dantzig-Wolfe decomposition and the earlier work on path flows in networks of Ford and Fulkerson (1958).

When S is unbounded, classical results of Minkowski and Weyl (see Nemhauser and Wolsey 1988) state that $conv(S^*)$ is a polyhedron and is represented by a convex combination of a finite set of points and a linear combination of a finite set of rays. Thus column generation for integer programming is still possible when S is unbounded. However, primarily for simplicity of exposition, we will assume throughout this paper that S is bounded. Vanderbeck (1994, 1995) gives more details on the unbounded case.

Given $S^* = \{y_1, \ldots, y_p\}$, any point $y \in S^*$ can be represented as

$$y = \sum_{1 \leqslant k \leqslant p} y_k \lambda_k,$$

subject to the convexity constraint

$$\begin{aligned} &\sum_{1 \leqslant k \leqslant p} \lambda_k = 1,\\ &\lambda_k \in \{0, 1\}, \quad k = 1, \ldots, p. \end{aligned}$$

Let $c_k = c(y_k)$ and let $a_k = Ay_k$. We obtain the column generation form of P given by

$$\begin{aligned} &\max \sum_{1 \leqslant k \leqslant p} c_k \lambda_k\\ &\sum_{1 \leqslant k \leqslant p} a_k \lambda_k \leqslant b,\\ &\sum_{1 \leqslant k \leqslant p} \lambda_k = 1,\\ &\lambda_k \in \{0, 1\}, \quad k = 1, \ldots, p. \end{aligned} \tag{4}$$

Note that the column generation form of P is an integer linear programming problem, whereas the initial formulation can have a nonlinear objective function. Linearization in this manner is possible because in the equation

$$y = \sum_{1 \leqslant k \leqslant p} y_k \lambda_k,$$

we obtain $y = y_k$ for some k since $\Sigma_{1 \leqslant k \leqslant p} \lambda_k = 1$ and $\lambda_k \in \{0, 1\}$. This cannot be done when S is unbounded so that in that case the initial formulation must have a linear objective function or other devices must be used to achieve linearity; see Vanderbeck and Wolsey (1996).

If S can be decomposed, i.e., $S = \cup_{1 \leqslant j \leqslant n} S_j$, we can represent each set

$$S_j^* = \{x_j \in S_j : x_j \text{ integer}\},$$

as

$$S_j^* = \{y_1^j, \ldots, y_{p_j}^j\}.$$

Now let $c(y_k^j) = c_k^j$ and $Ay_k^j = a_k^j$. This yields a column generation form of P with separate convexity constraints for each S_j given by

$$\begin{aligned} &\max \sum_{1 \leqslant j \leqslant n} \sum_{1 \leqslant k \leqslant p_j} c_k^j \lambda_k^j\\ &\sum_{1 \leqslant j \leqslant n} \sum_{1 \leqslant k \leqslant p_j} a_k^j \lambda_k^j \leqslant b,\\ &\sum_{1 \leqslant k \leqslant p_j} \lambda_k^j = 1, \quad j = 1, \ldots, n,\\ &\lambda_k^j \in \{0, 1\}, \quad j = 1, \ldots, n,\ k = 1, \ldots, p_j. \end{aligned} \tag{5}$$

If the subsets in the decomposition are identical, i.e., $S_j = \bar{S} = \{y_1, \ldots, y_p\}$ for $j = 1, \ldots, n$, then they can be

represented by one subset $\bar{S}$ with $\lambda_k = \Sigma_j \lambda_k^j$ and the convexity constraints replaced by an aggregated convexity constraint

$$\sum_{1 \leqslant k \leqslant p} \lambda_k = n,$$

where $\lambda_k \geqslant 0$ and integer. This results in the column generation form

$$\max \sum_{1 \leqslant k \leqslant p} c_k \lambda_k$$

$$\sum_{1 \leqslant k \leqslant p} a_k \lambda_k \leqslant b,$$

$$\sum_{1 \leqslant k \leqslant p} \lambda_k = n,$$

$$\lambda_k \geqslant 0 \quad \text{and} \quad \text{integer}, k = 1, \ldots, p. \tag{6}$$

In some formulations, the convexity constraint is written as an inequality

$$\sum_{1 \leqslant k \leqslant p} \lambda_k \leqslant 1.$$

Generally this is the case when $y = 0 \in S^*$ and $c(0) = 0$, so $y = 0$ is just left out of the formulation. In the case of identical subsets the aggregated convexity constraint may be omitted altogether when n is not fixed as part of the input.

The essential difference between P and its column generation form is that S^* has been replaced by a finite set of points. We see that any fractional solution to the linear programming relaxation of P is a feasible solution to the linear programming relaxation of its column generation form if and only if it can be represented by a convex combination of extreme points of $conv(S^*)$. In particular, Geoffrion (1974) has shown that if the polyhedron $conv(S)$ does not have all integral extreme points, then the linear programming relaxation of the column generation form of P will be tighter than that of P for some objective functions.

However, since the column generation form frequently contains a huge number of columns, it may be necessary to work with restricted versions that contain only a subset of its columns, and to generate additional columns only as they are needed. The column generation form is called the master problem (MP), and when it does not contain all of its columns it is called a restricted master problem (RMP). Column generation is done by solving pricing problems of the form

$$\max \{dx : x \in S^*\} \text{ or } \max \{dx : x \in \text{conv}(S^*)\},$$

where d is determined from optimal dual variables to an LP relaxation of an RMP.

2.2. Partitioning Models

Many combinatorial optimization problems can be formulated as set partitioning problems; see Balas and Padberg (1976). Since most of the branch-and-price algorithms we are aware of have been developed for set partitioning based formulations, they will be emphasized. In the general set partitioning problem, we have a ground set of elements and rules for generating feasible subsets and their profits, and we wish to find the maximum profit partitioning of the ground set into feasible subsets. Let $z_{ij} = 1$ if element i is in subset j, and 0 otherwise, and let z_j denote the characteristic vector of subset j, i.e., a vector with entries z_{ij} for each element i. The general partitioning problem is of the form

$$\max \sum_{1 \leqslant j \leqslant n} c(z_j)$$

$$\sum_{1 \leqslant j \leqslant n} z_{ij} = 1, \quad i = 1, \ldots, m,$$

$$z_j \in S,$$

$$z \quad \text{binary}, \tag{7}$$

where m is the number of elements in the ground set, n is the number of subsets, and S is the set of feasible subsets.

