ELSEVIER

Operations Research Letters 19 (1996) 1–9

operations research letters

Gomory cuts revisited[1]

E. Balas [a], S. Ceria [b,2], G. Cornuéjols [a,*], N. Natraj [c]

[a] *GSIA, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[b] *Graduate School of Business, Columbia University, NY 10027, USA*
[c] *US West Technologies, Boulder, CO 80303, USA*

Received 1 August 1994; revised 1 February 1996

Abstract

We investigate the use of Gomory's mixed integer cuts within a branch-and-cut framework. It has been argued in the literature that "a marriage of classical cutting planes and tree search is out of the question as far as the solution of large-scale combinatorial optimization problems is concerned" [16] because the cuts generated at one node of the search tree need not be valid at other nodes. We show that it is possible, by using a simple lifting procedure, to make Gomory cuts generated at a node of the enumeration tree globally valid in the case of mixed 0–1 programs. The procedure essentially amounts to treating the variables fixed at 0 or 1 as if they were free. We also show why this lifting procedure is not valid for general (other than 0–1) mixed integer programs. Other issues addressed in the paper are of a computational nature, such as strategies for generating the cutting planes, deciding between branching and cutting, etc. The result is a robust mixed integer program solver.

Keywords: Mixed 0–1 programming; Gomory cuts; Lift and project

1. Introduction

In the late fifties and early sixties, Gomory [6–8] proposed to solve integer programs by using cutting planes, thus reducing integer programming to the solution of a sequence of linear programs. After an initial enthusiasm for this idea, the research community became skeptical of its practical usefulness. Recently, lift-and-project cuts [1, 2] used in an appropriate context have been shown to be an effective way of solving mixed 0–1 programs. In this paper, we reconsider Gomory's mixed integer cuts in the light of new developments in linear and integer programming.

In his reminiscences of the early days of integer programming, Gomory [9] remembers his experience with fractional cuts. " In the summer of 1959, I joined IBM research and was able to compute in earnest... We started to experience the unpredictability of the computational results rather steadily." Subsequently, Gomory placed great hopes in his all-integer cuts, but they turned out to be uniformly inferior to fractional cuts. "This poor computational performance was a great disappointment to me" [9]. The fact that Gomory's fractional cuts outperform all-integer cuts is documented by several other authors [11, 18].

Nevertheless, the literature on computational experience with Gomory's fractional cuts remains scant. Only two papers on specially structured 0–1

* Corresponding author.

[1] Supported by NSF grants DMI-9424348 and DMS-9509581 and ONR contract N00014-89-J-1063.

[2] Research performed while at Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza" and IASI, CNR, Italy.

programs report success, namely Martin [13] on the set covering problem and Miliotis [14] on the travelling salesman problem. The textbooks of the eighties are generally negative. "Although cutting plane methods may appear mathematically fairly elegant, they have not proved very successful on large problems" (Williams [20]). "...they do not work well in practice. They fail because an extremely large number of these cuts frequently are required for convergence" (Nemhauser and Wolsey [15]). Different people offer different explanations of the perceived failure: "The main difficulty has come, not from the number of iterations, but from numerical errors in computer arithmetic" (Parker and Rardin [17]). "These cutting planes have poor convergence properties. While finite convergence can be proven, classical cutting planes furnish "weak" cuts" (Padberg and Rinaldi [16]).

These statements appear to be based on experiments dating back to the sixties. The next quote reflects the sentiment of the panel on cutting planes at the Discrete Optimization Symposium held in Vancouver and Banff in 1977. "There is a common contention that cutting plane methods have been a failure in practice... Solid computational success is practically limited to Martin's work on airline crew scheduling and the work on the travelling salesman problem. Yet the general concensus was that there are enough ideas and enough open avenues that cutting planes cannot be written off" [10].

Many options were proposed but few were tried. For example, when Gomory [8] discusses criteria for selecting a row to generate a cut, he also states "still another approach would be to throw on several or even many new inequalities." This is an excellent idea which we rediscovered 30 years later. Another interesting idea was proposed by Garfinkel and Nemhauser [5] but apparently never tried. "Although pure cutting plane algorithms appear to be of limited practical value, cuts can be very important in branch-and-bound algorithms. In the process of solving a mixed integer program by branch-and-bound, an alternative to separation is to generate mixed integer Gomory cuts ... Presumably, empirical rules could be established for choosing between these two options."

