Math. Prog. Comp. (2012) 4:151–179
DOI 10.1007/s12532-012-0037-0

FULL LENGTH PAPER

On optimizing over lift-and-project closures

Pierre Bonami

Received: 26 October 2010 / Accepted: 29 February 2012 / Published online: 16 March 2012

Abstract The strengthened lift-and-project closure of a mixed integer linear program is the polyhedron obtained by intersecting all strengthened lift-and-project cuts obtained from its initial formulation, or equivalently all mixed integer Gomory cuts read from all tableaux corresponding to feasible and infeasible bases of the LP relaxation. In this paper, we present an algorithm for approximately optimizing over the strengthened lift-and-project closure. The originality of our method is that it relies on a cut generation linear programming problem which is obtained from the original LP relaxation by only modifying the bounds on the variables and constraints. This separation LP can also be seen as dual to the cut generation LP used in disjunctive programming procedures with a particular normalization. We study properties of this separation LP, and discuss how to use it to approximately optimize over the strengthened lift-and-project closure. Finally, we present computational experiments and comparisons with recent related works.

Mathematics Subject Classification 90C11 · 90C57

1 Introduction

In this paper, we consider Mixed Integer Linear Programs of the form:

$$\begin{aligned} &\max\ c^\top x \\ &\text{s.t.:} \\ &Ax = b, \\ &x \in \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+, \end{aligned} \tag{MILP}$$

where A is an $m \times n$ rational matrix of full row rank, $c \in \mathbb{Q}^n$ and $b \in \mathbb{Q}^m$.

P. Bonami (✉)
LIF, CNRS, Aix Marseille University, 163 Avenue de Luminy - Case 901, 13008 Marseille, France
e-mail: pierre.bonami@lif.univ-mrs.fr

In the last 20 years, cutting plane methods have become one of the main ingredients for solving (MILP). State of the art solvers implement branch-and-cut algorithms that employ a number of different cutting plane algorithms to automatically strengthen formulations: Gomory Mixed Integer (GMI) Cuts [32], Mixed Integer Rounding cuts [37], Knapsack Covers [13], Flow-covers [38,39]. The positive impact of cutting plane methods on exact algorithms for MILP has been evaluated in several independent computational studies, for example [15].

A central concept in cutting planes for (MILP) is that of elementary closures. The elementary closure of a given family of cuts is the convex set obtained by applying all cuts of the family that can be directly obtained from the initial formulation of (MILP (i.e. without a recursive application). Two well known closures are the Chvátal–Gomory closure [20] obtained by applying all possible Gomory fractional cuts [33], and the split closure [21]. Elementary closures have been a fruitful concept from a theoretical standpoint. In particular, the Chvátal–Gomory and the split closures are polyhedra over which it is NP-hard to optimize [19–21,26]. More recently, these closures have also been considered from a computational point of view. Several authors evaluated empirically the strength of different closures on problem instances from the literature by optimizing over these closures: Fischetti and Lodi optimized over the Chvátal–Gomory closure [27], Bonami et. al. [17] the projected Chvátal–Gomory closure, Balas and Saxena [12] the split closure and Dash et al. [25] the MIR closure. These studies showed that elementary closures often give rise to strong relaxations. In particular, by optimizing over the split or MIR closure (which are equivalent) one often closes a significant portion of the integrality gap. Unfortunately, the computing times to optimize over these closures following the approaches proposed so far are often too large for practical purposes. An interesting line of research is to try to approximate the bound obtained by optimizing over these closures in faster computing times.

In this paper, we are interested in approximating the split closure by looking at a weaker closure: the *strengthened lift-and-project closure*. Balas and Perregaard [11] showed that this closure can be equivalently defined by looking at two families of cuts: GMI cuts read from LP tableaux or strengthened lift-and-project cuts [7]. The strengthened lift-and-project closure is a polyhedron, but to the best of our knowledge it is not known whether one can optimize over it in polynomial time. We also use a weaker closure, over which one can optimize in polynomial time: the lift-and-project closure. Our goal is to devise algorithms to optimize over the lift-and-project closure and approximate the value obtained by optimizing over the strengthened lift-and-project closure.

This problem was already studied by Bonami and Minoux [18], and we follow a similar approach to the one proposed there: we optimize over the lift-and-project closure by means of a cutting plane algorithm, and strengthen each cut individually to approximate the strengthened lift-and-project closure. Our main contribution with respect to [18] is that we use a different cut generation oracle. While in [18] a linear program in an extended space is solved to generate each cut, here we solve a linear program in the original space of variables which we call *membership LP*.

Our work is also related to the equivalence established by Balas and Perregaard [11]. The membership LP we use is related to disjunctive programming and the classical Cut Generation LP. We show that it is dual to a simplified version of the CGLP used in

[11]. The membership LP is also related to the LP relaxation of (MILP). It is obtained by modifying the bounds on the variables and the right-hand-side of the constraints of the LP relaxation. We study properties of the basic dual feasible solutions of the membership LP and show that those of negative dual cost correspond to simple intersection cuts [3] from basic solutions of the LP relaxation. This is an analogous to the equivalence proved in [11] in our simpler setting. Finally, an algorithm was proposed in [11] to separate lift-and-project cuts in the tableau of the LP relaxation of (MILP). In our setting, the analogous of this algorithm is simply to solve the membership LP by the simplex algorithm. One limitation of the algorithm in [11] is that it only works by improving an existing cut, our procedure removes this limitation.

Our work is also related to several recent computational studies. Dash and Goycoolea [24] and Fischetti and Salvagnin [30] proposed two different methods for separating GMI cuts from alternate bases of the LP relaxation. Our work can be seen as a third proposal along these lines. We note that the membership LP that we use was also independently used in a recent work by Fischetti and Salvagnin [29] to optimize over the lift-and-project closure.

The paper is organized as follows. In Sect. 2, we recall the definitions of the various cutting planes we study and their closures. In Sect. 3, we introduce the membership LP and study its relations to CGLP. In Sect. 4, we establish various properties of the solutions of the membership LP and relate it to intersection cuts. In Sect. 5, we discuss the practical solution of the membership LP and its use for optimizing over the lift-and-project closure. Finally, in Sect. 6, we present computational results.

2 Definitions and basic results

The Linear Programming relaxation of (MILP) is obtained by dropping the integrality constraints:

$$\begin{aligned} &\max\ c^\top x \\ &\text{s.t.:} \\ &Ax = b, \\ &x \in \mathbb{R}^n_+. \end{aligned} \tag{LP}$$

We denote by $P := \{x \in \mathbb{R}^n_+ : Ax = b\}$ the feasible set of (LP), by $N := \{1, \ldots, n\}$ the index set of variables, by $N^I := \{1, \ldots, p\}$ the index set of integer constrained variables and by $M := \{1, \ldots, m\}$ the index set of constraints. Given an index set I, we denote by x_I the vector formed by the components of x indexed by $i \in I$, by A^I the matrix formed by the columns of A indexed by $i \in I$, and by A_I the matrix formed by the rows of A indexed by $i \in I$. By abuse of notations we denote $A^{\{j\}}$ and $A_{\{i\}}$ by A^j and A_i respectively.

We recall that a subset of variables is a basis of (LP) if the sub-matrix formed by the columns of A corresponding to these variables is non-singular. We denote the index set of variables in a basis by $B := \{i_1, \ldots, i_m\}$ and the index set of nonbasic variables by $J := N \backslash B$. The matrix A^B is the basis matrix, and the LP tableau corresponding to B is given by

$$x_B = (A^B)^{-1}b - (A^B)^{-1}A^J x_J.$$

We denote $\overline{A} = (A^B)^{-1}A$ and $\overline{b} = (A^B)^{-1}b$. Finally, we define $P(B)$ to be the cone obtained by dropping the non-negativity requirements on the basic variables x_j ($j \in B$):

$$P(B) := \{x \in \mathbb{R}^{n+m} : x_B = \overline{b} - \overline{A}^J x_J, \quad x_J \geq 0\}.$$

Next, we define the four cutting plane families used in this paper.

2.1 Cutting planes

First, we briefly recall the concept of split cuts which is central to the cutting plane methods we use.

A *split disjunction* is a condition of the form $(x \in P, \pi^\top x \leq \pi_0) \vee (x \in P, \pi^\top x \geq \pi_0 + 1)$, with $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$ such that $\pi_{\{p+1,\ldots,n\}} = 0$. Clearly every solution of (MILP) satisfies one term of the split disjunction. *Split cuts* [21] are the inequalities that are valid for both terms of a split disjunction, or equivalently for the set:

$$P^{(\pi,\pi_0)} := \operatorname{conv}\left(\left\{x \in P : \pi^\top x \leq \pi_0\right\} \cup \left\{x \in P : \pi^\top x \geq \pi_0 + 1\right\}\right).$$

All the inequalities presented here are split cuts.

The simple intersection cut Let B be the index set of a basis of (LP), $\overline{x}$ be the corresponding basic solution, and $k \in B \cap N^I$ be such that $\overline{x}_k \notin \mathbb{Z}$. Assume that x_k is basic in the ith row of the simplex tableau. This row can therefore be written as:

$$x_k + \sum_{j \in J} \overline{a}_{ij} x_j = \overline{b}_i (= \overline{x}_k) \tag{1}$$

Let $f_0 = \overline{x}_k - \lfloor \overline{x}_k \rfloor$. The *intersection cut* [3] from the convex set $\{x \in R^n : \lfloor \overline{x}_k \rfloor \leq x_k \leq \lceil \overline{x}_k \rceil\}$ applied to (1) is

$$\sum_{j \in J} \max\left\{\overline{a}_{ij}(1 - f_0),\ -\overline{a}_{ij} f_0\right\} x_j \geq f_0(1 - f_0) \tag{2}$$

This cut is valid for all solutions to (MILP) and is also known as the *simple disjunctive cut* from the condition $x_k \leq \lfloor \overline{x}_k \rfloor \vee x_k \geq \lceil \overline{x}_k \rceil$.

