ON THE VALUE OF BINARY EXPANSIONS FOR GENERAL MIXED-INTEGER LINEAR PROGRAMS

JONATHAN H. OWEN

General Motors Research and Development Center, 30500 Mound Road, MC #480-106-359, Warren, Michigan 48090, jonathan.owen@gm.com

SANJAY MEHROTRA

Department of Industrial Engineering and Management Sciences, Robert R. McCormick School of Engineering, Northwestern University, Evanston, Illinois 60208, mehrotra@iems.nwu.edu

(Received November 2000; revision received May 2001; accepted June 2001)

We study the use of binary variables in reformulating general mixed-integer linear programs. We show that binary reformulations result in problems for which almost all the binary variables replacing a general integer variable need to be explored during branching. We also give computational results on the performance of such reformulations in solving the mixed-integer programs, which support our theoretical results.

1. INTRODUCTION

Practitioners frequently create models by replacing general integer variables in a mixed-integer linear program by introducing new binary variables (Rothberg 2000). In this paper we provide theoretical and computational evidence demonstrating that remodeling of mixed-integer programs by binary variables should be avoided in practice unless special techniques are used to handle these variables.

Given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $p \leqslant n$, the general mixed-integer linear programming problem is

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && x \in K^0, \end{aligned} \qquad \text{(MILP)}$$

where

$$K^0 = \{x \in K \mid x_i \in \mathbb{Z} \quad \text{for } i \in \{1, \dots, p\}\} \qquad (1)$$

and

$$K = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}. \qquad (2)$$

The set K^0 describes feasible mixed-integer solutions and the set K gives the feasible region of the LP-relaxation for MILP. We assume that K is bounded. Under this assumption, without loss of generality, we assume that the bound constraints $x_j \geqslant 0$, for $j = 1, \dots, n$, and $x_j \leqslant M_j$, for $j = 1, \dots, p$, are included in the constraint set defining K. By $\text{conv}(K^0)$ we denote the convex hull of points in K^0.

In recent years methods using variable disjunctions to generate cuts have emerged as a powerful tool for solving mixed binary linear and nonlinear programs (Balas et al. 1993, 1996; Sherali and Adams 1990, 1994; Stubbs and Mehrotra 1999). The disjunctive cuts generated at the root node using variable disjunctions have proven useful in reducing the size of the tree generated in a branch-and-cut algorithm (Balas et al. 1993, 1996). From a theoretical point of view, the use of variable disjunctions allows for a complete description of the convex hull of the mixed binary set in a finite number of sequential convexification steps (Balas et al. 1993; Sherali and Adams 1990, 1994). Balas et al. (1993) also give a finite cutting plane algorithm for mixed binary linear programs, providing further theoretical support for using disjunctive cuts.

The complete description of the convex hull of the general mixed-integer set is not possible in a finite number of sequential convexification steps using variable disjunctions (Balas 1979, Owen and Mehrotra 2001) unless general integer variables are modeled as binary variables. Sherali and Adams (1999) provide one such technique, which uses a full expansion of the general integer variables in terms of binary variables. When convexification is done using variable disjunctions on general integer variables, the sequential convexification procedure converges to the convex hull only in the limit (Owen and Mehrotra 2001), in contrast with finite convergence for the binary case. Knowing this theoretical advantage of mixed binary problems, it is tempting for practitioners to reformulate a MILP as a mixed binary linear problem using standard techniques. Furthermore, one may think to use the binary reformulation of MILP and use disjunctive cuts in a branch-and-cut algorithm to solve this reformulation. On the contrary, there is an understanding among experts that this remodeling of general integer variables results in binary problems which are considerably more time consuming to solve (Rothberg 2000).

Subject classifications: Programming, integer: remodeling of general integer variables.
Area of review: Optimization.

In this paper we study the effectiveness of a compact and a full binary reformulation of MILP. We show that to generate $\operatorname{conv}(K^0)$ in the compact reformulation of MILP by choosing binary variables in the best possible order, the sequential convexification procedure of Balas et al. (1993) and the reformulation technique of Sherali and Adams (1990, 1994) have to be carried out for all the binary variables (replacing a general integer variable) to eliminate more than half of the possible ranges for extreme points that are fractional in an integer variable. This negative result indicates that we need to explore the entire depth of the branch-and-bound tree while branching on binary variables replacing a general integer variable. It also indicates that disjunctive cuts that make good progress toward an optimal solution in the mixed-integer space may not be available early (e.g., at the root node). A result on the variable selection order is also given for the full reformulation of MILP. This result suggests that in the full reformulation all binary variables corresponding to integer values larger than the optimal integer value need to be fixed during branching.

As a consequence, we conclude that the binary reformulations are unlikely to be of practical (computational) value and should be discouraged in practice unless special techniques are used to handle these variables. This is further supported by computational evidence obtained on MIPLIB test problems (Bixby et al. 1998).

This paper is organized as follows. In the next section we give the compact reformulation and describe a use of the convexification procedures of Sherali and Adams (1990) and Balas et al. (1993). Here we also show that if sets from the binary reformulation are prematurely projected, then we get the sets generated in the procedure described in Owen and Mehrotra (2001), which only converges in the limit. In §3 we consider an application of the binary variable convexification procedure to a two-variable example problem. In §4 we prove the main results of this paper. Here we show that a particular variable selection order is best in terms of generating early cuts in the original space. We also demonstrate that in most cases we need to generate the hull for all expansion variables in order to eliminate most fractional extreme points in the original space. We state a result on variable selection order for the full reformulation in §4.3. In §5 we study computational results on the performance of binary reformulations.

2. BINARY REFORMULATION AND CONVEXIFICATION PROCEDURE

2.1. Binary Reformulation

The binary reformulation of MILP that we primarily focus on in our theoretical discussion is obtained by defining each integer variable by

$$x_j = u_0^j + 2u_1^j + 4u_2^j + 8u_3^j + \cdots + 2^{k_j} u_{k_j}^j,$$

where variables $u_i^j \in \mathbb{B}$ for $i = 0, 1, 2, \ldots, k_j$, and $k_j = \lfloor \log_2 M_j \rfloor$. We call this a compact reformulation since it requires us to introduce the least number of new binary variables. An alternative full reformulation introducing M_j binary variables (one for each integer value) was used by Sherali and Adams (1999); we will discuss implications of our analysis to the full reformulation in §4.3.

The feasible set of the reformulated problem can be written in n continuous and $k^* \equiv \sum_{j=1}^{p}(1+k_j)$ binary variables as follows:

$$K_B^0 = \{(x,u) \in K_B \mid u_i^j \in \mathbb{B} \quad \text{for } j = 1, 2, \ldots, p, \text{ and } i = 0, 1, \ldots, k_j\},$$

where

$$K_B = \left\{ x \in \mathbb{R}^n, u \in \mathbb{R}^{k^*} \;\middle|\; \begin{array}{ll} Ax \leqslant b & \\ x_j = \sum_{i=0}^{k_j} 2^i u_i^j & \text{for } j = 1, 2, \ldots, p, \\ 0 \leqslant u_i^j \leqslant 1 & \text{for } j = 1, 2, \ldots, p, \\ & \text{and } i = 0, 1, \ldots, k_j \end{array} \right\}.$$

In general, we will have a set $S \subseteq K_B$ in the (x,u)-space whose projection onto the x-space is defined as

$$\operatorname{Proj}_x(S) \equiv \{x \mid (x,u) \in S\}.$$

For example, $K = \operatorname{Proj}_x(K_B)$.

