Explicit-Constraint Branching for Solving Mixed-Integer Programs

(Appears in *Computing Tools for Modeling, Optimization and Simulation*, M. Laguna and J.L. González-Velarde, Kluwer Academic Publishers, Boston, 2000.)

Jeffrey A. Appleget[1] and R. Kevin Wood[2]
[1] *TRADOC Analysis Center, Monterey, CA 93943*
[2] *Operations Research Department, Naval Postgraduate School, Monterey, CA 93943*

Key words: integer programming, branch and bound

Abstract: This paper develops a new generalized-branching technique called "explicit-constraint branching" (ECB) to improve the performance of branch-and-bound algorithms for solving mixed-integer programs (MIPs). ECB adds structure to a MIP, in the form of auxiliary constraints and auxiliary integer variables, to allow branching on groups of (original) integer variables that would not otherwise be possible. Computational tests on three sets of real-world MIPs demonstrate that ECB often improves solution times over standard branch and bound, sometimes dramatically.

1. INTRODUCTION

This paper develops a simple new technique called "explicit-constraint branching" (ECB) to improve the performance of branch-and-bound algorithms for solving certain mixed-integer programs (MIPs). The technique can be classified as a type of "generalized branching" (e.g., Sol 1994, Jörnsten and Larsson 1988) or "constraint branching" (Foster and Ryan 1981). It is neither a specialization nor generalization of the branching techniques based on "special ordered sets" (Beale and Tomlin 1970). We believe that ECB is a tool that should be in every integer programmer's toolkit.

Standard variable-based branch and bound solves the linear-programming (LP) relaxation of a MIP and partitions the MIP's feasible region by branching on individual integer variables. As an example,

suppose we are solving a MIP for solution vector (**x***, **v***) where **x** and **v** represent integer and continuous variables, respectively. We have just obtained a solution $(\mathbf{x}^*, \mathbf{v}^*)$ to the MIP's LP relaxation, and some variable $\hat{x}_j$ is fractional, i.e., $m < \hat{x}_j < m + 1$ for some integer m. The standard branching (partitioning) choices are

$$x_j \le m \text{ or } x_j \ge m + 1. \tag{1}$$

Branching on individual variables is attractive since its implementation is quite simple within the bounded-variable simplex algorithm that is normally embedded in a MIP solver. Empirically, however, this type of branching can create an unbalanced enumeration tree, which can lead to excessive enumeration (Foster and Ryan 1981). We propose another branching technique that has the potential to yield significant computational improvements for some MIPs.

Suppose that we are solving a generic MIP with integer variables $x_j \ge 0$ for $j \in J$. Let $J' \subseteq J$ be an arbitrary, nonempty subset of J and define integer coefficients α_j for each x_j, $j \in J'$. Then, $\Sigma_{j \in J'} \alpha_j x_j$ must be integer in any solution to the MIP, and a valid partition of the MIP's feasible region is derived from

$$\sum_{j \in J'} \alpha_j x_j \le m \text{ or } \sum_{j \in J'} \alpha_j x_j \ge m+1, \tag{2}$$

for any integer m. We call this partitioning scheme "constraint branching" although, strictly speaking, it is a generalization of constraint branching as defined by Foster and Ryan (1981).

Previous applications of constraint branching, to problems with special structure, have proven to be highly effective. Such applications, with one exception to be mentioned later, do not require that explicit constraints of the form (2) be added. Examples include the special ordered sets (SOS) of Beale and Tomlin (1970) and the techniques of Foster and Ryan (1981). In a simplified fashion, we illustrate the constraint branching of Foster and Ryan on the set-partitioning problem:

$$(\text{SPP}) \qquad \min \quad \sum_{j \in J} c_j x_j$$

$$\text{s.t.} \quad \sum_{j \in J_i} x_j = 1 \,\forall\, i \in I \tag{3}$$

$$x_j \in \{0,1\} \,\forall\, j \in J$$

where $J_i \subset J$ for all i.

Suppose we are at the initial node of the branch-and-bound tree, have just solved the LP relaxation of SPP for $\hat{\mathbf{x}}$, and $\hat{x}_j$ is fractional for some j. Select subset index f such that $j \in J_f$ (there are at least two fractional variables in J_f), and partition J_f into two disjoint subsets, J_f^1 and J_f^2, each containing one or more fractional variables. Then, a valid partition of the SPP's feasible region is

$$\sum_{j \in J_f^1} x_j \le 0 \;\text{ or }\; \sum_{j \in J_f^1} x_j \ge 1. \tag{4}$$

Those restrictions are equivalent to

$$\sum_{j \in J_f^1} x_j \le 0 \;\text{ or }\; \sum_{j \in J_f^2} x_j \le 0, \tag{5}$$

because $\Sigma_{j \in Jf} x_j = 1$. Clearly, we can perform this type of branching recursively on the subsets, and explicit constraints of the form (4) or (5) are not needed: Simply fix variables in J_f^1 or J_f^2 to 0. We refer to this special case of constraint branching as "implicit-constraint branching" since it can be accomplished without adding explicit constraints like (4).