Gomory Mixed Integer Cut The simple intersection cut from row (1) defined by (2) can be strengthened using the integrality of the variables in $J \cap N^I$. For each $j \in N^I$, we define $f_j := \overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor$. The Gomory mixed integer split $\pi(B, k)$ corresponding to the basis indexed by B and the basic variable x_k is defined to be

$$\pi(B,k)_j := \begin{cases} \lfloor \overline{a}_{ij} \rfloor & \text{if } j \in N^I \text{ and } f_j \leq f_0, \\ \lceil \overline{a}_{ij} \rceil & \text{if } j \in N^I \text{ and } f_j > f_0, \\ 0 & \text{if } j \in N \backslash N^I \end{cases}$$

(note that $\pi(B,k)_k = 1$ and $\pi(B,k)_j = 0, \forall j \in B\backslash\{k\}$). The Gomory Mixed-Integer Cut [32] is the intersection cut from the convex set $\{x \in R^n : \lfloor \overline{x}_k \rfloor \leq \sum_{j\in J} \pi(B,k)_j x_j \leq \lceil \overline{x}_k \rceil\}$ applied to (1). The GMI cut dominates all intersection cuts obtained from row (1) and convex sets of the form $\{x \in R^{n+m} : \lfloor \overline{x}_k \rfloor \leq \pi^\top x \leq \lceil \overline{x}_k \rceil\}$ with $\pi_{(J\backslash N^I)\cup(B\backslash\{k\})} = 0$ and $\pi_k = 1$ (see [1] for a proof).

The lift-and-project cut Let e_k be the kth unit vector of the canonical basis. Lift-and-project cuts are the valid inequalities for any set $P^{(e_k,\pi_0)}$ with $k = 1, \ldots, p$ and $\pi_0 \in \mathbb{Z}$.

The fundamental theorem of Balas [4,5] on unions of polyhedra allows the formulation of the separation problem for $P^{(e_k,\pi_0)}$ as a linear program.

Theorem 1 ([5,7]) *$\hat{x} \in P^{(\pi,\pi_0)}$ if and only if the optimal solution to*

$$\begin{aligned} &\min \alpha^\top \hat{x} - \beta \\ &s.t.: \\ &\alpha = A^\top u + s - u_0\pi, \\ &\alpha = A^\top v + t + v_0\pi, \\ &\beta = b^\top u - u_0\pi_0, \\ &\beta = b^\top v + v_0(\pi_0 + 1), \\ &\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}, u, v \in \mathbb{R}^m, s, t \in \mathbb{R}^n_+, u_0, v_0 \in \mathbb{R}_+, \end{aligned} \qquad \text{(CGLP}(\pi,\pi_0)\text{)}$$

has a non-negative objective value.

If the solution to the cut generation linear program is negative, $\alpha^\top x \geq \beta$ defines a valid inequality that cuts off the point $\hat{x}$.

Lift-and-project cuts are separated by solving (CGLP(e_k,π_0)). Since this program is unbounded whenever it has a negative solution, a *normalization constraint* is often added to truncate the cone of feasible solutions. Following [10], the most commonly used constraint is $\sum_{i\in M}(u_i + v_i) + \sum_{i\in N}(s_i + t_i) + u_0 + v_0 = 1$. Since it is the most commonly encountered form, we call in the remainder (CGLP(e_k,π_0)) augmented with this normalization condition simply (CGLP).

The strengthened lift-and-project cut Lift-and-project cuts can be strengthened in a similar fashion as simple intersection cuts can be strengthened to GMI cuts.

Consider a basic solution $(\hat{\alpha}, \hat{\beta}, \hat{u}, \hat{v}, \hat{s}, \hat{t}, \hat{u}_0, \hat{v}_0)$ to (CGLP(e_k,π_0)). The coefficients $\hat{\alpha}$ satisfy:

$$\begin{aligned} \hat{\alpha}_k &= \max\{\hat{u}^\top A^k - \hat{u}_0, \hat{v}^\top A^k + \hat{v}_0\}, \\ \hat{\alpha}_j &= \max\{\hat{u}^\top A^j, \hat{v}^\top A^j\}, \qquad j \in N\backslash\{k\}. \end{aligned}$$

These coefficients can be strengthened by replacing $\hat{\alpha}_j$, for $j \in \{1, \ldots, p\} \backslash \{k\}$, with

$$\overline{\alpha}_j := \min\{\hat{u}^\top A^j - \hat{u}_0 \lfloor m_j \rfloor, \hat{v}^\top A^j + \hat{v}_0 \lceil m_j \rceil\}$$

where

$$m_j := \frac{\hat{u}^\top A^j - \hat{v}^\top A^j}{\hat{u}_0 + \hat{v}_0},$$

(see [8,9] for justifications.)

2.2 Equivalences and closures

Balas and Perregaard showed that every simple intersection (resp. GMI) cut from a basic solution (feasible or not) of (LP) is equivalent to a lift-and-project (resp. strengthened lift-and-project) cut obtained from a basic feasible solution of (CGLP); and conversely every lift-and-project (resp. strengthened) cut derived from a basic feasible solution of (CGLP) is equivalent to a simple intersection (resp. GMI) cut obtained from a well defined basic solution of (LP). Here equivalent means that the two cuts are identical up to multiplication by a positive scalar. The equivalence gives the precise correspondence between the two bases of (LP) and (CGLP). In view of this result, solving (CGLP) can be re-interpreted as a method for finding a good basis from which to generate a GMI cut.

Note that this result is stated in [11] for 0-1 MILP with inequalities constraints but its generalization to (MILP) is straightforward.

For a given family of cuts, we call rank 1 cuts those cuts that can directly be deduced from the initial formulation of (MILP). The elementary closure of a family of cuts consists of the intersection of all rank 1 cuts of that family. A direct consequence of the result aforementioned is that lift-and-project cuts and simple intersection cuts on one hand, GMI cuts and strengthened lift-and-project cuts on the other hand are equivalent in terms of their closures [22].

The *lift-and-project closure* is the intersection of the polyhedra $P^{(e_k,\pi_0)}$ for all $k \in \{1, \ldots, p\}$ and all $\pi_0 \in \mathbb{Z}$, or equivalently the intersection of all simple intersection cuts from all bases (feasible and infeasible) of (LP). We denote it by P_e. Defining $\mathcal{B}$ to be the set of all index sets of bases of (LP), we have:

$$\begin{aligned} P_e &= \bigcap_{\substack{k \in N^I \\ \pi_0 \in \mathbb{Z}}} P^{(e_k,\pi_0)} \\ &= \bigcap_{B \in \mathcal{B}} \bigcap_{k \in N^I \cap B} \operatorname{conv}\left(\{x \in P\,(B) : x_k \leq \lfloor \overline{x}_k \rfloor \vee x_k \leq \lceil \overline{x}_k \rceil\}\right). \end{aligned}$$

The *strengthened lift-and-project closure* is defined by taking all strengthened lift-and-project cuts obtained from all basic feasible solutions to (CGLP), or equivalently all GMI cuts defined from all bases of (LP):

$$P_{e^*} = \bigcap_{B \in \mathcal{B}} \bigcap_{k \in N^I \cap B} \operatorname{conv}\left(\left\{x \in P\,(B) : \pi(B,k)^\top x \leq \lfloor \overline{x}_k \rfloor \vee \pi(B,k)^\top x \geq \lceil \overline{x}_k \rceil\right\}\right).$$

Since P_e and P_{e^*} are the intersection of a finite number of polyhedra they are clearly polyhedra. Moreover, one can optimize over P_e in polynomial time. To the best of our knowledge, it is not known whether one can optimize over P_{e^*} in polynomial time.

Finally, we recall that the split closure P_S is obtained by intersecting all split cuts:

$$P_S := \bigcap_{\substack{(\pi,\pi_0) \in \mathbb{Z}^n \times \mathbb{Z} \\ \pi_{\{p+1,\ldots,n\}} = 0}} P^{(\pi,\pi_0)}.$$

P_S is a polyhedron [21]. Andersen, Cornuéjols and Li [2] have shown that P_S can also be defined in terms of intersection cuts. From the definitions, it should be clear that $P_S \subseteq P_{e^*} \subseteq P_e$. It is NP-Hard to optimize over P_S. Computational studies such as [12,25] have shown that the relaxation obtained by optimizing over P_S can be strong, but computational times are often prohibitive.

3 The membership LP

In this section, we establish a linear program which given $(\pi, \pi_0) \in \mathbb{Z} \times \mathbb{Z}_0$ and a point $\hat{x} \in P$ answers the question of whether $\hat{x} \in P^{(\pi,\pi_0)}$. It should be clear that this question can be answered by solving (CGLP(π, π_0)) using Theorem 1. Our linear program is indeed equivalent to (CGLP(π, π_0)) using duality, but it is formulated in the same space as (LP).

Our membership LP is based on the following proposition.

Proposition 1 *Let $\hat{x} \in P$ and $(\pi, \pi_0) \in \mathbb{Z} \times \mathbb{Z}_0$ be such that $\pi_0 < \pi^\top \hat{x} < \pi_0 + 1$. $\hat{x} \in P^{(\pi,\pi_0)}$ if and only if there exists $y \in \mathbb{R}^n$ such that*

$$\begin{aligned} &\pi^\top y - (\pi_0 + 1)\left(\pi^\top \hat{x} - \pi_0\right) \geq 0, \\ &Ay = b\left(\pi^\top \hat{x} - \pi_0\right), \\ &0 \leq y \leq \hat{x}. \end{aligned} \tag{3}$$

This proposition can be proved in different ways. In particular, it is closely related to classical results of disjunctive programming. We will discuss this connection in the following paragraphs. First, we give a short and self-contained proof of Proposition 1 as it gives a more direct interpretation of (3).

