Noname manuscript No.
(will be inserted by the editor)

A Relax-and-Cut Framework for Gomory Mixed-Integer Cuts

Matteo Fischetti · Domenico Salvagnin

the date of receipt and acceptance should be inserted later

Abstract Gomory Mixed-Integer Cuts (GMICs) are widely used in modern branch-and-cut codes for the solution of Mixed-Integer Programs. Typically, GMICs are iteratively generated from the optimal basis of the current Linear Programming (LP) relaxation, and immediately added to the LP before the next round of cuts is generated. Unfortunately, this approach is prone to instability. In this paper we analyze a different scheme for the generation of rank-1 GMICs read from a basis of the original LP—the one before the addition of any cut. We adopt a relax-and-cut approach where the generated GMICs are not added to the current LP, but immediately relaxed in a Lagrangian fashion. Various elaborations of the basic idea are presented, that lead to very fast—yet accurate—variants of the basic scheme. Very encouraging computational results are presented, with a comparison with alternative techniques from the literature also aimed at improving the GMIC quality. We also show how our method can be integrated with other cut generators, and successfully used in a cut-and-branch enumerative framework.

Keywords :
Mixed-Integer Programming, Gomory cuts, Lagrangian relaxation, Relax and Cut

1 Introduction

Gomory Mixed-Integer Cuts (GMICs) for general MIPs have been introduced by Ralph Gomory about 50 years ago in his seminal paper [22]. However,

Matteo Fischetti
DEI, University of Padova, Italy,
E-mail: matteo.fischetti@unipd.it

Domenico Salvagnin
DMPA, University of Padova, Italy
E-mail: salvagni@math.unipd.it

these cuts were not used in practice until the seminal work of Balas, Ceria, Cornuéjols and Natraj [5], who found for the first time an effective way to exploit them in a branch-and-cut context [14], stressing the importance of their generation in *rounds*, i.e., from all the tableau rows with fractional right hand side. Nowadays GMICs are of fundamental importance for branch-and-cut Mixed-Integer Program (MIP) solvers, that are however quite conservative in their use because of known issues due to the iterative accumulation of GMICs in the optimal Linear Programming (LP) basis, which leads to numerical instability due a typically exponential growth of the determinant of the LP basis.

Recent work on the subject suggests however that stability issues are largely due to the overall framework where GMICs are used, rather than to the GMICs themselves. Indeed, the two main cutting plane modules (the LP solver and the cut generator) form a closed-loop system that is intrinsically prone to instability—unless a "decoupling filter" is introduced in the loop. Breaking the feedback is therefore a must if one wants to really exploit the power of GMICs.

In this paper we propose a new mechanism to break the entanglement between LP bases and GMICs. More specifically, in our framework the generated GMICs are not added to the current LP, but immediately relaxed in a Lagrangian fashion. In this way, GMICs are always generated from a (Lagrangian near-optimal) basis of the original LP, hence their quality is not likely to deteriorate in the long run as we do not allow GMICs to accumulate in the LP basis.

The approach of relaxing cuts right after their separation is known in the literature as *Relax-and-Cut*. It was introduced independently by Lucena [26] and by Escudero, Guignard and Malik [19]—who actually proposed the relax-and-cut name; see Lucena [27] for a survey of the technique and of its applications. Very recently, Lucena [28] applied a relax-and-cut approach to the solution of hard single 0-1 knapsack problems, where fractional Gomory cuts were used, for the first time, in a Lagrangian framework.

The paper is organized as follows. Section 2 briefly reviews standard GMIC frameworks from the literature. In Section 3 we introduce our notation and describe the relax-and-cut framework. Various elaborations of the basic idea are presented in Section 4, that lead to faster yet accurate variants of the basic relax-and-cut scheme. Very encouraging computational results are presented in Section 5, with a comparison with alternative techniques from the literature also aimed at improving the GMIC quality, namely those proposed very recently by Balas and Perregaard [7] and by Dash and Goycoolea [17]. In the same section we also discuss how our method can be integrated with other cut generators, and successfully used in a cut-and-branch enumerative framework. Some conclusions and possible directions of work are finally addressed in Section 6.

We assume the reader has some familiarity with MIP cuts; see, e.g., Cornuéjols [15] for an in-depth treatment of the subject.

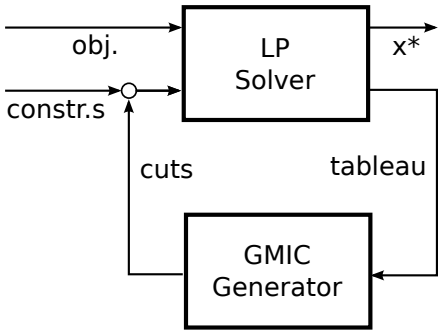

Fig. 1 Traditional closed-loop scheme for GMICs.

2 Standard frameworks for GMICs

An explanation of GMIC instability in terms of closed-loop dynamic systems was pointed out by Zanette, Fischetti and Balas [30], who presented computational experiments showing that reading the LP optimal solution to cut *and* the Gomory cuts themselves from the same LP basis almost invariably creates a dangerous feedback in the long run. Figure 1 gives an illustration of the closed-loop nature of the standard GMIC framework.

It is worth observing that feedback issues are common to other cutting plane procedures that derive cuts directly from tableau information of the enlarged LP that includes previously-generated cuts (e.g., those related to Gomory's corner polyhedron, including cyclic-group cuts, intersection cuts, multi-row cuts, etc.). This is not necessarily the case when using methods based on an external cut generation LP (e.g., disjunctive methods using disjunctions not read from the optimal tableau), or when the derived cuts are post-processed so as to reduce their correlation with the optimal tableau (e.g., through cut strengthening methods [7,17] or by lexicographic search [30]). In particular, for 0-1 MIPs, Bonami and Minoux [10] successfully generated violated lift-and-project cuts (i.e., disjunctive cuts from a trivial disjunction) by first solving a cut generation LP, and then by strengthening them through the Balas and Jeroslow procedure [6].

A different framework for Gomory cuts was recently proposed by Fischetti and Lodi in [20]. The aim of their work was actually to compute the best possible bound obtainable with rank-1 fractional Gomory cuts. The fact of restricting to rank-1 cuts forced the authors to get rid of the classical separation scheme, and to model the separation problem through an auxiliary MIP to be solved by an external module. The surprising outcome was a numerically stable cutting plane method where rank-1 fractional Gomory cuts alone produced very tight bounds—though the separation overhead was too large to be practical in most cases. Note that, in that scheme, the separation

procedure did not have access to the optimal LP basis, but only received on input the point to be separated—hence loosening the optimization and separation entanglement. As a consequence, even if the point x^* to be separated possibly did not change significantly at some iterations, it was unlikely that the separated cuts were as heavily correlated as in the classical scheme—in this context, the well-known erratic behavior of MIP solvers that often return quite different solutions for almost identical input, turned out to be beneficial in that it acted as a diversification in the cut selection policy. These results were later confirmed for general (i.e., possibly not read from a tableau row) GMICs by Balas and Saxena [8] and by Dash, Günlük, and Lodi [18], who adopted the same scheme but generalized the MIP separation module so as to deal with GMIC separation.

The above discussion suggests that an improved performance can be attained if one does not insist on reading GMICs from the optimal basis of the current LP, that includes previously generated GMICs. Progresses in this direction have been obtained recently by using one of the following two approaches. Let x^* be an optimal vertex of the "large LP" (the one defined by the original constraints plus the GMICs generated in the previous iterations), and let B^* be an associated optimal basis.

(i) Balas and Perregaard [7] perform a sequence of pivots on the tableau of the large LP leading to a (possibly non-optimal or even infeasible) basis of the same large LP that produces a deeper cut with respect to the given x^*.
(ii) Dash and Goycoolea [17] heuristically look for a basis B of the *original* LP that is "close enough to B^*", in the hope of cutting the given x^* with rank-1 GMICs associated with B; this is done, e.g., by removing from A all the columns that are nonbasic with respect to x^*, thus restricting B to be a submatrix of B^*.

3 A relax-and-cut framework for GMICs

Consider a generic MIP of the form

$$\min\{cx : Ax = b,\, x \geq 0,\, x_j \text{ integer } \forall j \in J\}$$

where $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, and $J \subseteq \{1, \cdots, n\}$ is the index set of the integer variables. As customary, let $P := \{x \in \mathbb{R}^n_+ : Ax = b\}$ denote the LP relaxation polyhedron, that we assume to be bounded and nonempty, so as to guarantee the existence of an optimal solution for any linear objective function.