2.2. Convexification Procedure

For any $j \in \{1, 2, \ldots, p\}$ and $i \in \{0, 1, \ldots, k_j\}$, define

$$\mathscr{C}_i^j(S) \equiv \operatorname{conv}\big(\{(x,u) \in S \mid u_i^j = 0\}, \{(x,u) \in S \mid u_i^j = 1\}\big).$$

A convexification procedure to generate $\operatorname{conv}(K^0)$ based on the binary convexification procedure of Balas et al. (1993), and the reformulation-linearization technique of Sherali and Adams (1990), can be described as follows.

Given a general-integer problem, we first generate a binary expansion in the manner described in §2.1; let the convex set K_B represent the linear relaxation of the expansion feasible region. Let C_B represent the current polyhedral approximation of K_B^0. We initialize C_B to be K_B. At each iteration of the procedure, if there are any expansion variables that are fractional at an extreme point of C_B, then we select such a fractional variable u_i^j. In this case, we update $C_B := \operatorname{conv}(\{(x,u) \in C_B \mid u_i^j = 0\}, \{(x,u) \in C_B \mid u_i^j = 1\})$. This update does not add any new fractional extreme points and it removes all extreme points of current C_B with fractional u_i^j (see Theorem 2.1 and Theorem 2.2 of Balas et al. 1993). When all extreme points of C_B are integer valued in the expansion variables, the region C_B is equal to $\operatorname{conv}(K_B^0)$. At this point in the procedure, we project the final expansion region C_B onto the x-space, giving $C_G^* = \operatorname{Proj}_x(C_B)$. It is easy to see that $C_G^* = \operatorname{conv}(K^0)$. A pseudocode description of the procedure is described in Figure 1. In this description, C_B^t represents the current expansion region during the tth iteration.

Theorem 1 and Corollary 2 below show that we do not benefit by projecting the set at an intermediate stage of the binary expansion convexification procedure. These results show that if the set in the (x,u)-space is projected after each iteration of the binary expansion convexification procedure, we get the infinite convergence procedure of Owen

Figure 1. Pseudocode of the binary expansion convexification procedure.

```
algorithm binary expansion convexification ($K$)
{
    /* lift to (mixed) binary space */
    $K_B := \mathrm{conv}(\{(x,u)$ in binary expansion of $K\})$;

    /* initialization */
    $t := 0$; $C_B^0 := K_B$;

    while ($\{(x,u) \in \mathrm{ext}(C_B^t) | u_i^j \notin \mathbb{Z}$ for some $i$, $j$ $\} \neq \emptyset$) {
        select a fractional expansion variable $u_i^j$;

        /* binary convexification */
        $C_B^{t+1} := \mathrm{conv}\left(\{(x,u) \in C_B^t | u_i^j = 0\}, \{(x,u) \in C_B^t | u_i^j = 1\}\right)$;

        $t := t + 1$;
    }

    /* project to general-integer space */
    $C_G^* := \mathrm{Proj}_x(C_B^t)$;

    STOP; /* $C_G^* = \mathrm{conv}(K^0)$ */
}
```

and Mehrotra (2001). In particular, Theorem 1 states that after using the binary convexification procedure for all the binary variables corresponding to a general-integer variable x_j, if we project the set onto the x-space, we get the convex hull of C_G restricted to integer values of x_j. This same set is generated at an iteration of the convexification procedure in Owen and Mehrotra (2001). Corollary 2 states the equivalence of generated sets when the procedure is applied repeatedly.

Theorem 1. *For a polyhedral set C_G in x-space, let $C_B(C_G)$ be the corresponding polyhedral set in (x, u)-space of its binary reformulation. For each $j \in \{1, 2, \ldots, p\}$ and each ordering $i_0, i_1, \ldots, i_{k_j}$ of the indices $\{0, 1, \ldots, k_j\}$, let $S_j(C_B(C_G)) \equiv \mathscr{C}_{i_{k_j}}^j(\mathscr{C}_{i_{k_j-1}}^j(\ldots(\mathscr{C}_{i_0}^j(C_B(C_G)))\ldots))$. Then,*

$$\mathrm{Proj}_x(S_j(C_B(C_G))) = \mathscr{K}_j(C_G),$$

where $\mathscr{K}_j(C_G) \equiv \mathrm{conv}\{x \in C_G | x_j \in \mathbb{Z}\}$.

Corollary 2. *For $l \leqslant p$, let $\{j_1, j_2, \ldots, j_l\} \subseteq \{1, 2, \ldots, p\}$. Then,*

$$\mathscr{K}_{j_l}(\mathscr{K}_{j_{l-1}}(\ldots(\mathscr{K}_{j_1}(C_G))\ldots)) = \mathrm{Proj}_x(S_{j_l}(C_B(\mathscr{K}_{j_{l-1}}(\mathscr{K}_{j_{l-2}}(\ldots(\mathscr{K}_{j_1}(C_G))\ldots))))).$$

Corollary 2 suggests that it is necessary to generate $\mathrm{conv}(K_B^0)$ before projecting it onto the x-space in order to generate the hull $\mathrm{conv}(K^0)$ in a finite number of steps. Therefore, we focus on studying the efficiency with which the binary expansion convexification procedure generates $\mathrm{conv}(K_B^0)$.

We note that our interest is in solving MILP. In the branch-and-cut algorithm $\mathrm{conv}(K_B^0)$ is not generated while solving MILP, and constraints describing $\mathrm{conv}(K_B^0)$ are generated as we progress. However, we would like to know when it is possible to generate constraints that would help strengthen the set of solutions in the x-space. In this context the study of how $\mathrm{conv}(K_B^0)$ is generated in the (x, u)-space becomes useful.

We illustrate our main results with the help of an example in the next section.

3. GENERATING THE CONVEX HULL FOR AN EXAMPLE MILP

We apply the convexification procedure in Figure 1 to a two-variable example problem. The sets C_B^t in this procedure are generated using version 1.3.1 of PORTA (Christof and Löebel 1997). As the convexification procedure progresses, we want to know the cutting planes that are generated (if any) in the x-space. For this purpose we project the mixed-binary set C_B^t at each iteration of the convexification procedure to the x-space, however, we continue with the original binary formulation (with added inequalities) in the next iteration. This determines the progress being made in the x-space when applying the expansion convexification procedure. Consider the following example problem:

