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

Cynthia Barnhart [1]
Ellis L. Johnson
George L. Nemhauser
Martin W.P. Savelsbergh
Pamela H. Vance [2]

Georgia Institute of Technology
School of Industrial and Systems Engineering
Atlanta, GA 30332-0205

Abstract

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.

1 Introduction

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.

[01] Currently at MIT
[02] Currently at Auburn University
[0*] This research has been supported by the following grants and contracts: NSF SES-91-22674, NSF and AFORS DDM-9115768, NSF DDM-9058074 and IBM MHV2379

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 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 sub problems. 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:

- 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 and describing applications. It is by no means an extensive survey, but does develop some general ideas that have only appeared in very special contexts. Routing and scheduling has been a particularly fruitful application area of branch-and-price, see Desrosiers et al. [1994] for a survey of these results.

Section 2 presents 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 discusses the efficient solution of LPs in branch-and-price algorithms. Section 6 considers the implementation of branch-and-price in mixed-integer programming codes. Section 7 summarizes computational experience with branch-and-price algorithms for binary cutting stock problems, generalized assignment problems, urban transit crew scheduling problems and bandwidth packing problems.

2 Suitable Models for Column Generation

2.1 General Models

The general problem P we consider is of the form

$$\begin{aligned} \max\ & cx \\ Ax &\le b, \\ x &\in S, \\ x &\quad \text{integer}, \end{aligned} \tag{1}$$

where S is a bounded polyhedron. The boundedness assumption is not necessary and is made purely for simplicity of exposition.

The fundamental construct of column generation is that the set

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

is represented by the extreme points $y_1, ..., y_p$ of its convex hull. Note that if x is binary, then S^* coincides with the extreme points of $conv(S^*)$.

Any point y in $conv(S^*)$ can be represented as

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

subject to the convexity constraint

$$\begin{aligned} \sum_{1 \le k \le p} \lambda_k &= 1, \\ \lambda_k &\ge 0 \qquad k = 1, ..., p. \end{aligned}$$

This yields the column generation form of P given by

$$\begin{aligned} \max \sum_{1 \le k \le p} (cy_k)\lambda_k & \\ \sum_{1 \le k \le p} (Ay_k)\lambda_k &\le b, \\ \sum_{1 \le k \le p} y_k\lambda_k &\quad \text{integer}, \\ \sum_{1 \le k \le p} \lambda_k &= 1, \\ \lambda_k \ge 0 &\qquad k = 1, ..., p. \end{aligned} \tag{2}$$

When x is binary, the condition $\sum_{1 \le k \le p} y_k\lambda_k$ integer is equivalent to $\lambda_k \in \{0, 1\}$ for $k = 1, ..., p$. If the null vector is an extreme point of $conv(S^*)$, it may not be explicitly included in the formulation, in which case the convexity constraint can be written as an inequality, i.e., $\sum_{1 \le k \le p} \lambda_k \le 1$.

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

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

by the extreme points $y_1^j, ..., y_{p_j}^j$ of its convex hull, i.e., any point y^j in $conv(S_j^*)$ can be represented as

$$y^j = \sum_{1 \le k \le p_j} y_k^j \lambda_k^j,$$

subject to the convexity constraint

$$\begin{aligned} \sum_{1\le k\le p_j} \lambda_k^j &= 1, \\ \lambda_k^j &\ge 0 \quad k=1,\dots,p_j. \end{aligned}$$

This yields a column generation form of P with separate convexity constraints for each S_j given by

$$\begin{aligned} \max \sum_{1\le j\le n}\sum_{1\le k\le p_j} (cy_k^j)\lambda_k^j & \\ \sum_{1\le j\le n}\sum_{1\le k\le p_j} (Ay_k^j)\lambda_k^j &\le b, \\ \sum_{1\le k\le p_j} y_k^j\lambda_k^j \quad & \text{integer} \quad j=1,\dots,n, \\ \sum_{1\le k\le p_j} \lambda_k^j &= 1 \quad j=1,\dots,n, \\ \lambda_k^j \ge 0 \quad & j=1,\dots,n,\ k=1,\dots,p_j. \end{aligned} \tag{3}$$

If the subsets in the decomposition are identical, i.e., $S_j = \overline{S}$ for $j=1,\dots,n$, then they can be combined into one subset $\overline{S}$ with the convexity constraints

$$\sum_{1\le k\le p} \lambda_k^j = 1, \quad j=1,\dots,n$$

replaced by an aggregated convexity constraint

$$\sum_{1\le k\le p} \lambda_k = n.$$

This results in the column generation form

$$\begin{aligned} \max \sum_{1\le k\le p} (cy_k)\lambda_k & \\ \sum_{1\le k\le p} (Ay_k)\lambda_k &\le b, \\ \sum_{1\le k\le p} y_k\lambda_k \quad & \text{integer}, \\ \sum_{1\le k\le p} \lambda_k &\le n, \\ \lambda_k \ge 0 \quad & k=1,\dots,p, \end{aligned} \tag{4}$$

where $y_1, ..., y_p$ are the extreme points of $conv(\overline{S}^*)$. Here we have chosen the inequality form of the aggregated convexity constraint because in most applications no elements are assigned to some subsets. Moreover, if n is not fixed as part of the input, then the aggregated convexity constraint can be omitted altogether.

The essential difference between P and its column generation form is that S has been replaced by the extreme point representation of its convex hull. 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 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

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

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

2.2 Partitioning models

Many interesting 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 costs, and we wish to find the minimum cost 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. Similarly, let c_{ij} denote the profit associated with having element i in subset j and let c_j denote the corresponding profit vector. The general partitioning problem is of the form

$$\begin{aligned} \max \sum_{1 \le j \le n} c_j z_j & \\ \sum_{1 \le j \le n} z_{ij} &= 1 \quad i = 1, ..., m, \end{aligned}$$

$$\begin{array}{rcl} z_j & \in & S, \\ z_j & & \text{binary}, \end{array} \qquad (5)$$

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 Enumerated subsets

One important class of partitioning problems for which column generation is desirable occurs when we do not know a description of S by linear inequalities, but we do know a way of enumerating S. Hopefully, the enumeration can be done cleverly, but even a brute force approach may suffice.

This structure occurs, for example, in crew pairing problems, where a sequence of flights, called a pairing, has to be constructed and assigned to a crew. The first flight in the sequence must depart from the crew's base, each subsequent flight departs from the station where the previous flight arrived and the last flight must return to the base. The sequence can represent several days of flying. Pairings are subject to a number of constraints resulting from safety regulations and labor 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. The main point is that these restrictions are not efficiently described by linear inequalities. In addition, the cost of pairings is a messy function of several attributes of the sequence.

Although enumerating pairings is complex because of all the rules that must be checked, 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 difficulty is the total number of pairings, which grows exponentially with the number of flights. For example, in a typical problem with 253 flights, there are 5,833,004 pairings [Vance 1993]. However, it is possible to represent pairings as paths in a graph, and to evaluate their costs with a multilabel shortest path or dynamic programming algorithm, see Desrochers and Soumis [1989], Barnhart et al. [1993], and Vance [1993].