Given a large (possibly exponential) family of linear cuts

$$\alpha^i x \geq \alpha^i_0, \quad i = 1, \ldots, M \tag{1}$$

we aim at computing—possibly in an approximate way—the value

$$z_1 := \begin{cases} \min cx \\ x \in P \\ \alpha^i x \geq \alpha_0^i, \quad i = 1, \ldots, M \end{cases} \tag{2}$$

In our basic application, family (1) consists of the GMICs associated with all possible primal-feasible bases of system $Ax = b$, i.e., z_1 is a lower bound on the first GMIC closure addressed by Balas and Saxena [8] and by Dash, Günlük, and Lodi [18]. However, as discussed in the computational section, family (1) is in principle allowed to contain any kind of linear inequalities, including problem-specific cuts and/or GMICs of any rank, or even invalid linear conditions related, e.g., to branching conditions.

A standard solution approach for (2) consists in dualizing cuts (1) in a Lagrangian fashion, thus obtaining the Lagrangian dual problem

$$\max_{u \geq 0} \left\{ L(u) := \min\{cx + \sum_{i=1}^{M} u_i(\alpha_0^i - \alpha^i x) : \ x \in P\} \right\} \tag{3}$$

whose optimal value is known to coincide with z_1.

The solution of (3) can be attempted through very simple iterative procedures known as subgradient methods, or through more sophisticated and accurate schemes such as the bundle method; see e.g. [24]. All the above solution schemes generate a sequence of dual points $u^k \geq 0$ meant to converge to an optimal dual solution u^*. For each u^k in the sequence, an optimal solution $x^k \in P$ of the inner-minimization in (3) is computed, along with the associated Lagrangian value

$$L(u^k) = cx^k + \sum_{i=1}^{M} u_i^k(\alpha_0^i - \alpha^i x^k)$$

and subgradient $s^k \in \mathbb{R}^M$, whose components are just the cut violations

$$s_i^k := \alpha_0^i - \alpha^i x^k, \quad i = 1, \ldots, M$$

($s_i^k > 0$ for violated cuts, and $s_i^k \leq 0$ for slack/tight cuts). In particular, the ability of computing the subgradient is essential for the convergence of overall scheme—this is not a trivial task when the cut family is described only implicitly.

In our context, family (1) is by far too large to be computed explicitly, so we store only some of its members, using a data structure called *cut pool*. Cut duplication in the pool is heuristically avoided by normalizing the cut right-hand-side and by using a hash function.

The cut pool is initially empty, or it can contain some heuristic collection of warm-start cuts. The pool is then iteratively extended by means of rank-1 GMICs that are heuristically generated, on the fly, during the process of solving the Lagrangian dual problem. More specifically, if the Lagrangian subproblem at a certain u^k is solved by the simplex method and an optimal vertex x^k of

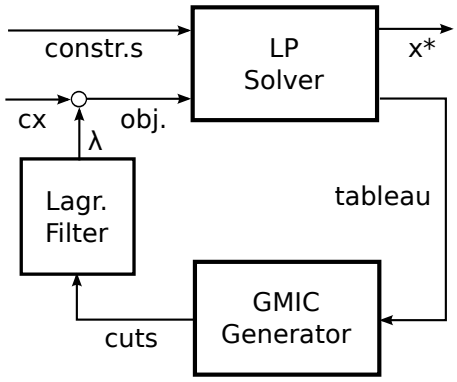

Fig. 2 Basic relax-and-cut scheme for GMICs.

P with fractional components is found, we can just read a round of rank-1 GMICs from the optimal LP basis and feed the cut pool. Note that these cuts are always associated with a primal-feasible basis of the original system P, so they are globally valid for our MIP problem even if the cut pool contains invalid cuts (e.g., branching conditions or temporary diversification cuts). Also note that, although violated by x^k, some of these cuts can actually belong already to the current pool—an indication that their Lagrangian multiplier should be increased in the next iterations.

Our relax-and-cut GMIC framework is depicted in Figure 2, to be compared with the "traditional one" as in Figure 1. A notable characteristic of this scheme is that, differently from traditional cutting plane schemes, there is no natural "fractional point to cut", and the discovery of new cuts to be added to the pool is beneficial mainly because new components of the "true" subgradient s^k are computed, thus improving the chances of convergence to the "true" optimal dual value z_1 of the overall Lagrangian scheme.

4 Implementations

We next describe three very basic heuristics for the Lagrangian dual problem (3), that are intended to evaluate the potential of using GMICs in a relax-and-cut framework. The investigation of more sophisticated schemes such as the bundle method is left to future investigation.

4.1 Subgradient optimization

The basic algorithm underlying our heuristics is the subgradient method. The subgradient method is an adaptation of the gradient method to the case of maximization of nondifferentiable concave functions, such as $L(u)$. It starts

with a tentative point $u^0 \geq 0$ ($u^0 = 0$ in our implementation) and then iteratively constructs a sequence of points u^k according to the following rule:

$$u^{k+1} = (u^k + \lambda_k s^k)_+$$

where s^k is a subgradient of $L(\cdot)$ in u^k, $\lambda_k > 0$ is an appropriate parameter called *step length*, and $(\cdot)_+$ denotes the projection onto the nonnegative orthant.

The asymptotic convergence of the method is guaranteed by the properties of the subgradient and by the choice of appropriate step sizes. A step-size rule often used in practice, usually known as *relaxation step length* or *Polyak's step length*, computes

$$\lambda_k = \frac{\mu_k(UB - L(u^k))}{||s^k||^2}$$

where μ_k is a parameter satisfying $0 < \mu_k \leq 2$, and UB is the unknown optimal dual value z_1, typically replaced by an upper bound on z_1. In our code, this upper bound is computed at the very beginning by taking the best integer solution found by a MIP solver at the root node, or an appropriate multiple of the LP relaxation if none is found (in particular, the multiplier was set to 2.0 if the LP relaxation value was positive, and to 0.5 otherwise). Such upper bound information is however never used to perform reduced-cost fixing of variables. As to μ_k, it is adjusted dynamically in order to try to speed up convergence, using quite an elaborate update strategy inspired by the computational studies reported in [11,23]. In particular, let $bestLB$ denote the best Lagrangian lower bound so far. At the beginning of each Lagrangian iteration, we compute a "reference" interval $\Delta = UB - bestLB$, that we use to guide our strategy. If $L(u) < bestLB - \Delta$ for 10 consecutive iterations, we halve μ_k and backtrack to the best u^k so far. Otherwise, let $avgLB$ be the average value of $L(u)$ in the last $p = 100$ iterations. If $bestLB$ has improved by less than 0.01Δ in the last p iterations, then we update μ_k as

$$\mu_k = \begin{cases} 10\mu_k & \text{if } bestLB - avgLB < 0.001\Delta \\ 2\mu_k & \text{if } 0.001\Delta \leq bestLB - avgLB < 0.01\Delta \\ \mu_k/2 & \text{otherwise} \end{cases}$$

In the following, we will denote by `subg` our subgradient implementation with a limit of $10,000$ iterations. The starting step size parameter is aggressively set to $\mu_0 = 10$. This is justified by the fact that in our scenario the convergence of the method is not guaranteed (and is also unlikely in practice), because we are dealing explicitly only with a small subset of cuts. In particular, we always deal with truncated subgradients and, even more importantly, we have no way of generating violated GMICs apart from reading them from the LP tableau. According to our computational experience, in this scenario a small initial value for μ is quite inappropriate because it causes the method to saturate far from an optimal Lagrangian dual solution u^*, with no possibility for recovery.

Finally, to avoid overloading the cut pool we read a round of GMICs at every K-th subgradient iteration, where $K = 10$ in our implementation. In addition, we do not add new entries to the Lagrangian vector u^k every time new cuts are added to the pool, but only every 50 subgradient iterations, so as to let the subgradient method stabilize somehow before dealing with new Lagrangian components. In this view, our implementation is between the so-called *delayed* and *non-delayed* relax-and-cut methods [27].

4.2 Hybrid LP and subgradient optimization

The basic subgradient method presented in the previous subsection has several drawbacks when applied to (3). In particular, finding the right combination of parameters to obtain a satisfactory convergence is definitely tricky. In our setting, we found beneficial to recompute, from time to time, the optimal Lagrangian multipliers u_i for *all* the cuts in the current pool. This amounts to solving the *large LP* defined by the original constraints plus all the cuts in the pool; the optimal dual values of the cuts then correspond to the optimal Lagrangian multipliers. The solution of the large LP can be done quite efficiently in practice by means of a dynamic selection policy of the pool cuts based on the concepts of efficacy and orthogonality, as described in [1, 3, 5]. Note that this policy is usually not attractive in a classical setting where the number of dualized constraints is fixed—solving the large LP would be just as hard as solving (3). This is however not the case in our context, because the pool is extended dynamically and stores a (large but) manageable subset of cuts.