Of course, as pointed out by Padberg and Rinaldi [16], there is a big problem with trying to implement this strategy. At a generic node of the search tree, some variables have been fixed "and consequently the resulting Gomory cuts need not be valid for the original integer polyhedron. Using cuts which are not valid for the full integer polyhedron makes it necessary to treat and store separately the cuts related to each node of the search tree. This requires additional bookkeeping and a much larger amount of memory. Thus a marriage of classical cutting planes and tree search is out of the question as far as the solution of large-scale combinatorial optimization problems is concerned" [16]. This point is reemphasized in a recent paper of Hoffman and Padberg [12]: " To share the set of generated constraints across different branches of the search tree is mathematically simply *not correct* when traditional cutting planes of integer programming, such as Gomory cuts or intersection cuts, are used."

The main result of this paper is that Gomory cuts can be shared across different branches of the search tree for a mixed 0–1 program. Therefore, in the context of mixed 0–1 programming, the objections raised in [16] and [12] are moot. In the second part of the paper, we report on our computational experience with this approach.

2. Lifting Gomory's mixed integer cuts

In this section we prove our main result, namely a lifting theorem for Gomory cuts in the case of mixed 0–1 programs. A similar result for general (pure or mixed) integer programs is not known. Indeed, this is an intriguing open question.

Consider the mixed 0–1 program (MIP)

$$\begin{aligned}&\text{Min } cx\\&\text{subject to}\\&\quad Ax = b,\\&\quad x \geqslant 0,\\&\quad x_i \in \{0,1\},\quad i = 1,\dots,p,\end{aligned}$$

where the first p variables are 0–1 constrained and the remaining variables x_i, $i = p+1,\dots,n$, are continuous.

We denote by $F_0, F_1 \subseteq \{1,\dots,p\}$ the sets of variables that have been fixed at 0 and 1 in the search tree. In general, a cut generated at a node (F_0, F_1) of the search tree is valid only at those other nodes where the variables in $F_0 \cup F_1$ remain at their fixed values. When cuts can be made valid throughout the search tree, this not only reduces the need for extensive bookkeeping, but such "shared" cuts may improve the bounds at many nodes of the tree. A cut generated at node (F_0, F_1) is made valid for (MIP), and therefore for the whole search tree, by computing appropriate coefficients for the variables x_j for $j \in F_0 \cup F_1$. This operation is called *lifting* the cut.

Example 1.

$$\begin{aligned} \text{(MIP)}\ \text{Min}\quad & 3x_1 + x_2 + 3x_3 + 4x_4 \\ \text{s.t.}\quad & 2x_1 + 3x_2 + x_3 + x_4 = 4, \\ & x_i \in \{0,1\},\ i = 1,2,3, \\ & x_4 \geqslant 0. \end{aligned}$$

At the root node ($F_0 = F_1 = \emptyset$), the optimal LP solution is $x_1 = \frac{1}{2}$, $x_2 = 1$, $x_3 = x_4 = 0$. After branching on variable x_1, consider the LP solved at node ($F_0 = \emptyset$, $F_1 = \{1\}$):

$$\begin{aligned} \text{Min}\quad & x_2 + 3x_3 + 4x_4 \\ \text{s.t.}\quad & x_2 = \tfrac{2}{3} - \tfrac{1}{3}x_3 - \tfrac{1}{3}x_4, \\ & 0 \leqslant x_2, x_3 \leqslant 1, \\ & x_4 \geqslant 0. \end{aligned}$$

The Gomory mixed integer cut generated at this node is

$$x_3 + x_4 \geqslant 2.$$

It is only valid at those nodes of the search tree where $x_1 = 1$. In order to make this cut valid for MIP (i.e. globally valid throughout the search tree), it must be lifted, i.e. a coefficient must be calculated for x_1:

$$-x_1 + x_3 + x_4 \geqslant 1.$$

In this section we show how to lift mixed integer Gomory cuts for the case of mixed 0–1 programs. We describe the basic steps to be followed in generating a mixed integer Gomory cut at a node (F_0, F_1) of the search tree, and lifting it so that it becomes valid throughout the branch-and-cut tree.