The enumeration yields the following column generation form

$$\begin{array}{l} \max \sum_{1 \le k \le p} (c_k y^k) \lambda_k \\ \qquad \begin{array}{rcll} \sum_{1 \le k \le p} y_i^k \lambda_k & = & 1 & i = 1, ..., m, \\ \lambda_k & \in & \{0,1\}, & k = 1, ..., p, \end{array} \end{array}$$

where each y^k is an element of S. This column generation form corresponds to the 'standard' formulation of the set partitioning problem.

2.2.2 Linearly constrained subsets

Now we suppose that the rules on feasible subsets in a set partitioning problem can be described by linear inequalities.

Different restrictions on subsets
First, we assume that the feasible subsets have different requirements. Assume the requirements are given by

$$S_j = \{z_j : D_j z_j \le d_j \quad j = 1, ..., n, \quad z_j \text{ binary}\}. \tag{6}$$

More explicitly, problem P is given by

$$\begin{aligned} \max \sum_{1 \le j \le n} c_j z_j & & & \\ \sum_{1 \le j \le n} z_{ij} &= 1 & & i = 1, ..., m, \\ D_j z_j &\le d_j & & j = 1, ..., n, \\ z & & \text{binary}, & \end{aligned} \tag{7}$$

and its column generation form by

$$\begin{aligned} \max \sum_{1 \le j \le n} \sum_{1 \le k \le p_j} (c_j y_k^j) \lambda_k^j & & & \\ \sum_{1 \le j \le n} \sum_{1 \le k \le p_j} y_{ik}^j \lambda_k^j &= 1 & & i = 1, ..., m, \\ \sum_{1 \le k \le p_j} \lambda_k^j &\le 1 & & j = 1, ..., n, \\ \lambda_k^j &\in \{0, 1\} & & j = 1, ..., n, \; k = 1, ..., p_j, \end{aligned} \tag{8}$$

where the $\{y_k^j\}$, $1 \le k \le p_j$ are the extreme points of $conv(S_j^*)$ with elements y_{ik}^j for $i = 1, ..., m$. We have chosen to write the convexity constraint as an inequality, since in many of these applications we may not assign any elements to a given subset.

To illustrate, consider the generalized assignment problem (GAP). In the 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.

The standard integer programming formulation of GAP is

$$\begin{aligned}
\max \sum_{1\le i\le m}\sum_{1\le j\le n} p_{ij}z_{ij} & & & \\
\sum_{1\le j\le n} z_{ij} &= 1 & & i=1,...,m,\\
\sum_{1\le i\le m} w_{ij}z_{ij} &\le d_j & & j=1,...,n,\\
z_{ij} &\in \{0,1\} & & i=1,...,m,\ j=1,...,n,
\end{aligned}$$

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.

The column generation form is

$$\begin{aligned}
\max \sum_{1\le j\le n}\sum_{1\le k\le K_j}\Big(\sum_{1\le i\le m} p_{ij}y^j_{ik}\Big)\lambda^j_k & & & \\
\sum_{1\le j\le n}\sum_{1\le k\le K_j} y^j_{ik}\lambda^j_k &= 1 & & i=1,...,m,\\
\sum_{1\le k\le K_j} \lambda^j_k &\le 1 & & j=1,...,n,\\
\lambda^j_k &\in \{0,1\} & & j=1,...,n,\ k=1,...,K_j,
\end{aligned}$$

where the first m entries of a column, given by $y^j_k = (y^j_{1k}, y^j_{2k}, ..., y^j_{mk})$, form a feasible solution to the knapsack problem

$$\begin{aligned}
\sum_{1\le i\le m} w_{ij}y^j_i &\le d_j,\\
y^j_i &\in \{0,1\} \quad i=1,...,m.
\end{aligned}$$

In other words, a column represents a feasible assignment of tasks to a machine. Note that by replacing the knapsack constraint by its feasible solutions, we have improved the quality of the linear programming relaxation.

Identical restrictions on subsets
Now, we assume that the feasible subsets have identical requirements. Then (6) is replaced by the single set of inequalities

$$\overline{S} = \{z_j : Dz_j \le d \quad j=1,...,n, \quad z_j \text{ binary}\}. \tag{9}$$

More explicitly, problem P is given by

$$\begin{aligned} \max \sum_{1\le j\le n} & c_j z_j \\ \sum_{1\le j\le n} z_{ij} &= 1 \quad i = 1, ..., m, \\ D z_j &\le d \quad j = 1, ..., n, \\ z & \quad \text{binary}, \end{aligned} \tag{10}$$

and its column generation form by

$$\begin{aligned} \max \sum_{1\le k\le p} & (c y_k)\lambda_k \\ \sum_{1\le k\le p} y_{ik}\lambda_k &= 1 \quad i = 1, ..., m, \\ \lambda_k &\in \{0,1\} \quad k = 1, ..., p. \end{aligned} \tag{11}$$

Here we have chosen to omit the convexity constraint because it is common in these applications for n not to be fixed.

Consider the 0-1 cutting stock problem where item i has length d_i, the demand for each item is 1, the length of each stock roll is d and the objective is to meet demand using the minimum number of stock rolls. An integer programming formulation is

$$\begin{aligned} \min \sum_{1\le j\le n} & w_j \\ \sum_{1\le j\le n} z_{ij} &= 1 \quad i = 1, ..., m, \\ \sum_{1\le i\le m} d_i z_{ij} &\le d w_j \quad j = 1, ..., n, \\ z_{ij}, w_j &\in \{0,1\} \quad i = 1, ..., m,\ j = 1, ..., n, \end{aligned}$$

where $w_j = 1$ if roll j is selected and $z_{ij} = 1$ if item i is assigned to roll j.

The column generation formulation is

$$\begin{aligned} \min \sum_{1\le k\le p} & \lambda_k \\ \sum_{1\le k\le p} y_{ik}\lambda_k &= 1, \quad i = 1, ..., m, \\ \lambda_k &\in \{0,1\} \quad k = 1, ..., p, \end{aligned}$$

where each y_k is a binary solution to the knapsack inequality

$$\sum_{1\le i\le m} d_i y_{ik} \le d.$$

That is, each y_k represents a feasible pattern for cutting one of the stock rolls into some subset of the items. MP replaces the knapsack inequality by all of its 0-1 solutions. The resulting problem has an LP relaxation which very frequently provides a bound whose round up equals the value of an optimal solution, Marcotte [1985]. On the other hand, the LP relaxation of the formulation P yields the trivial bound of $\lceil(\sum_{1\le i\le m} d_i)/d\rceil$.

Another 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.

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, it is easy to verify that 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.

2.3 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, see Junger, Reinelt, and Rinaldi [1994]. 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 6 million variables was solved in less than an hour on an IBM 3090E vector facility, see Anbil, Tanga, and Johnson [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 is polynomial in the size of the input, in contrast to the other formulations we have discussed where the number of variables grows exponentially.