In what follows, we will denote by `hybr` our implementation of a hybrid subgradient method for solving (3), where we periodically compute the optimal multipliers of the pool cuts by solving the large LP (note however that we do not read GMICs from the optimal basis of the large LP). In our code this is done every $1,000$ subgradient iterations. All other parameters are the same as in `subg`.

4.3 Fast hybrid framework

Although the hybrid version `hybr` is definitely an improvement over `subg`, both methods are still quite demanding as far as running time is concerned. The reason is twofold. First, we may spend a lot of time generating GMICs from useless bases (contrarily to popular belief, reading cuts from the tableau comes *not* for free, although it is very cheap compared to other separation methods). Second, the LPs change significantly from one iteration to the next one, because of the zig-zagging nature of the dual multipliers induced by the standard subgradient algorithm, hence the usual warm-start of the simplex algorithm is less effective—note that this drawback may be reduced by using more stabilized algorithms like the bundle method.

We developed some variants of `hybr` tweaked for speed, trying to sacrifice the quality of the computed bound on z_1 as little as possible. Speed is obtained

by drastically reducing the number of subgradient iterations and by using a very small step length parameter ($\mu_k = 0.01$ in our code). The small step size yields more parameterized LPs where warm-start is more effective, and the reduced number of iterations speeds up the whole approach. In a sense, we are no longer relying on the step size for the convergence of the method—which is taken care of by the large LPs used to get the optimal multipliers—and we use the subgradient method just to sample near-optimal Lagrangian bases of the original system generating rank-1 GMICs (this will be called the *sampling phase* in the sequel). It is worth noting that the small step length parameter and the reduced number of iterations essentially turn off the step-length update strategy that we have described in Section 4.1.

We implemented two variants of the above method, namely `fast` and `faster`. In both variants we solve the large LP to compute the Lagrangian optimal multipliers only 10 times, and we generate GMICs at every subgradient iteration. The difference is in the number of subgradient iterations in the sampling phase between two consecutive large-LP resolutions, which is 100 for `fast`, and just 50 for `faster`.

It is worth observing that the methods above can be interpreted as a way to decompose *à la Dantzig-Wolfe* the optimal solution x^* of the large LP into a suitable convex combination $\sum_j \lambda_j x^j$ of vertices x^j of P, and to separate these x^j in the attempt of finding valid cuts violated by x^*. This links those variants to the work of Ralphs, Kopman, Pulleyblank, and Trotter [29], where a similar idea was applied to separate capacity cuts for the Capacitated Vehicle Routing Problem—the fractional CVRP vertex being expressed as the convex combination of m-TSP integer solutions, each of which is easily evaluated to find violated capacity cuts.

4.4 The overall scheme

The four variants described in the present section correspond to different parameter choices of a common scheme, as described in Algorithm 1. According to the scheme, a first round of GMICs is read from the optimal tableau of the original relaxation, in order to initialize the pool (lines 1–5). If the problem is not solved, a main loop of at most L iterations is started at line 6: at each iteration, $I_{\max}$ subgradient iterations are performed, starting from the Lagrangian multiplier vector u_0 for the cuts in the pool. Note that the subgradient algorithm is considered here as a black-box solver, which is given in input the original relaxation P, an initial list of cuts (*pool*) with the corresponding initial Lagrangian multiplier vector u_0, and some parameters (an initial value μ_0 for the Lagrangian step-size parameter μ, the maximum number of subgradient iterations $I_{\max}$, and the iteration interval K for new cut generation). The subgradient procedure updates the cut pool on return. Finally, at lines 9–11 the current relaxation is solved to optimality.

Algorithm 1: Relax-and-cut for GMICs—the basic scheme.

input : MIP $\equiv \min\{c^T x : x \in P,\ x_j \text{ integer } \forall j \in J\}$, number of main iterations L, starting value μ_0 for the step-size parameter μ, subgradient iteration limit $I_{\max}$, separation interval K

output: a polyhedron P' approximating the first GMIC closure

```
x* = LPSolve (P)
pool = ReadGMICs (P)
P' = P ∩ {cuts in pool}
x* = LPSolve (P')
if x*_j is integer ∀j ∈ J then return P'
for i=1 to L do
    u_0 = optimal LP dual vector for cuts in pool
    pool = SubgradientRun (P, pool, u_0, μ_0, I_max, K)
    P'= P ∩ {cuts in pool}
    x* = LPSolve (P')
    if x*_j is integer ∀j ∈ J then return P'
end
return P'
```

It is easy to easy that the four variants described in the present section correspond to the four different settings of the parameters of Algorithm 1 presented in Table 1.

Table 1 Different parameter settings for Algorithm 1.

	parameters			
variant	L	μ	$I_{\max}$	K
`subg`	1	10	10000	10
`hybr`	10	10	1000	10
`fast`	10	0.01	100	1
`faster`	10	0.01	50	1

5 Computational results

We tested our variants of the relax-and-cut framework for GMICs on the problem instances in MIPLIB 3.0 [9] and MIPLIB 2003 [2]. Following [17], we omitted all instances where there is no improvement after one round of GMICs read from the optimal tableau, or where no integer solution is known. In the end, we were left with 54 instances from MIPLIB 3.0, and 20 instances from MIPLIB 2003. To improve readability, we report the outcome of our experiments through average figures only, whereas instance-by-instance information are given in the appendix.

We implemented our code in C++, using IBM ILOG Cplex 11.2 as black box LP solver (its primal heuristics were also used to compute the subgradient

upper bound UB). All tests have been performed on a PC with an Intel Q6600 CPU running at 2.40GHz, with 4GB of RAM (only one CPU was used by each process). As far as the GMIC generation is concerned, for a given LP basis we try to generate a GMIC from every row where the corresponding basic variable has a fractionality of at least 0.001. The cut is however discarded if its final dynamism, i.e., the ratio between the greatest and smallest absolute value of the cut coefficients, is greater than 10^{10}. All the generated GMICs are stored in our cut pool, whose maximum size is fixed to 1GB of RAM: whenever this limit is reached, we perform a purge operation based on cut efficacy (with respect to the last large LP solution x^* available) and cut age. It is worth noting that some instances in our testbed do not fulfill the boundedness assumption on P, hence in some (rare) occasions the Lagrangian subproblem turns out to be unbounded. In this case, we simply interrupt the subgradient loop, and let the overall method continue.

5.1 Approximating the first GMIC closure

In our first set of experiments we compared the ability (and speed) of the proposed methods in approximating the first GMIC closure for the problems in our testbed. The first GMIC closure has received quite a lot of attention in the last years, and it was computationally proved that it can provide a tight approximation of the convex hull of the feasible solutions. In addition, rank-1 GMICs are read from the original tableau, hence they are generally considered safe from the numerical point of view. Note that our method can only generate cuts from *primal-feasible* bases, hence it can produce a weaker bound than that associated with the first GMIC closure [16].

Table 2 Average gap closed and computing times for rank-1 methods.

	MIPLIB 3.0		MIPLIB 2003	
method	cl.gap	time (s)	cl.gap	time (s)
`1gmi`	26.4%	0.03	18.3%	0.54
`faster`	57.1%	1.53	43.3%	33.33
`fast`	58.9%	2.57	45.5%	58.40
`hybr`	60.1%	15.78	48.6%	314.73
`subg`	55.0%	24.97	43.5%	290.82
`dgDef`	60.8%	23.78	39.7%	853.85

In Table 2 we report the average gap closed by all methods that generate rank-1 GMICs only, as well as the corresponding computing times (geometric means). We recall that for a given instance, the gap closed is defined as $100 \cdot (z - z_0)/(opt - z_0)$, where z_0 is the value of the initial LP relaxation, z is the value of the final LP relaxation, and opt is the best known solution. For comparison, we report also the average gap closed by one round of GMIC read

from the first optimal tableau (`1gmi`), as well as the average gap closed with the default method proposed by Dash and Goycoolea (`dgDef`), as reported in [17]. All computing times are given in CPU seconds on our Intel machine running at 2.4 GHz, except for `dgDef` where we just report the computing times given in [17], without any speed conversion—the entry for MIPLIB 3.0 refers to a 1.4 GHz PowerPC machine (about 2 times slower than our PC), while the entry for MIPLIB 2003 refers to a 4.2 GHz PowerPC machine (about twice as fast as our PC).