Consider the linear program LP(F_0, F_1) at a generic node of the search tree, and its associated basic solution

$$\begin{aligned} & x_i = \bar{a}_{i0} + \sum_{j \in J} \bar{a}_{ij}(-x_{ij}) \quad \text{for all } i \in I, \\ & x_k \geqslant 0 \quad \text{for all } k \in I \cup J, \\ & x_k \leqslant 0 \quad \text{for all } k \in F_0, \\ & x_k \geqslant 1 \quad \text{for all } k \in F_1, \end{aligned} \tag{1}$$

where I and J denote the sets of basic and nonbasic variables, respectively. We assume w.l.o.g. that $F_0 \cup F_1 \subseteq J$. As in the case of most branch-and-bound and branch-and-cut implementations, we assume that the fixing of the variables (in the sets F_0 and F_1) is performed by changing the upper and lower bounds, not by eliminating the fixed variables from the formulation.

By complementing all the variables x_k in F_1 (i.e. replacing x_k by $1 - x_k$) we get a new system where all the fixed variables are at 0. Thus we can assume w.l.o.g. that in (1), $F_1 = \emptyset$. We adopt this assumption so as to simplify the notation.

As usual, we denote by $\lfloor \bar{a}_{ij} \rfloor$ and f_{ij} the integer part and the fractional part of $\bar{a}_{ij}$, respectively. The next theorem shows that deriving a Gomory cut while treating the fixed variables as if they were free, yields an inequality valid throughout the search tree.

Theorem. *For $i \in I$, $i \leqslant p$, the Gomory mixed integer inequality $\gamma x \geqslant 1$ cuts off the point $\bar{x}$ and is valid for* (MIP), *where*

$$\gamma_j = \begin{cases} \min\left(\dfrac{f_{ij}}{f_{i0}}, \dfrac{1 - f_{ij}}{1 - f_{i0}}\right) & j \in J,\ j \leqslant p, \\[2ex] \max\left(\dfrac{\bar{a}_{ij}}{f_{i0}}, \dfrac{-\bar{a}_{ij}}{1 - f_{i0}}\right) & j \in J,\ j \geqslant p + 1, \\[2ex] 0 & j \in I. \end{cases}$$

Proof. We need some notation. Let $J_1 := J \cap \{1, \ldots, p\}$, $J_2 = J \setminus J_1$, $J_1^+ = \{j \in J_1 : f_{ij} < f_{i0}\}$, $J_1^- = J_1 \setminus J_1^+$, $J_2^+ = \{j \in J_2 : \bar{a}_{ij} > 0\}$ and $J_2^- = J_2 \setminus J_2^+$.

Assume first that $F_0 = \emptyset$. Gomory's mixed integer cut can be derived in three steps, using a standard argument.

First, by substituting $\lfloor \bar{a}_{ij} \rfloor + f_{ij}$ for $\bar{a}_{ij}$, $j \in J_1^+$, and $\lfloor -\bar{a}_{ij} \rfloor + 1 - f_{ij}$ for $-\bar{a}_{ij}$, $j \in J_1^-$, one obtains the congruence

$$\sum_{j\in J_1^+} f_{ij}x_j + \sum_{j\in J_1^-} (f_{ij}-1)x_j + \sum_{j\in J_2^+} \bar{a}_{ij}x_j + \sum_{j\in J_2^-} \bar{a}_{ij}x_j \equiv f_{i0} \pmod 1. \qquad (2)$$

Second, (2) and the nonnegativity of the variables implies that at least one of the inequalities

$$\sum_{j\in J_1^+} f_{ij}x_j + \sum_{j\in J_2^+} \bar{a}_{ij}x_j \geqslant f_{i0} \qquad (3)$$

or

$$\sum_{j\in J_1^-} (f_{ij}-1)x_j + \sum_{j\in J_2^-} \bar{a}_{ij}x_j \leqslant f_{i0} - 1 \qquad (4)$$

must hold.