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 (1) given by

$$\begin{aligned} g(\pi) = \max \quad & [cx - \pi(b - Ax)] \\ & x \in S \\ & x \text{ integer} \end{aligned} \tag{12}$$

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

$$g = \min\{g(\pi) : \pi \geq 0\}. \tag{13}$$

Using linear programming duality, Geoffrion [1974] proved that the maximum value of the linear programming relaxation of (2) equals g. Moreover, the Lagrangian relaxation (12) is exactly of the form of the column generation subproblem that is used in the solution of the linear programming relaxation of (2). 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. In Lagrangian duality, the standard approach is to solve (13) by subgradient optimization, see Held et al. [1974]. Here, given a solution to (12), we determine a subgradient direction by the magnitude of violation of $Ax \leq b$ and a new π vector by

$$\pi_i^{t+1} = \pi_i^t - \vartheta^t[A_i x^t - b_i]^+$$

where $u^+ = \max(0, u)$ and ϑ^t is the step size. With an appropriate choice of step size, the subgradient algorithm converges in the limit but the sequence $g(\pi^t)$ is not monotone.

In the column generation approach, π^t is an optimal solution to the t^{th} restricted linear programming relaxation of the restricted master problem. This procedure converges finitely and monotonically, although column generation linear programs are known to stall near termination.

Since both methods produce the same theoretical bounds and solve subproblems of the same form, other criteria are needed to choose between them. There are clear tradeoffs:

- The subgradient algorithm is much simpler than the simplex algorithm with column generation and therefore is much easier to code.

- Linear programming uses global information, i.e. $x^1, \dots, x^t$ to determine π^{t+1}, while the subgradient algorithm uses only local information (π^t, x^t).
- The subgradient algorithm is faster per iteration but generally requires many more iterations and practical implementations are not finitely convergent.

Only extensive empirical tests may settle the issue, but here are some observations. Over the past 25 years the Lagrangian approach has been used much more extensively. We believe the reasons were the absence of efficient simplex codes with column generation capabilities and the lack of sufficient computer memory. While it is difficult to implement an efficient simplex procedure, a subgradient optimization program only requires a few lines of code. However, the situation has changed dramatically with the modern simplex codes, see Section 6. 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 of π. One can see this in comparing the results of Savelsbergh [1993] and Guignard and Rosenwein [1989] on the generalized assignment problem, see Section 7 on computational experience. 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.

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. Nonetheless, many problems have been solved successfully, but not to proven optimality, by the heuristic of limiting the column generation to the root node of the branch-and-bound tree.

Consider a branch-and-bound algorithm that has the possibility of generating columns at any node of the tree. In particular, suppose using the conventional branching rule based on variable dichotomy, we branch on fractional variable λ_k, and we are in the branch in which λ_k is fixed to zero. In the column generation phase, it is possible (and quite likely) that the optimal solution to the subproblem will be the set represented by λ_k. In that case, it becomes necessary to generate the column with the 2^{nd} highest reduced cost. At depth l in the branch-and-bound tree we may need to find the column with l^{th} highest reduced cost. In order to prevent columns that have been branched on from being regenerated, we must choose a branching rule that is *compatible* with the pricing problem. By compatible, we mean that we must be able to modify the pricing problem so that columns that are infeasible due to the branching constraints will not be generated and the pricing problem will remain tractable.

So the challenge in formulating a branching strategy is to find one that excludes the current solution, validly partitions the solution space of the problem, and provides a pricing problem that is still tractable. We need a guarantee that a feasible integer solution will be found (or infeasibility proved) after a finite number of branches and we need to be able to encode the branching information into the pricing problem. In addition, a branch-and-bound algorithm is more likely to be effective if the branching scheme divides the feasible set of solutions to the problem evenly, i.e. each new subproblem created has approximately the same number of feasible solutions.

4.1 Set partitioning master problems

Ryan and Foster [1981] suggested a branching strategy for set partitioning problems based on the following proposition.

Proposition 1 *If Y is a 0–1 matrix, and a basic solution to $Y\lambda = 1$ 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 $y_{rk''} = 1$ as illustrated by the submatrix in Figure 1. Since there

	k	k’
r	1	1
s	0	1

Figure 1: Submatrix Present in Candidate Branching Pairs

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. This leads to the following sequence of relations:

$$1 = \sum_{1\leq k\leq p} y_{rk}\lambda_k$$

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

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

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

This branching is analogous to requiring that rows r and s be covered by the same column on the first (left) branch and by different columns on the second (right) branch, i.e., 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; see Section 7.

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. The number of branches necessary will generally be considerably fewer than would be necessary if individual variables were branched on since there are in general many fewer pairs of rows than variables. In addition, 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, Kolen, and Sakarovitch [1985]. Total balanceness 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.

Applications of this branching rule can be found for urban transit crew scheduling in Desrochers and Soumis [1989]; for airline crew scheduling in Anbil, Tanga and Johnson [1992], Barnhart et al. [1993] and Vance [1993]; for vehicle routing in Dumas, Desrosiers and Soumis [1989], for graph coloring in Mehrotra and Trick [1993]; and for the binary cutting stock problem in Vance et al. [1994].

Different requirements on subsets

Now consider the situation where the rules on feasible subsets can be described by linear inequalities and where different subsets may have different requirements, i.e., the formulation for P has the block diagonal structure given by (7) and the associated explicit column generation form, with separate convexity constraints for each subset, is given by (8).

In this situation, if we apply the branching rule discussed above 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

$$\sum_{1 \le k \le p_s : y^s_{rk} = 1} \lambda^s_k = 1 \qquad \text{and} \qquad \sum_{1 \le k \le p_s : y^s_{rk} = 1} \lambda^s_k = 0. \tag{14}$$

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 based on the following proposition.

Proposition 2 *Let λ be a feasible solution to the LP-relaxation of (8) and let $z_{ij} = \sum_{1 \le k \le p_j} y^j_{ik} \lambda^j_k$, then z constitutes a feasible solution to the LP-relaxation of (7) and z is integral if and only if λ is integral.*

Proof As z is a convex combination of extremal solutions it is feasible. If λ is integral, then z is integral. Thus z_j is integral for all j. Consider any element i of z_j with $z_{ij} = 1$. Consequently, $\sum_{1 \le k \le p_j} y^j_{ik} \lambda^j_k = z_{ij} = 1$. Since $\sum_{1 \le k \le p_j} \lambda^j_k \le 1$, we must have $y^j_{ik} = 1$ for all $k = 1, ..., p_j$ with $\lambda^j_k > 0$. Therefore, all columns associated with variables with positive values are identical. Since there are no duplicate columns in the basis, there can be only one positive variable, which implies that λ is integral. □

Consequently, the branching strategy given by (14) corresponds precisely to performing standard branching in (7), since

$$\sum_{1\le k\le p_s : y^s_{rk}=1} \lambda^s_k = 1 \Rightarrow \sum_{1\le k\le p_s} y^s_{rk}\lambda^s_k = 1 \Rightarrow z_{rs} = 1$$

and

$$\sum_{1\le k\le p_s : y^s_{rk}=1} \lambda^s_k = 0 \Rightarrow \sum_{1\le k\le p_s} y^s_{rk}\lambda^s_k = 0 \Rightarrow z_{rs} = 0.$$

Furthermore, this branching strategy does not increase the difficulty of solving the pricing problem. In fact, Sol and Savelsbergh [1994] show that any algorithm for the pricing problem used in the root node can also be used in subsequent nodes. To prevent an element from being in a generated solution, we just ignore it altogether.