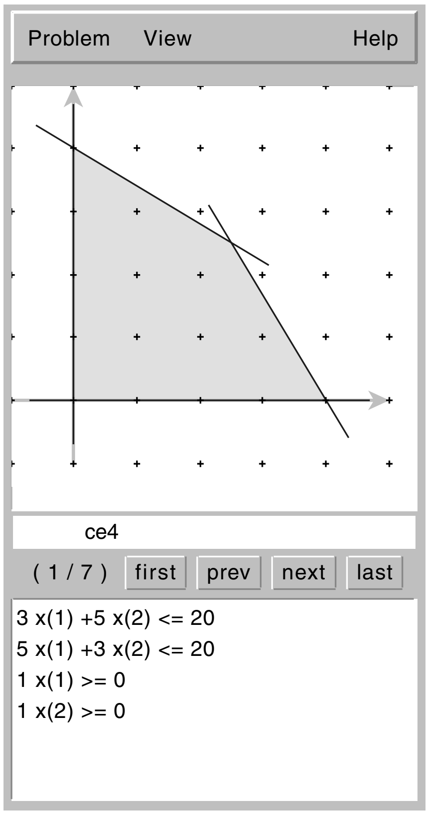
$$K^0 = \left\{ x \in \mathbb{Z}^2 \,\middle|\, \begin{array}{rcl} 3x_1 + 5x_2 & \leqslant & 20 \\ 5x_1 + 3x_2 & \leqslant & 20 \\ x_1 & \geqslant & 0 \\ x_2 & \geqslant & 0 \end{array} \right\}.$$

The relaxation K of this region, pictured in Figure 2, has a single fractional extreme point, $\bar{x} = (2.5, 2.5)$. The linear relaxation of the binary expansion of this region is given by

$$K_B = \left\{ x \in \mathbb{R}^2, u \in \mathbb{R}^6 \,\middle|\, \begin{array}{rcl} 3x_1 + 5x_2 & \leqslant & 20 \\ 5x_1 + 3x_2 & \leqslant & 20 \\ x_1 & \geqslant & 0 \\ x_2 & \geqslant & 0 \\ u_0^1 + 2u_1^1 + 4u_2^1 - x_1 & = & 0 \\ u_0^2 + 2u_1^2 + 4u_2^2 - x_2 & = & 0 \\ 0 \leqslant u & \leqslant & 1 \end{array} \right\}.$$

Since $K \subseteq \mathbb{R}^2$ we can visually observe the region C_G^t at each iteration of the modified binary convexification procedure. The illustrations in Figure 3 show the set C_G^t after each iteration of the procedure, in the case where expansion variables are selected in the order $u_2^2, u_1^2, u_0^2, u_2^1, u_1^1, u_0^1$. Observe that cuts are not generated in the original space until the third iteration of the procedure (Figure 3c), after we have convexified with respect to all expansion variables for x_2. At this point we have the convex hull with respect to x_2, thus no extreme points have fractional x_2 components. No cut in the x-space is generated after we convexify for variable u_2^1, and we get the convex hull $\mathrm{conv}(K^0)$ only after all the binary variables are considered. This leads to the interesting question: Is the property that (almost) all binary variables need to be considered to generate the convex hull true in general? Unfortunately, the answer to this question is yes, as it is proved in the next section.

Figure 2. The LP-relaxation of the example feasible region.



```
3 x(1) +5 x(2) <= 20
5 x(1) +3 x(2) <= 20
1 x(1) >= 0
1 x(2) >= 0
```

4. PROPERTIES OF THE BINARY EXPANSIONS

The previous example demonstrated the progress in x-space of the binary expansion convexification procedure, using a fixed selection order of the binary variables $(u_2^2, u_1^2, u_0^2, u_2^1, u_1^1, u_0^1)$. In §4.1 we show that this selection order of variables u_2^2, u_1^2, u_0^2 and u_2^1, u_1^1, u_0^1 is a "best" ordering for the example, from the viewpoint of generating cuts in the original space as early as possible.

The result presented in §4.2 shows that even with a best variable selection order, we still need to consider all expansion variables in order to eliminate more than half of the possible choices of fractional values of general-integer variables at an extreme point of K. This suggests that all binary variables corresponding to a general-integer variable must be explored in the standard branching process; such a behavior is confirmed in the computational experiments reported in §5. It may also be inferred from the result in §4.2 that mixing the order of binary variables representing different general-integer variables (for example, $u_2^2, u_2^1, u_1^2, u_1^1, u_0^2, u_0^1$) will not significantly improve the situation.

4.1. Best Variable Selection

We state the main result of this section in the following theorem.

Theorem 3. *Assume that all binary variables corresponding to a general-integer variable are selected together for convexification and no prior efficient ordering of the x variables is known in MILP. Then, cuts in the x-space are generated earliest if the convexification order for binary expansion variables of x is given as*

$$\left(u_{k_{j_1}}^{j_1}, u_{k_{j_1}-1}^{j_1}, \ldots, u_0^{j_1}, u_{k_{j_2}}^{j_2}, u_{k_{j_2}-1}^{j_2}, \ldots, u_0^{j_2}, \ldots, u_{k_{j_p}}^{j_p}, u_{k_{j_p}-1}^{j_p}, \ldots, u_0^{j_p}\right)$$

for some ordering $(j_1, j_2, \ldots, j_p)$ of the variable indices $\{1, 2, \ldots, p\}$ of x in MILP.

The remainder of this section is dedicated to establishing this result. Proposition 4 below is a consequence of the facial disjunctive property of Balas (1979).

Proposition 4. *For any index set $\{i_0, i_1, \ldots, i_l\} \subseteq \{0, 1, \ldots, k_j\}$,*

$$\mathscr{C}_{i_l}^j\left(\mathscr{C}_{i_{l-1}}^j(\ldots(\mathscr{C}_{i_0}^j(K_B))\ldots)\right) = \operatorname{conv}\left\{(x, u) \in K_B \mid u_{i_0}^j, u_{i_1}^j, \ldots, u_{i_l}^j \in \{0, 1\}\right\}.$$

Let $\operatorname{ext}(S)$ represent the extreme points of a set S. Lemma 5 shows that an extreme point of $\operatorname{Proj}_x(S)$ is obtained from the projection of an extreme point of S. As a consequence of this lemma, the subsequent corollary shows that the set $\operatorname{Proj}_x(S)$ can be generated by taking the convex hull of the projection of extreme points of S on to the x-space.

Lemma 5. *For a polyhedron $S \subseteq K_B$*

$$\bar{x} \in \operatorname{ext}(\operatorname{Proj}_x(S)) \Rightarrow \textit{there exists } \bar{u} \textit{ such that } (\bar{x}, \bar{u}) \in \operatorname{ext}(S).$$

Proof. Consider an arbitrary $\bar{x} \in \operatorname{ext}(\operatorname{Proj}_x(S))$. Since $\bar{x}$ is an extreme point of $\operatorname{Proj}_x(S)$, there must exist some $\bar{c} \in \mathbb{R}^n$ such that $\bar{x}$ is the unique optimal solution to the following problem:

$$\begin{aligned} &\text{minimize} \quad \bar{c}^T x \\ &\text{subject to} \quad x \in \operatorname{Proj}_x(S). \end{aligned}$$

Furthermore, since $\bar{x} \in \operatorname{Proj}_x(S)$ there must exist some $\bar{u}$ such that $(\bar{x}, \bar{u}) \in S$. Now consider the following problem:

$$\begin{aligned} &\text{minimize} \quad \bar{c}^T x \\ &\text{subject to} \quad (x, u) \in S. \end{aligned}$$