Implicit-constraint branching has been shown to be computationally effective through the technique just described (Foster and Ryan 1981) and through the techniques of special ordered sets (e.g., Hummeltenberg 1984, Escudero 1988). In fact, the "SOS-Type-3" (S3) constraint-branching of Escudero (1987, 1988), which is based on SOS Type 1 (S1) as defined by Beale and Tomlin (1970), is quite similar to our set-partitioning example, except: In SOS branching, the subsets of any J_i are created in a more rigid fashion that depends on a predefined ordering of the variables. (Escudero defines an S3 set to mean that all variables in the set are binary and are part

of a set-partitioning constraint. See the discussion on SOS in the next section.)

Standard, variable-based branch and bound has difficulties with the SPP because it sets individual variables to 0 or to 1. Setting a variable x_j, $j \in J_i$, to 0 can be a weak decision because all but one of the variables (there may be thousands) in constraint i will be 0 in the final solution anyway; there are typically many other variables in the constraint that can, collectively or individually, replace the fractional value $\hat{x}_j$ and achieve nearly the same objective function value as when $x_j = 0$. Setting x_j to 1 is a very strong decision because it forces all other variables in J_i to 0. In fact, it forces $x_{j'} = 0$ for all $j' \in \bigcup_{i':\, j \in J_{i'}} J_{i'} - \{j\}$, and often has a significant effect on the objective function value. An unbalanced enumeration tree results in such a problem because a "1-branch" eliminates a very large number of possible solutions from consideration further down the tree, while a "0-branch" eliminates only a few (Foster and Ryan 1981).

Constraint branching described above moderates the strength of branching decisions. Branching from a node in the enumeration tree restricts all the variables in one subset to 0, and allows any combination of the variables in the other subset to sum to 1. As the branching progresses, the strong decision of setting a particular variable to 1 is deferred, while unpromising variables are culled from the unrestricted "sum-to-1" subset and placed in the new "set-to-0" subset. As a result, the enumeration tree is better balanced and often smaller than the standard tree (Foster and Ryan 1981).

Implicit-constraint branching is useful but it has one severe drawback: Special problem structure is required. It is natural to prefer implicit-constraint branching to explicitly adding constraints of the form (2) because of the computational overhead involved. But, when structure is lacking, constraint branching with explicitly added constraints can reduce enumeration significantly, and the overhead can be modest, especially with today's efficient LP solvers. We demonstrate explicit-constraint branching in the next section.

2. EXPLICIT-CONSTRAINT BRANCHING

"Explicit-constraint branching" (ECB) allows the benefits of constraint branching for MIPs lacking the special structure required of known implicit-constraint branching techniques. ECB using (2) is set up by (a) defining subsets J_k of the index set J of integer variables, (b) defining integer coefficients α_{kj} for each k and each for $j \in J_k$, (c) defining general integer "branching variables" y_k for each k, and (d) adding ECB constraints:

$$\sum_{j \in J_k} \alpha_{kj} x_j - y_k = 0 \; \forall \; k \; . \tag{6}$$

(Note: Additional information may imply tight bounds on the y_k and such bounds should be included; in some instances, the y_k may even be binary.) Constraint branching as in (2) is then performed by standard variable branching on the variables y_k. Our implementations form ECB constraints before solving the MIP's LP relaxation ("static ECB"), or after solving the LP relaxation, but before the start of branching ("semi-dynamic ECB"). We have not implemented a completely dynamic version of ECB that would add (and delete) constraints within the enumeration tree. The following example shows how ECB can exponentially reduce the enumeration required to solve a simple MIP.

Consider the following MIP, a knapsack problem with $\binom{2n}{n}$ alternate optimal solutions:

$$(\text{P1}) \qquad \max \quad \sum_{j=1}^{2n} 3x_j \tag{7}$$

$$\text{s.t.} \quad \sum_{j=1}^{2n} 2x_j \le 2n+1 \tag{8}$$

$$x_j \in \{0,1\}, \; j = 1,\ldots,2n \, . \tag{9}$$

The LP relaxation of P1 has extreme point solutions with n variables equal to 1, one variable equal to 0.5, and $n - 1$ variables equal to 0. The objective value of the LP relaxation is $3n + 1.5$, which can be rounded down to $3n + 1$ since the data are integer. Variable-based branch and bound forms a partition based on the one fractional variable, designated x_f here, and branches using the restrictions "$x_f = 0$ or $x_f = 1$." Each subsequent LP relaxation solved at a node of the enumeration tree has one of the remaining unfixed variables fractional, until n variables are fixed at 0 and an integer solution is obtained. This solution is optimal and is obtained quickly, but it takes much more work to prove that it is optimal: The upper bound from an LP relaxation (of a restriction) does not change from $3n + 1$ to the tight

value of $3n$ until n variables are fixed to 0. Such a solution is integral anyway so, essentially, we cannot use the LP bound to trim the enumeration tree at all. Consequently, all $\binom{2n}{n}$ alternate optimal solutions must be enumerated before the first solution is proven optimal. Of course, we can solve this problem trivially by tightening the right-hand side to $2n$, or by adding a cut $\sum_{j=1}^{2n} x_j \le n$. But, excessive enumeration like this can arise in problems where tightening right-hand sides is cumbersome or has little effect, where useful cuts are difficult to find, and where the bound provided by the LP relaxation changes often (by small amounts), rather than rarely, as in the example.