To force an element to be in the solution, we modify the dual variables such that every solution that does not use the element has a nonpositive reduced cost. Let u be the dual vector associated with the partitioning constraints, v the dual vector associated with the convexity constraints, and w the dual variable associated with constraint

$$\sum_{1\le k\le p_{\hat{j}}} (1 - y^{\hat{j}}_{\hat{i}k})\lambda^{\hat{j}}_k + \sum_{1\le j\ne\hat{j}\le n} \sum_{1\le k\le p_j} y^j_{\hat{i}k}\lambda^j_k \le 0$$

which forces element $\hat{i}$ to be in subset $\hat{j}$.

Proposition 3 *Let (u, v, w) be an optimal dual solution to the extended restricted master problem and let $\eta > 0$, then (u', v', w') with $u' := u - \eta e_{\hat{i}}$, $v' := v + \eta e_{\hat{j}}$, and $w' := w + \eta$, where e_k denotes the kth unit vector, is an alternative optimal dual solution.*

Proof When the extended restricted master problem is solved the reduced costs satisfy

$$\overline{c^j_k} = c^j_k - \sum_i y^j_{ik}u_i - v_j - y^j_{\hat{i}k}w \le 0 \qquad 1 \le j \ne \hat{j} \le n,\ k = 1, ..., p_j,$$

and

$$\overline{c^{\hat{j}}_k} = c^{\hat{j}}_k - \sum_i y^{\hat{j}}_{ik}u_i - v_{\hat{j}} - (1 - y^{\hat{j}}_{\hat{i}k})w \le 0 \qquad k = 1, ..., p_{\hat{j}}.$$

To see that the alternative dual solution is feasible for the case $y^{\hat{j}}_{\hat{i}k} = 1$, we have

$$\overline{c'^j_k} = c^j_k - \sum_i y^j_{ik}u'_i - v'_j - y^j_{\hat{i}k}w' =$$

$$c^j_k - \sum_{i\ne\hat{i}} y^j_{ik}u_i - y^j_{\hat{i}k}(u_{\hat{i}} - \eta) - v_j - y^j_{\hat{i}k}(w + \eta) = \overline{c^j_k} \qquad 1 \le j \ne \hat{j} \le n,\ k = 1, ..., p_j$$

and

$$\overline{c'^{\hat{j}}_k} = c^{\hat{j}}_k - \sum_i y^{\hat{j}}_{ik} u'_i - v'_{\hat{j}} - (1 - y^{\hat{j}}_{\hat{i}k}) w' =$$

$$c^{\hat{j}}_k - \sum_{i \neq \hat{i}} y^{\hat{j}}_{ik} u_i - y^{\hat{j}}_{\hat{i}k}(u_{\hat{i}} - \eta) - (v_{\hat{j}} + \eta) - (1 - y^{\hat{j}}_{\hat{i}k})(w + \eta) = \overline{c^{\hat{j}}_k} \qquad k = 1, ..., p_{\hat{j}}.$$

For the case $y^{\hat{j}}_{\hat{i}k} = 0$, we have

$$\overline{c'^{\hat{j}}_k} = c^{\hat{j}}_k - \sum_i y^{\hat{j}}_{ik} u'_i - v'_{\hat{j}} - (1 - y^{\hat{j}}_{\hat{i}k}) w' =$$

$$c^{\hat{j}}_k - \sum_{i \neq \hat{i}} y^{\hat{j}}_{ik} u_i - y^{\hat{j}}_{\hat{i}k}(u_{\hat{i}} - \eta) - (v_{\hat{j}} + \eta) - (1 - y^{\hat{j}}_{\hat{i}k})(w + \eta) = \overline{c^{\hat{j}}_k} - 2\eta \qquad k = 1, ..., p_{\hat{j}}.$$

It is optimal because $\sum_i u'_i + \sum_j v'_j = \sum_i u_i + \sum_j v_j$. $\square$

Now by Proposition 3, it follows that by choosing η large enough, we can always ensure that $c^{\hat{j}}_k \leq 0$ if $y^{\hat{j}}_{\hat{i}k} = 0$.

Applications of this branching strategy are presented for crew scheduling in Vance [1993]; for generalized assignment in Savelsbergh [1993]; for multi-commodity flow in Barnhart, Hane, Johnson and Sigismondi [1991] and Parker and Ryan [1993]; for vehicle routing in Desrosiers, Soumis and Desrochers [1984] and Desrochers, Desrosiers and Solomon [1992]; and for pickup and delivery problems in Sol and Savelsbergh [1994].

4.2 General mixed integer master problems

So far, we have discussed branching strategies for set partitioning master problems. A branching strategy for general mixed integer master problems with different requirements on subsets can be derived directly from (3) as follows [Johnson 1989]. The optimal solution to the linear programming relaxation is infeasible if and only if

$$\sum_{1 \leq k \leq p_j} y^j_k \lambda^j_k$$

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

$$\sum_{1 \leq k \leq p_j} y^j_{rk} \lambda^j_k \leq \lfloor \alpha \rfloor$$

and on the other branch we require

$$\sum_{1 \le k \le p_j} y^j_{rk} \lambda^j_k \ge \lceil \alpha \rceil.$$

This implies that when a new extreme point is generated an upper bound of $\lfloor \alpha \rfloor$ on component r has to be enforced on the first branch and a lower bound of $\lceil \alpha \rceil$ on the second branch. Note that each pricing problem differs only in the lower and upper bounds on the components.

General models with identical restrictions on subsets
Developing a branching strategy for general mixed integer master problems with identical restrictions on subsets is more complex. If the solution to (2) is fractional, we may be able to identify a single row r and an integer α_r such that

$$\sum_{k:(Ay_k)_r \ge \alpha_r} \lambda_k = \beta_r$$

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

$$\sum_{k:(Ay_k)_r \ge \alpha_r} \lambda_k \le \lfloor \beta_r \rfloor \text{ and } \sum_{k:(Ay_k)_r \ge \alpha_r} \lambda_k \ge \lceil \beta_r \rceil.$$

These constraints place upper and lower bounds on the number of columns with $(Ay_k)_r \ge \alpha_r$ that can be present in the solution. In general, these constraints will not 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 $(Ay_k)_r \ge \alpha_r$. 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, $\sum_{k:y_{kr} \ge 1} \lambda_k = 1$ for every row. Thus we may have to branch on multiple rows.