Proof (i) Suppose that $\hat{x} \in P^{(\pi,\pi_0)}$. Since $\pi^\top \hat{x} - \pi_0 \in]0, 1[$, there exists $x^0 \in P \cap \left\{\pi^\top x \leq \pi_0\right\}$, $x^1 \in P \cap \left\{\pi^\top x \geq \pi_0 + 1\right\}$ and $\lambda \in]0, 1[$ such that $\hat{x} = \lambda x^1 + (1 - \lambda)x^0$. Without loss of generality, we can assume that $\pi^\top x^0 = \pi_0$ and $\pi^\top x^1 = \pi_0 + 1$. Then, since $\pi^\top(\lambda x^1 + (1 - \lambda)x^0) = \pi^\top \hat{x}$, we have $\lambda = \pi^\top \hat{x} - \pi_0$. We take $\overline{y} = \lambda x^1$ and verify that it satisfies (3). First, $\pi^\top \overline{y} - (\pi_0 + 1)(\pi^\top \hat{x} - \pi_0) = \lambda(\pi_0 + 1) - (\pi_0 + 1)\lambda = 0$. Also, $A\overline{y} - b(\pi^\top \hat{x} - \pi_0) =$

$(Ax^1 - b)\lambda = 0$ and $\overline{y} = \lambda x^1 \geq 0$, since $Ax^1 = b$, $x^1 \geq 0$ and $\lambda > 0$. Finally, since $\hat{x} = \overline{y} + (1 - \lambda)x^0$, $x^0 \geq 0$ and $\lambda < 1$ we have $\overline{y} \leq \hat{x}$.

(ii) Suppose now that $\overline{y}$ satisfies (3). Let $\lambda = \pi^\top \hat{x} - \pi_0$. Note that $0 < \lambda < 1$ by hypothesis. We take $x^1 = \frac{\overline{y}}{\lambda}$ and $x^0 = \frac{\hat{x}-\overline{y}}{1-\lambda}$. Clearly $\hat{x} = \lambda x^1 + (1 - \lambda)x^0$. Furthermore, $x^1 \in P$ and $\pi^\top x^1 \geq \pi_0 + 1$.
It remains to show that $x^0 \in P \cap \{\pi^\top x \leq \pi_0\}$. $x^0 \in P$, since $(1-\lambda)(Ax^0 - b) = A(\hat{x} - \overline{y}) - b + b\lambda = A\hat{x} - b - A\overline{y} + b\lambda = 0$ and $(1 - \lambda)x^0 = \hat{x} - \overline{y} \geq 0$. Finally

$$\begin{aligned}\pi^\top x^0 - \pi_0 &= \frac{\pi^\top \hat{x} - \pi^\top \overline{y} - (1 - \lambda)\pi_0}{1 - \lambda} \\ &= \frac{-\pi^\top \overline{y} + \lambda + \lambda\pi_0}{1 - \lambda} = -\frac{\pi^\top \overline{y} - (\pi_0 + 1)\lambda}{1 - \lambda} \leq 0.\end{aligned}$$

□

Using Proposition 1, we formulate a linear program which answers the question of membership for $\hat{x}$ and $P^{(\pi,\pi_0)}$:

$$\begin{aligned}&\max\ \pi^\top y - (\pi_0 + 1)\left(\pi^\top \hat{x} - \pi_0\right) \\ &\text{s.t.:} \\ &Ay = b\left(\pi^\top \hat{x} - \pi_0\right), \\ &0 \leq y \leq \hat{x}.\end{aligned} \tag{MLP}$$

By Proposition 1, $\hat{x} \in P^{(\pi,\pi_0)}$ if and only if the optimal solution to (MLP) is non-negative.

An alternative way to prove Proposition 1 is to see that (MLP) is closely related to the dual of (CGLP(π, π_0)) with the normalization condition $u_0 + v_0 = 1$:

$$\begin{aligned}&\min \alpha^\top \hat{x} - \beta \\ &\text{s.t.:} \\ &\alpha = A^\top u + s - u_0\pi, \\ &\alpha = A^\top v + t + v_0\pi, \\ &\beta = b^\top u - \pi_0 u_0, \\ &\beta = b^\top v + (\pi_0 + 1)v_0, \\ &u_0 + v_0 = 1, \\ &\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}, u, v \in \mathbb{R}^m, s, t \in \mathbb{R}^n_+, u_0, v_0 \in \mathbb{R}_+\end{aligned} \tag{CGLP'(π, π_0))}$$

A basic result stated in [11] (Lemma 1) is that in any solution of (CGLP(π, π_0)) that yields an inequality $\alpha^\top x \geq \beta$ not dominated by the constraints of (LP), both u_0 and v_0 are positive. The following lemma shows that in fact, if we assume $\hat{x} \in P$ and $\pi_0 < \pi^\top \hat{x} < \pi_0 + 1$, the conditions $u_0, v_0 \geq 0$ can be dropped from (CGLP(π, π_0)).

Lemma 1 *Let $\hat{x} \in P$ be such that $\pi_0 < \pi^\top \hat{x} < \pi_0+1$. Suppose that $(\alpha, \beta, u, v, u_0, v_0)$ satisfies the following system of equations:*

$$\begin{aligned}
&\alpha = A^\top u + s - u_0\pi,\\
&\alpha = A^\top v + t + v_0\pi,\\
&\beta = b^\top u - \pi_0 u_0,\\
&\beta = b^\top v + (\pi_0 + 1)v_0,\\
&\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}, u, v \in \mathbb{R}^m, s, t \in \mathbb{R}^n_+, u_0, v_0 \in \mathbb{R}
\end{aligned}$$

with $u_0 \leq 0$ or $v_0 \leq 0$. Then $\alpha^\top \hat{x} \geq \beta$.

Proof The proof is similar for $u_0 \leq 0$ and $v_0 \leq 0$; we only consider the case $u_0 \leq 0$.

Let $(\alpha, \beta, u, v, u_0, v_0)$ be a solution that satisfies the conditions of the statement. Then, by definition of $(\alpha, \beta, u, v, u_0, v_0)$

$$\begin{aligned}
\alpha^\top \hat{x} - \beta = (A^\top u + s - u_0\pi)^\top \hat{x} - b^\top u + \pi_0 u_0 = u^\top (A\hat{x} - b)\\
+ s^\top \hat{x} - u_0(\pi^\top \hat{x} - \pi_0).
\end{aligned}$$

This last quantity is non-negative since $A\hat{x} = b$, $\hat{x} \geq 0$, $s \geq 0$, $\pi^\top \hat{x} > \pi_0$ and $u_0 \leq 0$. □

Since we only exploit solutions (CGLP'(π, π_0)) with negative objective values, by application of the lemma, we can remove the conditions $u_0, v_0 \geq 0$ in (CGLP'(π, π_0)).

After having removed the non-negativity constraints on u_0 and v_0, we can eliminate α, β, u_0, and v_0 from (CGLP'(π, π_0)) to obtain

$$\begin{aligned}
&\min u^\top (A\hat{x} - b) + s^\top \hat{x} + \left(\pi^\top \hat{x} - \pi_0\right)\left(b^\top (u - v) - \pi_0 - 1\right),\\
&\text{s.t.:}\\
&A^\top (u - v) + (s - t) = \pi\\
&\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}, u, v \in \mathbb{R}^m, s, t \in \mathbb{R}^n_+.
\end{aligned} \tag{4}$$

Finally, we eliminate the term $u^\top (A\hat{x} - b) = 0$ from the objective of (4) and replace $(u - v)$ by a single vector of variables λ:

$$\begin{aligned}
&\min s^\top \hat{x} + \left(\pi^\top \hat{x} - \pi_0\right)\left(b^\top \lambda - \pi_0 - 1\right),\\
&\text{s.t.:}\\
&A^\top \lambda + (s - t) = \pi\\
&\lambda \in \mathbb{R}^m, s, t \in \mathbb{R}^n_+.
\end{aligned} \tag{5}$$

Taking the dual of (5) with multipliers y associated to the constraints, one obtains (MLP). It should be clear from this development that a dual feasible solution to (MLP) with negative dual cost gives rise to a valid cut for $P^{(\pi,\pi_0)}$.

Remark 1 Note that one should be careful when dropping the non-negativity conditions on u_0 and v_0 in (CGLP'(π, π_0)). While solutions to (CGLP'(π, π_0)) of non-negative cost define valid (non-violated) inequalities, solution to (5) of non-negative cost do not. For example, the trivial solution $\lambda = 0, s - t = \pi$ is always feasible for (5) but the inequality it defines is not necessarily valid. For example take $\pi = e_k$ and $\pi_0 = 1$, the solution $\lambda = 0, t = 0, s = e_k$ defines the inequality $x_k \leq 2$ which certainly cannot be valid for all integer programs! Nevertheless, if the solution has a negative objective value, it yields a valid cut.

(CGLP'(π, π_0)) (with that normalization condition) has been studied previously in [7,10,28]. A well known property of the normalization condition $u_0 + v_0 = 1$ is that if $\hat{x}$ is an extreme point of P such that $\hat{x}_k \notin \mathbb{Z}$ then an optimal solution to (CGLP'$(e_k, \lfloor \hat{x}_k \rfloor)$) is the intersection cut obtained from the row of the tableau in which k is basic in any basis defining $\hat{x}$ and the corresponding strengthened cut is the GMI cut from the same row and same basis (see [28] for a precise statement and a complete proof). In the next proposition, we re-state the property in the context of (MLP). Looking at (MLP), we are able to give a slightly more precise view: if $\hat{x}$ is an extreme point, (MLP) has a unique solution. We present a proof of the result in the context of (MLP) since it is short and gives insights into the connections between (LP) and (MLP) (the fact that the associated cut is the intersection cut will follow directly from the results of the next section).