2.2.1. Different Restrictions on Subsets. First we assume that the feasible subsets have different requirements given by

$$S_j = \{z_j : D_j z_j \leqslant d_j, z_j \text{ binary}\} \quad j = 1, \ldots, n. \tag{8}$$

Then problem P is

$$\max \sum_{1 \leqslant j \leqslant n} c_j(z_j)$$

$$\sum_{1 \leqslant j \leqslant n} z_{ij} = 1, \quad i = 1, \ldots, m,$$

$$D_j z_j \leqslant d_j, \quad j = 1, \ldots, n,$$

$$z \quad \text{binary}, \tag{9}$$

and its column generation form is

$$\max \sum_{1 \leqslant j \leqslant n} \sum_{1 \leqslant k \leqslant p_j} (c_j^k y_k^j) \lambda_k^j$$

$$\sum_{1 \leqslant j \leqslant n} \sum_{1 \leqslant k \leqslant p_j} y_{ik}^j \lambda_k^j = 1, \quad i = 1, \ldots, m,$$

$$\sum_{1 \leqslant k \leqslant p_j} \lambda_k^j \leqslant 1, \quad j = 1, \ldots, n,$$

$$\lambda_k^j \in \{0, 1\}, \quad j = 1, \ldots, n, \; k = 1, \ldots, p_j, \tag{10}$$

where $c_j^k = c_j(y_k^j)$ and the $\{y_k^j\}$, $1 \leqslant k \leqslant p_j$ are the points of S_j^* with elements y_{ik}^j for $i = 1, \ldots, m$. We have chosen to write the convexity constraints as inequalities since, in many of these applications, we may not assign any elements to a given subset. This structure occurs, for example, in the generalized assignment problem with unrelated machines.

2.2.2. Identical Restrictions on Subsets. Now we assume that the feasible subsets have identical requirements. Then (8) is replaced by the single set of inequalities

$$\bar{S} = \{z_j : Dz_j \leqslant d, z_j \text{ binary}\}. \tag{11}$$

Problem P is

$$\max \sum_{1 \leq j \leq n} c(z_j)$$

$$\sum_{1 \leq j \leq n} z_{ij} = 1, \quad i = 1, \ldots, m,$$

$$Dz_j \leq d, \quad j = 1, \ldots, n,$$

$$z \quad \text{binary}, \tag{12}$$

and its column generation form is

$$\max \sum_{1 \leq k \leq p} c_k \lambda_k$$

$$\sum_{1 \leq k \leq p} y_{ik} \lambda_k = 1, \quad i = 1, \ldots, m,$$

$$\lambda_k \in \{0, 1\}, \quad k = 1, \ldots, p, \tag{13}$$

where $c_k = c(y_k)$. Here we have chosen to omit the aggregated convexity constraint because it is common in these applications for n not to be fixed. This structure occurs, for example, in the generalized assignment problem with identical machines and the 0-1 cutting stock problem.

A major advantage of MP for these problems with identical subset rules is that it eliminates some of the inherent symmetry of P that causes branch-and-bound to perform very poorly. By this we mean that any solution to P or its LP relaxation has an exponential number of representations as a function of the number of subsets. Therefore branching on a variable z_{ij} to remove a fractional solution will likely produce the same fractional solution with z_{ik} equal to the old value of z_{ij} and vice-versa, unless z_{ij} is fractional for all j. Formulation MP eliminates this symmetry and is therefore much more amenable to branching rules in which meaningful progress in improving the LP bound can be made as we go deeper in the tree.

2.3. Covering Models

Although the discussion above has focused on set partitioning type master problems, in many applications the problem structure allows the master problem to be formulated either as a set partitioning problem or as a set covering problem. Consider, for example, vehicle routing and scheduling problems, where several vehicles are located at one or more depots and must serve geographically dispersed customers. Each vehicle has a given capacity and is available in a specified time interval. Each customer has a given demand and must be served within a specified time window. The objective is to minimize the total cost of travel. A solution to a vehicle routing and scheduling problem partitions the set of customers into a set of routes for vehicles. This naturally leads to a set partitioning formulation in which the columns correspond to feasible routes and the rows correspond to the requirement that each customer is visited *precisely* once. Alternatively, the problem can be formulated as a set covering problem in which the columns correspond to feasible routes and the rows correspond to the requirement that each customer is visited *at least* once. Since deleting a customer from a route, i.e., not visiting that customer, results in another shorter less costly feasible route, an optimal set covering solution will be an optimal set partitioning.

In general, if any subcolumn of a feasible column defines another feasible column with lower cost, an optimal solution to the set covering problem will be an optimal set partitioning and we can work with either one of the formulations. When there is a choice, the set covering formulation is preferred since

- its linear programming relaxation is numerically far more stable and thus easier to solve;
- it is trivial to construct a feasible integer solution from a solution to the linear programming relaxation.

2.4. Nondecomposable Models

Column generation is also used for very large nondecomposable models. Here it may be the only practical approach. If a model contains so many variables that the storage requirements are enormous, then it is usually impossible to solve the LP relaxation directly and a column generation approach may be the only alternative.

This approach has been used to solve large instances of the traveling salesman problem; for example, see Junger et al. (1995). To handle the enormous number of variables (for a thousand city instance there are a million variables) only variables associated with a small subset of the edges, the k shortest edges associated with each vertex, are maintained. When the LP is solved for this reduced edge set, it is necessary to price out all the edges not in this set to verify that the true optimum has been found. If edges with favorable reduced costs are identified, they are added to the reduced edge set and the process is repeated.

A related approach, called SPRINT, has proven very successful for huge set partitioning problems. The SPRINT approach solves subproblems consisting of a subset of the columns, e.g., 10,000 out of 5 million. A new subproblem is formed by retaining the columns in the optimal basis of the old subproblem and collecting a set of good columns based on the reduced costs. This is repeated until all columns have been considered and then finally, and only once, the full problem is optimized. Using the SPRINT approach a linear programming relaxation of a set partitioning problem with nearly six million variables was solved in less than an hour on an IBM 3090E vector facility, see Anbil et al. (1992), and even more quickly using a combined interior point/simplex sprint approach by Bixby et al. (1992).

Note that in both cases standard simplex pricing is used to price out the variables that are initially left out of the formulation. This can be done because the total number of variables does not grow exponentially with the size of the input.

3. LAGRANGIAN DUALITY

Lagrangian duality is an alternative to column generation for getting improved relaxations of MIPs. In fact, a standard dualization gives the identical bound to the one obtained from column generation; see Brooks and Geoffrion (1966) and Geoffrion (1974).