Third, multiplying (4) by -1 and dividing each inequality by its right hand side produces two inequalities (3′) and (4′), with a nonnegative left hand side and a right hand side of 1, at least one of which must hold. Adding the left hand sides of (3′) and (4′) yields the valid inequality $\gamma x \geqslant 1$.

Now, let $F_0 \neq \emptyset$. The congruence of the first step is derived from the fact that the integer parts of the coefficients, multiplied with integers, add up to an integer, and this remains true. The disjunction between (3) and (4) in the second step follows from (2) because all the terms on the left hand side of (3) are nonnegative, and all the terms on the left hand side of (4) are nonpositive. But this still remains the case if $F_0 \neq \emptyset$. Finally, the multiplications of Step 3 are clearly unaffected by the fact that $F_0 \neq \emptyset$.

Thus the inequality $\gamma x \geqslant 1$ is valid. □

The fact that (MIP) is a mixed 0–1 program is key to this argument. Unfortunately, the argument does not carry over to general mixed integer programs, since the slack variables of a variable fixing constraint, like $x_j \leqslant \lfloor \bar{x}_j \rfloor$ or $x_j \geqslant \lceil \bar{x}_j \rceil$, cease to be nonnegative when the variable x_j is set free from a value $\lfloor \bar{x}_j \rfloor$ or $\lceil \bar{x}_j \rceil$ that ceases to be its lower or upper bound.

Next we illustrate the fact that in the case of a general (as opposed to 0–1) mixed integer program the above theorem is not valid.

Example 2. Consider the problem [5, p. 115]

$$\begin{aligned} \min\ & 7x + 3y + 4z \\ \text{s.t.}\ & x + 2y + 3z \geqslant 8, \\ & 3x + y + z \geqslant 5, \end{aligned}$$

whose optimal LP solution is $x = 2/5$, $y = 19/5$, $z = 0$. Branching on $y \leqslant 3$ and reoptimizing yields the solution $x = z = \frac{1}{2}$, $y = 3$ where the row corresponding to x is

$$x = \tfrac{1}{2} - \tfrac{1}{8}s_1 + \tfrac{3}{8}s_2 + \tfrac{1}{8}s_y.$$

Here s_1 and s_2 are the surplus variables associated with the two original inequalities, while s_y is the slack variable in $y \leqslant 3$. Generating a mixed integer Gomory cut from this row yields

$$\tfrac{1}{4}s_1 + \tfrac{3}{4}s_2 + \tfrac{1}{4}s_y \geqslant 1$$

or, expressed in terms of the original variables,

$$5x + 2y + 3z \geqslant 12.$$

But this cut, valid at the given node and its descendants in the search tree, is globally invalid. In particular, it cuts off the solution $x = z = 0$, $y = 5$, which happens to be the unique optimum.

3. Implementation

Empirical principles for the efficient use of general cutting planes within a branch-and-cut framework are developed in [2]. Here we adopt these principles, summarized below.

Traditionally, Gomory's algorithm generates one cut per iteration, adds it to the formulation and resolves. Of course, an alternative is to generate several cuts from the current basis (possibly one for every 0–1 variable which is currently fractional). An interesting issue is to decide how many cuts should be generated at each iteration. We define a *round* of cuts to be a set of cutting planes generated for some or all of the fractional 0–1 variables in the current solution to the