Proposition 2 *Let $P = \{x \in \mathbb{R}^n_+ : Ax = b\}$, and $\hat{x}$ be an extreme point of P such that $0 < \pi^\top \hat{x} - \pi_0 < 1$, then the unique solution to*

$$\begin{aligned} &\max \ \pi^\top y - (\pi_0 + 1)(\pi^\top \hat{x} - \pi_0) \\ &\textit{s.t.:} \\ &Ay = b(\pi^\top \hat{x} - \pi_0) \\ &0 \leq y \leq \hat{x} \end{aligned} \tag{6}$$

is $\overline{y} = \hat{x}(\pi^\top \hat{x} - \pi_0)$. Furthermore $\pi^\top \overline{y} - (\pi_0 + 1)(\pi^\top \hat{x} - \pi_0) < 0$.

Proof First, we prove that $\overline{y}$ is a solution. Since $0 < \pi^\top \hat{x} - \pi_0 < 1$ and $A\hat{x} - b = 0$, $(\pi^\top \hat{x} - \pi_0)(A\hat{x} - b) = 0$, and $0 \leq (\pi^\top \hat{x} - \pi_0)\hat{x} \leq \hat{x}$.

Second, we show that this solution is unique. We denote by $x_J = 0$, the subset of the inequalities defining P satisfied at equality by $\hat{x}$. Since $\hat{x}$ is an extreme point of P, it is the unique solution of $Ax = b$, $x_J = 0$. A solution of (6) satisfies:

$$\begin{aligned} Ay &= b(\pi^\top \hat{x} - \pi_0) \\ y_J &= 0, \end{aligned}$$

which in turn has a unique solution.

Finally, it is trivial to check that $\pi^\top \overline{y} - (\pi_0 + 1)(\pi^\top \hat{x} - \pi_0) = (\pi^\top \hat{x} - \pi_0 - 1)(\pi^\top \hat{x} - \pi_0) < 0$. □

Proposition 2 has important consequences for the practical use of (MLP). In particular, it exposes its main weakness with respect to more classical lift-and-project cut generation LPs such as the one used in [11].

The standard way to apply lift-and-project cuts (and GMI cuts), is to generate them recursively by rounds where at each round the basic optimal solution to the LP relaxation (computed by the simplex algorithm) is cut by separating one lift-and-project cut for each basic integer constrained variable with a fractional value in the optimal solution. At the end of the round all the lift-and-project cuts generated are added to the LP relaxation and the process is iterated recursively. In such an algorithm, using (MLP) to generate cuts brings little interest since the point to cut is always an extreme point of the polyhedron used to separate and the cuts generated would therefore be GMI cuts.

Nevertheless, there are contexts where solving (MLP) could present a practical interest. In particular one cannot always assume that $\hat{x}$ is an extreme point of P. This is the case, for example, if the objective function in (MILP) is replaced with a nonlinear convex function, or if the LP relaxation is solved with an interior point method. In this work, we focus on another case of interest: generating violated rank 1 cuts. Indeed, if in the procedure outlined above, one does not use the previously generated cuts to generate new cuts, then the point to cut is not an extreme point of the polyhedron used to separate.

4 Dual feasible bases of the membership LP

Our goal in this section is to study properties of the basic solutions of (MLP). In particular, we characterize the dual feasible bases that give rise to cuts. This allows us to give a proof of the equivalence between the cuts obtained from solving (MLP) and intersection cuts. This last result is a special case of the generalization of Theorems 4A, 4B in [11] made in [6]. The interest of the proof here is that it is shorter, and gives a self-contained understanding of (MLP). It also gives a more complete understanding of the solutions (MLP) than in [11], in particular of those that do not correspond to cuts

From now on, we only consider elementary disjunctions of the form $x_k \leq \pi_0 \vee x_k \geq \pi_0 + 1$. Note that this assumption is not restrictive since any split $\pi^\top x \leq \pi_0 \vee \pi^\top x \geq \pi_0 + 1$ can always be put into this form by adding an extra variable equal to $\pi^\top x$. For this elementary disjunction, the membership LP is

$$\begin{aligned} &\max \ y_k - \lceil \hat{x}_k \rceil (\hat{x}_k - \lfloor \hat{x}_k \rfloor), \\ &\text{s.t.:} \\ &Ay = b(\hat{x}_k - \lfloor \hat{x}_k \rfloor), \qquad (\text{MLP}_k) \\ &0 \leq y \leq \hat{x}, \\ &y \in \mathbb{R}^n. \end{aligned}$$

For ease of notation, in the sequel, we denote $(\hat{x}_k - \lfloor \hat{x}_k \rfloor)$ by f_k.

The dual of (MLP$_k$) is the Cut Generation Linear Program

$$\begin{aligned} &\min s^\top \hat{x} + b^\top \lambda f_k - \lceil \hat{x}_k \rceil f_k \\ &\text{s.t.:} \\ &A^\top \lambda + (s - t) = e_k \\ &\lambda \in \mathbb{R}^m,\ s,\ t \in \mathbb{R}^n_+. \end{aligned} \tag{CGLP'$_k$}$$

A solution (λ, s, t) to (CGLP'$_k$) defines a cut of the form:

$$s^\top x + \left(b^\top \lambda - \lceil \hat{x}_k \rceil\right) x_k \geq \left(b^\top \lambda - \lceil \hat{x}_k \rceil\right) \lfloor \hat{x}_k \rfloor. \tag{7}$$

Note that (MLP$_k$) is in the form of a general LP with lower and upper bounds on the variables. In a basic solution, we denote by J^+ the index set of nonbasic variables at their upper bound (corresponding to variables s in the dual) and by J^- index the set of nonbasic variables at their lower bound (corresponding to variables t). We remind the reader that a basic solution of (MLP$_k$) is dual feasible if the reduced costs of every variable indexed by J^- is non-positive and the reduced cost of every variable indexed by J^+ is non-negative.

The next Lemma gives a characterization of dual feasible bases of (MLP$_k$).

Lemma 2 *Let* $B = \{i_1, \ldots, i_m\} \subseteq N$.

a. *The variables indexed by B form a basis of* (MLP$_k$) *if and only if they form a basis of* (LP).
b. *Suppose that B indexes a basis. This basis is dual feasible for* (MLP$_k$) *if either:*
   (i) *the variable* y_k *is non-basic at its upper bound, or*
   (ii) y_k *is basic with corresponding tableau row* $y_k + \sum_{j \in J} \overline{a}_{ij} y_j = \overline{a}_{i0}$ *in* (MLP$_k$) *and* $j \in J$ *is non-basic at its upper bound if* $\overline{a}_{ij} < 0$ *and nonbasic at its lower bound if* $\overline{a}_{ij} > 0$.

Proof a. Since (MLP$_k$) and (LP) have the same matrix of constraints this is true by definition of a basis.

b. Let J^+ be the index set of nonbasic variables at their upper bound and J^- the index set of nonbasic variables at their lower bound.
   (i) If $k \notin B$, all variables of (MLP$_k$) have a reduced cost of 0 except y_k which has a reduced cost of 1. Therefore y_k has to be at its upper bound for the solution to be dual feasible.
   (ii) Now suppose that $k \in B$, and y_k is basic in row i. The reduced cost of a variable x_j, $j \in J$, is given by $0 - e_i^\top A^{B^{-1}} A^j = 0 - \overline{a}_{ij}$. Therefore, the basis is dual feasible if for all $j \in J$ such that $\overline{a}_{ij} > 0$, $j \in J^-$ and for all $j \in J$ such that $\overline{a}_{ij} < 0$, $j \in J^+$. □

Using the reduced cost computed in Lemma 2, we can obtain the values for the dual variables and compute the cut (7) (keeping in mind that by Lemma 1 and Remark 1, only solutions of negative dual cost give rise to valid inequalities). In the next lemma, we give a more precise description of dual feasible bases of (MLP$_k$) with negative dual cost.

Lemma 3 *Let B be the index set of a dual feasible basis of* (MLP$_k$). *Let $\overline{x} \in \mathbb{R}^n$ be the basic solution of* (LP) *corresponding to the basis indexed by B. If the dual basic solution of* (MLP$_k$) *associated to B has negative dual objective value then $k \in B$ and $\overline{x}_k \in \left]\lfloor \hat{x}_k \rfloor, \lceil \hat{x}_k \rceil\right[$.*

Proof First, we uppose that $k \notin B$. Then we are in case *(i)* of Lemma 2: all dual variables have value 0 except s_k which has value 1. In that case, the dual objective value of the solution is:

$$\hat{x}_k - \lceil \hat{x}_k \rceil (\hat{x}_k - \lfloor \hat{x}_k \rfloor) = \lfloor \hat{x}_k \rfloor (\lceil \hat{x}_k \rceil - \hat{x}_k),$$

and is non-negative (this solution is the one exhibited in Remark 1 and does not lead to a valid inequality).

Now, we suppose that $k \in B$. Then we are in case (ii) of Lemma 2. The values of the dual variables s and t are given by the reduced costs computed in the proof of Lemma 2:

$$s_j = \begin{cases} -\overline{a}_{ij} & \text{if } j \in J^+ \cap N, \\ 0 & \text{if } j \in N \backslash J^+; \end{cases} \tag{8}$$

$$t_j = \begin{cases} \overline{a}_{ij} & \text{if } j \in J^- \cap N, \\ 0 & \text{if } j \in N \backslash J^-, \end{cases} \tag{9}$$

(with $J^+ = \{j \in J : \overline{a}_{ij} < 0\}$ and $J^- = \{j \in J : \overline{a}_{ij} > 0\}$). The values of λ are deduced from (8–9) to be $\lambda^\top = e_i (A^B)^{-1}$ (i.e. $\lambda^\top$ is row i of the basis inverse).