The idea is to consider the Lagrangian relaxation of (3) given by

$$g(\pi) = \max [cx + \pi(b - Ax)],$$
$$x \in S,$$
$$x \text{ integer}, \tag{14}$$

where $\pi \geq 0$, and the Lagrangian dual given by

$$g = \min \{g(\pi) : \pi \geq 0\}. \tag{15}$$

Using linear programming duality, Geoffrion (1974) proved that the maximum value of the linear programming relaxation of (4) equals g. Moreover, the Lagrangian relaxation (14) is exactly of the form of the column generation subproblem that is used in the solution of the linear programming relaxation of (4). Also the Lagrange multiplier vector π corresponds to the dual variables for the constraints

$$\sum_k (Ay_k)\lambda_k \leq b,$$

in the linear programming relaxation of the restricted master problem.

Thus the choice between Lagrangian duality and column generation rests largely on the issue of which approach produces an optimal π more efficiently. Subgradient optimization is the traditional approach to solving the piecewise linear Lagrangian dual; see Held et al. (1974). The subgradient algorithm is simple and easy to code for special purpose applications. However, it can be slow and have trouble with convergence for large problems. Other recently developed methods for nondifferentiable optimization, such as bundle methods (see Lemarechal 1989), appear to be very promising but have not been widely tested on integer and combinatorial optimization problems.

Only extensive empirical tests may settle the issue of whether to use linear programming, as in the column generation approach, or a nonlinear method, e.g., subgradient optimization or a bundle method. Over the past 25 years subgradient optimization has been used extensively. We believe the reasons were the absence of efficient simplex codes with column generation capabilities and the lack of sufficient computer memory. However, the situation has changed dramatically with the modern simplex codes. With these capabilities in LP-based branch-and-bound codes, it is now feasible to take advantage of the global information used by the simplex codes to speed up the convergence. One can see this in comparing the results of Savelsbergh (1997) and Guignard and Rosenwein (1989) on the generalized assignment problem, where a linear programming method clearly outperforms a nonlinear method. This is in contrast with the results obtained by Held and Karp (1970, 1971) on the traveling salesman problem more than twenty years ago where the limitations of LP solving with column generation led to the conclusion that the subgradient algorithm was preferred. It still remains to be seen whether the more advanced nonlinear methods may provide an even better alternative.

	k'	k"
r	1	1
s	0	1

Figure 1. Submatrix present in candidate branching pairs.

4. BRANCHING

An LP relaxation solved by column generation is not necessarily integral and 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 branching, it may be the case that there exists a column that would price out favorably, but is not present in the master problem. Therefore, to find an optimal solution we must generate columns after branching.

4.1. Set Partitioning Master Problems

Ryan and Foster (1981) suggested a branching strategy for set partitioning problems based on the following proposition. Although they were not considering column generation, it turns out that their branching rule is very useful in this context.

Proposition 1. *If Y is a* 0-1 *matrix, and a basic solution to $Y\lambda = 1$ is fractional, i.e., at least one of the components of λ is fractional, then there exist two rows r and s of the master problem such that*

$$0 < \sum_{k:y_{rk}=1,y_{sk}=1} \lambda_k < 1.$$

Proof. Consider fractional variable $\lambda_{k'}$. Let row r be any row with $y_{rk'} = 1$. Since $\sum_{1\leq k\leq p} y_{rk}\lambda_k = 1$ and $\lambda_{k'}$ is fractional, there must exist some other basic column k'' with $0 < \lambda_{k''} < 1$ and $y_{rk''} = 1$. Since there are no duplicate columns in the basis, there must exist a row s such that either $y_{sk'} = 1$ or $y_{sk''} = 1$ but not both as illustrated by the submatrix in Figure 1. This leads to the following sequence of relations:

$$\begin{aligned} 1 &= \sum_{1\leq k\leq p} y_{rk}\lambda_k \\ &= \sum_{k:y_{rk}=1} \lambda_k \\ &> \sum_{k:y_{rk}=1,y_{sk}=1} \lambda_k, \end{aligned}$$

where the inequality follows from the fact that the last summation includes either $\lambda_{k'}$ or $\lambda_{k''}$, but not both. □

The pair r, s gives the pair of branching constraints

$$\sum_{k:y_{rk}=1,y_{sk}=1} \lambda_k = 1 \quad \text{and} \quad \sum_{k:y_{rk}=1,y_{sk}=1} \lambda_k = 0,$$

i.e., the rows r and s have to be covered by the same column on the first (left) branch and by different columns on the second (right) branch.

Proposition 1 implies that if no branching pair can be identified, then the solution to the master problem must be

integer. The branch-and-bound algorithm must terminate after a finite number of branches since there are only a finite number of pairs of rows. Note that each branching decision eliminates a large number of variables from consideration.

A theoretical justification for this branching rule is that the submatrix shown in Figure 1 is precisely the excluded submatrix in the characterization of totally balanced matrices; see Hoffman et al. (1985). Total balancedness of the coefficient matrix is a sufficient condition for the LP relaxation of a set partitioning problem to have only integral extreme points and the branching rule eventually gives totally balanced matrices.

4.1.1. Identical Restrictions on Subsets. The branching scheme suggested by Ryan and Foster requires that elements r and s belong to the same subset on the left branch and to different subsets on the right branch. Thus on the left branch, all feasible columns must have $y_{rk} = y_{sk} = 0$ or $y_{rk} = y_{sk} = 1$, while on the right branch all feasible columns must have $y_{rk} = y_{sk} = 0$ or $y_{rk} = 0$, $y_{sk} = 1$ or $y_{rk} = 1$, $y_{sk} = 0$. Rather than adding the branching constraints to the master problem explicitly, the infeasible columns in the master problem can be eliminated. On the left branch, this is identical to combining rows r and s in the master problem, giving a smaller set partitioning problem. On the right branch, rows r and s are restricted to be disjoint, which may yield an easier master problem since set partitioning problems with disjoint rows (sets) are more likely to be integral. Not adding the branching constraints explicitly has the advantage of not introducing new dual variables that have to be dealt with in the pricing problem.

Usually, enforcing the branching constraints in the pricing problem, i.e., forcing two elements to be in the same subset on one branch and forcing two elements to be in different subsets on the other branch, is fairly easy to accomplish. However, the pricing problem on one branch may be more complicated than on the other branch.