According to the table, the relax-and-cut methods performed surprisingly well, in particular for the hard MIPLIB 2003 instances where all of them outperformed `dgDef` in terms of both quality and speed. As far as the bound quality is concerned, the best method appears to be `hybr`, mainly because of its improved convergence with respect to `subg`, and of the much larger number of subgradient iterations (and hence of LP bases) generated with respect to the two fast versions. As a reference, Balas and Saxena [8] and Dash, Günlük and Lodi [18] report a closed gap of 77.9% for the MIPLIB 3.0 testbed when optimizing over the split closure.

The two fast versions also performed very well, in particular `faster` that proved to be really fast (more than 10 times faster than `dgDef`) and quite accurate. It is worth observing that about 75% of the computing time for `fast` and `faster` was spent in the sampling phase: 40% for LP reoptimizations, and 35% for actually reading the GMICs from the tableau and projecting slack variables away. Quite surprisingly, the solution of the large LPs through a dynamic pricing of the pool cuts required just 15% of the total computing time.

5.2 Higher rank GMICs

In this subsection we investigate the possibility of generating GMICs of rank greater than 1. Unfortunately there is no fast way to compute the exact rank of a cut, hence we use an easy upper bound where the rows of the original system $Ax = b$ are defined to be of rank 0, and the rank of a GMIC is computed as the maximum rank of the involved rows, plus one. Having computed the above upper bound for each GMIC, we avoid storing in the pool any GMIC whose upper bound exceeds an input rank limit k (k =2 or 5, in our tests).

Our relax-and-cut framework can be extended in many different ways to generate higher-rank GMICs. In particular, given a maximum allowed rank k, it is possible to:

a) Generate k rounds of GMICs in a standard way, use them to initialize the cut pool, and then apply our method to add rank-1 GMICs on top of them. This very simple strategy turned out not to work very well in practice, closing significantly less gap than the rank-1 version.
b) Apply one of the relax-and-cut variants of the previous subsection until a termination condition is reached. At this point add to the original formulation (some of) the GMICs that are tight at the large-LP optimal solution,

and repeat k times. This approach works quite well as far as the final bound is concerned, but it is computationally expensive because we soon have to work with bigger (and denser) tableaux.

c) Stick to rank-1 GMICs in the sampling phase, never enlarging the original system. However, each time the large LP is solved to recompute the dual multipliers (this can happen at most k times), add to the pool (but not to the original formulation) all the GMICs read from the large-LP optimal basis.

d) As before, stick to rank-1 GMICs in the sampling phase. If however no cut separating the previous large-LP solution x^* is found in the sampling phase, then add to the pool all GMICs read from the large LP optimal basis, and continue. This way, the generation of higher-rank cuts acts as a diversification step, used to escape a local deadlock, after which standard rank-1 separation is resumed.

According to our preliminary computational experience, the last two schemes give the best compromise between bound quality and speed. In particular, c) takes almost the same computing time as its rank-1 counterpart in Table 2, and produces slightly improved bounds. Although slower, option d) closes significantly more gap than c), hence it is more attractive for a comparison with rank-1 cuts.

Table 3 Average gap closed and computing times for higher rank methods.

		MIPLIB 3.0		MIPLIB 2003	
method	rank	cl.gap	time (s)	cl.gap	time (s)
`gmi`	1	26.4%	0.03	18.3%	0.54
`faster`	1	57.1%	1.53	43.3%	33.33
`fast`	1	58.9%	2.57	45.5%	58.40
`gmi`	2	35.2%	0.04	24.0%	0.88
`faster`	2	61.4%	3.19	47.2%	58.37
`fast`	2	63.7%	5.98	48.5%	106.76
`gmi`	5	46.6%	0.08	30.3%	2.17
`faster`	5	65.1%	6.45	49.9%	126.65
`fast`	5	66.8%	12.03	51.1%	238.33
`L&P`	10	55.6%	3.75	30.7%	95.23

In Table 3 we report the average gap closed by our fast versions when higher-rank GMICs are generated according to scheme d) above. Computing times (geometric means) are also reported. Rank-1 rows are taken from the previous table.

In the table, row `gmi` refers to 1, 2 or 5 rounds of GMICs. For the sake of comparison, we also report the average gap closed by 10 rounds of Lift&Project cuts (`L&P`), as described in [4]. To obtain the Lift&Project bounds and running times we ran the latest version of separator `CglLandP` [12] contained in

the COIN-OR [13] package Cgl 0.55, using Clp 1.11 as black box LP solver (the separator did not work with Cplex because of the lack of some pivoting procedures). This separation procedure was run with default settings, apart from the minimum fractionality of the basic variables used to generate cuts, which was set to 0.001 as in the other separators. All computing times are given in seconds on our Intel machine running at 2.4 GHz.

Our fast procedures proved quite effective also in this setting, providing significantly better bounds than `L&P` in a comparable or shorter amount of time, even when restricting to rank-1 GMICs. As expected, increasing the cut rank improves the quality of the bound by a significant amount, though it is not clear whether this improvement is worth the time overhead—also taking into account that GMICs of higher rank tend to be numerically less reliable. Similarly, it is not clear whether the bound improvement achieved by `fast` with respect to `faster` is worth the increased computing time.

5.3 Pool initialization by pseudo-branching

In order to start our relax-and-cut procedure, the cut pool has to be initialized with some cuts. In the computational results presented so far, the cut pool was just fed with the GMICs that could be generated from the optimal tableau of the very first LP relaxation. More elaborate strategies can however be used.

We next address a very simple mechanism for gathering a significant number of LP bases "around the root-node LP optimum". Roughly speaking, we simulate a simple branching strategy and derive a collection of LP bases whose associated GMICs are used to initialize our cut pool. A similar idea was applied already in [17], a main difference being however that we do not pretend the generated GMICs be violated by the root-node LP optimum, but simply use them to warm-start our relax-and-cut method.

More specifically, our pseudo-branching scheme implements the following *diving* procedure; see Figure 3 for an illustration. Assume the MIP to be solved involves binary variables, as it is the case in most practical situations. After solving the LP relaxation, binary variables are ranked in order of decreasing fractionality (the closer to 0.5 the better) and we randomly choose the branching variable x_b (say) among the first 5 in such a sorted list. Then, we randomly choose a branching direction and perform a branching step by increasing (in case of a left-branch) or decreasing (in case of a right-branch) cost c_b by a large constant. The procedure is repeated until a depth limit isreached (10 in our tests). Note that our branching mechanism guarantees the global validity of the generated GMICs, and could heuristically be applied to general-integer branching variables as well—as a rough way to generate new LP bases not too far from the current one.

For each instance in our testbed, we performed 10 dives with the algorithm just described and collected the GMICs associated with all the generated bases. Results are shown in Table 4, both for the procedure used as a standalone tool, namely `dive (alone)`, or used to initialize the `fast` and `faster` cut

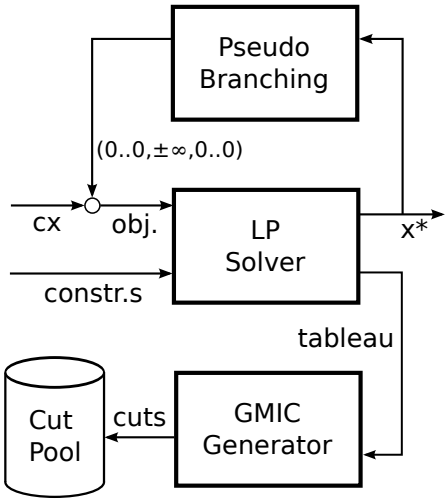

Fig. 3 Pseudo-branching scheme.

pools (`dive+fast` and `dive+faster`, respectively). The format of the table is the same as in Table 2; rows `L&P`, `dgDef`, `fast` and `faster` are copied from previous tables. The results show that `dive (alone)` is a very fast way to close a significant portion of the gap, and appears quite competitive with previous methods such as `L&P` and `dgDef` on MIPLIB-2003 instances. It also turns out to be effective when used together with `faster`, being able to close essentially the same gap as `fast` in a significantly shorter computing time. It usefulness as warm start of `fast` is instead less clear (at least in its current tuning), in that it only allows to close about 1% more gap than `fast` alone.

Table 4 Effect of pseudo-branching on the average gap closed and computing time.