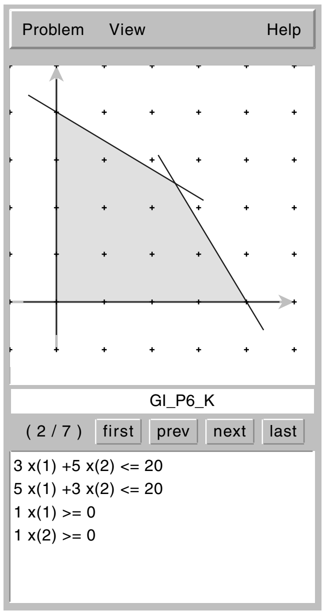
Since S is nonempty, we can solve this problem using the simplex method to yield a solution $(\hat{x}, \hat{u}) \in \operatorname{ext}(S)$ where $\bar{c}^T\hat{x} = \bar{c}^T\bar{x}$. Since $(\hat{x}, \hat{u}) \in S$, we have that $\hat{x} \in \operatorname{Proj}_x(S)$. It follows from our selection of $\bar{c}$ that $\hat{x} = \bar{x}$. Thus, since $(\hat{x}, \hat{u}) \in \operatorname{ext}(S)$ we know that $(\bar{x}, \hat{u}) \in \operatorname{ext}(S)$. ☐

Corollary 6. *For a polyhedron $S \subseteq K_B$, $\operatorname{Proj}_x(S) = \operatorname{conv}(\operatorname{Proj}_x(\operatorname{ext}(S)))$.*

Proof. By Lemma 5, we have that $\operatorname{ext}(\operatorname{Proj}_x(S)) \subseteq \operatorname{Proj}_x(\operatorname{ext}(S))$. Clearly $\operatorname{Proj}_x(\operatorname{ext}(S)) \subseteq \operatorname{Proj}_x(S)$, thus $\operatorname{Proj}_x(S) \equiv \operatorname{conv}(\operatorname{ext}(\operatorname{Proj}_x(S))) \subseteq \operatorname{conv}(\operatorname{Proj}_x(\operatorname{ext}(S))) \subseteq \operatorname{conv}(\operatorname{Proj}_x(S))$. Since $\operatorname{Proj}_x(S)$ is a convex set, $\operatorname{conv}(\operatorname{Proj}_x(\operatorname{ext}(S))) = \operatorname{Proj}_x(S)$. The result follows immediately. ☐

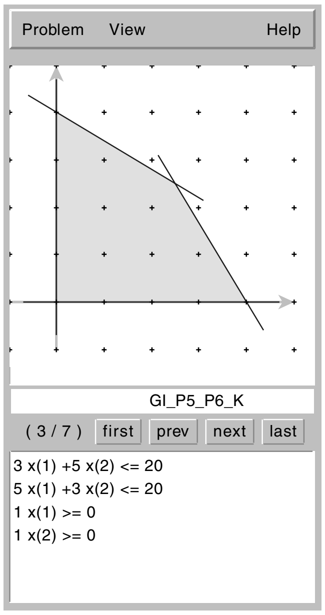
Figure 3. Projections of convexified regions onto the original problem space $\mathbb{R}^2$. (a) After selecting u_2^2. (b) After selecting u_1^2. (c) After selecting u_0^2. (d) After selecting u_2^1. (e) After selecting u_1^1. (f) After selecting u_0^1.

(a)



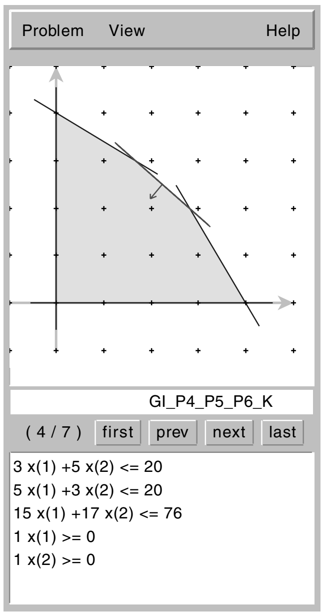
```
3 x(1) +5 x(2) <= 20
5 x(1) +3 x(2) <= 20
1 x(1) >= 0
1 x(2) >= 0
```

(b)



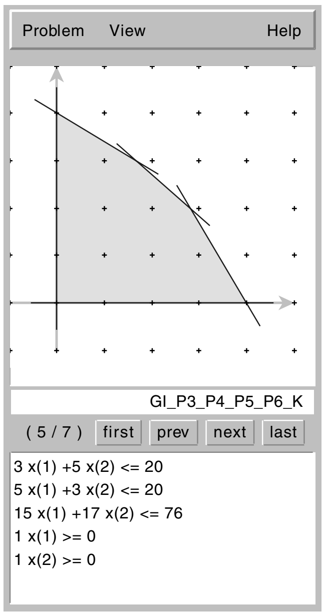
```
3 x(1) +5 x(2) <= 20
5 x(1) +3 x(2) <= 20
1 x(1) >= 0
1 x(2) >= 0
```

(c)



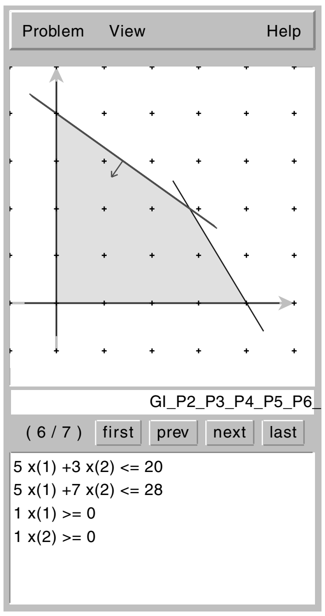
```
3 x(1) +5 x(2) <= 20
5 x(1) +3 x(2) <= 20
15 x(1) +17 x(2) <= 76
1 x(1) >= 0
1 x(2) >= 0
```

(d)



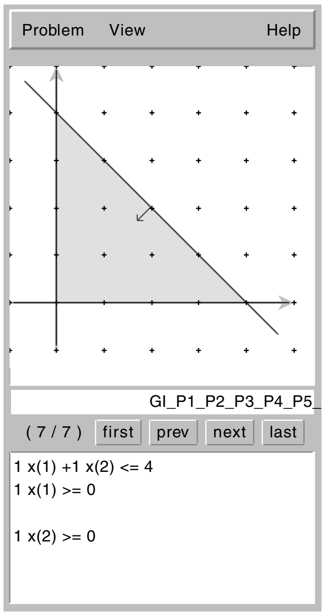
```
3 x(1) +5 x(2) <= 20
5 x(1) +3 x(2) <= 20
15 x(1) +17 x(2) <= 76
1 x(1) >= 0
1 x(2) >= 0
```

(e)



```
5 x(1) +3 x(2) <= 20
5 x(1) +7 x(2) <= 28
1 x(1) >= 0
1 x(2) >= 0
```

(f)



```
1 x(1) +1 x(2) <= 4
1 x(1) >= 0

1 x(2) >= 0
```

The following is a technical lemma. It quantifies the range of values x_j can take as its highest-order expansion variables are restricted to binary values.