We employ ECB on P1 by adding a single ECB constraint

$$\sum_{j \in J} x_j - y = 0, \tag{10}$$

where $y \ge 0$ and y is integer. The LP relaxation solves as before for $\hat{\mathbf{x}}$, and the corresponding value for the branching variable y is $\hat{y} = n + 0.5$. If we partition the IP's feasible region based on y, rather than on a fractional x_f — this is accomplished within a branch-and-bound solver by setting the "branching priority" higher for y than for the x_j (e.g., Brooke *et al.* 1992, pp. 281-283) — we derive the restrictions "$y \le n$ or $y \ge n + 1$." The second restriction is infeasible, and the first yields an LP relaxation with integer-optimal extreme points. Thus, ECB with variable-based branch and bound enumerates at most three LPs to solve this problem.

We note that the above example cannot be interpreted as an SOS technique: (a) Type 1 SOS requires that exactly one variable in a specified subset be positive (all others are 0) (Tomlin 1970), (b) Type 2 requires that exactly two variables in a subset are positive (all others are 0) and those two variables must be adjacent in some ordering of the subset (Tomlin 1970), and (c) Type 3 is essentially the same as Type 1 except that all variables are binary and the variables are automatically identified as part of a set-partitioning constraint (Escudero 1988) or a constraint that is equivalent to a set-partitioning constraint through variable reflection (e.g., GAMS 1996). (These definitions vary in practice; for example, "exactly" may be replaced by "at most," and "positive" may be replaced by "1.") One special case of ECB and one variant of SOS are related, however, and we explore the similarities and differences in Section 3.3.

It should also be noted that implicit-constraint branching for the SPP, as described in the previous section, is "complete" in the sense that every feasible solution can be enumerated by the partitioning scheme, and in

theory, the method must converge to an optimal solution. However, we typically intend to add only a modest number of ECB constraints to help solve a problem, and thus our version of ECB is usually "incomplete," i.e., standard branching on the original integer variables of the problem is usually necessary for convergence. A dynamic version of ECB could be made complete, but this is beyond the scope of the current research.

3. ILLUSTRATIVE EXAMPLES

Here we demonstrate the application of ECB to three real-world MIPs and provide computational results. In each case, we exploit some knowledge of the model to formulate potentially useful ECB constructs. All three problems are also candidates for SOS branching, although only one instance is related to ECB. We delay discussion of SOS to Section 3.3.

3.1 ECB for the Set-Partitioning Problem

The requirement that $\Sigma_{j\in J}x_j$ be an integer quantity in any solution to the SPP is a less coercive requirement than is "each x_j must be 0 or 1." We can implement the former requirement, just as in P1, by defining a general integer variable $y \geq 0$ and adding the single ECB constraint (10). By setting the branching priority higher for y than for any of the x_j, the branch-and-bound algorithm will ensure that $y = \Sigma_{j\in J}x_j$ is integer before branching on any individual x_j. If y is or becomes integer, subsequent branching on an x_j may cause y to become fractional at which point branching reverts to y.

To create the ECB constraint above, we are taking advantage of this knowledge about SPPs: *A priori*, every variable looks pretty much the same. (This would also be the case for set-packing, set-covering and knapsack problems, for instance.) As will be seen, even a single constraint like (10) can substantially reduce computational effort. However, we can try to extend the basic technique further.

The ECB constraint (10), along with branching priorities on integer variables, expresses: "The sum of all relaxed integer variables must be integer before requiring any individual variable to be integer." When this statement is satisfied, it is natural to ask for a somewhat stronger requirement: "The sum of some subset of relaxed integer variables x_j should be integer before requiring any variable in that subset to be integer." We accomplish this through "nested ECB."

Nested ECB separates the integer variables in one ECB constraint into disjoint subsets and creates a new ECB constraint for each subset. (It is not a requirement that the subsets be disjoint, but it is a logical and compelling step in our initial exploration of this technique.) For the SPP, we create a partition of J with subsets $J_l \subset J$, $l \in L$, and add ECB constraints

$$\sum_{j \in J_l} x_j - y_l = 0 \; \forall \; l \in L \, .$$

(Actually, any one of the nested constraints defined this way can be omitted because it is implied by the others and the initial constraint defined over all $j \in J$.) We set the branching priority high for y, intermediate for the y_l, and low for the x_j. Of course, nesting can be carried out recursively many times, but we have found in our test problems that one or two levels is the limit of its usefulness.