	MIPLIB 3.0		MIPLIB 2003	
method	cl.gap	time (s)	cl.gap	time (s)
`1gmi`	26.4%	0.03	18.3%	0.54
`dive (alone)`	37.8%	0.31	31.5%	7.45
`faster`	57.1%	1.53	43.3%	33.33
`dive+faster`	59.2%	1.70	46.4%	40.24
`fast`	58.9%	2.57	45.5%	58.40
`dive+fast`	60.6%	2.79	46.5%	69.73
`L&P`	55.6%	3.75	30.7%	95.23
`dgDef`	60.8%	23.78	39.7%	853.85

5.4 Working on top of other cuts

Our method is meant to add rank-1 GMICs on top of a collection of other cuts collected in a cut pool. In our previous experiments the cut pool only contained GMICs collected in the previous iterations. However, it seems reasonable to allow the pool to contain other classes of (more combinatorial) cuts, e.g., all those generated at the root node by a modern MIP solver. In this setting, the preprocessed model and the generated cuts (stored in the cut pool) can be provided on input to our relax-and-cut scheme, in the attempt of reducing even further the integrality gap at the root node.

This is indeed the experiment we tried, using IBM ILOG Cplex 11.2 to perform the root node processing of the testbed problems. After the root node processing is finished, we reload the preprocessed model, store its additional cuts into our own cut pool, and launch our `fast` method to add rank-1 GMICs on top of the Cplex's one. Note that Cplex's cuts are *not* incorporated in the rank-0 model, i.e., the additional GMICs we generate are of rank 1 with respect to the preprocessed formulation *without* Cplex's cuts (this improves the numerical stability of the overall approach). Note that, in this case, our cuts are not necessarily of rank 1 for the original formulation.

We considered two Cplex variants to perform the root node processing before the addition of our cuts: in `cpx` we used the default setting, while in `cpx2` we increased the cut generation emphasis to 2 (aggressive cut policy). The results of our experiment are presented in Table 5.

Table 5 GMICs on top of other cuts.

	MIPLIB 3.0		MIPLIB 2003	
method	cl.gap	time (s)	cl.gap	time (s)
`cpx`	55.5%	0.17	49.0%	6.57
`cpx+fast`	70.0%	2.14	58.2%	56.03
`cpx2`	65.3%	0.56	52.9%	22.48
`cpx2+fast`	72.2%	1.98	59.6%	59.75

According to the table, both `cpx` and `cpx2` are effective in reducing the integrality gap in a short computing time, which is not surprising because they exploit a rich arsenal of cut separation procedures that go far beyond GMICs. The combination `cpx+fast` reduces significantly more gap than `cpx2` (58.2% vs. 52.9% on the hard MIPLIB 2003 testbed), but requires about 3 times more computing time. Interestingly, `cpx2+fast` is able to close about 15% of the residual gap after `cpx2`.

5.5 Use in a Cut&Branch scheme

The ultimate use of cuts is of course the exact solution of the MIP at hand. The state-of-the-art approach for exact MIP solvers is Branch&Cut, where cuts are possibly generated at every branching node. In practice, however, cuts are extensively used at the root node only, and a very limited effort to generate them is performed at the other branching nodes. Following other authors, including [4], we therefore decided to evaluate the potential of our new cut generation framework by using the so-called Cut&Branch scheme. In other words, cut generation is only allowed at the root node to improve the current LP formulation, as we did in the previous subsection, whereas all other nodes are processed by a black-box MIP solver (Cplex 11.2 in our case) with cut generation turned off.

In our experiments, we removed from our testbed all the "easy" instances that can be solved by Cplex 11.2 (default setting) is less than one hour on our PC. We also removed instance `glass4` that created numerical problems for all the compared methods. The remaining 18 instances have been processed with a total time limit of 10,000 seconds each. The root node of each instance has been processed by any of the three codes `cpx`, `cpx2` and `cpx+fast` described in the previous subsection.

Cumulative results are presented in Table 6, whereas detailed results are reported in Table 7. In these tables, the following information is reported: `solved` is the number of solved instances within the time limit (out of 18), `time` is the total computing time, including the root node processing, in seconds (geometric mean), `nodes` is the number of nodes enumerated (geometric mean), and `fin.gap` is the final gap between the primal and dual bounds at the end of the computation (arithmetic mean). For the sake of comparison, in Table 6 we also report the cumulative results for Cplex 11.2 branch-and-cut with default settings (`cpx_def`) on the original models.

Table 6 Cumulative results in a Cut&Branch context.

method	solved	time	nodes	fin.gap
`cpx`	1	9,780	253,090	21.1%
`cpx2`	2	9,183	52,605	22.8%
`cpx+fast`	3	6,723	104,406	16.2%
`cpx_def`	2	9,278	77,171	15.7%

The experiments show that the cuts generated by our procedure are rather effective on these hard instances. Indeed, `cpx+fast` is able to solve to proven optimality more instances—and to close more average gap—than the other two methods. Our method turns out to be particularly effective on instances `nsrand-ipx` and `rll3000`, where it is able to prove optimality in 102 and 877 seconds respectively, while `cpx` did not solve any of them within the 10,000-second time limit, and `cpx2` was only able to solve `rll3000` in 4,138 seconds.

Table 7 Detailed results in a Cut&Branch context.

	cpx			cpx2			cpx+fast		
problem	time	nodes	fin.gap	time	nodes	fin.gap	time	nodes	fin.gap
a1c1s1	10,000	1,817,501	7.87%	10,000	113,988	3.54%	10,000	499,301	4.72%
aflow40b	6,706	2,249,450	0.00%	5,212	136,650	0.00%	8,832	1,281,509	0.00%
atlanta-ip	10,000	7,994	9.35%	10,000	2,458	13.06%	10,000	6,312	10.31%
dano3mip	10,000	13,200	21.07%	10,000	5,365	20.29%	10,000	1,627	24.22%
danoint	10,000	1,201,099	1.49%	10,000	117,261	2.69%	10,000	690,565	2.46%
mkc	10,000	7,956,401	0.35%	10,000	736,801	0.15%	10,000	6,997,001	0.16%
momentum1	10,000	81,344	6.14%	10,000	50,100	0.04%	10,000	57,921	6.08%
momentum2	10,000	11,555	27.43%	10,000	4,121	100.00%	10,000	18,891	13.59%
msc98-ip	10,000	2,552	5.43%	10,000	1,451	12.11%	10,000	4,075	5.43%
net12	10,000	5,419	91.80%	10,000	2,501	43.78%	10,000	5,040	30.44%
nsrand-ipx	10,000	5,820,201	0.69%	10,000	662,731	0.51%	102	8,421	0.00%
protfold	10,000	20,119	36.54%	10,000	12,381	40.30%	10,000	15,942	37.12%
rd-rplusc-21	10,000	335,425	100.00%	10,000	30,651	100.00%	10,000	230,703	100.00%
roll3000	10,000	3,972,801	1.62%	4,138	164,675	0.00%	877	81,552	0.00%
seymour	10,000	159,308	1.65%	10,000	68,374	1.35%	10,000	184,992	1.90%
sp97ar	10,000	390,501	0.76%	10,000	241,501	0.79%	10,000	346,901	0.75%
swath	10,000	2,889,801	21.22%	10,000	1,126,301	13.86%	10,000	3,151,801	16.93%
timtab2	10,000	12,075,901	45.74%	10,000	929,301	57.53%	10,000	5,961,601	37.82%
	9,780	253,090	21.06%	9,183	52,605	22.78%	6,723	104,406	16.22%

6 Conclusions and future work

We have considered Gomory Mixed-Integer Cuts (GMICs) read from LP bases, as it is done customary in branch-and-cut methods, but in a new shell aimed at overcoming the notoriously bad behavior of these cuts in the long run. The new shell uses a relax-and-cut approach where the generated GMICs are not added to the current LP, but are stored in a cut pool and immediately relaxed in a Lagrangian fashion.

We have presented some variants of our basic method and we have computationally compared them with other methods from the literature. The results have shown that even simple implementations of the new idea are quite effective, and outperform their competitors in terms of both bound quality and speed. We are confident however that there is still room for improvement of our basic methods.