Applications of this branching rule can be found for urban transit crew scheduling in Desrochers and Soumis (1989); for airline crew scheduling in Anbil et al. (1992), Vance et al. (1997), and Vance (1993); for vehicle routing in Dumas et al. (1991); for graph coloring in Mehrotra and Trick (1996); and for the binary cutting stock problem in Vance et al. (1994).

4.1.2. Different Restrictions on Subsets. Now consider the situation where different subsets may have different requirements, i.e., the formulation for P has the block diagonal structure given by (9) and the associated explicit column generation form, with separate convexity constraints for each subset, is given by (10).

In this situation, if we apply the branching scheme suggested by Ryan and Foster but always select one partitioning row, say row r, and one convexity row, say s, we obtain a special branching scheme that has a natural interpretation in the original formulation and some nice computational properties. The pair of branching constraints that results is given by

$$\sum_{1\leqslant k\leqslant p_s : y_{rk}^s = 1} \lambda_k^s = 1 \quad \text{and} \quad \sum_{1\leqslant k\leqslant p_s : y_{rk}^s = 1} \lambda_k^s = 0. \tag{16}$$

This branching rule corresponds to requiring element r to be in subset s on the left branch and requiring element r to be in any subset but s on the right branch. This branching strategy has a very natural interpretation. It corresponds precisely to performing standard branching in (9), since

$$\sum_{1\leqslant k\leqslant p_s : y_{rk}^s = 1} \lambda_k^s = 1 \quad \text{f} \quad \sum_{1\leqslant k\leqslant p_s} y_{rk}^s \lambda_k^s = 1 \quad \text{f} \quad z_{rs} = 1,$$

and

$$\sum_{1\leqslant k\leqslant p_s : y_{rk}^s = 1} \lambda_k^s = 0 \quad \text{f} \quad \sum_{1\leqslant k\leqslant p_s} y_{rk}^s \lambda_k^s = 0 \quad \text{f} \quad z_{rs} = 0.$$

We have already shown how to apply this branching rule in the generalized assignment problem by adjusting the knapsack pricing problems. Sol and Savelsbergh (1994) show how to apply this branching rule in more general settings without increasing the difficulty of the pricing problem. To be more precise, they show that if this branching rule is applied, any algorithm for the pricing problem used in the root node can also be used in subsequent nodes.

Applications of this branching strategy are presented for crew scheduling in Vance (1993); for generalized assignment in Savelsbergh (1997); for multicommodity flow in Barnhart et al. (1995) and Parker and Ryan (1994); for vehicle routing in Desrosiers et al. (1984) and Desrochers et al. (1992); and for pickup and delivery problems in Sol and Savelsbergh (1998).

4.2. General Mixed Integer Master Problems

So far, we have discussed branching strategies for set partitioning master problems. In this section, we discuss branching strategies for general mixed integer master problems.

4.2.1. Different Restrictions on Subsets. A branching strategy for general mixed integer master problems with different restrictions on subsets can be derived directly from the integrality requirements on the original variables; see Johnson (1989). The optimal solution to the linear programming relaxation is infeasible if and only if

$$x_j = \sum_{1\leqslant k\leqslant p_j} y_k^j \lambda_k^j,$$

has a fractional component r for some j, say with value α. This suggests the following branching rule: on one branch we require

$$\sum_{1\leqslant k\leqslant p_j} y_{rk}^j \lambda_k^j \leqslant \lfloor \alpha \rfloor,$$

and on the other branch we require

$$\sum_{1\leqslant k\leqslant p_j} y_{rk}^j \lambda_k^j \geqslant \lceil \alpha \rceil.$$

4.2.2. Identical Restrictions on Subsets. Developing a branching strategy for general mixed integer master problems with identical restrictions on subsets is more complex, because we do not want to branch on original variables for reasons of symmetry. If the solution to (6) is fractional, we may be able to identify a single row r and an integer α_r such that

$$\sum_{k:(a_k)_r \geq \alpha_r} \lambda_k = \beta_r,$$

and β_r is fractional. We can then branch on the constraints

$$\sum_{k:(a_k)_r \geq \alpha_r} \lambda_k \leq \lfloor \beta_r \rfloor \quad \text{and} \quad \sum_{k:(a_k)_r \geq \alpha_r} \lambda_k \geq \lceil \beta_r \rceil.$$

These constraints place upper and lower bounds on the number of columns with $(a_k)_r \geq \alpha_r$ that can be present in the solution. In general, these constraints cannot be used to eliminate variables and have to be added to the formulation explicitly. Each branching constraint will contribute an additional dual variable to the reduced cost of any new column with $(a_k)_r \geq \alpha_r$. This may complicate the pricing problem.

It is easy to see that a single row may not be sufficient to define a branching rule. Consider a set partitioning master problem that has a fractional solution. The only possible value for α_r is 1. However, $\sum_{k:y_{kr} \geq 1} \lambda_k = 1$ for every row. Thus we may have to branch on multiple rows.

Assume that

$$\sum_{k:(a_k)_r \geq \alpha_r} \lambda_k = \beta_r,$$

and β_r is integer for every row r and integer α_r. Pick an arbitrary row, say r, and search for a row s and integer α_s such that

$$\sum_{k:(a_k)_r \geq \alpha_r \wedge (a_k)_s \geq \alpha_s} \lambda_k = \beta_s,$$

and β_s is fractional. If such a row exists, we branch on the constraints

$$\sum_{k:(a_k)_r \geq \alpha_r \wedge (a_k)_s \geq \alpha_s} \lambda_k \leq \lfloor \beta_s \rfloor \quad \text{and}$$

$$\sum_{k:(a_k)_r \geq \alpha_r \wedge (a_k)_s \geq \alpha_s} \lambda_k \geq \lceil \beta_s \rceil.$$

Otherwise we seek a third row. We note that if the solution is fractional it is always possible to find a set of rows to branch on. This branching rule was developed by Vanderbeck and Wolsey (1996) (see also Vanderbeck 1995). A slightly different version of this branching rule was developed independently by Barnhart et al. (1994).

The branching scheme just presented applied to set partitioning master problems gives precisely the branching scheme of Ryan and Foster (1981) discussed in Section 4.1. To see this, note that by Proposition 1 we can always branch on two rows, say r and s, and that the two branches are defined by

$$\sum_{k:y_{rk} \geq 1 \wedge y_{sk} \geq 1} \lambda_k \leq \lfloor \beta_s \rfloor = 0 \quad \text{and}$$

$$\sum_{k:y_{rk} \geq 1 \wedge y_{sk} \geq 1} \lambda_k \geq \lceil \beta_s \rceil = 1.$$

5. COMPUTATIONAL ISSUES

In the previous sections, we have discussed the foundations of branch-and-price algorithms. In this section, we discuss some important computational issues that need to be considered when implementing a branch-and-price algorithm.