The basic ECB constraint (10) for the SPP can certainly be added without solving the problem's LP relaxation. In fact, that constraint can become a static part of the basic model and is thus a "static ECB constraint." Static nested constraints can be created by defining the subsets J_l before solving any LP relaxation. Because these ECB constraints are static, however, little benefit may be gained from them if $\hat{y}$ and all $\hat{y}_l$ are integer in the initial LP relaxation yet $\hat{\mathbf{x}}$ is fractional: The branch-and-bound algorithm will immediately branch on some x_j and ECB can only make a contribution at some lower level in the enumeration tree. But, if we solve the LP relaxation for $\hat{\mathbf{x}}$ and define the subsets J_l so that at least some of the $\hat{y}_l$ are fractional, ECB must come into play immediately, even if $\hat{y}$ is initially integer. We test this "semi-dynamic nested ECB" by splitting J into two subsets J_1 and J_2 using this procedure: (a) Evenly divide (roughly) the variables with $\hat{x}_j = 0$ between J_1 and J_2, (b) evenly divide (roughly) the variables with $\hat{x}_j = 1$ between J_1 and J_2, (c) evenly divide (roughly) the fractional variables $\hat{x}_j$ between J_1 and J_2, and (d) if neither $\sum_{j \in J_1} \hat{x}_j$ nor $\sum_{j \in J_2} \hat{x}_j$ is fractional, move one fractional variable between J_1 and J_2 to ensure that both sums are fractional.

ECB is implemented for the SPP using the CPLEX Callable Library (Version 3.0 1993). We use the "maximum-infeasibility" variable-selection rule for branching which is, empirically, the best option for our test problems among the four alternatives available. A relative optimality criterion of 0.1% is used (CPLEX 1993, p. 123). All computation is performed on an IBM RS/6000 Model 590 with 512 megabytes of RAM.

Our test problems are modest-sized SPPs from the meat-packing industry (Ronen 1997). The results are displayed in Table 1 and show that "Basic ECB" improves solution times significantly for the four largest problems. The value of semi-dynamic ECB is dubious but more experimentation is warranted: Perhaps different rules for defining the nested subsets would yield better results. Results with a static, nested ECB were uniformly worse than "Basic ECB" and are not listed.

Table 1. Solving set-partitioning problems with ECB.

Model			BandB		Basic ECB		SD2 ECB	
Name	Rows	Cols	Time	Nodes	Time	Nodes	Time	Nodes
PIB01	19	377	0.5	23	**0.5**	**4**	0.7	5
PIB02	30	660	**0.6**	**9**	0.9	14	1.0	21
PIBA0	38	2564	71.6	1757	**7.6**	43	8.0	**37**
PIB03	49	3308	29.3	275	**13.1**	**119**	19.6	213
PIB04	49	2196	4.8	67	**4.4**	27	4.6	**20**
PIB05	50	2158	>1K	>21K	**7.9**	**109**	18.2	253

"Time" is in CPU seconds. "Nodes" are the number of nodes in the branch and bound enumeration tree. "BandB" uses standard branch and bound alone, while all other columns use a variant of ECB. "Basic ECB" adds a single branching constraint over all the variables. "SD2" adds one level of semi-dynamic nesting to basic ECB, by splitting J into two disjoint sets. Boldface values indicate the most efficient, minimum-time or minimum-node, solutions. "K" indicates thousands.

3.2 ECB for the Elastic Generalized Assignment Problem

The problem that instigated our search for better branching techniques is a version of the generalized assignment problem (GAP) (e.g., Amini and Racer 1994, Ross and Soland 1975) that we call the "elastic GAP" or "EGAP." The GAP arises in a number of contexts but is described here as a minimum cost assignment of a collection of orders to delivery trucks. Each order must be delivered, and the number of orders any truck can deliver is constrained by the amount of time the truck has available to make deliveries. Deliveries are made from a single depot and each order o requires one out-and-back trip of known duration. All orders must be delivered and it is assumed that there is sufficient time available to make these deliveries in the basic GAP; otherwise the problem is infeasible. In the EGAP, penalized overtime on some or all trucks is allowed and this ensures that all deliveries can be met. The model is

$$
\begin{aligned}
(\text{EGAP}) \qquad \min \quad & \sum_{o\in O}\sum_{t\in T} c_{ot}\, x_{ot} + \sum_{t\in T} p_t z_t \\
\text{s.t.} \quad & \sum_{t\in T_o} x_{ot} = 1 \;\forall\, o \in O \qquad (11)\\
& \sum_{o\in O_t} h_{ot}\, x_{ot} - z_t \le H_t \;\forall\, t \in T \\
& x_{ot} \in \{0,1\} \;\forall\, o \in O, t \in T_o \qquad (12)\\
& z_t \ge 0 \;\forall\, t \in T,
\end{aligned}
$$

where

- $o \in O$ is the set of orders to be delivered,
- $t \in T$ is the set of trucks that can make the deliveries,
- O_t is the set of orders that truck t is capable of delivering,
- T_o is the set of trucks with which order o can be delivered,
- c_{ot} is the cost (in dollars) of delivering order o with truck t,
- p_{ot} is the overtime penalty (in dollars per tenths of an hour) for truck t,
- h_{ot} denotes the hours (in tenths) required by truck t to deliver order o,
- H_t denotes the total regular-time hours (in tenths) available for deliveries on truck t,
- x_{ot} is 1 if order o is delivered by truck t, and is 0 otherwise, and
- z_t are the hours (in tenths) of overtime on truck t.