Future work should investigate the following research topics:

- The use of a more sophisticated Lagrangian dual optimizer to replace the simple subgradient procedure we implemented.
- The use of the Lagrangian solutions for primal heuristics. Indeed, during Lagrangian optimization a large number of (possibly slightly fractional or even integer) vertices of P are generated, that could be used heuristically to provide good primal MIP solutions. As a quick shot, we tried to round each vertex with a single iteration of Feasibility Pump 2.0—see [21] for details on the rounding algorithm—and compared the results with the objective Feasibility Pump 2.0 (`objFP2`) itself. Although, as expected, the success rate is definitely lower—we found a feasible solution in 40 out of 72 instances, while `objFP2` was successful in 69 instances—the quality of the

solutions is quite good. In particular, on the 40 instances where both methods are successful, the geometric mean of the gap of the solutions found by Lagrangian optimization is 2.54%, while it is 6.73% for the `objFP2`.
- The incorporation of the pseudo-branching mechanism described in Subsection 5.3 in a restart scheme akin to one investigated by Karzan, Nemhauser and Savelsbergh [25] for 0-1 MIPs. There, an incomplete branch-and-bound tree is run beforehand to gather information to be used, after a restart, to make more effective branching decisions. With a little overhead, in the preliminary phase one could also generate and store (without actually using) a large number of globally-valid rank-1 GMICs read from the optimal LP bases at the enumeration nodes. This is in fact possible because, for 0-1 MIPs, all branching variables are fixed to their lower or upper bound, hence one can easily get rid of the GMIC dependency on the branching constraints; see [5]. After restart, the collected GMICs can therefore be used, and in Subsection 5.3, to initialize the relax-and-cut pool.
- The possibility of using our relax-and-cut scheme for a more aggressive generation of rank-1 GMICs at branching nodes different from the root, as a continuation of the line of research introduced by Balas, Ceria, Cornuéjols and Natraj [5]. Indeed, we observe that the generation of GMICs during enumeration inherits from pure cutting plane methods all the issues deriving from "adding GMICs on top of other GMICs" (the latter generated in the parent nodes), hence numerical instability is likely to occur if no "decoupling filter" is introduced. In [5], cut decorrelation is achieved by a skip policy that avoids invoking GMIC separation for "similar" branching nodes, and by purging from the LP certain previously-generated cuts whose slacks are basic. In our relax-and-cut context, a natural strategy would be the one exploited in our `fast` variants of Subsection 4.3. Specifically, at certain branching nodes all the cuts added to the original LP (possibly including branching conditions) are just dualized, with their optimal dual multiplier, and the original LP with the Lagrangian costs is solved to produce a new round of rank-1 GMICs.

Finally, in the process of developing our method we realized that cutting plane schemes miss an overall "meta-scheme" to control cut generation and to escape "local optima" by means of diversification phases—very well in the spirit of Tabu or Variable Neighborhood Search meta-schemes for primal heuristics. The development of sound meta-schemes on top of a basic separation tool is therefore an interesting topic for future investigations—our relax-and-cut framework for GMICs can be viewed as a first step in this direction.

Acknowledgements This work was partially supported by the *Progetto di Ateneo* on "Computational Integer Programming" of the University of Padova; the second author was also partially supported by the *Progetto di Eccellenza 2008-2009* of the "Fondazione Cassa di Risparmio di Padova e Rovigo".

References

1. Achterberg, T.: Scip: solving constraint integer programs. Mathematical Programming Computation **1**, 1–41 (2009)
2. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters **34**(4), 1–12 (2006). DOI 10.1016/j.orl.2005.07.009. URL `http://www.zib.de/Publications/abstracts/ZR-05-28/`. See `http://miplib.zib.de`
3. Andreello, G., Caprara, A., Fischetti, M.: Embedding cuts in a branch and cut framework: a computational study with {0,1/2}-cuts. INFORMS Journal on Computing (19), 229–238 (2007)
4. Balas, E., Bonami, P.: Generating lift-and-project cuts from the LP simplex tableau: open source implementation and testing of new variants. Mathematical Programming Computation **1**(2–3), 165–199 (2009)
5. Balas, E., Ceria, S., Cornuéjols, G., Natraj, N.: Gomory cuts revisited. Operations Research Letters **19**, 1–9 (1996)
6. Balas, E., Jeroslow, R.G.: Strengthening cuts for mixed integer programs. European Journal of Operational Research **4**(4), 224–234 (1980)
7. Balas, E., Perregaard, M.: A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. Mathematical Programming **94**(2–3), 221–245 (2003)
8. Balas, E., Saxena, A.: Optimizing over the split closure. Mathematical Programming **113**(2), 219–240 (2008)
9. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Optima **58**, 12–15 (1998). See `http://www.caam.rice.edu/bixby/miplib/miplib.html`
10. Bonami, P., Minoux, M.: Using rank-1 lift-and-project closures to generate cuts for 0-1 MIPs, a computational investigation. Discrete Optimization **2**(4), 288–307 (2005)
11. Caprara, A., Fischetti, M., Toth, P.: A heuristic method for the set covering problem. Operations Research **47**(5), 730–743 (1999)
12. CglLandP: website. `https://projects.coin-or.org/Cgl/wiki/CglLandP`
13. COIN-OR: website. `http://www.coin-or.org/`
14. Cornuéjols, G.: Revival of the Gomory cuts in the 1990's. Annals of Operations Research **149**(1), 63–66 (2006)
15. Cornuéjols, G.: Valid inequalities for mixed integer linear programs. Mathematical Programming **112**(1), 3–44 (2008)
16. Cornuéjols, G., Li, Y.: Elementary closures for integer programs. Operations Research Letters **28**(1), 1–8 (2001)
17. Dash, S., Goycoolea, M.: A heuristic to generate rank-1 GMI cuts. Tech. rep., IBM (to appear in Mathematical Programming Computation) (2009)
18. Dash, S., Günlük, O., Lodi, A.: MIR closures of polyhedral sets. Mathematical Programming **121**(1), 33–60 (2010)
19. Escudero, L.F., Guignard, M., Malik, K.: A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. Annals of Operations Research **50**(1), 219–237 (1994)
20. Fischetti, M., Lodi, A.: Optimizing over the first Chvàtal closure. Mathematical Programming **110**(1), 3–20 (2007)
21. Fischetti, M., Salvagnin, D.: Feasibility pump 2.0. Mathematical Programming Computation **1**(2–3), 201–222 (2009)
22. Gomory, R.E.: An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation (1960)
23. Guta, B.: Subgradient optimization methods in integer programming with an application to a radiation therapy problem. Ph.D. thesis, University of Kaiserslautern (2003)
24. Hiriart-Hurruty, Lemaréchal, C.: Convex Analysis and Minimization Algorithms. Springer-Verlag (1993)
25. Karzan, F.K., Nemhauser, G.L., Savelsbergh, M.W.P.: Information-based branching schemes for binary linear mixed integer problems. Mathematical Programming Computation **1**(4), 249–293 (2009)

26. Lucena, A.: Steiner problems in graphs: Lagrangian optimization and cutting planes. COAL Bulletin (21), 2–8 (1982)
27. Lucena, A.: Non delayed relax-and-cut algorithms. Annals of Operations Research **140**(1), 375–410 (2005)
28. Lucena, A.: Lagrangian relax-and-cut algorithms. In: Handbook of Optimization in Telecommunications, pp. 129–145. Springer (2006)
29. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: On the capacitated vehicle routing problem. Mathematical Programming **94**(2–3), 343–359 (2003)
30. Zanette, A., Fischetti, M., Balas, E.: Lexicography and degeneracy: can a pure cutting plane algorithm work? Mathematical Programming (2009)

Appendix

Tables 8 to 11 give instance-by-instance results corresponding to the cumulative figures reported in the previous computational sections.

Table 8: Detailed rank-1 methods comparison for MIPLIB 3.0 (cumulative results in Table 2).