By Lemma 1 the dual solution has a negative dual cost only if the quantities u_0 and v_0 are positive. u_0 is given by $u_0 = -\lambda^\top b + \lceil \hat{x}_k \rceil$. Note that $-\lambda^\top b = -\overline{b}_i = -\overline{x}_k$. Therefore $u_0 > 0$ only if $\overline{x}_k < \lceil \hat{x}_k \rceil$. Similarly since $v_0 = 1 - u_0$, $v_0 > 0$ only if $\overline{x}_k > \lfloor \hat{x}_k \rfloor$. □

In the next theorem, we relate dual feasible bases of (MLP$_k$) with negative dual cost to intersection cuts of (LP).

Theorem 2 *Let B be the index set of a dual feasible basis of* (MLP$_k$) *with negative dual cost. The cut obtained from the dual solution of* (MLP$_k$) *corresponding to the basis indexed by B is the simple intersection cut for x_k of* (LP) *obtained from the basis indexed by B.*

Proof Let $\overline{x} \in \mathbb{R}^n$ be the primal basic solution of (LP) corresponding to the basis indexed by B.

Since B indexes a dual feasible basis of (MLP$_k$) with negative dual cost, by Lemma 3, $k \in B$ and $\overline{x}_k \in \left]\lfloor \hat{x}_k \rfloor, \lceil \hat{x}_k \rceil\right[$. Therefore in the tableau of (LP) corresponding to B, the intersection cut (2) for variable x_k is well defined.

Now, we consider the cut (7) defined by the dual basic solution of (MLP$_k$) indexed by B:

$$s^\top x + \left(\lambda^\top b - \lceil \hat{x}_k \rceil\right) x_k \geq \left(\lambda^\top b - \lceil \hat{x}_k \rceil\right) \lfloor \hat{x}_k \rfloor. \tag{10}$$

Using the values of s given by (8) and the equality $\lambda^\top b = \overline{x}_k$, (10) can be rewritten as

$$-\sum_{j\in J^+} \overline{a}_{ij}x_j + \left(\overline{x}_k - \lceil \hat{x}_k \rceil\right) x_k \geq \left(\overline{x}_k - \lceil \hat{x}_k \rceil\right) \lfloor \hat{x}_k \rfloor. \tag{11}$$

Now using the tableau row of (LP) $x_k + \sum_{j\in J} \overline{a}_{ij}x_j = \overline{x}_k$, we eliminate x_k from (11):

$$-\sum_{j\in J^+} \overline{a}_{ij}x_j + \left(\overline{x}_k - \lceil \hat{x}_k \rceil\right) \left(\overline{x}_k - \sum_{j\in J} \overline{a}_{ij}x_j\right) \geq \left(\overline{x}_k - \lceil \hat{x}_k \rceil\right) \lfloor \hat{x}_k \rfloor. \tag{12}$$

Finally, re-grouping the coefficients in (12), we obtain:

$$-(\overline{x}_k - \lfloor \hat{x}_k \rfloor) \sum_{j\in J^+} \overline{a}_{ij}x_j + (\lceil \hat{x}_k \rceil - \overline{x}_k) \sum_{j\in J^-} \overline{a}_{ij}x_j \geq (\lfloor \hat{x}_k \rfloor - \overline{x}_k)(\overline{x}_k - \lceil \hat{x}_k \rceil). \tag{13}$$

Since $\overline{x}_k \in \left]\lfloor \hat{x}_k \rfloor, \lceil \hat{x}_k \rceil\right[$, we have $\overline{x}_k - \lfloor \hat{x}_k \rfloor = \overline{x}_k - \lfloor \overline{x}_k \rfloor$. Defining $f_0 := \overline{x}_k - \lfloor \overline{x}_k \rfloor$ as in (2), we have that (13) is:

$$-f_0 \sum_{j\in J^+} \overline{a}_{ij}x_j + (1 - f_0) \sum_{j\in J^-} \overline{a}_{ij}x_j \geq (1 - f_0) f_0. \tag{14}$$

This last cut is identical to the intersection cut (2). Indeed, if $j \in J^+, \overline{a}_{ij} \leq 0$ and $\max\{-f_0\overline{a}_{ij}, (1 - f_0)\overline{a}_{ij}\} = -f_0\overline{a}_{ij}$; if $j \in J^-, \overline{a}_{ij} \geq 0$ and $\max\{-f_0\overline{a}_{ij}, (1 - f_0)\overline{a}_{ij}\} = (1 - f_0)\overline{a}_{ij}$. This ends the proof of the Theorem. □

Theorem 2 can be seen as the counterpart of Theorems 4A, 4B in [11]. Indeed, the statements are almost equivalent except that the normalization condition $u_0+v_0 = 1$ is used here whereas the normalization condition $\sum(u_i+v_i)+\sum(s_i+t_i)+u_0+v_0 = 1$ was used in [11]. In [6], the theorems of [11] were generalized to weighted normalizations of the form $\sum \psi_i(u_i+v_i)+\sum \xi_i(s_i+t_i)+u_0+v_0 = \psi_0$ where $\psi \in \mathbb{R}^m_+$, $\xi \in R^n_+$ and $\psi_0 \in \mathbb{Z}_+$. Theorem 2 is a particular case of these theorems. In [11], an equivalence is also established between the strengthened lift-and-project cuts and GMI cuts. Of course this correspondence carries out to (MLP$_k$) when the cut (14) is strengthened.

We note that Lemmas 2 and 3 give a slightly better understanding of the characterizations in [6,11] by describing dual feasible bases of non-negative dual cost.

Another important result in [11] is an algorithm to solve (CGLP) by pivoting in the tableau of (LP). Our results shows that, with the normalization $u_0 + v_0 = 1$, a similar algorithm is to solve (MLP$_k$) by the simplex method. A limitation of the algorithm in [11] is that it starts from a basis of (LP) that gives a cut (i.e. a basic feasible solution of (CGLP'(π, π_0)) of negative cost). If such a basis is not available, the solution of (MLP$_k$) can be seen as a way to find one (or certify that there is none).

5 Optimizing over P_e and approximating P_{e^*}

We now turn back to our initial goal which is the optimization over P_e and P_{e^*}. We follow a simple Kelley cutting plane algorithm [34] where the master problem is the LP relaxation of (MILP) augmented with cuts and the cut generation procedure is the solution of (MLP$_k$). More precisely, at each iteration, we first solve the master problem and obtain a solution $\hat{x}$. If $\hat{x}$ is integer feasible, we stop. Otherwise, we solve the separation problem (MLP$_k$) for a number of fractional components of $\hat{x}$ which should be integral. If some of the separation problems lead to a cut, we add those cuts to the master problem and iterate. Otherwise, if all (MLP$_k$) for all $k \in \{1, \ldots, p\}$ such that $\hat{x}_k \notin \mathbb{Z}$ have non-negative objective value, we have a proof that $\hat{x} \in P_e$. To approximate the optimal value over P_{e^*}, we just strengthen every cut individually by the usual strengthening procedure. Note that cuts are never added to (MLP$_k$) in the course of the procedure (i.e. we stick to generating rank 1 cuts). This simple algorithm should terminate in a finite number of iterations (since there is only a finite number of cuts of rank 1), but there is no complexity guarantee. Also, there is no guarantee of how good the approximation of the value over P_{e^*} is. Nevertheless, it is guaranteed that strengthening each cut in this fashion gives a better bound than optimizing over P_e. Since the extra computational cost is negligible, it should always be better.

Note that this approach is similar to the one presented in [18], although it is presented there from a different point of view. In [18], a Benders decomposition is applied to a linear programming model of P_e defined in a higher dimensional space. Applying this decomposition amounts to performing the cut generation method outlined above. The main difference between the two approaches is the separation problem that is solved. In [18], a (CGLP) is solved, while here we solve (MLP$_k$) (other differences are the details of our algorithm that are described below).

In order to make this algorithm somewhat efficient, one drawback has to be taken care of. In the course of the algorithm, it often happens that although $\hat{x}_k$ is fractional the solution of (MLP$_k$) does not lead to a cut. If at a given iteration, this happens for all $k \in \{1, \ldots, p\}$, $\hat{x} \in P_e$ and the algorithm stops but if it happens for some k's it does not help the algorithm in progressing. After the first iteration, this phenomenon usually happens very frequently, and a significant part of the computation can be spent in solving those separation problems that do not lead to cuts. To try to overcome it, at each iteration we solve (MLP$_k$) only for those k which led to a cut in the previous iteration. To maintain the validity of the algorithm, in the case when no cuts were generated, we need to test all the variables (we also test all the variables in the case where some tailing off is detected). This strategy is obviously not perfect as still some problems (MLP$_k$) will not lead to cuts but it improves the efficiency of the procedure (more comments on the practical performance will be made in Sect. 6.1).