Constraints (11) ensure that each order is assigned to exactly one truck, and "resource constraints" (12) ensure that the hours available on the truck are not exceeded unless a linear overtime penalty is paid. The GAP is simply the EGAP with $z_t \equiv 0$ for all t.

Because constraints (11) do not overlap in the EGAP, the sum of all x_{ot} will be integer in any relaxation or restriction. Thus, an ECB constraint over all the variables, as in (10) for the SPP, would be of no value. However, the structure of the problem leads one to the following idea: The sum of (relaxed) orders on a truck, $\sum_{o\in O_t} x_{ot}$, should be integer before any individual variable x_{ot}, $o \in O_t$, is required to take on a binary value. This idea can be implemented as basic ECB by adding the following constraints to the EGAP:

$$\sum_{o \in O_t} x_{ot} - y_t = 0 \quad \forall\, t \in T\,, \tag{13}$$

where each $y_t \geq 0$ is a general integer variable. This modified EGAP has $|T|$ new constraints and $|T|$ new integer variables. We hope that the extra burden placed on the LP solver by these constraints and variables is outweighed by a significant reduction in the number of LPs that must be solved.

Actually, Jörnsten and Värbrand (1991) have applied a dynamic version of the ECB methodology just described to GAPs. In their branch-and-bound procedure, they branch by explicitly adding constraints to the problem whenever the sum of orders on a truck is fractional. Our technique is essentially equivalent, but is obviously much simpler and needs no specialized code to implement.

We first test ECB, implemented with constraints (13), on a set of eight EGAPs from the petroleum industry (Brown 1995). The EGAP is difficult and problems with as few as 200 variables cannot be solved with standard branch and bound in a reasonable amount of time. Table 2 lists problem statistics and computational results for these EGAPs (the number of trucks ranges from 6 to 35, and the number of orders from 21 to 151). The same hardware, software and software settings are used as for the SPP computational tests. Results labeled with "ECB" use $|T|$ ECB constraints, one for each constraint of type (12). Results labeled with "KS" use elastic knapsack cuts (Appleget 1997) which extend standard knapsack cuts or "cover cuts" (Balas 1975, Balas and Zemel 1978). A set of at most 200 such cuts is added to the problem before branch and bound begins. The column annotated solely with "KS" is included for comparison. The results in Table 2 demonstrate that significant computational improvements can be obtained with ECB and that there can be a synergistic effect between ECB and knapsack cuts.

Table 2. EGAP solution results for eight real-world problems.

Model			BandB		ECB		ECB+KS		KS	
Name	Rows	Cols	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes
LONGD	30	46	**1.7**	259	1.9	**4**	2.2	12	2.2	6
LONGN	27	37	**1.5**	82	**1.5**	12	2.1	**4**	2.1	95
BOSTD	73	330	>1K	>170K	**2.5**	159	3.5	**63**	>1K	>129K
BOSTN	65	266	>1K	>183K	**1.8**	57	3.0	**50**	>1K	>132K
DLWRD	89	469	>1K	>113K	>1K	>137K	**8.6**	**492**	>1K	>102K
DLWRN	59	200	28.9	10441	**1.8**	89	3.1	**65**	8.0	1393
LOSAD	185	1835	>1K	>79K	>1K	>35K	**199.0**	**3725**	>1K	>49K
LOSAN	182	1790	>1K	>73K	>1K	>36K	**815.9**	**13627**	>1K	>34K

Boldface values denote minimum-time or minimum-node solutions. "ECB" indicates that ECB constraints (13) were used, and "KS" indicates that elastic knapsack cuts were used. ECB+KS is clearly the best overall technique.

We have also obtained a set of 84 randomly generated GAPs from the literature (Osman 1995, Beasley and Chu 1995, Cattrysse *et al.* 1994), which have been compiled by Beasley (1997). These problems range in size from 20 constraints and 75 variables to 220 constraints and 4,000 variables. Using our terminology, the number of trucks in each problem ranges from 5 to 20 and the number of orders ranges from 15 to 200; each order can be delivered by each truck. Table 3 summarizes the computational results for these problems. As with the EGAPs, ECB can improve solution times significantly for these GAPS, particularly when combined with knapsack cuts.

Table 3. Summary of computational results for 84 randomly generated GAPs.

	Small (< 1000 Variables)				Large (≥ 1000 Variables)			
	BandB	KS	ECB	ECB+KS	BandB	KS	ECB	ECB+KS
Solved	40.6%	100%	100%	100%	30%	80%	90%	100%
Time	88.2	4.3	6.9	2.9	352.5	113.0	26.5	70.1
Nodes	33K	451	1022	33	49K	3731	1596	724

"Solved" is the percentage of problems successfully solved in under 1,000 CPU seconds, "Time" is the average CPU seconds required to solve those problems that could be solved in under 1000 seconds, and "Nodes" is the average number of nodes in the branch-and-bound tree required by each successfully solved problem. As in Table 2, ECB+KS is the most efficient procedure.