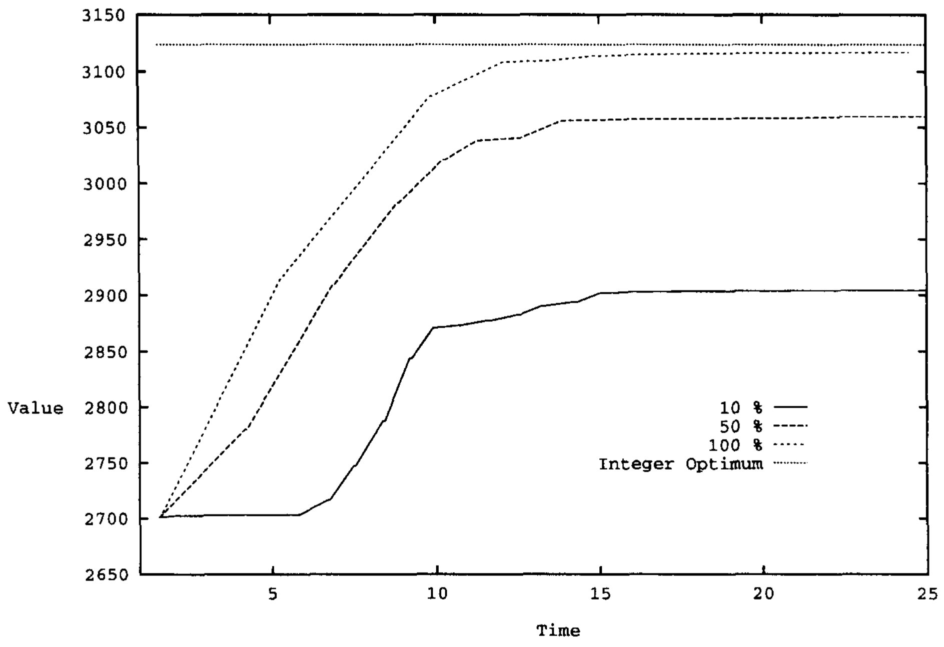

Fig. 1. Cutting plane comparison

linear program LP(F_0,F_1). For lift-and-project cuts, computational experience in [2] showed that, given a fixed amount of computing time, the best improvements in objective function value were found when cuts were generated for all fractional 0–1 variables. For mixed integer Gomory cuts, our experience is similar. To illustrate this, consider the graph of Fig. 1 obtained for problem instance P2756 (originating with [4]) as follows: First we ran 10 rounds of cuts at the node LP($\emptyset,\emptyset$), where in every round we generated a cut for every fractional 0–1 variable in the current LP relaxation solution. Let TTIME denote the total time taken to generate the cuts in these 10 rounds. We then ran for TTIME an experiment where in every round we generated cuts for 50% of the fractional components and for 10%. The fractional components closest to 1/2 are those selected to generate the cuts. It is clear from the figure that larger rounds are better: In 25s, 98% of the gap between the IP optimum value and the value of the initial LP relaxation is closed with large rounds whereas less than 50% is closed with small rounds. Furthermore, with small rounds, almost no progress is made after the first 10s of CPU time. We observed a similar behaviour with other instances.

When incorporating general cutting planes in a branch-and-cut framework, one has to face the key decision of when to stop generating cutting planes at a given node. In [2] we observed that, even though it is a good strategy to generate cuts in rounds, it is not necessarily best to generate them at every node of the enumeration tree. Define the *skip factor* to be the number of nodes of the enumeration tree that are enumerated before a new round of cuts is generated. For each instance, we determine the skip factor as a function of several parameters. The most important of these parameters is *cut quality*, which is measured by computing the euclidean distance between the hyperplane defining the cut and the point it cuts off. Whenever (MIP) is not solved by the initial LP relaxation, we generate a round of cuts at the root node of the enumeration tree. Let d be the average cut quality at the root node (as defined above) and f be the number of fractional 0–1 variables. Then the skip factor is determined by the following formula:

$$k = \min\left\{\mathrm{KMAX}, \left\lceil \frac{f}{cd\log_{10} p} \right\rceil\right\},$$

where KMAX and c are constants. In our implementation, we set KMAX = 32 and $c = 15$. All cuts are stored in a common *pool* and the nodes of the enumeration tree are retrieved using a best bound criterion. When a node is retrieved, the subproblem to be solved is constructed by adding to the original constraints the cuts from the pool that are tight for $\bar{x}$, the solution of the subproblem at the time when the node was generated, plus any cut from the pool that is violated by $\bar{x}$. These cuts are said to be *active*. In our implementation, we set the maximum pool size to 500 and when this limit is reached, cuts which are not active at any node are removed. If no such cut exists, the pool size is increased.