A second aspect where we try to make our algorithm more efficient is in trying to make the solutions time of the problem (MLP$_k$) small. At each iteration of the algorithm in Step 4, we solve a sequence of such problem for a number of the integer variables $x_{k_1}, \ldots, x_{k_{p'}}$. Each problem is solved by the simplex algorithm. Note that from one problem to the next, in general, all bounds on variables and constraints are changed as well as the objective function. Therefore, the optimal basis of (MLP'$_{k_i}$) is in general neither primal nor dual feasible for (MLP'$_{k_{i+1}}$). There is one simple

exception to this, if $\hat{x}_{k_i} = \hat{x}_{k_{i+1}}$, both problems have the same constraint system and therefore the solution of (MLP'$_{k_i}$) is primal feasible for (MLP'$_{k_{i+1}}$). We try to benefit from this by solving the problems (MLP$_k$) in increasing order of the value $\hat{x}_k$ (also hoping that if the value $\hat{x}_{k_i}$ is close to the one of $\hat{x}_{k_{i+1}}$, the solution of (MLP'$_{k_i}$) will be close to being primal feasible for (MLP'$_{k_{i+1}}$)). Again, we do not claim that our strategies are the best possible but just try to follow some sound heuristics rules. The computation in the next section will illustrate the practical performance.

The pseudo-code of the algorithm we use to approximately optimize over P_{e^*} is given in Algorithm 1. For optimizing over P_e, we just skip the strengthening in Step 4 of the algorithm.

Algorithm 1 Cutting plane algorithm for P_{e^*}

0. **Initialize feasible region.**
   $\mathcal{C} \leftarrow \{x \in R^n_+ : Ax = b\}$.
1. **Initialize variables list**
   $\mathcal{K} \leftarrow \{1, \ldots, p\}$, `reinit` $\leftarrow$ *true*.
2. **Master Problem**
   Solve $\max\{{c'}^\top x : x \in \mathcal{C}\}$. Let $\hat{x}$ be the solution.
3. **Filter variables**
   $\mathcal{F} \leftarrow \{i \in \mathcal{K} : \hat{x}_i - \lfloor \hat{x}_i \rfloor \geq \epsilon\}$, $\mathcal{K} \leftarrow \emptyset$. Sort $\mathcal{F}$ by increasing values of $\hat{x}_k$.
4. **Separate**
   For each $k \in \mathcal{F}$ solve (MLP$_k$). If the solution has objective value $\leq -\epsilon$, construct strengthened cut $\alpha^T x \geq \beta$ from the dual solution, and let $\mathcal{K} \leftarrow \mathcal{K} \cup \{k\}$, $\mathcal{C} \leftarrow \mathcal{C} \cap \{x : \alpha^\top x \geq \beta\}$.
5. **Termination**
   If $\mathcal{K} = \emptyset$ and `reinit` $=$ *true*, then **STOP**.
6. **Loop**
   If $\mathcal{K} = \emptyset$ or tailing off is detected, then **go to 1**, else `reinit` $\leftarrow$ *false*, **go to 2.**

6 Computational results

We now present computational experiments aimed at assessing the effectiveness of our procedures for optimizing over P_e and approximately optimizating over P_{e^*}. In the remainder, by abuse of terminology, we simply call the two procedures for computing these two values using Algorithm 1 P_e and P_{e^*}. The computations are performed with an implementation of Algorithm 1 in C++. The code allows the use of two different LP solvers: Clp [31] from COIN-OR [35] or IBM ILOG CPLEX. For efficiency reasons, the code calls each solver directly by using their respective native C++ and C API. All experiments are conducted on a machine equipped with Intel Quad Core Xeon 2.93GHz processors and 120 GB of RAM, using only one thread for each run. We use Clp version 1.14.2 and Cplex version 12.1. The tolerance ϵ in Algorithm 1 is set to 10^{-4} and a time limit of 1 h is imposed.

Since the number of cuts generated can sometime grow very large, we use a cut pool to limit the size of the master problem. Our implementation of a cut pool is straightforward. The pool contains all the cuts that are not currently active in the master problem. When new cuts have been generated and the master problem is reoptimized, all

violated cuts of the pool are added to the master problem. This is iterated until all the cuts in the pool are satisfied by the optimal solution of the master. The master problem is then cleaned of all the non-active cuts that are put back in the pool. Cuts are never purged from the pool.

Our main goal in these experiment is to assess how strong the bound obtained by P_{e^*} can be and how quickly it is computed. First, we present computations of P_e and P_{e^*}. Next, we present several comparisons with different methods:

– a comparison with textbook GMI cuts and lift-and-project cuts generated by rounds;
– a comparison with several methods recently proposed to generate rank 1 split cuts.

Finally, we present some preliminary experiments on using P_{e^*} in a branch-and-cut framework. All experiments are performed on MIPLIB 3.0 [14] and MIPLIB 2003 [36] benchmark instances.

Computations of P_e and P_{e^*} were already conducted in [18]. Before proceeding to the results, we stress the main differences with the approach proposed here and the interesting features of the new experiments. The approach in [18] used a cut generation LP formulated in a higher dimensional space. Although it is possible that each cut separated was deeper there, the size of the cut generation LPs and their solution time limited the applicability of the approach to small and medium size problems. For that reason the computational experiments in [18] were limited to problems with up to 1000 variables. We did not carry out systematic comparison between the two approaches but let us note that on the test set used in [18] our code is about 12 times faster than the results reported there (although our tolerances are likely to be stricter). This is certainly in part due to faster machines and faster LP codes but we believe also to the use of a better separation LP. One main interest of the experiments here is that thanks to these faster computing time we are able to report numbers on the complete MIPLIB 3 and MIPLIB 2003 problems.

6.1 Computation of P_e and P_{e^*}

We ran our code to compute P_e and P_{e^*} on 62 MIPLIB 3.0 instances[1] and 20 MIPLIB 2003 instances that are not already in MIPLIB 3.0. We ran each instance with four variants: computation of P_e and P_{e^*} both with Clp and CPLEX as LP solver. The code is run on the original instance without any preprocessing. The results of these experiments are summarized in Tables 1 and 2. For each test problem, we report the CPU time and the fraction of the integrality gap closed by the method (two instances, `dsbmip` and `enigma` do not have integrality gap and we report no_gap for them).

Several remarks can be made from the results. There are only four instances (`stein27, stein45, noswot` and `rd-rplusc-21`) on which no gap is closed using P_{e^*}. On average, on MIPLIB 3.0 (not taking into account the two instances without integrality gap), P_e closes 47.42% of the integrality gap with Clp and 50.64% with

[1] We do not report computations on `markshare1`, `markshare2` and `pk1`, because our code ran into computational troubles on these two instances when using CPLEX as LP solver. Note however that no gap is closed neither by P_e nor by P_{e^*} on these three instances.

Table 1 CPU time and gap closed by P_e and P_{e^*} on MIPLIB 3.0

Name	P_e				P_{e^*}			
	Clp		CPLEX		Clp		CPLEX	
	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
10teams	3,600	15.10	3,600	30.41	3,600	100	3,600	100
air03	0.88	100	0.21	100	0.92	100	0.18	100
air04	3,600	44.46	3,600	87.38	3,600	70.20	3,600	95.31
air05	3,600	54.51	3,600	65.81	3,600	61.24	3,600	68.27
arki001	6.62	20.34	6.06	20.34	4.51	35.59	8.58	36.44
bell3a	0.02	64.56	0.01	64.56	0.01	64.56	0.01	64.56
bell5	0.06	86.25	0.03	86.25	0.04	86.26	0.03	86.55
blend2	0.12	21.82	0.05	21.82	0.11	22.12	0.06	22.54
cap6000	1.35	50	0.35	50	1.64	62.50	0.24	56.25
dano3mip	3,600	0.28	3,600	0.52	3,600	0.23	3,600	0.52
danoint	3,600	5.54	167.88	5.53	2,576.11	5.62	151.12	5.61
dcmulti	1.79	98.15	1.08	98.15	1.50	98.17	0.89	98.19
dsbmip	8.61	no_gap	0.47	no_gap	3.60	no_gap	0.54	no_gap
egout	0.04	93.85	0.02	93.85	0.04	93.85	0.02	93.85
enigma	0.71	no_gap	0.52	no_gap	2.87	no_gap	12.35	no_gap
fast0507	3,600	1.79	3,600	9.95	3,600	1.79	3,600	11.74
fiber	4.45	20.63	2.06	20.63	0.72	92.97	0.72	97.35
fixnet6	25.13	86.37	9.71	86.36	25.14	86.56	9.19	86.72
flugpl	0.00	11.72	0.00	11.72	0.00	11.72	0.00	11.72
gen	2.57	70.49	1.27	70.49	0.36	81.97	0.17	93.99
gesa2	1.35	59.10	0.64	59.10	1.08	66.06	0.52	67.38
gesa2_o	1.25	59.80	0.88	59.80	1.31	65.20	0.70	72.66
gesa3	7.18	79.89	3.65	79.89	3.18	94.40	1.05	94.59
gesa3_o	6.92	82.88	3.61	82.88	3.11	94.40	1.23	94.40
gt2	0.03	92.38	0.01	92.38	0.04	98.34	0.02	98.58
harp2	8.72	21.28	2.88	21.28	7.06	47.41	2.18	44.19
khb05250	0.28	99.86	0.16	99.86	0.33	99.95	0.18	99.95
l152lav	102.66	34.46	13.02	34.46	228.01	63.45	19.82	65.33
lseu	0.03	16.58	0.02	16.58	0.02	77.45	0.01	69.59
mas74	0.12	5.47	0.04	5.47	0.05	7.95	0.03	8.06
mas76	0.16	3.68	0.06	3.68	0.06	7.32	0.02	7.32
misc03	1.59	40.21	0.50	40.21	0.96	40.21	0.49	40.21
misc06	0.50	86.21	0.16	86.21	0.37	87.36	0.13	87.36
misc07	13.61	11.44	4.68	11.44	16.90	13.08	4.14	13.13
mitre	303.96	100	68.30	100	4.68	100	3.44	100
mkc	3,600	60.66	12.24	63.18	3600	63.53	52.71	63.69
mod008	0.06	9.02	0.03	9.02	0.04	37.36	0.02	37.83
mod010	43.13	52.51	7.04	52.51	2.29	100	0.07	100