5.1. Initial Solution

To start the column generation scheme, an initial restricted master problem has to be provided. This initial restricted master problem must have a feasible LP relaxation to ensure that proper dual information is passed to the pricing problem. Depending on the application, it is not always obvious how to construct such an initial restricted master. However, if it exists, such an initial restricted master can always be found using a two-phase method similar in spirit to the two-phase method incorporated in simplex algorithms to find an initial basic feasible solution: add a set of artificial variables with large negative costs and associated columns that form an identity matrix. The artificial variables ensure that a feasible solution to the LP relaxation exists. (In case of a set partitioning master problem, a single artificial variable with a large negative cost and an associated column consisting of all ones suffices.) Observe that an initial restricted master problem with a feasible LP relaxation has to be provided at each node in the branch-and-bound tree. Therefore, the artificial variables are usually kept at all nodes of the branch-and-bound tree.

The primary goal in defining an initial restricted master problem is to ensure the existence of a feasible LP relaxation. However, since the initial restricted master determines the initial dual variables that will be passed to the pricing problem, a "good" initial restricted master problem can be important.

5.2. Column Management

In a maximization linear program, any column with positive reduced cost is a candidate to enter the basis. The pricing problem is to find a column with highest reduced cost. Therefore, if a column with positive reduced cost exists the pricing problem will always identify it. This guarantees that the optimal solution to the linear program will be found.

However, it is not necessary to select the column with the highest reduced cost—any column with a positive reduced cost will do. Using this observation can improve the overall efficiency when the pricing problem is computationally intensive.

Various column generation schemes can be developed based on using approximation algorithms to solve the pricing problem. To guarantee optimality, a two-phase approach is applied. As long as an approximation algorithm for the pricing problem produces a column with positive reduced cost, that column will be added to the restricted master. If the approximation algorithm fails to produce a column with positive reduced cost, an optimization algorithm for the pricing problem is invoked to prove optimality or produce a column with positive reduced cost. Such a scheme reduces the computation time per iteration. However, the number of iterations may increase, and it is not certain that the overall effect is positive.

Depending on the pricing problem, it may even be possible to generate more than one column with positive reduced cost per iteration without a large increase in computation time. Such a scheme increases the time per iteration, since a larger restricted master has to be solved, but it may decrease the number of iterations.

During the column generation process, the restricted master problem keeps growing. It may be advantageous to delete nonbasic columns with very negative reduced cost from the restricted master problem in order to reduce the time per iteration.

These ideas can be combined into the following general column generation scheme:

Step 1. Determine an initial feasible restricted master problem.

Step 2. Initialize the column pool to be empty.

Step 3. Solve the current restricted master problem.

Step 4. Delete nonbasic columns with high negative reduced costs from the restricted master problem.

Step 5. If the column pool contains columns with positive reduced costs, select a subset of them, add them to the restricted master and go to 3.

Step 6. Empty the column pool.

Step 7. Invoke an approximation algorithm for the pricing problem to generate one or more columns with positive reduced cost. If columns are generated, add them to the column pool and go to 5.

Step 8. Invoke an optimization algorithm for the pricing problem to prove optimality or generate one or more columns with positive reduced costs. If columns are generated, add them to the column pool and go to 5.

Step 9. Stop.

Vanderbeck (1994) discusses many issues related to the selection of a subset of "good" columns. On the one hand we are trying to produce an integer solution to the master problem; on the other hand we are trying to solve the linear relaxation of the restricted master problem. Limited computational experience seems to suggest that when the pricing problem can be solved to optimality efficiently, then adding only the most profitable column works best, and when the pricing problem is computationally intensive using approximation algorithms and adding multiple columns works best.

Sol and Savelsbergh (1998) describe a fast heuristic approach for generating columns with positive reduced costs, which turns out to be very effective for their application. They take existing columns with reduced cost equal to zero (at least all basic columns satisfy this requirement) and employ fast local improvement algorithms to construct columns with a positive reduced cost.

Notice the similarity between the column management functions performed in branch-and-price algorithms and the row management functions performed in branch-and-cut algorithms.

5.3. LP Termination

The branch-and-bound framework has some inherent flexibility that can be exploited in branch-and-price algorithms. Observe that branch-and-bound is essentially an enumeration scheme that is enhanced by fathoming based on bound comparisons. To control the size of the branch-and-bound tree it is best to work with strong bounds, however the method will work with any bound. Clearly, there is a tradeoff between the computational efforts associated with computing strong bounds and evaluating small trees and computing weaker bounds and evaluating bigger trees. In the case of linear programming based branch-and-bound algorithms in which the linear programs are solved by column generation, there is a very natural way to explore this tradeoff, especially when the pricing problem is hard to solve. Instead of solving the linear program to optimality, i.e., generating columns as long as profitable columns exist, we can choose to prematurely end the column generation process and work with bounds on the final LP value. Lasdon (1970), Farley (1990), and Vanderbeck and Wolsey (1996) describe simple and relatively easy to compute bounds on the final LP value based on the LP value of the current restricted master problem and the current reduced costs. This is especially important in view of the tailing-off effect that many column generation schemes exhibit, i.e., requiring a large number of iterations to prove LP optimality.

5.4. Dual Solutions

Recall that the objective function of the pricing problem depends on the dual variables of the solution to the linear relaxation of the restricted master problem. Consequently, if there are alternative dual solutions, we may pick any point on the face defined by these solutions. Simplex algorithms will give a vertex of this face, whereas interior point algorithms give a point in the "center" of this face. A central point appears to have the advantage of giving a better representation of the face. Although no extensive computational tests have been done to investigate the differences, it seems that using interior point methods works somewhat better (Marsten 1994).

5.5. LP Solution

The computationally most intensive component of a branch-and-price algorithm is the solution of the linear programs, which includes the solution of many pricing problems. Therefore, we have to solve these linear programs efficiently to obtain efficient branch-and-price algorithms. We consider two alternatives to accomplish this.

- Employ specialized simplex procedures that exploit the problem structure.
- Alter the master problem formulation to reduce the number of columns.