Lemma 7. *For any $x_j \in \mathbb{R}$ such that $x_j = \sum_{i=0}^{k_j} 2^i u_i^j$, where $u_i^j \in [0, 1]$ for all $i \in \{0, 1, \ldots, k_j\}$ and $u_i^j \in \{0, 1\}$ for $i \in \{k_j, k_j - 1, \ldots, k_j - \delta\}$,*

$$x_j \notin \bigcup_{\beta=1}^{2^{\delta+1}-1} \left(\beta 2^{k_j-\delta} - 1, \beta 2^{k_j-\delta}\right),$$

where $\delta \in \{0, 1, \ldots, k_j\}$.

Proof. We prove the result by induction on the value of δ. As the base case, consider $\delta = 0$. In this case, $u_{k_j}^j \in \{0, 1\}$. Since $x_j = \sum_{i=0}^{k_j} 2^i u_i^j$ and $u_i^j \in [0, 1]$ for all $i \in \{0, 1, \ldots, k_j\}$, we know that $x_j \in [0, 2^{k_j+1} - 1]$. Thus, for $u_{k_j}^j \in \{0, 1\}$, the value x_j is either in the interval $[0, 2^{k_j} - 1]$ or in the interval $[2^{k_j}, 2^{k_j+1} - 1]$, when $u_{k_j}^j = 0$ or $u_{k_j}^j = 1$, respectively. It follows that x_j is not in the interval $(2^{k_j} - 1, 2^{k_j})$ when $u_{k_j}^j \in \{0, 1\}$, thus the result holds for $\delta = 0$. For the induction step, first assume that the claim is true for some arbitrary $\delta \in \{0, 1, \ldots, k_j - 1\}$. We will show that the result then holds for $\delta + 1$. By our assumption, we have that $x_j \notin \bigcup_{\beta=1}^{2^{\delta+1}-1}(\beta 2^{k_j-\delta}-1, \beta 2^{k_j-\delta})$, which is equivalent to the statement that $x_j \in \bigcup_{\beta=0}^{2^{\delta+1}-1}[\beta 2^{k_j-\delta}, (\beta+1)2^{k_j-\delta}-1]$. Observe that for each $\beta \in \{0, 1, \ldots, 2^{\delta+1}-1\}$, any value of x_j in the interval $[\beta 2^{k_j-\delta}, (\beta+1)2^{k_j-\delta}-1]$ is attainable only for fixed values of $u^j_{k_j}, u^j_{k_j-1}, \ldots, u^j_{k_j-\delta}$. In particular, if $x_j \in [\beta 2^{k_j-\delta}, (\beta+1)2^{k_j-\delta}-1]$, then for $i \in \{k_j, k_j-1, \ldots, k_j-\delta\}$, u^j_i is equal to the (k_j-i+1)th most significant digit of the $(\delta+1)$-digit binary representation of the integer $\beta 2^{k_j-\delta}$. Now let $\hat{u}^j_{k_j}, \hat{u}^j_{k_j-1}, \ldots, \hat{u}^j_{k_j-\delta}$ be fixed at the appropriate values for some arbitrary $\hat{\beta} \in \{0, 1, \ldots, 2^{\delta+1}-1\}$, and consider $\hat{u}^j_{k_j-(\delta+1)} \in \{0, 1\}$. Let (F_0, F_1) be the (unique) partition of the indices $\{k_j, k_j-1, \ldots, k_j-\delta\}$ such that $i \in F_0 \Leftrightarrow \hat{u}^j_i = 0$ and $i \in F_1 \Leftrightarrow \hat{u}^j_i = 1$. If $x_j \in [\hat{\beta}2^{k_j-\delta}, (\hat{\beta}+1)2^{k_j-\delta}-1]$, then it follows that

$$x_j = \underbrace{\sum_{i\in F_0} 2^i \hat{u}^j_i}_{=0} + \underbrace{\sum_{i\in F_1} 2^i \hat{u}^j_i}_{=\hat{\beta}2^{k_j-\delta}} + 2^{k_j-(\delta+1)}\hat{u}^j_{k_j-(\delta+1)} + \underbrace{\sum_{i=0}^{k_j-(\delta+2)} 2^i \hat{u}^j_i}_{\in[0,\,2^{k_j-(\delta+1)}-1]}.$$

It thus follows that if $\hat{u}^j_{k_j-(\delta+1)} = 0$, then

$$\begin{aligned} x_j &\in [\hat{\beta}2^{k_j-\delta}, \hat{\beta}2^{k_j-\delta} + 2^{k_j-(\delta+1)} - 1] \\ &\in [2\hat{\beta}(2^{k_j-(\delta+1)}), (2\hat{\beta}+1)(2^{k_j-(\delta+1)}) - 1], \end{aligned}$$

and if $\hat{u}^j_{k_j-(\delta+1)} = 1$, then

$$\begin{aligned} x_j \in [&\hat{\beta}(2^{k_j-\delta}) + 2^{k_j-(\delta+1)}, \hat{\beta}(2^{k_j-\delta}) + 2^{k_j-(\delta+1)} \\ &+ 2^{k_j-(\delta+1)} - 1] \\ \in [&(2\hat{\beta}+1)2^{k_j-(\delta+1)}, (2\hat{\beta}+2)2^{k_j-(\delta+1)} - 1], \end{aligned}$$

for each $\hat{\beta} \in \{0, 1, \ldots, 2^{\delta+1}-1\}$. It follows that

$$\begin{aligned} x_j \in \bigcup_{\beta=0}^{2^{\delta+1}-1} \Big(&\big[2\beta(2^{k_j-(\delta+1)}), (2\beta+1)(2^{k_j-(\delta+1)}) - 1\big] \\ &\cup \big[(2\beta+1)(2^{k_j-(\delta+1)}), (2\beta+2)(2^{k_j-(\delta+1)}) - 1\big]\Big), \end{aligned}$$

which implies that

$$x_j \in \bigcup_{\beta=0}^{2^{\delta+2}-1} [\beta 2^{k_j-(\delta+1)}, (\beta+1)2^{k_j-(\delta+1)} - 1].$$

This is equivalent to the statement that

$$x_j \notin \bigcup_{\beta=1}^{2^{\delta+2}-1} \big(\beta 2^{k_j-(\delta+1)} - 1, \beta 2^{k_j-(\delta+1)}\big),$$

thus the claim is true for $\delta+1$. The result follows. □

In Theorem 8 we show that it is necessary to generate the hull in the (x, u)-space with respect to the highest-order expansion variable, in order to see any tightening of the set in x-space. In Theorem 9 we show that the most effective approach for tightening the set in x-space is to generate the convex hull in the (x, u)-space with the highest-order variables first.

Theorem 8. *For any index set* $\{i_0, i_1, \ldots, i_l\} \subset \{0, 1, \ldots, k_j\}$ *such that* $k_j \notin \{i_0, i_1, \ldots, i_l\}$, *let*

$$S = \mathscr{C}^j_{i_l}(\mathscr{C}^j_{i_{l-1}}(\ldots(\mathscr{C}^j_{i_0}(K_B))\ldots)).$$

Then, $\text{Proj}_x(S) = K$.