Since all the data in our version of the EGAP is integer, another ECB partitioning scheme quickly comes to mind: Ensure that the number of hours (in tenths) assigned to any truck is integer before requiring that the variables x_{ot} take on binary values. If we were dealing with a standard GAP with inelastic, inequality resource constraints ($z_t \equiv 0$ in constraints (12)), simply adding a slack variable to each resource constraint and branching on these variables would implement ECB. (We might call this implicit-constraint branching since we would add no constraints, just variables.) Because of the overtime variables z_t, however, this will not work for the EGAP. Therefore, we implement ECB by defining integer ECB variables $y'_t \geq 0$ and adding ECB constraints:

$$\sum_{o \in O_t} h_{ot} x_{ot} - y'_t = 0 \;\forall\, t \in T\,, \tag{14}$$

and by setting the branching priority higher for y'_t than for the x_{ot}. Since $h_{ot} > 1$ is usual, this type of branching corresponds to using ECB constraints (6) with $\alpha_{kj} > 1$.

This variant of ECB is a failure, by itself and in conjunction with ECB constraints (13). (We omit these computational results; most run times exceed 1000 seconds.) We conjecture that ECB fails in this instance because: If $m < \hat{y}'_t < m + 1$ for some LP relaxation to the EGAP and for some integer m, branching can occur via $y'_t \le m$ or $y'_t \ge m + 1$. But, given the nature of constraints (12) and coefficients h_{ot}, there may be no solution to EGAP with $y'_t = m$ or $y'_t = m + 1$. Thus, the algorithm wastes time investigating infeasible values for y'_t.

There may be a way around the difficulty just described: Dynamic programming could be used first to determine the set of feasible values for each y'_t, say $m_{t_1} < m_{t_2} < \ldots < m_{t_L}$, based on the elastic knapsack constraints (12). Then, when $m_{t_l} < \hat{y}'_t < m_{t_{l+1}}$ the branching rule would be "$y'_t \le m_{t_l}$ or $y'_t \ge m_{t_{l+1}}$." Valid values for the m_{t_l} could even be determined dynamically within a branch-and-bound algorithm. We have not yet investigated these possibilities.

3.3 A Plant-Line Scheduling Model

GLS (Generic Line Scheduler) models plant-line scheduling in the food processing industry on a shift-by-shift basis, over a two-week horizon. The key model variable is $X_{ptp't'}$, which is 1 if "production pattern" p finishes production at time t and is immediately followed by pattern p' in shifts $t + 1$ through t'. A production pattern produces a single product on a processing line and packs that product into one or more stock-keeping units (skus) on one or more packing lines. The use of variables $X_{p't'pt}$ allows accurate modeling of product and packing changeovers.

GLS includes standard production/inventory/demand constraints and uses flow-balance constraints to route one unit of "pattern flow" on each processing line over the shifts in the time horizon:

$$-\sum_{(p',t')\in PT^-_{pt}} X_{p't'pt} + \sum_{(p',t')\in PT^+_{pt}} X_{ptp't'} = 0 \;\forall\, l \in L, t \in T, p \in P_{lt}, \qquad (15)$$

where L is the set of processing lines, T is the set of shifts, P_{lt} is the set of products that may be produced on line l during shift t, PT^-_{pt} is the set of patterns and ending shifts for those patterns that may directly precede

pattern p beginning on shift t, and PT_{pt}^{+} is analogous, but applies to immediately succeeding patterns and shifts. (Other constraints start each line off with one unit of flow, limit each pattern from appearing more than once during the week, ensure that each sku is packed, etc.)

The model formulation is complicated, but the wisdom of the formulation is validated by tight LP relaxations, never worse than 10% of the optimal solution value in any test. However, the sheer number of binary variables in this model makes branch and bound difficult to use. We can reduce this number with ECB.

Constraints (15) are flow-balance constraints for "nodes," indexed by p and t, in a directed acyclic graph. If we fix the flow transiting each node to 0 or 1, the LP relaxation of GLS will have an intrinsically integer solution, or it will be infeasible. This observation leads to the following model modifications:

1. Add binary variables Y_{pt} defined to be 1 if pattern p starts at time t, else 0.
2. Redefine variables $X_{p't'pt}$ to be non-negative and continuous. (Upper bounds are unnecessary.)
3. Add the following set of constraints:

$$\sum_{(p',t')\in PT_{pt}^{+}} X_{ptp't'} - Y_{pt} = 0 \;\forall\; l \in L, t \in T, p \in P_{lt}\,. \tag{16}$$

Now, instead of branching on 4,000–15,000 binary variables $X_{p't'pt}$, we have a complete branching scheme that involves only 500 or so variables, Y_{pt}. The price we pay is the addition of 500 or so constraints.

At this point we must point out the relationship between ECB for this problem and SOS branching. The set of variables $X_{ptp't'}$ plus Y_{pt} in each constraint (16) can be classified as a set of SOS-Type-1 (S1) variables and as a set of SOS-Type-3 (S3) variables if Y_{pt} is reflected: Exactly one of the variables in the set must be 1. CPLEX, our chosen solver, uses a different definition of a set of S1 variables however: At most one variable in the set may be strictly between its bounds (GAMS 1996). On the other hand, CPLEX will automatically identify sets of S3 variables (to include reflections) via the constraints (16) and perform S1 branching on those sets, as described by Tomlin (1970). As we shall see, SOS branching has an erratic effect on run times for GLS.