Again consider the master problem given by (10), but with equality convexity constraints.

$$\max \sum_{1\leqslant j\leqslant n} \sum_{1\leqslant k\leqslant p_j} (c_j y_k^j)\lambda_k^j$$

$$\sum_{1\leqslant j\leqslant n} \sum_{1\leqslant k\leqslant p_j} y_{ik}^j \lambda_k^j = 1, \quad i = 1, \ldots, m,$$

$$\sum_{1\leqslant k\leqslant p_j} \lambda_k^j = 1, \quad j = 1, \ldots, n,$$

$$\lambda_k^j \in \{0, 1\}, \quad j = 1, \ldots, n,\ k = 1, \ldots, p_j. \tag{17}$$

Since the column generation form of P has the Dantzig-Wolfe master program structure, it can be solved using specialized solution procedures such as the generalized upper bounding procedure of Dantzig and Van Slyke (1967) or the partitioning procedure of Rosen (1964). Both of these procedures exploit the block-diagonal convexity constraint structure of (17) and perform all steps of the simplex method on a reduced working basis of dimension m. Both methods transform the MP problem formulation by selecting *key* extreme point solutions defined by $\lambda_{k_j^*}^j = 1$, $j = 1, \ldots, n$ and substituting $\lambda_{k_j^*}^j = 1 - \sum_{1\leqslant k\leqslant p_j, k\neq k_j^*} \lambda_k^j$ for each subproblem j. With this substitution, the key formulation of (17) becomes

$$\max \sum_{1\leqslant j\leqslant n} \sum_{1\leqslant k\leqslant p_j} (c_j(y_k^j - y_{k_j^*}^j))\lambda_k^j + \sum_{1\leqslant j\leqslant n} c_j y_{k_j^*}^j$$

$$\sum_{1\leqslant j\leqslant n} \sum_{1\leqslant k\leqslant p_j, k\neq k_j^*} (y_{ik}^j - y_{ik_j^*}^j)\lambda_k^j$$

$$= 1 - \sum_{1\leqslant j\leqslant n} y_{ik_j^*}^j, \quad i = 1, \ldots, m,$$

$$\sum_{1\leqslant k\leqslant p_j, k\neq k_j^*} \lambda_k^j \leqslant 1, \quad j = 1, \ldots, n,$$

$$\lambda_k^j \in \{0, 1\}, \quad j = 1, \ldots, n,\ k = 1, \ldots, p_j. \tag{18}$$

The variable λ_k^j now indicates whether the key extreme point k_j^* of subproblem j should be transformed into extreme point k of subproblem j ($\lambda_k^j = 1$) or not ($\lambda_k^j = 0$). To enforce nonnegativity of $\lambda_{k_j^*}^j$, the *key nonnegativity* constraints $\sum_{1\leqslant k\leqslant p_j, k\neq k_j^*} \lambda_k^j \leqslant 1$ are required. They can be added explicitly, but in both the Dantzig-Van Slyke and Rosen procedures, the key nonnegativity constraints are handled implicitly by changing the key extreme point when they are about to be violated.

To illustrate, consider the multicommodity network flow problem. In a column generation formulation of the multicommodity flow problem, the variables represent origin-destination flows of commodities. In the associated key formulation, a specific origin-destination path p_j^* is selected for each commodity j to serve as a *key path*. Any other origin-destination path p_j for commodity j will be represented by a column with

$+1$ for each arc in p_j and not in p_j^*;
-1 for each arc in p_j^* and not in p_j;
0 for each arc in both or neither p_j and p_j^*; and
$+1$ for the key path nonnegativity constraint for j.

The variables of the key formulation represent the symmetric difference between the key path for a commodity and some other origin-destination path for that commodity. Since the symmetric difference of two paths that share a common origin and destination is a set of cycles, we can think of these variables as representing flow shifts around these cycles, i.e., flow is removed from the key path and placed on an alternative path. (A detailed description is provided in Barnhart et al. 1995.)

Although the Dantzig-Van Slyke and Rosen procedures may improve LP solution times, because of the smaller working basis, they do not prevent tailing-off, i.e., slow convergence to LP optimality, which is a major efficiency issue for column generation procedures. To reduce the tailing-off effect, an alternate key formulation having far fewer columns may be used. The idea behind the alternate key formulation is to allow columns to be represented as a combination of simpler columns.

Consider the multicommodity flow problem again. In the key formulation for the multicommodity flow problem, each column corresponds to sets of disjoint cycles. Refer to a column containing a single cycle as a simple cycle and to one containing multiple cycles as a compound cycle. Since every compound cycle can be represented as the sum of simple cycles, every possible multicommodity flow solution can be represented with just simple cycles. In the column generation framework, each column generated by the pricing problem has to be decomposed into simple cycles and only these simple cycles are added to the restricted simple cycle master problem are added.

Since several simple cycles can be chosen, the key path nonnegativity constraints have to be modified. The nonnegativity constraint for each key path p_j^* can be replaced by a set of constraints, one for each key path arc, ensuring that the flow on the arc is nonnegative. As above, these constraints can be handled implicitly.

The idea presented above for the multicommodity flow problem generalizes to any problem with the master program structure of (10). As before, the key formulation is obtained by selecting a key column k_j^* for subproblem j and substituting it out, i.e., replacing all other columns k of subproblem j by a column with

$+1$ for each element in k and not in k_j^*;
-1 for each element in k_j^* and not in k;
0 for each element in both or neither k and k_j^*; and
$+1$ for the key column nonnegativity constraint for j.

These columns are referred to as *exchanges* since one or more elements in the key column k_j^* may be removed from the solution and replaced by new elements in column k. Similar to the concept of simple cycles in the multicommodity flow context, a column in the key formulation that cannot be represented as the sum of other columns is called an *elementary exchange*. In solving the LP relaxation of (18) it suffices to work with elementary exchange columns and thereby obtaining a substantial reduction in the number of columns. See Vance (1993) for further details.