	1gmi		fast		faster		hybr		subg		dgDef	
problem	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time
air04	8.1	1.98	22.2	294.28	28.5	230.25	13.1	3,600.00	11.7	3,600.00	19.4	3,600.00
air05	4.7	0.69	10.9	169.31	11.2	122.34	11.4	3,600.00	8.9	3,600.00	12.9	3,600.00
arki001	27.2	0.04	44.5	6.63	44.5	5.08	44.5	22.53	39.6	0.91	35.5	394.84
bell3a	45.1	0.00	64.6	0.16	64.6	0.08	64.6	0.99	64.0	8.53	74.0	0.82
bell5	14.5	0.00	87.4	0.23	86.5	0.12	90.1	1.76	63.3	6.22	23.7	0.32
blend2	16.4	0.00	22.4	0.27	22.0	0.13	25.4	11.43	23.7	16.97	21.4	2.07
cap6000	42.9	0.05	55.2	2.41	55.2	1.51	55.2	71.93	55.2	156.55	62.4	14.65
dano3mip	0.0	174.66	0.3	1,290.87	0.3	1,999.63	0.3	370.35	0.0	144.70	0.3	3,600.00
danoint	0.3	0.07	2.7	16.53	0.9	5.20	0.8	1.67	0.5	0.30	1.7	235.69
dcmulti	47.7	0.01	93.2	3.51	91.1	1.73	92.4	41.05	89.7	124.07	91.4	12.40
egout	51.6	0.00	95.1	0.15	93.7	0.05	97.9	0.63	98.6	4.38	98.7	0.30
fast0507	1.4	10.99	2.4	145.17	3.6	212.65	2.2	3,600.00	1.9	3,600.00	1.7	3,600.00
fiber	70.6	0.01	98.0	1.36	97.1	0.95	98.6	29.01	98.0	77.82	98.1	14.78
fixnet6	10.5	0.01	82.7	1.95	82.5	1.23	86.8	48.11	72.8	60.39	86.6	15.55
flugpl	11.7	0.00	12.2	0.02	11.7	0.01	12.2	0.21	12.2	1.30	11.7	0.01
gen	59.4	0.01	82.9	1.01	78.0	0.68	89.4	17.46	87.8	81.34	91.8	10.50
gesa2	30.9	0.03	62.5	2.71	62.1	1.22	68.5	69.89	59.8	119.87	85.6	45.56
gesa2_o	30.9	0.02	72.7	4.43	69.9	2.23	77.0	98.56	72.1	95.73	58.8	78.47
gesa3	45.8	0.03	93.9	4.45	92.2	2.91	94.7	30.48	94.3	198.03	92.6	119.48
gesa3_o	49.2	0.02	94.7	3.74	64.2	0.81	95.0	53.03	94.6	190.17	93.4	108.54
gt2	91.9	0.00	98.3	0.10	98.2	0.08	95.3	0.86	95.0	0.42	96.6	0.29
harp2	13.6	0.03	41.3	5.75	38.4	2.87	40.6	42.93	42.0	124.80	67.3	240.81
khb05250	74.9	0.01	99.9	0.59	99.9	0.37	100.0	7.24	99.9	21.02	99.5	3.49
l152lav	11.1	0.04	61.2	29.23	55.5	11.10	60.8	218.45	55.6	924.12	48.7	402.71
lseu	53.4	0.00	89.7	0.19	84.8	0.07	89.0	1.46	88.6	4.40	84.6	0.12
mas74	6.4	0.00	8.8	0.66	8.8	0.34	9.0	2.65	9.0	5.97	8.5	0.63
mas76	6.3	0.00	10.6	0.54	10.2	0.55	10.3	3.97	9.9	5.67	10.3	1.13
misc03	8.6	0.00	26.4	1.48	26.1	0.78	24.1	3.27	25.4	50.21	20.6	6.15
misc06	30.6	0.02	85.1	1.13	88.4	0.83	41.5	0.10	37.1	0.06	86.2	34.46
misc07	0.7	0.01	6.7	3.35	4.4	1.50	7.7	38.12	5.8	89.41	2.1	11.08
mitre	83.8	0.22	100.0	4.88	100.0	3.71	100.0	7.93	100.0	14.03	100.0	84.74
mkc	1.7	0.10	59.8	44.96	47.5	12.36	78.4	717.86	65.1	955.14	49.3	2,058.05
mod008	20.1	0.00	57.3	0.15	66.0	0.10	73.7	0.94	73.2	12.66	62.5	0.62
mod011	17.1	0.17	56.8	386.22	54.0	182.74	75.8	1,060.66	53.4	727.89	32.2	3,600.00
modglob	17.3	0.01	83.6	1.17	81.6	0.71	83.6	18.03	78.8	25.67	75.3	8.59

Table 8: Detailed rank-1 methods comparison for MIPLIB 3.0 (cumulative results in Table 2).

	1gmi		fast		faster		hybr		subg		dgDef	
problem	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time
nw04	66.1	1.06	100.0	73.66	100.0	24.83	100.0	28.83	100.0	29.44	100.0	84.74
p0033	56.8	0.00	76.4	0.09	76.4	0.06	76.7	0.37	76.4	1.65	83.2	0.10
p0201	33.8	0.01	74.3	0.88	73.6	0.46	73.1	6.04	72.9	18.19	61.3	6.66
p0282	3.7	0.01	98.1	0.72	96.4	0.49	98.2	12.14	95.5	18.81	96.9	3.55
p0548	40.7	0.01	92.3	0.91	92.1	0.55	96.6	11.98	75.8	16.79	94.3	2.67
p2756	0.5	0.02	84.0	2.59	74.6	1.11	86.7	49.27	81.7	65.20	96.9	17.05
pp08a	52.1	0.00	71.5	0.04	71.5	0.04	70.2	0.04	61.1	0.01	93.8	3.12
pp08aCUTS	31.5	0.01	50.0	0.09	50.0	0.10	49.3	0.06	39.4	0.02	83.3	21.52
qiu	0.2	0.17	31.3	43.03	30.8	22.36	38.6	407.60	33.7	719.55	25.9	3,600.00
qnet1	12.6	0.06	78.4	14.10	60.3	5.88	81.5	168.77	72.0	340.93	79.3	437.03
qnet1_o	33.4	0.03	86.5	6.24	82.4	3.13	89.8	55.44	87.2	116.30	93.5	196.73
rentacar	5.0	0.30	29.0	17.08	21.7	13.62	7.9	2.13	5.3	0.57	45.0	1,123.07
rgn	5.0	0.00	85.5	0.17	98.8	0.19	100.0	1.76	48.3	0.63	99.6	0.54
rout	1.4	0.03	37.8	7.42	33.6	3.14	44.0	91.50	42.5	316.38	35.5	180.27
set1ch	38.1	0.01	39.9	0.81	39.9	0.58	42.4	0.27	38.9	0.03	83.6	30.66
seymour	7.8	1.69	32.2	202.93	34.1	105.80	44.0	2,500.83	25.4	1,357.82	20.8	3,600.00
swath	6.4	0.07	34.9	3.89	34.9	11.36	34.9	186.13	34.9	2,489.35	34.0	64.99
vpm1	10.0	0.00	36.9	0.61	36.6	0.46	35.9	7.50	35.0	17.35	89.8	5.10
vpm2	12.0	0.00	58.1	1.36	52.8	0.62	64.0	22.30	56.6	41.62	61.2	8.03
	26.4	0.03	58.9	2.57	57.1	1.53	60.1	15.78	55.0	24.97	60.8	23.78

Table 9 Detailed rank-1 methods comparison for MIPLIB 2003 (cumulative results in Table 2).

	`1gmi`		`fast`		`faster`		`hybr`		`subg`		`dgDef`	
problem	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time
a1c1s1	19.2	0.07	57.7	94.60	56.6	41.46	61.2	1,153.60	58.6	1,744.05	54.3	3,578.97
aflow30a	11.9	0.02	40.6	4.60	38.2	1.49	42.2	67.96	39.3	208.81	46.8	98.09
aflow40b	8.0	0.07	27.6	14.35	23.7	6.13	32.3	204.92	27.8	689.47	33.5	1,163.44
atlanta-ip	0.7	40.19	2.6	1,747.60	2.2	1,382.71	1.9	3,600.00	1.9	3,600.00	1.1	3,600.00
glass4	0.0	0.00	0.0	1.06	0.0	0.59	0.0	0.08	0.0	0.01	0.0	1.51
momentum1	32.3	4.40	64.5	208.10	58.6	90.14	64.5	3,600.00	64.5	809.16	41.1	3,600.00
momentum2	39.1	5.05	39.3	357.07	39.3	302.69	41.9	3,600.00	39.4	3,600.00	40.7	3,600.00
msc98-ip	45.1	82.98	54.8	1,795.88	54.2	1,096.94	57.1	3,600.00	54.2	3,600.00	46.8	3,600.00
mzzv11	9.8	20.02	82.5	425.20	79.9	308.82	97.9	3,600.00	92.6	3,631.02	24.7	3,600.00
mzzv42z	10.7	36.11	81.3	305.08	73.3	170.97	92.5	1,904.46	80.2	2,424.21	51.8	3,600.00
net12	7.3	5.35	22.7	465.75	21.9	244.98	21.8	3,600.00	20.3	3,600.00	13.6	3,600.00
nsrand-ipx	36.0	0.19	80.7	66.58	80.1	38.76	80.2	362.10	79.0	1,218.06	80.5	1,545.22
opt1217	22.8	0.01	34.7	0.48	34.9	0.76	35.9	4.34	35.9	61.96	32.3	3,600.00
protfold	5.0	6.64	18.2	162.72	22.6	152.63	30.3	3,600.00	27.6	3,600.00	11.7	3,600.00
rd-rplusc-21	0.0	2.50	0.0	896.12	0.0	296.84	0.0	3,600.00	0.0	3,600.00	0.0	1,491.32
roll3000	3.6	0.23	84.0	54.51	75.8	17.15	86.5	740.50	84.6	974.92	51.9	1,880.39
sp97ar	11.9	2.24	45.3	261.01	39.3	180.61	55.0	3,600.00	34.2	1,993.52	28.4	3,600.00
timtab1	23.6	0.00	51.8	1.50	51.8	0.84	51.1	1.21	34.6	0.11	77.8	21.62
timtab2	18.0	0.01	45.4	3.20	42.7	1.80	43.9	3.03	26.3	0.20	69.2	47.65
tr12-30	60.3	0.03	75.8	7.21	70.2	3.82	76.1	89.04	68.6	127.10	87.3	42.15
	18.3	0.54	45.5	58.40	43.3	33.33	48.6	314.73	43.5	290.82	39.7	853.85