Table 1 continued

Name	P_e				P_{e^*}			
	Clp		CPLEX		Clp		CPLEX	
	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
mod011	3,600	47.80	3,600	55.39	3,600	47.09	3,600	56.13
modglob	0.50	57.09	0.27	57.09	0.53	57.09	0.31	61.65
noswot	15.06	0	0.06	0	0.57	0	0.06	0
nw04	3,600	36.57	3,600	38.85	605.36	100	102.12	100
p0033	0.01	8.19	0.01	8.19	0.01	57.76	0.01	76.40
p0201	0.50	46.85	0.37	46.85	1.08	70.45	0.67	73.34
p0282	0.37	93.90	0.18	93.90	0.47	98.61	0.26	99
p0548	2.76	91.34	0.81	91.36	2.03	95.15	0.89	95.35
p2756	3.02	81.69	1.67	81.60	2.54	98.88	1.02	96.31
pp08a	0.26	79.29	0.12	79.29	0.22	79.46	0.12	79.29
pp08aCUTS	0.67	68.81	0.37	68.81	0.56	70.47	0.41	69.09
qiu	3600	76.78	690.82	78.09	3600	77.11	751.43	78.09
qnet1	13.00	94.28	10.71	94.28	28.09	96.09	4.23	94.49
qnet1_o	1.09	87.59	2.59	87.59	1.10	89.57	3.46	90.27
rentacar	3,600	67	3,600	91.35	3,600	61.29	3,600	99.51
rgn	0.19	11.88	0.10	11.88	0.11	73.65	0.03	69.88
rout	20.04	28.03	7.91	28.01	11.62	50.24	1.39	53.18
set1ch	0.52	39.88	0.29	39.88	0.54	39.88	0.30	39.88
seymour	3,600	26.95	2,508.03	55.94	3,600	30.04	2,191.19	57.67
stein27	0.21	0	0.31	0	0.24	0	0.27	0
stein45	10.92	0	10.22	0	24.15	0	11.77	0
swath	20.28	2.77	10.92	2.77	9.45	26.31	4.46	26.31
vpm1	0.17	31.42	0.07	31.42	0.09	35.64	0.08	33.17
vpm2	0.36	54.29	0.17	54.29	0.29	54.42	0.17	54.78

CPLEX. On MIPLIB 2003 it closes is 23.11% and 33.49% with Clp and Cplex respectively (note that the differences are mostly due to problems which go to the time limit). P_{e^*} closes significantly more gap: on MIPLIB 3.0 60.80% and 63.29% with Clp and CPLEX respectively; on MIPLIB 2003, 32.05% and 44.55%. The computations of P_{e^*} highlight the heuristic nature of the procedure, since for many instances the gap closed is significantly different depending on the LP solver used (while those numbers are usually equal or very close for P_e with the two solvers when the time limit is not attained). Finally, computing times vary a lot between instances but are usually close between the four methods. For P_{e^*} with CPLEX (our best method) 14 problems go to the time limit (7 in MIPLIB 3.0 and 7 in MIPLIB 2003). 7 out of these 14 can be solved to optimality in a shorter computing time with CPLEX (`10teams`, `air04`, `air05`, `fast0507`, `rentacar`, `qiu` and `mzzv11`), 6 would take longer than an hour (`atlanta-ip`, `momentum1`, `msc98-ip`, `net12`, `protfold` and `rd-rplus-sc21`) and one is an open problem (`dano3mip`). In spite of these

Table 2 CPU time and gap closed by P_e and P_{e^*} for MIPLIB 2003

Name	P_e				P_{e^*}			
	Clp		CPLEX		Clp		CPLEX	
	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
a1c1s1	613.47	78.76	78.22	78.76	723.86	78.75	75.60	78.76
aflow30a	4.93	42.43	1.48	42.41	4.00	42.95	1.91	43.27
aflow40b	27.14	34.29	7.78	34.29	22.38	35.39	7.69	35.87
atlanta-ip	3,600	0.18	1,531.34	1.09	3,600	0.29	3,600	1.77
glass4	9.37	0	1.00	0	14.01	0.01	1.11	0.13
momentum1	3,600	37.30	3,600	44.88	3,600	37.42	3,600	45.15
momentum2	3,600	22.97	226.78	41.47	3,600	37.21	546.47	41.84
msc98-ip	3,600	6.94	3,600	42.23	3,600	4.43	3,600	44.65
mzzv11	3,600	7.79	3,600	56.47	3,600	9.62	3,187.29	100
mzzv42z	3,600	7.44	2,426.39	87.73	3,600	10.30	140.22	100
net12	3,600	8.98	3,600	22.73	3,600	8.77	3,600	22.71
nsrand-ipx	594.21	36.88	152.32	36.88	65.24	77.43	16.58	77.09
opt1217	1.69	0.19	0.43	0.19	4.77	25.34	0.85	26.32
protfold	3,600	7.51	3,600	10.29	3,600	8.34	3,600	10.83
rd-rplusc-21	3,600	0	3,600	0	3,600	0	3,600	0
roll3000	401.82	16.30	103.69	16.31	318.29	54.25	109.69	55.90
sp97ar	2,324.45	42.06	657.63	42.06	1,838.60	61.08	377.99	59.91
timtab1	0.53	26.99	0.33	26.99	0.61	42.31	0.30	42.45
timtab2	1.65	20.98	1.04	20.98	2.46	43.04	1.01	40.18
tr12-30	5.95	64.12	3.18	64.12	6.04	64.13	3.21	64.12

Table 3 Detailed statistics of P_{e^*} procedure on MIPLIB3 problems

	MIPLIB 3	>1 s CPU	Finished
# Master iterations	55.90	142.67	46.26
Total resolve time	0.48	21.60	0.16
# Cuts	521.20	2,903.31	339.66
# Non-cut	105.43	230.61	134.06
Total (MLP_k) Time	0.52	11.61	0.23
Total (MLP_k) pivots	9,129.19	133,136.41	4,578.62

instances, computing times are typically small with a geometric mean of 3.44 and 1.55 s respectively for P_{e^*} with Clp and CPLEX on MIPLIB 3.0, and 229.51 and 84.78 s respectively on MIPLIB 2003. Note that computing times are often slightly higher for P_e. This is not surprising since our stopping criterion is that the point $\hat{x}$ belongs to P_e and that the cuts generated in P_{e^*} are always deeper.

Table 3 gives geometric means for a number of key statistics of our procedure: the number of master iterations of the procedure (i.e. the number of times step 2 of

Algorithm 1 is performed), the CPU time spent to solve the master problem (i.e. the total time spent in step 2), the number of times the separation problem (MLP_k) led to a cut, the number of times it did not lead to a cut, the CPU time spent to solve separation problems, and finally the number of simplex iterations for solving all the separation problems. Statistics refer to P_{e*} with CPLEX as LP solver. Averages figures are given on three sets of problems: all instances of MIPLIB 3.0, all instances in MIPLIB 3.0 that took more than 1 s of CPU time and all instances solved within the time limit.

The main observation we can make from Table 3 is that the solution of (MLP_k) is typically very quick. In addition to the cumulative numbers presented in the table, the geometric mean of the time for solving one (MLP_k) problem is 5.92×10^{-4} s if we consider the whole MIPLIB 3.0, and 2.79×10^{-3} s if we restrict to instances that took more than 1 s of CPU time. This can be explained by the low number of pivots: 10.33 pivots on average on MIPLIB 3.0, and 32.03 if we restrict to problems that took more that 1 s. This is also illustrated by the fact that in total resolving the LP relaxations after adding cuts takes about the same time as solving (MLP_k)'s. Another interesting figure is the proportion of separation problems that lead to a cut. Here one should only look at the statistics for problems that are finished (other statistics are biased by unfinished problems where the proportion of cuts is typically much higher), on average 39 % of (MLP_k)'s do not lead to a cut.

It is also interesting to measure the effects of the heuristic rules we use in Step 3. of Algorithm 1 to filter and order fractional variables. If at each iteration, we solve (MLP_k) for all fractional variables the number of (MLP_k) that do not lead to a cut increases by 168% and the total time spent solving (MLP_k) by 54% on problems that did not go to the time limit. If the variables are not sorted in Step 3 of the algorithm, the number of pivots increases by 54%. Overall, the impact of these rule is not dramatic as most computing times remain short without them but it is significant on the most difficult problems: for the four most difficult instances that take more than 100 s not sorting the variables increases the computation time by 48.34% on average and not filtering increases it by 254.98%.

6.2 Comparison with GMI and lift-and-project cuts

We now present the comparison with GMI and lift-and-project cuts. To perform this comparison, we use the publicly available code for generating lift-and-project cuts `CglLandP` [6,16]. GMI cuts are simply generated by setting the pivot limits to 0 in `CglLandP`. Although this setting might be slightly slower than other GMI cuts generator; it presents the advantage of making certain to have identical tolerances for both GMI and lift-and-project cuts generators. We report results with different numbers of rounds of cut generation: 1 rounds, 10 rounds and 50 rounds. The first setting is to compare the gap closed by the rank 1 cuts of P_e and P_{e*} with the gap closed by the rank 1 cuts generated by the classical methods. The second setting is chosen because it is a reasonable setting for cut generators in a branch and-cut procedure. The last setting is chosen to see the behavior of GMI and strengthened lift-and-project cuts after a number of rounds that is comparable to the average number of master iterations taken by Algorithm 1 in our computations. To make the comparison fair, we also use a cut

Table 4 Comparison between P_e, P_{e^*}, GMI cuts and lift-and-project cuts generated by rounds

Geometric means of CPU times and average gap closed

	Time sep.	Time resolve	Gap closed (%)
P_e	1.42	2.02	48.23
P_{e^*}	1.08	1.33	60.80
1 GMI round	0.01	0.02	28.79
10 GMI rounds	0.06	0.16	49.33
50 GMI rounds	0.28	1.24	51.94
1 l-a-p round	0.07	0.02	30.11
10 l-a-p rounds	2.31	0.29	60.77
50 l-a-p rounds	13.19	2.37	63.18

pool to deal with large numbers of cuts generated with GMI and lift-and-project. Since the code for GMI and lift-and-project cuts works only with `Clp`, we only compare with our runs using `Clp`.