5.6. Primal Heuristics

A branch-and-price algorithm can be easily turned into an effective approximation algorithm when finding a good feasible IP solution is the major concern and proving optimality is of lesser or no importance. This is accomplished by branching and searching the tree in a greedy fashion. For example, in set partitioning problems where branching is performed on a pair of rows, there are generally many pairs that can be chosen to eliminate the current fractional solution. If the goal is to prove optimality, it usually makes sense to choose a branching decision that divides the solution space evenly, i.e., we expect it to be equally likely to find a good solution at either of the two nodes created. If the goal is to find a good feasible solution, it makes sense to choose a branching decision that divides the solution space in such a way that we are more likely to find a good solution in one of the two nodes created and then choose this node for evaluation first. For set partitioning problems, this would correspond to choosing a pair of rows r and s such that

$$\sum_{k:y_{rk}=y_{sk}=1} \lambda_k$$

has a value close to one. When this is the case, we are much more likely to find a good integer solution on the $\Sigma_{k:y_{rk}=y_{sk}=1}\ \lambda_k = 1$ branch than on the $\Sigma_{k:y_{rk}=y_{sk}=1}\ \lambda_k = 0$ branch. We then greedily search the tree always following the branch that is more likely to yield a good feasible solution. This approach has been applied to crew pairing problems with good results, see for instance Krishna et al. (1995). In their implementation, the LP relaxation was not completely reoptimized at each node of the branch-and-bound tree, instead a maximum number of column generation iterations per node was set and columns were generated only when the LP bound increased significantly above the value of the root LP.

A somewhat similar greedy search strategy for finding good feasible solutions has been developed based on the standard branching rule. The approach is effective when maintaining the ability to backtrack in the tree search is not important. Each time a variable with a high fractional value is chosen and its value is permanently fixed to one. In Marsten (1994), this strategy was successfully applied to the crew pairing problem. Pairings with high fractional value were fixed into the solution sequentially. Whenever the value of the LP relaxation increased above a preset target, new pairings were generated. The pairing generation subproblem remained tractable since each fixed variable simply eliminates the flights it covers from the pairing generation subproblem.

5.7. Combining Column Generation and Row Generation

Combining column and row generation can yield very strong LP relaxations. However, synthesizing the two generation processes is nontrivial. The principle difficulty is that the pricing problem can become much harder after additional rows are added, because the new rows can destroy the structure of the pricing problem. To illustrate, consider the generalized assignment problem. The restricted master problem is a set partitioning problem and a well-known class of valid inequalities for the set partitioning problem are clique inequalities (Padberg 1973), which simply say in linear programming terms that the sum of the variables in the clique cannot exceed one. Now consider the pricing problem after the addition of a clique inequality to the restricted master problem. The reduced cost of a new column depends on whether it has a 0 or 1 in the row associated with the clique inequality. Therefore, two cases for this column need to be considered, and after k cliques have been added 2^k cases are required.

It is of course possible to do the pricing only over the original rows, i.e., assume that the new columns have 0 coefficients in the additional rows. However, it may be necessary to update the coefficients for the additional rows in order to maintain validity (see Mehrotra 1992), or it may be desirable to lift the coefficients to increase the strength of the valid inequality. Unfortunately, after the lifting is done, it may be the case that the column no longer prices out favorably.

Despite these difficulties, there have been some successful applications of combined row and column generation. In problem situations where the objective is to partition the ground set into a minimum number of feasible subsets, such as minimizing the number of vehicles required to satisfy customer demands in routing and scheduling problems, an LP solution with fractional objective function value v can be cut off by adding a constraint that bounds the LP solution from above by $\lfloor v \rfloor$. Because every column has a coefficient 1 in this additional constraint, the constraint does not complicate the pricing problem and can easily be handled.

Nemhauser and Park (1991) combine column and row generation in an LP-based algorithm for the edge coloring problem. The edge coloring problem requires a partitioning of the edges of a graph into a minimum cardinality set of matchings. Therefore, it can naturally be formulated as a set partitioning problem in which the columns correspond to matchings of the graph. Consequently, the pricing problem is a weighted matching problem. However, to strengthen the linear programming relaxation, they add

odd-circuit constraints to the restricted master, which destroys the pure matching structure of the pricing problem. The pricing problem now becomes a matching problem with an additional variable for each odd circuit constraint, and an additional constraint for each odd circuit variable which relates the odd circuit variable to the edge variables in the circuit. This problem is solved by branch-and-cut. The approach points out the need for recursive calling of integer programming systems for the solution of complex problems.

5.8. Implementation

Although implementing branch-and-price algorithms (or branch-and-cut algorithms) is still a nontrivial activity, the availability of flexible linear and integer programming systems has made it a less formidable task than it would have been five years ago.

Modern simplex codes, such as CPLEX (CPLEX Optimization 1990) and OSL (IBM Corporation 1990) not only permit column generation while solving an LP but also allow the embedding of column generation LP solving into a general branch-and-bound structure for solving MIPs.

The use of MINTO (Nemhauser et al. 1994, Savelsbergh and Nemhauser 1993) may reduce the implementation efforts even further. MINTO (Mixed INTeger Optimizer) is an effective general purpose mixed integer optimizer that can be customized through the incorporation of application functions. Its strength is that it allows users to concentrate on problem specific aspects rather than data structures and implementation details such as linear programming and branch-and-bound.

ACKNOWLEDGMENT

We would like to thank two anonymous referees for helpful suggestions related to the organization and clarity of the paper. This research was supported by NSF SES-91-22674, NSF and AFORS DDM-9115768, NSF DDM-9058074, NSF DMI-9410102, and IBM MHV2379.

REFERENCES

ANBIL, R., R. TANGA, AND E. L. JOHNSON. 1992. A Global Approach to Crew-pairing Optimization. *IBM Systems J.* **31,** 71–78.

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, AND G. SIGISMONDI. 1995. An Alternative Formulation and Solution Strategy for Multi-Commodity Network Flow Problems. *Telecommunications Systems* **3,** 239–258.

BARNHART, C., E. L. JOHNSON, G. L. NEMHAUSER, M. W. P. SAVELSBERGH, AND P. VANCE. 1994. Branch-and-Price: Column Generation for Solving Huge Integer Programs (abbreviated version). *Mathematical Programming, State of the Art.* J. R. Birge and K. G. Murty (eds.), Braun-Brumfield, 186–207.

BALAS, E., AND M. PADBERG. 1976. Set Partitioning: A Survey. *SIAM Review,* **18,** 710–760.

BIXBY, R. E., J. W. GREGORY, I. J. LUSTIG, R. E. MARSTEN, AND D. F. SHANNO. 1992. Very Large-scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods. *Opns. Res.* **40,** 885–897.

BROOKS, R., AND A. GEOFFRION. 1966. Finding Everett's Lagrange Multipliers by Linear Programming. *Opns. Res.* **14,** 1149–1153.

CPLEX OPTIMIZATION, INC. 1990. *Using the* CPLEX™ *Linear Optimizer*.

DANTZIG, G. B., AND R. M. VAN SLYKE. 1967. Generalized Upper Bounding Techniques. *J. Computer System Sci.* **1,** 213–226.