In the nodes where Gomory cuts are generated, we clean the main linear program as follows, after it has been retrieved and solved. We remove from the list of active constraints all Gomory cuts generated in previous iterations whose corresponding slack variables are basic at the current fractional point. The resulting linear program is then solved again. This "cleaning" of the main linear program has several positive effects. First and most importantly, it reduces the numerical instability of the basis, making the computation of Gomory cuts more accurate. Second, it keeps the linear programs small, thus reducing the computational effort as the LP is resolved after adding the cuts. Moreover, these cutting planes, and hence the smaller LP, will be inherited by the sons of the current node.

4. Computational experience

We use the same test-bed as in [2], where the bulk of the instances comes from MIPLIB [3]. All instances are preprocessed as explained in [2]. We compare two versions of the code, one where Gomory cuts are generated as described in Section 3, and one where no cuts are generated: all parameters which do not refer to cut generation are the same. The linear programs encountered during the procedure are solved using the CPLEX 2.1 callable library. In Table 1, m denotes the number of constraints, p the number of 0–1 variables and q the number of continuous variables. The times reported refer to seconds on an HP720 Apollo desktop workstation. Additional comparisons to OSL, CPLEXMIP, MINTO and MIPO can be obtained by looking at Table 8 in [2], where the same set of instances was run on the same Apollo workstation. This table is reproduced in the Appendix of this paper.

Table 1 proves the point we want to make in this paper, namely that the use of Gomory cuts within a branch-and-bound framework is possible and that it makes the resulting algorithm more robust (i.e. more instances can be solved).

One of the instances, namely MODGLOB, could not be solved by our branch-and-cut procedure because of numerical difficulties in cut generation. This problem is particularly poorly scaled, with several equality constraints involving data having ten significant digits. For the large scale instance L152LAV we had to reduce the maximum size of the pool to 100 because of space limitations. Indeed, the cuts generated tend to be fairly dense, and with a pool size of 500 we ran out of memory. For other large scale instances (AIR04 and AIR05), and for STEIN45, the algorithm did not finish because of space limitations.

Comparing the results in Table 1 to those of Table 8 in [2] (see the Appendix), we can make the following additional remarks.

Branch-and-cut with Gomory cuts (BCG for short) is able to solve more instances within the given time and space limitations than either OSL, CPLEXMIP or MINTO, though fewer than MIPO. The corresponding numbers are shown in Table 2.

Since the lift-and-project cuts used in MIPO are more elaborate and typically stronger than the Gomory cuts, it is not surprising that, as a rule, more cuts are generated per search tree node in the case of BCG than in that of MIPO. However, in terms of overall computing time, BCG does better than MIPO in 12 of the 29 instances, whereas MIPO does better in 16 cases (we excluded L152LAV for which the runs are not comparable, as explained above).

A natural question is to ask why Gomory cuts worked in our experience, while they did not work in the past. We believe that some of the reasons are:

- We generate cutting planes in rounds, rather than one at a time with reoptimization after each cut. As a result, the cuts tend to be less parallel to the objective function and each other.