We note also that each set of variables x_j for $j \in J_i$ in the SPP is an S3 set, as is each set of variables x_{ot} for $t \in O_t$ in the EGAP. Experiments with our SPP problems show that SOS branching does not provide a significant advantage over basic branch and bound. Furthermore, all EGAP problems

that "time out" at 1000 seconds for basic branch and bound also time out with SOS branching with only one exception. We believe that the long run times with SOS arise because SOS relies on a rigid and meaningful ordering of the S3 variables (Beale and Tomlin 1970) and there is no such ordering in these problems or, at least, we have not found one.

GLS can exploit another constraint-branching idea, too. Product changeovers are important and expensive, so we branch on Q'_w, defined to be the number of such changeovers during week *w*, using:

$$\sum_{t\in T_w}\sum_{p\in P_w}\sum_{(p',t')\in PT_{pt}^{-}\,|\,f_{p'}\neq f_p} X_{p't'pt} - Q'_w = 0 \quad w\in\{1,2\} \tag{17}$$

where $Q'_w \geq 0$ and integer for $w \in 1, 2$, T_w is the set of shifts in week *w*, P_w is the set of patterns that may be produced in week *w*, and f_p is the product associated with pattern *p*. (Lower bounds on Q'_w can sometimes be implied and exploited, but we do not use these bounds in the testing here.)

Model statistics for a set of GLS models are displayed in Table 4, with various combinations of the ECB constructs (16) and (17). Corresponding computational results displayed in Table 5. Tests are carried out using the same hardware as in the previous examples, but GLS is generated using GAMS (1996) and solved with CPLEX using a 5% optimality tolerance and using the "pseudo-cost" variable-selection rule for branching. Results indicate that the simple, product-changeover ECB constraints (17) do help solve the GLS model but typically interfere with rapid solutions when combined with ECB constraints (16). The most consistent results are obtained using ECB constraints (16) alone. However, SOS branching does show promise.

SOS branching is the best solution strategy for two of the five problems but is a horrible strategy for one of them. Our ordering of the S3 variables, which is based on time (see the caption of Table 5), works well in some cases. So, SOS may potentially improve run times for GLS, but more experimentation would be required to determine a consistently good ordering for the S3 variables, if one exists.

Table 4. GLS Model Statistics.

	Basic Model (a): No ECB			Model (b): With Constraints (17)			Model (c): With Constraints (16)			Model (d): With Constraints (16)+(17)		
Name	*m*	*n*	n_{int}	*m*	*n*	n_{int}	*m*	*n*	n_{int}	*m*	*n*	n_{int}
GLS1	567	4202	4157	569	4204	4159	1092	4727	527	1094	4729	529
GLS2	595	4494	4439	597	4496	4441	1141	4985	548	1143	5042	550
GLS3	639	5219	5162	641	5221	5164	1227	5750	590	1229	5809	592

	Basic Model (a): No ECB			Model (b): With Constraints (17)			Model (c): With Constraints (16)			Model (d): With Constraints (16)+(17)		
Name	m	n	n_{int}	m	n	n_{int}	m	n	n_{int}	m	n	n_{int}
GLS4	728	10231	10168	730	10233	10170	1400	10903	674	1402	10905	676
GLS5	772	14709	14644	774	14711	14646	1486	16423	716	1488	15425	718

All models pack 15-25 skus over two weeks. "m" in number of constraints, "n" is number of variables, and "n_{int}" is number of integer variables.

Table 5. GLS Computational Results.

	Model (a)		Model (b)		Model (c)		Model (d)		Mod.(c)+SOS	
Name	Sec.	Nodes	Sec.	Nodes	Sec.	Nodes	Sec.	Nodes	Sec.	Nodes
GLS1	27.8	125	7.9	**11**	**6.7**	13	18.6	18	17.9	53
GLS2	288.7	1730	71.2	182	**9.2**	**24**	57.3	121	389.3	1774
GLS3	131.4	715	120.7	231	20.4	57	46.3	63	**11.0**	**19**
GLS4	98.1	134	69.5	54	55.0	75	91.6	87	**16.7**	**13**
GLS5	6468.8	12694	1785.9	1888	**791.0**	**598**	3340.0	1504	> 9K	> 11K

Boldface values indicate the most efficient, minimum-time or minimum-node, solutions. All models are solved with a 5% relative optimality tolerance. "Mod.(c)+SOS" denotes Model (c) with each ECB constraint (16) used to identify a set of S3 variables on which S1 branching is performed. Times for "Mod.(c)+SOS" reflect the best ordering of the S3 variables found empirically: The variables $X_{ptp't'}$ are ordered by increasing t' and are followed by Y_{pt}.

4. SUMMARY AND COMMENTS

In summary, it appears that ECB holds much promise for reducing solution times for certain MIPs. Future work will explore the nested and dynamic versions of ECB more fully.