Proof. Let $\bar{x}$ be an arbitrary extreme point of K. First, suppose that $\bar{x}_j \in [0, 2^{k_j}]$. Since $k_j \notin \{i_0, i_1, \ldots, i_l\}$, we have that $(\bar{x}, \bar{u}) \in S$, where $\bar{u}^j_i = 0$ for $i = 0, 1, \ldots, k_j - 1$ and $\bar{u}^j_{k_j} = \bar{x}_j/2^{k_j}$. It follows that $\bar{x} \in \text{Proj}_x(S)$ for any $\bar{x} \in \text{ext}(K)$ such that $\bar{x}_j \in [0, 2^{k_j}]$. Now suppose that $\bar{x}_j \in (2^{k_j}, 2^{k_j+1}-1]$. Since $k_j \notin \{i_0, i_1, \ldots, i_l\}$, we have that $(\bar{x}, \bar{u}) \in S$, where $\bar{u}^j_i = 1$ for $i = 0, 1, \ldots, k_j - 1$ and $\bar{u}^j_{k_j} = (\bar{x}_j - (2^{k_j}-1))/2^{k_j}$. It follows that $\bar{x} \in \text{Proj}_x(S)$ for any $\bar{x} \in \text{ext}(K)$ such that $\bar{x}_j \in [2^{k_j}, 2^{k_j+1}-1]$. Thus, since $\text{Proj}_x(S)$ is a convex set containing all extreme points of K, we have $K \subseteq \text{Proj}_x(S)$. Also since $\text{Proj}_x(S) \subseteq \text{Proj}_x(K_B) = K$, we have $\text{Proj}_x(S) = K$ for all $\{i_0, i_1, \ldots, i_l\} \subseteq \{0, 1, \ldots, k_j\}$ such that $k_j \notin \{i_0, i_1, \ldots, i_l\}$. □

Theorem 9. *For any index set* $\{i_0, i_1, \ldots, i_l\} \subseteq \{0, 1, \ldots, k_j\}$ *such that* $k_j \in \{i_0, i_1, \ldots, i_l\}$, *let*

$$S = \mathscr{C}^j_{i_l}(\mathscr{C}^j_{i_{l-1}}(\ldots(\mathscr{C}^j_{i_0}(K_B))\ldots))$$

and

$$S_\delta = \mathscr{C}^j_{k_j-\delta}(\mathscr{C}^j_{k_j-\delta+1}(\ldots(\mathscr{C}^j_{k_j}(K_B))\ldots)),$$

where δ *is the largest integer such that* $\{k_j, k_j-1, \ldots, k_j-\delta\} \subseteq \{i_0, i_1, \ldots, i_l\}$. *Then,* $\text{Proj}_x(S_\delta) = \text{Proj}_x(S)$.

Proof. By construction, since $\{k_j, k_j-1, \ldots, k_j-\delta\} \subseteq \{i_0, i_1, \ldots, i_l\}$, we have that $S \subseteq S_\delta$. It follows that $\text{Proj}_x(S) \subseteq \text{Proj}_x(S_\delta)$. In the following, we show that $\text{Proj}_x(S_\delta) \subseteq \text{Proj}_x(S)$, and thus $\text{Proj}_x(S_\delta) = \text{Proj}_x(S)$. Let $\hat{x}$ be an arbitrary extreme point of $\text{Proj}_x(S_\delta)$. Since $\hat{x} \in \text{ext}(\text{Proj}_x(S_\delta))$, by Lemma 5 there exists some $\hat{u}$ such that $(\hat{x}, \hat{u}) \in \text{ext}(S_\delta)$. By Lemma 7, we know that $\hat{x}_j \notin \bigcup_{\beta=1}^{2^{\delta+1}-1}(\beta 2^{k_j-\delta}-1, \beta 2^{k_j-\delta})$, which implies that $\hat{x}_j$ is in the interval $[\beta 2^{k_j-\delta}, (\beta+1)2^{k_j-\delta}-1]$ for some $\beta \in \{0, 1, \ldots, 2^{\delta+1}-1\}$. If we restrict $u^j_i \in \{0, 1\}$, $i \in \{k_j, k_j-1, \ldots, k_j-\delta\}$, then observe that any value of $\hat{x}_j$ in this interval is attainable only for fixed values of $\hat{u}^j_i$ for $i \in \{k_j, k_j-1, \ldots, k_j-\delta\}$. In particular, if $\hat{x}_j \in [\hat{\beta}2^{k_j-\delta}, (\hat{\beta}+1)2^{k_j-\delta}-1]$, then for $i \in \{k_j, k_j-1, \ldots, k_j-\delta\}$, $\hat{u}^j_i$ is equal to the (k_j-i+1)th most significant digit of the $(\delta+1)$-digit binary representation of the number $\hat{\beta}2^{k_j-\delta}$. Let (F_0, F_1) be the (unique) partition of the indices $\{k_j, k_j-1, \dots, k_j-\delta\}$ such that $i \in F_0 \Leftrightarrow \hat{u}_i^j = 0$ and $i \in F_1 \Leftrightarrow \hat{u}_i^j = 1$. Then,

$$\hat{x}_j = \underbrace{\sum_{i\in F_0} 2^i \hat{u}_i^j}_{=0} + \underbrace{\sum_{i\in F_1} 2^i \hat{u}_i^j}_{=\hat{\beta}2^{k_j-\delta}} + 2^{k_j-(\delta+1)}\hat{u}_{k_j-(\delta+1)}^j + \underbrace{\sum_{i=0}^{k_j-(\delta+2)} 2^i \hat{u}_i^j}_{\in[0,2^{k_j-(\delta+1)}-1]}.$$

First suppose that $\hat{x}_j - \hat{\beta}2^{k_j-\delta} \leqslant 2^{k_j-(\delta+1)}$. In this case, let $\varepsilon = \hat{x}_j - \hat{\beta}2^{k_j-\delta}$ and take $\hat{u}$ such that

$\hat{u}_i^j = 0 \quad \forall\, i \in F_0,$

$\hat{u}_i^j = 1 \quad \forall\, i \in F_1,$

$\hat{u}_i^j = 0 \quad \forall\, i \in \{0, 1, \dots, k_j-(\delta+2)\},$ and

$\hat{u}_{k_j-(\delta+1)}^j = \varepsilon/2^{k_j-(\delta+1)}.$

Since $\hat{u}_i^j \in \{0,1\}$ for all $i \in \{i_0, i_1, \dots, i_l\}$, it is clear that $(\hat{x}, \hat{u}) \in S$, thus $\hat{x} \in \text{Proj}_x(S)$. Now suppose that $\hat{x}_j - \hat{\beta}2^{k_j-\delta} > 2^{k_j-(\delta+1)}$. In this case, let $\varepsilon = \hat{x}_j - \hat{\beta}2^{k_j-\delta} - 2^{k_j-(\delta+1)}$ and take $\hat{u}$ such that

$\hat{u}_i^j = 0 \quad \forall\, i \in F_0,$

$\hat{u}_i^j = 1 \quad \forall\, i \in F_1,$

$\hat{u}_i^j = 1 \quad \forall\, i \in \{0, 1, \dots, k_j-(\delta+2)\},$ and

$\hat{u}_{k_j-(\delta+1)}^j = \varepsilon/2^{k_j-(\delta+1)}.$

Since $\hat{u}_i^j \in \{0,1\}$ for all $i \in \{i_0, i_1, \dots, i_l\}$, it is clear that $(\hat{x}, \hat{u}) \in S$, thus $\hat{x} \in \text{Proj}_x(S)$. Since $\hat{x}_j \in [\hat{\beta}2^{k_j-\delta}, (\hat{\beta}+1)2^{k_j-\delta}-1]$ for some $\hat{\beta} \in \{0, 1, \dots, 2^{\delta+1}-1\}$, we have shown that there exists some corresponding $\hat{u}$ such that $(\hat{x}, \hat{u}) \in S$. Now the inclusion $\text{Proj}_x(S_\delta) \subseteq \text{Proj}_x(S)$ follows from the convexity of sets $S, S_\delta, \text{Proj}_x(S)$, and $\text{Proj}_x(S_\delta)$, and the use of the representation theorem (Bazaraa et al. 1993, Theorem 2.6.7) The result follows. □

Theorem 3 and Corollary 10 follow directly from Theorem 9.