DANTZIG, G. B., AND P. WOLFE. 1960. Decomposition Principle for Linear Programs. *Opns. Res.* **8,** 101–111.

DESROCHERS, M., J. DESROSIERS, AND M. SOLOMON. 1992. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Opns. Res.* **40,** 342–354.

DESROCHERS, M., AND F. SOUMIS. 1989. A Column Generation Approach to the Urban Transit Crew Scheduling Problem. *Transp. Sci.* **23,** 1–13.

DESROSIERS, J., Y. DUMAS, M. M. SOLOMON, AND F. SOUMIS. 1995. Time Constrained Routing and Scheduling. *Handbooks in Operations Research and Management Science, Volume 8: Network Routing.* M. Ball et al. (eds.), Elsevier, Amsterdam, 35–140.

DESROSIERS, J., F. SOUMIS, AND M. DESROCHERS. 1984. Routing with Time Windows by Column Generation. *Networks,* **14,** 545–565.

DUMAS, Y., J. DESROSIERS, AND F. SOUMIS. 1991. The Pickup and Delivery Problem with Time Windows. *European J. Opns. Res.* **54,** 7–22.

FARLEY, A. A. 1990. A Note on Bounding a Class of Linear Programming Problems, Including Cutting Stock Problems. *Opns. Res.* **38,** 922–924.

FORD, L. R., AND D. R. FULKERSON. 1958. A Suggested Computation for Maximal Multicommodity Network Flows. *Mgmt. Sci.* **5,** 97–101.

GEOFFRION, A. M. 1974. Lagrangean Relaxation for Integer Programming. *Math. Programming Studies,* **2,** 82–114.

GUIGNARD, M., AND M. ROSENWEIN. 1989. An Improved Dual-based Algorithm for the Generalized Assignment Problem. *Opns. Res.* **37,** 658–663.

HELD, M., AND R. M. KARP. 1970. The Traveling-salesman Problem and Minimum Spanning Trees. *Opns. Res.* **18,** 1138–1162.

HELD, M., AND R. M. KARP. 1971. The Traveling-salesman Problem and Minimum Spanning Trees: Part II. *Math. Programming,* **1,** 6–25.

HELD, M., P. WOLFE, AND H. P. CROWDER. 1974. Validation of Subgradient Optimization. *Math. Programming,* **6,** 62–88.

HOFFMAN, A. J., A. KOLEN, AND M. SAKAROVITCH. 1985. Totally Balanced and Greedy Matrices. *SIAM J. Algebraic and Discrete Methods,* **6,** 721–730.

HOFFMAN, K., AND M. PADBERG. 1985. LP-based Combinatorial Problem Solving. *Annals O. R.* **4,** 145–194.

IBM CORPORATION. 1990. *Optimization Subroutine Library, Guide and Reference.*

JOHNSON, E. L. 1989. Modeling and Strong Linear Programs for Mixed Integer Programming. *Algorithms and Model Formulations in Mathematical Programming.* S. W. Wallace (ed.), NATO ASI Series 51, 1–41.

JUNGER, M., G. REINELT, AND G. RINALDI. 1995. The Traveling Salesman Problem. In *Handbooks in Operations Research and Management Science, Volume 7: Network Models*. M. Ball et al. (eds.), Elsevier, Amsterdam, 225–330.

KRISHNA, A., C. BARNHART, E. L. JOHNSON, D. MAHIDARA, G. L. NEMHAUSER, R. REBELLO, A. SINGH, AND P. H. VANCE. 1995. *Advanced Techniques in Crew Scheduling*. Presentation at INFORMS, Los Angeles, CA.

LASDON, L. S. 1970. *Optimization Theory for Large Systems*. MacMillan, New York.

LEMARECHAL, C. 1989. Nondifferential Optimization. In *Handbooks in Operations Research and Management Science, Volume 1: Optimization*. G. Nemhauser et al. (eds.), Elsevier, Amsterdam, 529–572.

MARSTEN, R. 1994. *Crew Planning at Delta Airlines*. Presentation at Mathematical Programming Symposium XV, Ann Arbor, MI.

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

MEHROTRA, A., AND M. A. TRICK. 1996. *A Column Generation Approach for Exact Graph Coloring. INFORMS J. Computing* **8,** 344–354.

NEMHAUSER, G. L., AND S. PARK. 1991. A Polyhedral Approach to Edge Coloring. *O. R. Lett.* **10,** 315–322.

NEMHAUSER, G. L., M. W. P. SAVELSBERGH, AND G. C. SIGISMONDI. 1994. MINTO, a Mixed INTeger Optimizer. *O. R. Lett.* **15,** 47–58.

NEMHAUSER, G. L., AND L. A. WOLSEY. 1988. *Integer and Combinatorial Optimization*. Wiley, New York, NY.

PADBERG, M. W. 1973. On the Facial Structure of Set Packing Polyhedra. *Math. Programming,* **5,** 199–215.

PARKER, M., AND J. 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., AND B. A. FOSTER. 1981. An Integer Programming Approach to Scheduling. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. A. Wren (ed.), North-Holland, Amsterdam, 269–280.

SAVELSBERGH, M. W. P. 1997. A Branch-and-Price Algorithm for the Generalized Assignment Problem. *Opns. Res.* **6,** 831–841.

SOL, M., AND M. W. P. SAVELSBERGH. 1998. DRIVE: Dynamic Routing of Independent Vehicles. *Opns. Res.* **46,** to appear.

VANCE, P. H. 1993. Crew Scheduling, Cutting Stock, and Column Generation: Solving Huge Integer Programs. Ph.D. Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

VANCE, P. H., C. BARNHART, E. L. JOHNSON, AND G. L. NEMHAUSER. 1994. Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound. *Computational Optimization and Applications,* **3,** 111–130.

VANCE, P. H., C. BARNHART, E. L. JOHNSON, AND G. L. NEMHAUSER. 1997. Airline Crew Scheduling: A New Formulation and Decomposition Algorithm. *Opns. Res.* **45,** 188–200.

VANDERBECK, F. 1994. Decomposition and Column Generation for Integer Programs. Ph.D. Thesis, Universite Catholique de Louvain, Belgium.

VANDERBECK, F. 1995. On Integer Programming Decomposition and Ways to Enforce Integrality in the Master. Report 94-95-29, University of Cambridge, England.

VANDERBECK, F., AND L. A. WOLSEY. 1996. An Exact Algorithm for IP Column Generation. *Opns. Res. Letters* **19,** 151–160.