Assume that

$$\sum_{k:(Ay_k)_r \ge \alpha_r} \lambda_k = \beta_r$$

and β_r is integer for every row r and integer α_r. Pick an arbitrary row, say r, and branch on the constraints

$$\sum_{k:(Ay_k)_r \ge \alpha_r} \lambda_k \le \beta_r - 1 \text{ and } \sum_{k:(Ay_k)_r \ge \alpha_r} \lambda_k \ge \beta_r.$$

Because, the current fractional solution is still feasible in the latter branch, we look for a row s such that

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

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

$$\sum_{k:(Ay_k)_r \geq \alpha_r \wedge (Ay_k)_s \geq \alpha_s} \lambda_k \leq \lfloor \beta_s \rfloor \text{ and } \sum_{k:(Ay_k)_r \geq \alpha_r \wedge (Ay_k)_s \geq \alpha_s} \lambda_k \geq \lceil \beta_s \rceil.$$

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 and that a set of l rows gives rise to $l+1$ branches.

The branching scheme presented above 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, that the first branch defined by

$$\sum_{k:y_{rk} \geq 1} \lambda_k \leq 0$$

is empty, and that the other two branches defined by

$$\sum_{k:y_{rk} \geq 1 \wedge y_{sk} \geq 1} \lambda_k \leq 0 \text{ and } \sum_{k:y_{rk} \geq 1 \wedge y_{sk} \geq 1} \lambda_k \geq 1$$

are exactly those defined by the branching scheme of Ryan and Foster.

5 LP solution

The computationally most intensive component of a branch-and-price algorithm is the solution of the linear programs. Therefore, we have to concentrate on solving 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 columns.

5.1 Key Formulations

Again consider the master problem given by (8), but with an equality convexity constraint.

$$\begin{aligned} \max \sum_{1\le j\le n}\sum_{1\le k\le p_j} (c_j y_k^j)\lambda_k^j & & \\ \sum_{1\le j\le n}\sum_{1\le k\le p_j} y_{ik}^j \lambda_{ik}^j &= 1, \quad i=1,...,m, \\ \sum_{1\le k\le p_j} \lambda_k^j &= 1, \quad j=1,...,n, \\ \lambda_k^j \in \{0,1\}, & \quad j=1,...,n,\ \ k=1,...,p_j. \end{aligned} \tag{15}$$

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 problem structure of (15) 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 a *key* extreme point solution $*_j$ and substituting $\lambda_{*_j} = 1 - \sum_{1\le k\le p_j, k\ne *_j} \lambda_k^j$ for each subproblem j. With this substitution, the key formulation of (15) becomes

$$\begin{aligned} \max \sum_{1\le j\le n}\sum_{1\le k\le p_j} (c_j(y_k^j - y_{*_j}))\lambda_k^j &+ \sum_{1\le j\le n} c_j y_{*_j} \\ \sum_{1\le j\le n}\sum_{1\le k\le p_j, k\ne *_j} (y_{ik}^j - y_{i*_j})\lambda_k^j &= 1 - \sum_{1\le j\le n} y_{i*_j} \quad i=1,...m, \\ \sum_{1\le k\le p_j, k\ne *_j} \lambda_k^j &\le 1 \quad j=1,\ldots,n, \\ \lambda_k^j &\in \{0,1\} \quad j=1,\ldots,n,\ \ k=1,\ldots,p_j. \end{aligned} \tag{16}$$

The variable λ_k^j indicates whether the key extreme point $*_j$ 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 λ_{*_j}, the *key nonnegativity* constraints $\sum_{1\le k\le p_j, k\ne *_j} \lambda_k^j \le 1$ 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.

To illustrate, consider the multi-commodity network flow problem. In a column generation formulation of the multi-commodity 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. [1991].)

Although the Dantzig-Van Slyke and Rosen procedures will 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 multi-commodity flow problem again. In the key formulation for the multi-commodity 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 multi-commodity 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 the simple cycles not already present in 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. As will be explained later, the LP relaxations of the original key formulation and this alternate simple cycle formulation have equal optimal objective function values.

The idea presented above for the multi-commodity flow problem generalizes to any problem with the master program structure of (8). As before, the key formulation is obtained by selecting a key column $*_j$ 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 $*_j$;

-1 for each element in k and not in $_j$;

0 for each element in both or neither k and $*_j$; 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 $*_j$ may be removed from the solution and replaced by new elements in column k. Similar to the concept of simple cycles in the multi-commodity flow context, a column in the key formulation that cannot be represented as the sum of other columns is called an *elementary exchange.* In a column generation procedure, if recognizing elementary exchanges is difficult, a column generated by the pricing problem can be added as is. This will result in additional columns but will not affect the validity of the formulation.

To satisfy the key column nonnegativity constraints, we can add a constraint for each element in a key column that ensures that at most one elementary exchange contains it, i.e.,

$$\sum_{k:\, i \in k} \lambda_k^j \le 1 \qquad i \in *_j, j = 1, ..., n. \tag{17}$$

For example, if the key column contains elements 1,2,3 and exchanges 1 and 2 remove element 1, exchanges 1 and 3 remove element 2, and exchanges 2 and 3 remove element 3, then (17) yields

$$\begin{array}{llll} \lambda_1^j + & \lambda_2^j & & \le 1 \\ \lambda_1^j + & & \lambda_3^j & \le 1 \\ & \lambda_2^j + & \lambda_3^j & \le 1. \end{array}$$

While the key formulation (16) is equivalent to the elementary exchange formulation, the linear programming relaxation of the elementary exchange formulation may be weaker. To see this, note that in the above example $\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{2}$ is feasible to (17) but would be excluded by the nonnegativity constraints of (16). However, in the case of multi-commoddity flows, the constraints of (17) are totally unimodular and the linear programming relaxations give the same bound. To see this, observe that any simple cycle meets a set of consecutive arcs on a key path and therefore the rows of (17) can be organized to have the consecutive ones property.

Fractional solutions to the elementary exchange formulation can be removed by the addition of cutting planes derived from the node packing polytope or by branching. The computational advantage of working with a key formulation based on elementary exchanges can be substantial, as demonstrated by Vance [1993].

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 finds the 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:

1. Determine an initial feasible restricted master problem.

2. Initialize the column pool to be empty.

3. Solve the current restricted master problem.

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

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

6. Empty the column pool.

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.

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.

9. Stop.

A very fast and promising approach to generate columns with positive reduced costs is to use improvement algorithms that take existing columns with reduced cost equal to zero (at least all basic columns satisfy this requirement) and try to construct columns with a positive reduced cost by performing some simple changes [Sol and Savelsbergh 1994].

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 Alternative bounds

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 substantially 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 nonoptimal linear programming solutions.

In some situations prematurely ending the column generation process will not even affect the size of the branch-and-bound tree. Suppose we have an upper bound on the optimal value of the unrestricted master LP. If the objective function value of the integer program is known to be integer, column generation can be stopped once the optimal value of the restricted master problem exceeds the round down of this upper bound.