Corollary 10. *Cuts in the x-space of MILP are generated earliest if the convexification order for binary expansion variables of x_j is given as $\{i_0, i_1, \dots, i_{k_j}\} = \{k_j, k_j-1, \dots, 0\}$.*

4.2. Convexification for All Integer Values

In the previous section we showed that the best convexification order for the binary expansion variables of x_j is given as $(u_{k_j}^j, u_{k_j-1}^j, \dots, u_0^j)$. Intuitively, this order makes sense because at each step we are generating the convex hull with respect to the expansion variable with the largest coefficient that we have not yet considered, and thus we are restricting the value of x_j as much as possible at each step. As a result, it may seem that we would typically not need to generate the convex hull with respect to all of the expansion variables in order to eliminate the fractional extreme points of K, and hence we may be able to generate the convex hull earlier by considering (all) expansion variables with the largest coefficients first. The following theorem contradicts this expectation.

Theorem 11. *It is necessary to generate the convex hull with respect to all expansion variables u_i^j, $i \in \{0, 1, \dots, k_j\}$, in order to eliminate more than half of the possible ranges for extreme points in K that are fractional in x_j.*

Proof. This theorem follows from Lemma 7. In particular, suppose that we have generated the convex hull in the lifted space with respect to u_i^j for all $i \in \{1, 2, \dots, k_j\}$. In the corresponding projection, we have eliminated only the fractional extreme points with $x_j \in \bigcup_{\beta=1}^{2^{k_j}-1}(\beta 2^{k_j-\delta}-1, \beta 2^{k_j-\delta})$, a range of size $2^{k_j}-1$ units. Thus, since the range of $x_j = \sum_{i=0}^{k_j} 2^i u_i^j$ is a total of size $2^{k_j+1}-1$ units, by considering all expansion variables except for u_0^j we have eliminated less than half of the possible ranges for extreme points in K that are fractional in x_j. □

The above result, together with Theorem 3, has important implications for practitioners. It suggests that remodeling integer variables in MILP using additional binary variables results in a problem for which almost all the binary variables need to be explored in the standard branching process, or while generating cuts using variable disjunctions in a branch-and-cut algorithm. Therefore, such reformulations should be avoided, unless special techniques are used to handle these variables. Note however, that the need to explore all the binary variables corresponding to an integer variable does not mean that the size of the branch-and-bound tree will grow exponentially in these variables, since large segments of this tree would be pruned easily.

4.3. Full-Expansion Reformulation

The binary reformulation of MILP used by Sherali and Adams (1999) defines each integer variable x_j by

$$x_j = \sum_{i=0}^{M_j} i v_i^j,$$

$$\sum_{i=0}^{M_j} v_i^j = 1,$$

where variables $v_i^j \in \mathbb{B}$ for $i = 0, 1, 2, \dots, M_j$. We call this reformulation a full reformulation of MILP. A result similar to Theorem 3 is also valid for the full reformulation.

Theorem 12. *Assume that all binary variables corresponding to a general-integer variable are selected together for convexification and no prior efficient ordering of the x variables is known in MILP. Then, cuts in the x-space are generated earliest if the convexification order for binary expansion variables of x is given as*

$$\left(v_{M_{j_1}}^{j_1}, v_{M_{j_1}-1}^{j_1}, \dots, v_0^{j_1}, v_{M_{j_2}}^{j_2}, v_{M_{j_2}-1}^{j_2}, \dots, v_0^{j_2}, \dots, v_{M_{j_p}}^{j_p}, v_{M_{j_p}-1}^{j_p}, \dots, v_0^{j_p}\right)$$

for some ordering $(j_1, j_2, \ldots, j_p)$ of the variable indices $\{1, 2, \ldots, p\}$ of x in MILP.

Similar to the compact-reformulation case, this result is based on the result (left for the reader) that binary expansion variables corresponding to a general integer variable x_j need to be convexified in descending order to see tightening in the set obtained upon projection in the x-space.

This result suggests that in the full reformulation all binary variables corresponding to integer values larger than the optimal integer value need to be fixed during branching before we can prove optimality of a solution. Because of this phenomenon, in the branching process we are likely to consider many of the binary variables that replace an integer variable. The computational results in the next section confirm such behavior.

We also note that the reformulation-linearization technique of Sherali and Adams (1999) generates the convex hull with respect to all the binary variables u_i^j replacing a general integer variable x_j at once. The size of the linear program resulting from applying this technique to only one set of u_i^j is very large, and it is not clear if this technique can be used in practice.

5. COMPUTATIONAL EXPERIMENTS

The computational results presented in this section are based on problem instances from the MIPLIB-3.0 library (Bixby et al. 1998). For each problem instance considered, we generated both a compact- and full-expansion reformulation based on the variable bounds stated in the original problem file.[1] Each problem from the library that contains general-integer variables is considered except for `dsbmip` and `arki001`. The problem `dsbmip` is not considered because the value of each of its 32 general-integer variables is explicitly fixed. The problem `arki001` is excluded due to errors encountered when trying to solve its binary reformulations.[2] All problems were solved using version 6.6 of the general purpose solver CPLEX (ILOG Inc. 1998) with the default settings.

Table 1 gives a summary of the problem instances and computational results. The third column gives the total number of variables, and columns 4 and 5, respectively, give the number of general- and binary-integer variables for each instance. The penultimate column shows the number of branch-and-bound nodes required to solve the problem as reported by the CPLEX library routine *CPXgetnodecnt()*. The last column shows the "user time" needed to solve the problem as returned by the UNIX system call *getrusage()*; times are reported for solution on a Sun Ultra-60 workstation with 384Mb RAM (and 1Gb virtual memory) running Solaris 5.7.

The first observation from the results in Table 1 is that solution times for binary reformulations were always more than the time required by the original formulation. This suggests that solving either of the two reformulations is not effective. In its worst relative performance (problem `gesa3_o`), the compact expansion solved approximately 19 times slower than the original formulation. For problem `bell5` the full expansion solved 1,088 times slower than the original formulation.

The compact expansion was solved faster than the full expansion on 10 problems, while the full expansion was solved faster on the remaining four. The compact expansion of problem `bell3a` was solved 194 times faster than its full expansion, while the full expansion of problem `gesa3_o` was solved 3.7 times faster than its compact expansion. Although compact expansion generally seems to perform better, these results suggest that superiority of either of the two reformulations is not predictable.