Table 1
Computational results

Problem	Size $m \times (p+q)$	Pure branch-and-bound # of nodes	Pure branch-and-bound CPU time	Branch-and-cut with Gomory cuts # of nodes	Branch-and-cut with Gomory cuts # of cuts	Branch-and-cut with Gomory cuts CPU time
AIR04	823 ×(8904 + 0)	***	***	***	***	***
AIR05	426 ×(7195 + 0)	***	***	***	***	***
BM23	20 ×(27 + 0)	321	3.3	239	336	6.3
CFAT200-1	1919 ×(200 + 0)	85	455	83	167	597
CTN2	150 ×(120 + 120)	4975	163	179	888	52
CTN3	182 ×(142 + 142)	***	***	2013	8928	998
EGOUT	98 ×(55 + 86)	225	3.4	11	34	0.9
FXCH3	161 ×(141 + 141)	***	***	373	1555	117
GENOVA6	98 ×(904 + 0)	12485	1418	9425	5062	2466
L152LAV	97 ×(1989 + 0)	5985	1169	3209	2359	1371
LSEU	28 ×(89 + 0)	22175	309	519	454	27
MISC05	300 ×(74 + 62)	899	97	217	88	42
MISC07	212 ×(259 + 1)	22795	3107	15873	5277	3536
MOD008	6 ×(319 + 0)	17367	385	2283	800	201
MOD010	146 ×(2655 + 0)	663	197	1	37	9
MODGLOB	291 ×(98 + 324)	***	***	***	***	***
P0033	15 ×(33 + 0)	853	6.3	85	261	4.4
P0201	133 ×(201 + 0)	3307	207	1045	1306	211
P0282	241 ×(282 + 0)	***	***	671	2729	173
P0291	252 ×(291 + 0)	***	***	99	457	42
P0548	176 ×(548 + 0)	***	***	545	153	41
P2756	755 ×(2756 + 0)	***	***	463	376	776
RGN	24 ×(100 + 80)	2455	55	545	153	41
SAN200-0.9-3	1126 ×(200 + 0)	***	***	2771	4487	7286
SCPC2S	385 ×(468 + 0)	107	43	107	195	62
SET1AL	492 ×(240 + 472)	***	***	409	334	329
STEIN45	331 ×(45 + 0)	***	***	***	***	***
TSP43	143 ×(1117 + 0)	2243	247	421	332	101
VPM1	234 ×(168 + 210)	***	***	2511	4030	1006

Table 2
Number of instances solved out of 29

OSL	CPLEXMIP2.1	MINTO	MIPO	BCG
17	17	20	29	25

- The cuts are incorporated in a branch-and-cut framework. Generating the cuts at different nodes of the search tree also helps break the trend towards parallelization.
- The cuts generated are globally valid, which implies that cuts generated at a node of the enumeration tree are also used to improve the formulation at other nodes of the tree. Indeed, we ran a computational experiment where the cutting planes are used only locally and the code failed to terminate in almost half of the instances.
- The cuts are used selectively, i.e. not all cuts from the pool are part of the formulation solved at every iteration. This helps us avoid the "clogging" of the linear programs with constraints that are locally useless. Keeping the linear programs small may also improve the accuracy of the cuts that are generated, avoiding numerical difficulties.
- LP solvers are more reliable now than 30 years ago. They are faster and numerically more stable.

Appendix

This contains Table 8 from [2].

Problem	OSL		CPLEXMIP 2.1		MINTO 1.4			MIPO		
	# of nodes	CPU time	# of nodes	CPU time	# of nodes	# of cuts	CPU time	# of nodes	# of cuts	CPU time
AIR04	***	***	***	***	***	***	***	308	394	7662
AIR05	***	***	***	***	***	***	***	886	1123	18734
BM23	35	11	571	2	155	922	17	254	72	4
CFAT200-1	4	2726	30	962	***	***	***	76	80	848
CTN2	***	***	41071	1300	197	165	23	188	305	86
CTN3	***	***	***	***	4007	302	650	496	625	243
EGOUT	1	1.7	84	0.6	3	19	0.6	16	14	1.3
FXCH3	***	***	***	***	565	241	86	880	587	237
GENOVA6	3787	2375	18826	2006	7533	0	1281	3970	2111	1508
L152LAV	1842	7413	27255	5591	***	***	***	4716	3457	4772
LSEU	87	641	24995	674	161	153	13	1040	112	24
MISC05	454	220	948	37	505	100	79	788	74	88
MISC07	***	***	40713	3832	***	***	***	7766	2354	2634
MOD008	53	107	15203	263	537	479	293	1376	215	50
MOD010	18	212	577	146	47	6	970	18	30	58
MODGLOB	***	***	***	***	263	360	56	960	3346	7413
P0033	8	2.6	1058	2.5	15	34	0.8	138	23	1.6
P0201	164	282	2476	63	1233	116	160	998	292	145
P0282	210	341	***	***	3367	438	1007	218	186	36
P0291	8	10	***	***	31	101	8.5	60	185	17
P0548	0	12	***	***	***	***	***	6422	2433	11036
P2756	37	305	***	***	1188	875	1867	724	595	1933
RGN	2836	284	5213	49	3607	211	255	846	134	55
SAN200-0.9-3	***	***	2149	4074	***	***	***	310	201	995
SCPC2S	202	277	673	123	833	0	600	62	137	93
SET1AL	***	***	***	***	27	400	10	68	201	161
STEIN45	***	***	90922	3877	***	***	***	55824	16248	9010
TSP43	***	***	***	***	***	***	***	746	568	324
VPM1	***	***	***	***	183	44	8.4	2934	1438	1848