Table 10: Detailed results for GMICs on top of other cuts, on MIPLIB 3.0 (cumulative results in Table 5).

	cpx		cpx+fast		cpx2		cpx2+fast	
problem	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time
air04	17.3	2.27	43.8	195.27	19.8	7.57	42.2	207.39
air05	17.0	2.58	23.7	124.56	20.3	5.98	24.5	197.58
arki001	31.3	1.08	48.6	12.24	40.4	2.74	50.3	13.26
bell3a	69.3	0.01	70.7	0.45	70.7	0.01	70.7	0.24
bell5	92.3	0.01	92.4	0.08	96.7	0.03	97.2	0.31
blend2	19.1	0.10	26.3	0.78	28.6	0.18	30.9	0.82
cap6000	42.9	0.32	55.2	6.39	42.9	0.33	55.2	6.68
dano3mip	1.1	349.16	1.1	1,096.14	1.1	4,314.83	1.1	1,599.57
danoint	2.7	1.18	3.6	37.55	2.9	20.13	3.5	34.10
dcmulti	75.1	0.21	94.6	4.81	90.1	1.32	96.1	3.99
egout	100.0	0.00	100.0	0.00	100.0	0.01	100.0	0.01
fast0507	0.0	21.22	3.5	183.47	1.8	64.51	3.5	250.19
fiber	91.1	0.11	98.7	1.33	92.0	0.19	98.9	1.28
fixnet6	86.9	0.84	95.4	2.07	90.1	1.12	94.9	1.81
flugpl	46.5	0.00	48.5	0.03	54.3	0.01	54.4	0.14
gen	100.0	0.01	100.0	0.01	100.0	0.02	100.0	0.02
gesa2	79.6	0.21	81.7	3.83	98.7	0.59	98.8	3.66
gesa2_o	73.9	0.19	77.1	3.88	93.7	1.01	96.8	4.46
gesa3	71.3	0.33	91.0	5.73	74.6	0.99	86.6	5.75
gesa3_o	70.8	0.26	88.2	5.19	76.0	0.58	97.4	3.50
gt2	100.0	0.01	100.0	0.01	100.0	0.01	100.0	0.01
harp2	33.0	0.15	59.7	4.21	45.2	0.15	60.4	4.72
khb05250	99.9	0.06	100.0	1.08	100.0	0.69	100.0	0.69
l152lav	0.2	0.12	58.5	31.16	15.6	0.60	58.7	22.44
lseu	68.4	0.02	88.9	0.35	88.4	0.13	91.7	0.33
mas74	3.5	0.03	9.2	0.81	6.4	0.17	9.2	0.93
mas76	3.5	0.03	8.9	0.31	6.3	0.13	11.1	1.35
misc03	21.9	0.11	28.3	1.88	26.3	2.53	28.5	1.79
misc06	27.3	0.06	81.9	1.70	62.2	0.39	87.3	1.91
misc07	3.2	0.17	7.3	3.29	7.0	1.23	9.1	3.80
mitre	100.0	0.45	100.0	0.45	100.0	0.49	100.0	0.49
mkc	43.1	0.44	76.9	18.36	57.1	0.94	83.3	17.53
mod008	22.9	0.02	71.3	0.34	32.8	0.02	78.4	0.38
mod011	98.3	18.01	98.7	127.36	99.1	22.64	99.1	103.06
modglob	91.7	0.07	95.2	0.52	95.1	0.63	97.9	0.73
nw04	8.6	12.13	108.3	12.28	100.0	39.39	100.0	39.39
p0033	88.1	0.01	100.0	0.02	100.0	0.00	100.0	0.00
p0201	33.8	0.07	78.3	1.56	66.7	1.36	80.4	1.41
p0282	97.0	0.06	99.9	0.47	97.1	0.08	99.8	0.39
p0548	99.8	0.03	100.0	0.08	100.0	0.02	100.0	0.02
p2756	98.6	0.20	99.9	2.83	99.2	0.26	99.9	2.09
pp08a	95.7	0.13	96.2	0.71	96.4	0.95	96.8	0.75
pp08aCUTS	82.2	0.18	84.9	1.15	86.7	1.48	87.5	1.03
qiu	0.1	0.69	35.2	46.57	42.8	58.27	46.8	55.42
qnet1	71.6	0.90	91.6	8.45	80.0	1.27	92.8	7.37
qnet1_o	88.8	0.85	92.7	2.89	91.3	0.75	95.7	3.42
rentacar	54.1	0.47	67.6	17.32	62.0	5.27	73.1	18.29
rgn	26.6	0.01	100.0	0.27	93.5	0.08	100.0	0.09
rout	1.9	0.20	40.5	8.66	8.7	0.66	39.9	5.88
set1ch	97.8	0.11	97.8	2.15	99.5	0.24	99.5	1.83
seymour	41.0	8.53	44.8	147.86	43.8	102.21	47.1	203.91
swath	34.6	0.45	34.9	6.56	34.9	1.53	34.9	1.55
vpm1	100.0	0.01	100.0	0.58	100.0	0.01	100.0	0.01
vpm2	72.9	0.02	78.9	0.72	84.8	0.37	86.1	0.83
	55.5	0.17	70.0	2.14	65.3	0.56	72.2	1.98

Table 11 Detailed results for GMICs on top of other cuts, on MIPLIB 2003 (cumulative results in Table 5).

	`cpx`		`cpx+fast`		`cpx2`		`cpx2+fast`	
problem	cl.gap	time	cl.gap	time	cl.gap	time	cl.gap	time
a1c1s1	84.0	3.57	88.1	40.01	86.0	476.92	88.9	37.75
aflow30a	57.1	3.03	61.8	4.40	62.4	3.47	65.2	4.33
aflow40b	35.2	7.10	43.3	15.86	45.9	7.76	51.7	18.65
atlanta-ip	1.1	1,394.00	1.4	1,226.27	1.0	1,232.67	1.6	1,187.05
glass4	0.0	0.41	0.0	1.78	0.0	2.59	0.0	1.97
momentum1	64.5	12.05	64.5	241.32	64.6	195.33	64.6	289.34
momentum2	55.1	67.09	59.6	418.83	68.2	485.27	68.4	422.68
msc98-ip	50.5	164.71	52.5	704.31	57.1	286.25	57.1	547.79
mzzv11	81.6	34.70	98.1	369.08	82.5	37.12	98.4	303.07
mzzv42z	74.9	17.23	97.0	211.82	79.4	28.69	94.6	222.77
net12	30.9	17.88	41.6	511.41	34.3	84.11	42.1	552.30
nsrand-ipx	54.3	2.58	83.0	51.05	65.6	4.80	82.7	50.78
opt1217	100.0	0.11	100.0	1.52	100.0	0.26	100.0	1.53
protfold	8.5	42.36	22.4	193.45	18.9	110.69	24.4	457.71
rd-rplusc-21	0.0	9.23	0.0	175.73	0.0	29.71	0.0	237.48
roll3000	62.0	1.64	85.2	34.29	72.4	5.45	85.3	41.58
sp97ar	11.4	10.59	41.2	303.05	13.4	20.97	46.8	296.34
timtab1	63.0	0.30	67.2	3.04	58.3	0.98	64.7	2.91
timtab2	46.0	0.68	57.9	6.42	48.6	2.75	57.2	6.75
tr12-30	99.2	0.90	99.2	10.05	99.2	8.55	99.2	9.25
	49.0	6.57	58.2	56.03	52.9	22.48	59.6	59.75