Lasdon [1970] and Farley [1990] describe simple and 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. Vance et al. [1993] used Farley's observation to prematurely end the column generation process in the root node in their algorithm for the cutting stock problem. At nodes deeper in the tree, the LP bound of the parent can be used as an upper bound.

In many situations, the pricing problem is extremely hard to solve and it is only feasible, computationally, to solve it approximately. Obviously, in that situation, we cannot guarantee optimality of the current LP solution when we cannot identify any profitable columns, but we can branch anyway.

5.4 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 their incompatibility. That is, the pricing (separation) problem can be much harder after additional rows (columns) are added, because the new rows (columns) can destroy the structure of the pricing (separation) problem.

One remedy is to dualize the additional constraints using Lagrangian relaxation. Another is to do the pricing only over the original rows, i.e., assuming that the new columns have 0 coefficients in the additional rows. But then it may be necessary to update the columns coefficients over 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. Then 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.

The most successful optimization algorithms for the traveling salesman problem use branch-and-cut, see Junger, Reinelt and Rinaldi [1994] for a recent survey. However, these algorithms only maintain columns for a small subset of the edges. Consequently, 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 a true lower bound has been found. If edges with favorable reduced costs are identified, they are added to the reduced edge set and the process is repeated.

Nemhauser and Park [1991] combine column and row generation in an LP based algorithm for the edge coloring problem. No branching in the master problem is required on the instances they solve. 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.

6 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 three 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, Savelsbergh, and Sigismondi 1994, Savelsbergh and Nemhauser 1993] may reduce the implementation efforts even further. MINTO (Mixed INTeger Optimizer) is based on the belief that to solve large mixed-integer programs efficiently, without having to develop a full-blown special purpose code in each case, you need 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.

Figures 2 and 3 give the basic flow of control in MINTO and the associated application functions. A call to an application function temporarily transfers control to the application program, which can either accept control or decline control. If control is accepted, the application program performs the associated task. If control is declined, MINTO performs a default action, which in many cases will be "do nothing". To differentiate between actions carried out by the system and those carried out by the application program, there are different "boxes". System actions are in solid line boxes and application program actions are in dashed line boxes. A solid line box with a dashed line box enclosed is used whenever an action can be performed by both the system and the application program. Finally, to indicate that an action has to be performed by either

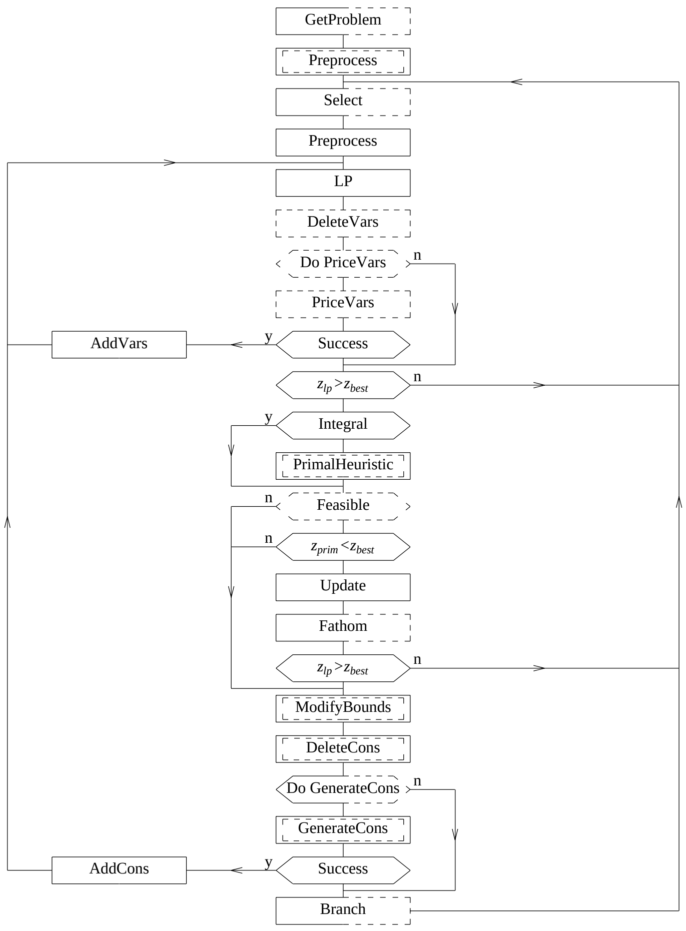

Figure 2: The underlying algorithm

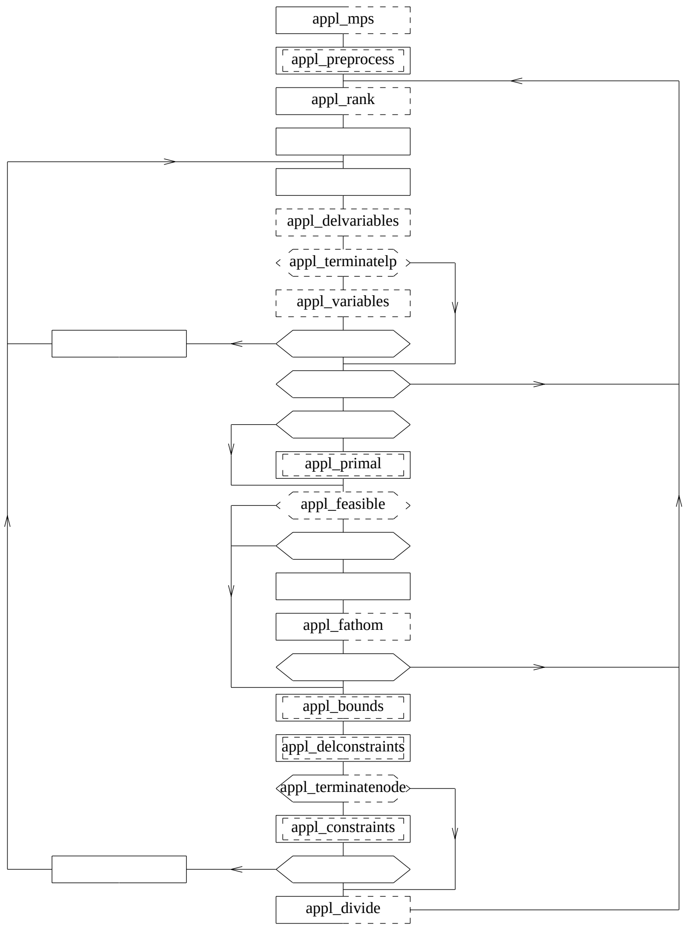

Figure 3: The application functions

the system or the application program, but not both, a box with one half in solid lines and the other half in dashed lines is used. If an application program does not carry out an action, but one is required, the system falls back to a default action. For instance, if an application program does not provide a division scheme for the branching task, the system will apply the default branching scheme.