A further analysis of the computational results in Table 1 for the compact expansion show a near-linear growth in the number of examined nodes in the branch-and-bound enumeration tree with growth in the binary variables due to reformulation. For this expansion the growth in the number of examined nodes is a good indicator for the increase in required CPU time to solve the problem. The growth in the number of nodes for the full expansion is less predictable. Also, for the full expansion the growth in CPU time as a function of growth in the number of examined nodes is less predictable. Although the problems in the MIPLIB test set have very different characteristics, our computational results on these problems support the performance indicated from the theory developed in the previous section.

As noted earlier, the binary reformulations for problems in Table 1 were generated using the bounds provided in the original MIPLIB problem files. It is also possible to generate implied bounds by solving linear programs corresponding to all general-integer variables. We performed a second set of experiments where we generated reformulations using these implied bounds. Our qualitative conclusions from the runs using the implied bounds are the same.

6. CONCLUSIONS

We have given theoretical explanations that suggest that binary reformulations of mixed-integer linear programs are not a practical alternative for solving these problems. With the help of computational results we showed that the solution times for these reformulations generally grow linearly with the increase in the number of binary variables introduced during reformulation. The performance of the full reformulation is less predictable; however, computational results suggest that it frequently performs worse than the compact reformulation.

We point out that our theoretical and computational results are under the assumption that the structure of constraints on binary variables in the reformulation is not exploited in the branching process. It may be possible to exploit the structure of the binary variables introduced in the problem reformulation. For example, in full expansion the resulting problem introduces a special-ordered (type 1) constraint. In order to exploit this in CPLEX one

Table 1. Computational results.

Problem Name	Problem Formulation	Total Number of Variables	General-Integer Variables	Binary Variables	Branch-and-Bound Nodes	Solution Time
bell3a	original	133	32	39	21,358	48.03
	compact expansion	453	0	359	100,165	223.67
	full expansion	32,165	0	32,071	2,460,931	43,563.30
bell5	original	104	28	30	7,999	4.99
	compact expansion	398	0	324	24,155	28.19
	full expansion	94,732	0	94,658	486,061	5,431.48
blend2	original	353	33	231	1,831	7.62
	compact expansion	407	0	285	5,783	20.77
	full expansion	444	0	322	1,821	8.53
flugpl	original	18	11	0	318	0.08
	compact expansion	73	0	55	395	0.24
	full expansion	227	0	209	688	0.62
gen	original	870	6	144	1	0.15
	compact expansion	900	0	174	4	0.22
	full expansion	1,020	0	294	29	0.76
gesa2_o	original	1,224	336	384	75,819	505.82
	compact expansion	1,944	0	1,104	132,169	1,341.41
	full expansion	2,424	0	1,584	122,094	1,278.63
gesa2	original	1,224	168	240	19,997	151.31
	compact expansion	1,584	0	600	49,102	376.72
	full expansion	1,824	0	840	137,324	1,126.37
gesa3_o	original	1,152	336	336	1,371	13.81
	compact expansion	1,920	0	1,104	28,110	261.68
	full expansion	2,496	0	1,680	5,365	71.47
gesa3	original	1,152	168	216	588	9.64
	compact expansion	1,536	0	600	889	12.48
	full expansion	1,824	0	888	7,161	72.97
gt2	original	188	164	24	2,277	1.92
	compact expansion	720	0	556	1,208	3.36
	full expansion	1,500	0	1,336	50,875	248.29
noswot	original	128	25	75	8,090,079	8,671.98
	compact expansion	553	0	500	5,864,631	11,649.50
	full expansion$^{(a)}$	240	0	192	8,242,054	13,248.90
qnet1_o	original	1,541	129	1,288	440	8.16
	compact expansion	1,913	0	1,660	973	15.91
	full expansion	2,345	0	2,092	1,938	38.30
qnet1	original	1,541	129	1,288	233	10.61
	compact expansion	1,913	0	1,660	319	13.55
	full expansion	2,345	0	2,092	610	18.83
rout	original	556	15	300	177,353	4,369.09
	compact expansion	586	0	330	269,318	10,396.90
	full expansion	601	0	345	114,282	4,578.15

(a)—full expansion using implied bounds; see endnote #1.

needs to give preassigned weights on variables in special-ordered branching (ILOG Inc. 1998). In our context, the natural choice for these weights are the integer values corresponding to each binary variable introduced while writing full expansions. This choice of weights results in a branching that is equivalent (provided the branching order of variables in x-space and (x, u)-space is same) to branching on the value of fractional integer variables in MILP directly. A branching scheme with similar properties can also be constructed for the compact expansion.

The comparative performance of methods working directly on MILP and compact- and full-expansion reformulations under different schemes for exploiting the binary structure remains a topic of future research.

ENDNOTES

1. The full-expansion reformulation for the problem `noswot` could not be solved using variable upper bounds from the original problem. For this instance, we generated

a full-expansion reformulation that was based on implied variable bounds determined by solving linear programs corresponding to each general integer variable.

2. The solver terminated with a nonoptimal objective value for the full expansion, and an attempt to solve the compact expansion of `arki001` was interrupted after several days when the solver ran out of memory.

ACKNOWLEDGMENTS

The authors sincerely thank an anonymous referee and the associate editor handling this paper for several constructive suggestions. In particular, the discussion on full formulation, and the computational section, were added due to their comments.

The work of the second author was supported in part by ONR grant N00014-01-1-0048 and NSF grants DMI-9500396 and DMI-9908038.

REFERENCES

Balas, E. 1979. Disjunctive programming. *Ann. Discrete Math.* **5** 3–51.

——, S. Ceria, G. Cornuéjols. 1993. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Math. Programming* **58** 295–324.

——, ——, ——. 1996. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Sci.* **42**(9) 1229–1246.

Bazaraa, M. S., H. D. Sherali, C. M. Shetty. 1993. *Nonlinear Programming Theory and Algorithms*, 2nd ed. John Wiley and Sons, New York.

Bixby, R. E., S. Ceria, C. M. McZeal, M. W. P. Savelsbergh. 1998. An updated mixed integer programming library: MIPLIB 3.0. Technical Report TR98-03, The Department of Computational and Applied Mathematics, Rice University, Houston, TX.

Christof, T., A. Löebel. 1997. PORTA: Polyhedron representation transformation algorithm. Version 1.3.1 of the software and documentation distributed via the Internet ⟨http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/soft.html⟩.

ILOG Inc. 1998. *Using the CPLEX Callable Library, Version 6.0*. ILOG Inc.

Owen, J. H., S. Mehrotra. 2001. A disjunctive cutting plane procedure for general mixed-integer linear programs. *Math. Programming* **89**(3) 437–448.

Rothberg, E. 2000. Using cuts to remove symmetry. Presentation at 17th International Symposium on Mathematical Programming, Atlanta, GA.

Sherali, H. D., W. P. Adams. 1990. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discrete Math.* **3**(3) 411–430.

——, ——. 1994. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems. *Discrete Appl. Math.* **52** 83–106.

——, ——. 1999. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*, Ch, 4. Kluwer Academic Publishers, Norwell, MA.

Stubbs, R. A., S. Mehrotra. 1999. A branch-and-cut method for 0-1 mixed convex programming. *Math. Programming* **86**(3) 515–532.