We mention one issue that is not discussed in the body of this paper: ECB can sometimes interfere with the rapid identification of good incumbent solutions within a branch-and-bound procedure. Apparently, an ECB variable is branched on in the "wrong direction" in these cases, and much time is wasted exploring a large, unfruitful portion of the enumeration tree. On the other hand, ECB may still lead to the quick fathoming of nodes through strong changes in local lower bounds, if a good incumbent is known. We have found that this conflict can sometimes be resolved by finding a good incumbent using a model without ECB, and then returning to the model with ECB and with the initial incumbent.

ACKNOWLEDGMENTS

Jeffrey Appleget's research was partially supported by the Faculty Development and Research Fund, U.S. Military Academy. Kevin Wood's

research was partially supported by the Office of Naval Research and the Air Force Office of Scientific Research. The authors thank their respective sponsors. The authors also thank John Tomlin for providing several references.

REFERENCES

Amini, M. and Racer, M. (1994) "A rigorous computational comparison of alternative solution methods for the generalized assignment problem", *Management Science*, vol. 40, pp. 868-890.

Appleget, J.A. (1997) "Knapsack cuts and explicit-constraint branching for solving integer programs", Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, June 1997.

Balas, E. (1975) "Facets of the knapsack polytope", *Mathematical Programming*, vol. 8, pp. 146-164.

Balas, E. and Zemel, E. (1978) "Facets of the knapsack polytope from minimal covers", *SIAM Journal of Applied Mathematics*, vol. 34, pp. 119-148.

Beale, E.M.L. and Tomlin, J.A. (1970) "Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables", in Proceedings of the 5th International Operations Research Conference, J. R. Lawrence (ed.), Tavistock Publications Limited, London, pp. 447-454.

Beasley, J.E. "Generalised assignment problem", OR-Library, The Management School at Imperial College of Science, Technology, and Medicine, London, United Kingdom, available from http://mscmga.ms.ic.ac.uk/, Internet, accessed 10 February 1997.

Beasley, J.E. and Chu, P.C. (1995) "A genetic algorithm for the generalised assignment problem", working paper, The Management School, Imperial College, London SW7 2AZ, England.

Bixby, R.E., Ceria, S., McZeal, C.M., and Savelsbergh, M.W.P. *MIPLIB 3.0*, Computational and Applied Mathematics Department, Rice University, Houston, Texas, available from http://www.caam.rice.edu/~bixby/miplib/miplib.html, Internet, accessed 10 February 1997.

Brooke, A., Kendrick, D., and Meeraus, A. (1992) *Release 2.25, GAMS, A Users Guide*, GAMS Development Corporation, Washington, DC.

Brown, G., personal communication, October 14, 1995.

Cattrysse, D., Salomon, M., and Van Wassenhove, L.N. (1994) "A set partitioning heuristic for the generalized assignment problem", *European Journal of Operational Research*, vol. 72, pp. 167-174.

CPLEX Optimization, Inc., *CPLEX Manual, Using the CPLEX™ Callable Library and CPLEX™ Mixed Integer Library*, Incline Village, Nevada, 1993.

Escudero, L.F. (1987) "On extensions of the Beale-Tomlin special ordered sets and strategies for LP-based solving a general class of scheduling problems", Martin Beale Memorial Symposium, London, July.

Escudero, L.F. (1988) "S3 sets, an extension of the Beale-Tomlin special ordered sets", *Mathematical Programming: Series B*, vol. 42, pp. 113-123.

Foster, B.A. and Ryan, D.M. (1981) "An integer programming approach to scheduling", in *Computer Scheduling of Public Transport* (A. Wren, ed.) North-Holland Publishing Company, pp. 269-280.

GAMS Development Corporation (1996) *GAMS—The Solver Manuals*, Washington, DC.
Hummeltenberg, W. (1984) "Implementations of special ordered sets in MP software", *European Journal of Operations Research*, vol. 17, pp. 1-15.
Jörnsten, K.O. and Larsson, T. (1988) "A generalized branching technique", Department of Mathematics Research Report, Linkoping Institute of Technology.
Jörnsten, K.O. and Värbrand, P. (1991) "A hybrid algorithm for the generalized assignment problem", *Optimization*, vol. 22, pp. 273-282.
Osman, I.H. (1995) "Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches", *OR Spektrum*, vol. 17, pp. 211-225.
Ross, G.T. and Soland, R.M. (1975) "A branch and bound algorithm for the generalized assignment problem", *Mathematical Programming*, vol. 8, pp. 91-103.
Ronen, D., personal communication, March 20, 1997.
Sol, M. (1994) "Column generation techniques for pickup and delivery problems", Ph.D. thesis, Technische Universiteit Eindhoven.
Tomlin, J.A. (1970) "Branch and Bound Methods for Integer and Non-Convex Programming", in *Integer and Nonlinear Programming*, J. Abadie (ed.), North-Holland Publishing Company, Amsterdam.
Vössner, S., and Wood, R. K. (1999) "A Plant-Line Scheduling Algorithm Using a Genetic Algorithm and Integer Programming", Working Paper, 15 March.