The flow-chart shows two main loops: an inner loop to allow column generation and an outer loop to allow row generation. Column management can be done using the functions **appl_delvars, appl_vars, appl_terminatelp** and row management can be done using the functions **appl_delcons, appl_cons, appl_terminatenode**. Developing a special branching scheme can be done using the function **appl_divide**.

To start the column generation scheme, an initial restricted master problem has to be provided by **appl_mps**. This initial restricted master problem must have a feasible LP relaxation to ensure that proper dual information is passed to the pricing problem (**appl_vars**). 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: either by adding a single artificial variable with a large negative cost and associated column consisting of all ones or by adding 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. The artificial variables are kept at all nodes of the branch-and-bound tree for the same reason.

7 Computational Experience

7.1 The Binary Cutting Stock Problem

Vance et al. [1993] solve binary cutting stock problems using a branch-and-price algorithm. Columns are generated by solving a binary knapsack problem with the optimal dual prices from the rows of the master problem as the prices on the items. The branching rule is identical to the Ryan and Foster rule presented earlier. In cutting stock terms, this rule requires two items to be contained in the same pattern on one branch and different patterns on the other branch. On the branch where the two items must be in the same pattern, the resulting knapsack pricing problem has a new super item replacing those two items. Thus, on this branch, the column generation problem is a knapsack problem with one fewer item. On the other branch a constraint is added to the knapsack problem that allows at most one of the items to be chosen. While this pricing problem is somewhat more difficult than a knapsack problem, it can still be solved quickly if there are not too many additional constraints.

Computational results are reported for randomly generated test problems. The branch-and-price algorithm was able to solve in seconds problems that could not be solved using a standard branch-and-bound procedure on an explicit formulation. Standard branch-and-bound procedures were unable to solve problems with more than 70 items. The largest problems solved using the branch-and-price algorithm had 500 items and they were solved in less than an hour on an IBM RS/6000.

7.2 The Generalized Assignment Problem

Savelsbergh [1993] develops a branch-and-price algorithm for the generalized assignment problem discussed in Section 2. The pricing problem is given by

$$\max_{1 \le j \le n} \{z(KP_j) - v_j\},$$

where v_j is the optimal dual price from the solution to the restricted master problem associated with the convexity constraint of machine j and $z(KP_j)$ is the value of the optimal solution to the knapsack problem

$$\max \sum_{1 \le i \le m} (p_{ij} - u_i) y_i^j$$

subject to

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

$$y_i^j \in \{0, 1\} \qquad j = 1, ..., n$$

with u_i being the optimal dual price from the solution to the restricted master problem associated with the partitioning constraint of task i. A column prices out to enter the basis if its reduced cost is positive. Consequently, if the objective value of the pricing problem is less than or equal to zero, then the current optimal solution for the restricted master problem is also optimal for the (unrestricted) master problem. The branching rule described in Section 4 for master problems with several convexity rows is used. This rule assigns task i to machine j on one branch and forbids machine j to perform task i on the other. In both cases, the size of the pricing problem for machine j is reduced by one task. Furthermore, on the branch where task i must be performed by machine j, task i may also be deleted from the pricing problem for each of the other machines.

Computational results indicate that the branch-and-price algorithm clearly outperforms existing algorithms and is able to solve much larger instances. In one of the computational experiments the average number of nodes required by the branch-and-price algorithm was compared with the average number of nodes required by the dual ascent algorithm of Guignard and Rosenwein [1989] for ten randomly generated instances in four different problem classes. The results are given in Table 1. Although in theory both algorithms use the same bounds, the branch-and-price algorithm clearly does better. Possible explanations for this phenomenon have been discussed in Section 3.

Table 1: Dual ascent versus branch-and-price

	Dual ascent	branch-and-price
problem class	#nodes	#nodes
A,5,50	7.	1.0
B,10,50	101.	1.6
C,10,50	156.	5.7
D,5,30	102.	40.0

7.3 The Urban Transit Crew Scheduling Problem

Desrochers and Soumis [1989] use a branch-and-price algorithm to solve the urban transit crew scheduling problem (UTCS). An instance of UTCS is defined by a bus schedule and the collective agreement between the drivers and management. The schedule defines a set of tasks that must be performed and the collective agreement places restrictions on feasible workdays for the drivers and dictates the cost of those workdays. The agreement may also place global restrictions on the types of workdays included in the schedule. The master problem is a set covering problem with additional constraints. There is a set covering constraint for each task to be performed, and columns representing feasible workdays for a bus driver. The set covering constraints ensure that at least one driver is assigned to each task. The additional constraints enforce any global restrictions on the characteristics of the final solution. For example, the number of workdays in the solution whose total elapsed time is less than a given threshold may be limited to a certain percentage of the total number of workdays. Columns are generated by solving a constrained shortest path problem on a specially constructed network. The branching rule is similar to the Ryan and Foster rule presented earlier except that instead of branching on whether two tasks are executed in the same workday, they branch on whether two tasks are executed *consecutively* in the same workday. This rule is more easily enforced in the constrained shortest path procedure than the more general rule.

The authors present computational results for two real-world problems. In both cases, the branch-and-price algorithm constructed solutions with lower cost than the best known solutions.

7.4 The Bandwidth Packing Problem

The bandwidth packing problem is to decide which calls on a list of requests should be chosen to route on a capacitated network. An example is the routing of video data for teleconferencing within a private network. The objective is to minimize the costs of routing the selected calls plus the revenue lost from unrouted calls. Parker and Ryan [1994] formulate this problem as an integer program as follows.

Let P_i denote a set of feasible paths for call i. Let r_i denote the revenue of call i, d_i the demand of call i in units of bandwidth, u_e denote the capacity of link e in bandwidth, and c_e denote the usage cost of transmitting one bandwidth on link e. Let L be the set of all links. There are two types of variables

$$x_{ij} = \begin{cases} 1 & \text{if call } i \text{ uses path } j \\ 0 & \text{otherwise} \end{cases}$$

and

$$y_i = \begin{cases} 1 & \text{if call } i \text{ is not routed} \\ 0 & \text{otherwise} \end{cases}.$$

Let δ_{ej} be the indicator that is 1 if link e is in path j and 0 otherwise and assume that there are n calls to be routed. The problem formulation is

$$\min \sum_{e \in L} c_e \sum_{i=1}^{n} \sum_{j \in P_i} \delta_{ej} d_i x_{ij} + \sum_{i=1}^{n} r_i y_i$$

subject to

$$\sum_{j \in P_i} x_{ij} + y_i = 1 \text{ for each call } i$$

$$\sum_{i=1}^{n} \sum_{j \in P_i} \delta_{ej} d_i x_{ij} \leq u_e \text{ for each link } e.$$

The first set of constraints ensures that each call is either routed or not, and the second set ensures the satisfaction of link capacities. The solution method uses column generation to solve the LP relaxations within a branch and bound scheme. The LP relaxations are similar to multi-commodity network flow problems and are solved using standard column generation solution techniques. To obtain integer solutions, Parker and Ryan use a hybrid branching strategy. Specifically, they first create one branch setting

x_{ij} to 1. This rule is easy to enforce in the pricing subproblem by deleting call i from the problem, and removing d_i from the capacity of all links on path j. Since forcing x_{ij} to 0 is difficult to enforce in the pricing subproblem, $x_{ij} = 0$ is satisfied by creating several branches, one for each link of path j. If the links of path j are $e_1, e_2, \ldots, e_k$, k new branches are created. At the ℓth branch, they delete the column corresponding to x_{ij}, and any other column in which call i uses link e_ℓ. The pricing subproblem is prevented from generating any of the deleted paths by removing link e_ℓ from the network.