*** Space limit or time limit exceeded.

References

[1] E. Balas, S. Ceria and G. Cornuéjols, "A lift-and-project cutting plane algorithm for mixed 0–1 programs", *Math. Programming* **58**, 295–324 (1993).

[2] E. Balas, S. Ceria and G. Cornuéjols, "Mixed 0–1 programming by lift-and-project in a branch-and-cut framework", Management Science Research Report MSRR-603, GSIA, Carnegie Mellon University (1994); to appear in *Manage. Sci.*

[3] R.E. Bixby, E.A. Boyd and R.R. Indovina, "MIPLIB: a test set of mixed integer programming problems", *SIAM News* (March 1992) 16.

[4] H. Crowder, E. Johnson and M. Padberg, "Solving large-scale zero–one linear programming problems", *Oper. Res.* **31**, 803–834 (1983).

[5] R.S. Garfinkel and G.L. Nemhauser, *Integer Programming*, Wiley, New York, 1972.

[6] R. Gomory, "Outline of an algorithm for integer solutions to linear programs", *Bull. Amer. Math. Soc.* **64**, 275–278 (1958).

[7] R. Gomory, "An algorithm for the mixed integer problem", Technical Report RM-2597, The Rand Corporation, 1960.

[8] R. Gomory, "An algorithm for integer solutions to linear programs", in: R.L. Graves and P. Wolfe (eds.), *Recent Advances in Mathematical Programming*, McGraw-Hill, New York, 1963, pp. 269–302.

[9] R. Gomory, "Early integer programming", in: J.K. Lenstra, A.H.G. Rinnooy Kan and A. Schrijver (eds.), *History of Mathematical Programming, A Collection of Personal Reminiscences*, North-Holland, Amsterdam, 1991, pp. 55–61.

[10] M. Gondran and B. Simeone, "Report on the session on cutting planes", *Ann. Discrete Math.* **5**, 193–194 (1979).

[11] J. Haldi and L.M. Isaacson, "A computer code for integer solutions to linear programs", *Oper. Res.* **13**, 946–959 (1965).

[12] K.L. Hoffman and M. Padberg, "Solving airline crew scheduling problems by branch-and-cut, *Manage. Sci.* **39**, 657–682 (1993).

[13] G.T. Martin, "An accelerated euclidean algorithm for integer linear programming", in: R.L. Graves and P. Wolfe (eds.), *Recent Advances in Mathematical Programming*, McGraw-Hill, New York, 1963, pp. 311–318.

[14] P. Miliotis, "Using cutting planes to solve the symmetric traveling salesman problem", *Math. Programming* **15**, 177–188 (1978).

[15] G.L. Nemhauser and L.A. Wolsey, "Integer programming", in: G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd (eds.), *Handbooks in Operations Research and Management Science 1: Optimization*, North-Holland, Amsterdam, 1989, pp. 447–527.

[16] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems", *SIAM Rev.* **33**, 60–100 (1991).

[17] R.G. Parker and R.L. Rardin, *Discrete Optimization*, Academic Press, New York, 1988.

[18] C.A. Trauth Jr. and R.E. Woolsey, "Integer linear programming: a study in computational efficiency", *Manage. Sci.* **15**, 481–493 (1963).

[19] T.J. Van Roy and L.A. Wolsey, "Solving mixed integer programming problems using automatic reformulation", *Oper. Res.* **35**, 45–57 (1987).

[20] H.P. Williams, *Model Building in Mathematical Programming*, Wiley, New York, 1985.