Parker and Ryan tested their algorithm on 14 problems ranging in size from 14 to 30 nodes and 23 to 93 calls. They report running times to find optimal solutions from 8 seconds to over 8 hours on a VAX 8800, and conclude that the algorithm is a practical procedure for solving a class of real world problem. Further investigations are proposed to use cutting planes at selected nodes in the branch and bound tree to reduce the computational effort.

References

R. Anbil, R. Tanga, and E.L. Johnson (1992). A global approach to crew-pairing optimization. *IBM Systems Journal 31*, 71-78.

L.H. Appelgren (1969). A column generation algorithm for a ship scheduling problem. *Transportation Science 3*, 53-68.

C. Barnhart, C.A. Hane, E.L. Johnson, and G. Sigismondi (1991). *An Alternative Formulation and Solution Strategy for Multi-Commodity Network Flow Problems.* Report COC-9102, Georgia Institute of Technology, Atlanta, Georgia.

E. Balas and M. Padberg (1976). Set Partitioning: A Survey. *SIAM Review 18*, 710-760.

C. Barnhart, E.L. Johnson, R. Anbil, and L. Hatay (1994). A column generation technique for the long-haul crew assignment problem. T. Ciriano and R. Leachman (eds.). *Optimization in Industry, Volume II*, John Wiley and Son, to appear.

R.E. Bixby, 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. *Operations Research 40*, 885-897.

R. Brooks and A. Geoffrion (1966). Finding Everett's Lagrange multipliers by linear programming. *Operations Research 14*, 1149-1153.

CPLEX Optimization, Inc. (1990). *Using the* CPLEX$^{\text{TM}}$ *Linear Optimizer.*

G.B. Dantzig and R.M. Van Slyke (1967). Generalized Upper Bounding Techniques. *Journal Computer System Sci. 1*, 213-226.

M. Desrochers, J. Desrosiers, and M. Solomon (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research 40*, 342-354.

M. Desrochers and F. Soumis (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science 23*, 1-13.

J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis (1994). Time constrained routing and scheduling. M.E. Ball, T.L Magnanti, C. Monma, and G.L. Nemhauser (eds.). *Handbooks in Operations Research and Management Science, Volume on Networks*, to appear.

J. Desrosiers, F. Soumis, and M. Desrochers (1984). Routing with time windows by column generation. *Networks 14*, 545-565.

Y. Dumas, J. Desrosiers, and F. Soumis (1991). The pickup and delivery problem with time windows. *European Journal of Operations Research 54*, 7-22.

A.A. Farley (1990). A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research 38*, 922-924.

A.M. Geoffrion (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Studies 2*, 82-114.

M. Guignard and M. Rosenwein (1989). An improved dual-based algorithm for the generalized assignment problem. *Operations Research 37*, 658-663.

M. Held and R.M. Karp (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research 18*, 1138-1162.

M. Held and R.M. Karp (1971). The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming 1*, 6-25.

M. Held, P. Wolfe, and H.P. Crowder (1974). Validation of subgradient optimization. *Mathematical Programming 6*, 62-88.

A.J. Hoffman, A. Kolen, and M. Sakarovitch (1985). Totally balanced and greedy matrices. *SIAM Journal on Algebraic and Discrete Methods 6*, 721-730.

K. Hoffman and M. Padberg (1985). LP-based combinatorial problem solving. *Annals of Operations Research 4*, 145-194.

IBM Corporation (1990). *Optimization Subroutine Library, Guide and Reference.*

E.L. Johnson (1989). Modeling and strong linear programs for mixed integer programming. S.W. Wallace (ed.). *Algorithms and Model Formulations in Mathematical*

Programming, NATO ASI Series 51, 1-41.

E.L. Johnson, A. Mehrotra, and G.L. Nemhauser (1993). Min-cut clustering. *Mathematical Programming 62*, 133-152.

M. Junger, G. Reinelt, and G. Rinaldi (1994). The traveling salesman problem. M.E. Ball, T.L Magnanti, C. Monma, and G.L. Nemhauser (eds.). *Handbooks in Operations Research and Management Science, Volume on Networks*, to appear.

L.S. Lasdon (1970). *Optimization Theory for Large Systems.* MacMillan, New York.

O. Marcotte (1985). The cutting stock problem and integer rounding. *Mathematical Programming 33*, 82-92.

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

A. Mehrotra and M.A. Trick (1993). *A Column Generation Approach to Graph Coloring.* Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

G.L. Nemhauser and S. Park (1991). A polyhedral approach to edge coloring. *Operations Research Letters 10*, 315-322.

G.L. Nemhauser and L.A. Wolsey (1988). *Integer and Combinatorial Optimization.* Wiley, Chichester.

G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi (1994). MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, to appear.

M. Parker and J. Ryan (1994). A column generation algorithm for bandwidth packing. *Telecommunications Systems*, to appear.

C. Ribeiro, M. Minoux, and M. Penna (1989). An optimal column generation with ranking algorithm for very large set partitioning problems in traffic assignment. *European Journal of Operations Research 41*, 232-239.

J.B. Rosen (1964) Primal Partition Programming for Block Diagonal Matrices. *Numerische Mathematik 6*, 250-260.

D.M. Ryan and B.A.Foster (1981). An integer programming approach to scheduling. A. Wren (ed.) *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, North-Holland, Amsterdam, 269-280.

M.W.P. Savelsbergh (1993). *A Branch-and-Price Algorithm for the Generalized Assignment Problem.* Report COC-9302, Georgia Institute of Technology, Atlanta, Georgia.

M.W.P. SAVELSBERGH AND G.L. NEMHAUSER (1993). *Functional description of MINTO, a Mixed INTeger Optimizer.* Report COC-91-03A, Georgia Institute of Technology.

M. SOL AND M.W.P. SAVELSBERGH (1994). *A Branch-and-Price Algorithm for the Pickup and Delivery Problem.* In preparation.

P.H. VANCE (1993). *Crew Scheduling, Cutting Stock, and Column Generation: Solving Huge Integer Programs.* PhD dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

P.H. VANCE, 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*, to appear.

A. WREN, B.M. SMITH, AND A.J. MILLER (1985). Complementary approaches to crew scheduling. J.-M. ROUSSEAU (ed.), *Computer Scheduling of Public Transport 2*, North-Holland, Amsterdam, 263-278.