In Table 4, we report the geometric means of separation times, reoptimization times and arithmetical mean of gap closed for the instances of MIPLIB 3.0.

Several remarks are in order. First, comparing rank 1 cuts only, we note that the gap closed with P_e and P_{e^*} is considerably larger than the one closed by one round of either GMI or lift-and-project cuts. Of course, this requires a much higher computing time. Iterating GMI the gap closed never reaches the one closed by P_{e^*}. After 50 rounds of GMI it is slightly higher than with P_e. The gap closed by lift-and-project cuts after 10 and 50 rounds is significantly bigger than with GMI cuts and with P_e, comparable to P_{e^*} for 10 rounds and a bit bigger for 50 rounds. Regarding CPU times, we note that although GMI cuts generation is certainly faster than our procedure, for 50 rounds the separation time is only about 4 times faster. On the other hand, our procedure is faster than 50 or even 10 rounds lift-and-project cuts.

Of course, as more rounds are performed, the rank of the GMI and lift-and-project cuts increases. Fischetti, Lodi and Tramontani experimentally found that the rank typically increases faster with GMI cuts [28]. A common belief is that rank 1 cuts are numerically more stable than higher rank cuts. This is in particular observed when GMI or lift-and-project cuts are generated by rounds. Using our procedure, we can verify if this is also true when one considers the rank 1 cuts generated by our procedure for approximately optimizing over P_{e^*}. In Table 5, we report the geometric means of several statistics of the cuts generated by the different procedures: the total number of cuts generated, the number of active cuts at LP relaxation after all cuts have been applied, the average density of those tight cuts and the condition number κ of the optimal basis matrix.

From Table 5, we observe that indeed the numerical stability of the optimal basis deteriorates as more rounds of cuts are performed. For GMI cuts, the conditioning number of the basis matrix goes from 4.53×10^4 with 1 round to 1.34×10^8 with 50. For lift-and-project, it goes from 5.19×10^4 to 1.39×10^8. The density of the cuts generated also increases (although it is smaller for GMI with 50 rounds than with 10). It is particularly interesting to note that the cuts generated with P_e and P_{e^*} have very different behaviors. It is surprising to note that many more cuts are active in the

Table 5 Comparison between P_e, P_{e^*}, GMI cuts and lift-and-project cuts generated by rounds

	# Cuts	# Active	Density	κ
P_e	380.10	65.96	83.25	1.24×10^6
P_{e^*}	309.49	60.98	86.75	6.02×10^5
1 GMI round	20.60	9.08	95.95	4.53×10^4
10 GMI rounds	212.52	20.62	216.12	5.64×10^6
50 GMI rounds	828.14	16.84	187.44	1.34×10^8
1 l-a-p round	25.33	11.90	79.44	5.19×10^4
10 l-a-p rounds	310.49	38.87	170.40	2.37×10^6
50 l-a-p rounds	1,155.05	45.85	232.13	1.39×10^8

Geometric means of total number of cuts generated, number of active cuts, cuts density, and the condition number κ of the optimal basis

optimal basis (60.98 cuts for P_{e^*} vs. 16.84 and 45.84 after 50 rounds of GMI and lift-and-project respectively). The density of the cuts generated by P_{e^*} is of the same order as with 1 rounds of GMI or lift-and-project and much smaller than when more rounds are performed. Finally, although the condition number is bigger than for 1 round of cuts, it is smaller for P_{e^*} than with 10 rounds of either GMI or lift-and-project.

6.3 Comparison with other methods based on rank 1 split cuts

Now, we compare our results with several recent related works that also aim at separating rank 1 split cuts. First, we consider the computations over the split (or equivalently MIR) closure performed by Balas and Saxena [12] and Dash, Günlük and Lodi [25]. The split closure is a tighter relaxation than ours, but the computing times are usually very high. It is therefore interesting to know how close we are able to approach those results with our comparatively faster procedure. Secondly, we consider two methods that are aimed at generating rank 1 GMI cuts from basic tableaux. The first of these was proposed by Dash and Goycoolea [24] and consists of several heuristics for finding violated rank 1 GMI cuts. The second of these is a framework based on generating GMI cuts and relaxing them in a Lagrangian fashion proposed by Fischetti and Salvagnin [30]. Finally, we consider a recent work by Cornuéjols and Nannicini [23]. The test instances we consider are a subset of MIPLIB 3.0 and the MIPLIB 2003 that were used in [24,30] (split closure computations are only available for MIPLIB 3.0).

In Table 6, we report the average gap closed by the split closure (for each problem we took the strongest value from [12] and [25]), Dash and Goycoolea's method, Fischetti and Salvagnin's method, and our P_{e^*} procedure. Both [24] and [30] have proposed several variants of their methods, to keep the comparison concise we selected only one for each approach: the `default` approach of Dash and Goycoolea and the `fast` approach of Fischetti and Salvagnin. We also report geometric means of computing times. Note that these times should be considered with care because experiments have been performed on different machines with different CPUs and different LP solvers.

Table 6 Comparison of methods for rank-1 GMI cuts from tableaux

Method	MIPLIB 3.0		MIPLIB 2003	
	Time[a]	Gap (%)	Time[a]	Gap (%)
Split closure (best from [12,25])	7,556	79.16	–	–
Heuristic rank-1 GMI cuts [24]	23.78	60.80	875.43	39.68
Relax and GMI cuts [30] (`fast`)	2.56	58.95	58.42	45.47
P_{e^*}	1.54	64.77	84.78	44.55

[a] Timings on different machines.

Table 7 Comparison with reduce and split cuts on mixed models in MIPLIB 3.0

Method	Gap (%)
Split closure	78.64
P_{e^*} procedure	63.87
Existing split cuts + reduce and split [23]	49.30

The computing times in [24] refer to a 1.4 GHz PowerPC machine for MIPLIB 3.0 and to a 4.2 GHz PowerPC machine for MIPLIB 2003. The computing times in [30], refer to a 2.4 GHz Intel Q6600.

As can be seen from the table, the split closure is significantly stronger than any of the three methods using GMI cuts from tableaux: the split closure closes 79.2% of the integrality gap on MIPLIB 3.0 versus between 58.95% and 64.77% for the three GMI based methods. On the other hand, the three GMI based methods are orders of magnitude faster than the methods used to compute the split closure.

Comparing the three methods based on GMI from the tableau, P_{e^*} closes the most gap on MIPLIB 3.0 and [30] closes the most gap on MIPLIB 2003. The computing times of [30] and ours are close ([30] is slower on MIPLIB 3, but faster on MIPLIB 2003), while [24] seems to be somewhat slower (note that besides the different CPUs, the three methods have different termination criterions). An important element not conveyed by average figures is that none of the three methods dominates the others: each closes the most gap on some instances. On MIPLIB 3.0, P_{e^*} closes more gap than the other two on 24 instances, [30] closes the most gap on 12 and [24] closes the most gap on 16. On MIPLIB 2003, P_{e^*} closes the most gap on 8 instances, [30] on 7 and [24] for 4.

Another method to compute split cuts that differs from traditional GMI cuts from the tableau is the reduce-and-split method [1]. The method was recently re-explored and enhanced by Cornuéjols and Nannicini [23]. In Table 7, we compare the gap closed by our method to the gap closed in [23] and the Split Closure. The test set consists only of the mixed models in MIPLIB 3.0 since reduce-and-split are aimed at those. The gap closed reported in [23] corresponds to the gap closed by one round of several split cuts generators available in the Cgl library (GMI cuts, MIR cuts, lift-and-project, covers, flow covers) plus several variants of the reduce-and-split cuts. On these instances our method closes more gap on average than the combination of rank 1 cuts in [23]. Again

it is important to note that our method does not dominate [23] and we close less gap on 4 instances.

6.4 Branch-and-cut computations

The experiments reported in the last three subsections show that our approach based on P_{e^*} seems competitive for obtaining strong relaxations, at least of problems in the MIPLIB. Our ultimate goal would be to also solve those problems to optimality faster. Applying our method in a branch-and-cut setting is not straightforward. First, the computing times, although relatively small on average, are certainly higher than any of the cutting plane techniques already available in solvers. It is sometimes even larger than the total CPU time taken by state-of-the-art solvers such as CPLEX to solve the instance to optimality. Second, solvers combine many different cut generators and we need to include our method in the cut generation loop. Finally, although the gap closed is intuitively a good measure of the quality of a relaxation, many different factors can influence the performance of branch-and-cut procedures. Here, we present a preliminary attempt to deal with those issues and use P_{e^*} in a branch-and-cut setting.

To setup the experiment, we use the callback system of CPLEX in order to include our procedure in its branch-and-cut loop. Note that our procedure could be included in two ways. Since the outer loop of our procedure is similar to a classical cut generation loop, we could try to include our procedure by just adding the inner loop (steps 3 and 4 of Algorithm 1) as a cut callback. This would have the disadvantage of losing some control on the termination of the procedure. Therefore, we follow the simpler approach of running the complete procedure described by Algorithm 1 in a callback.

We call our procedure only once at the root node. Since CPLEX has its own cutting plane procedures, which are usually faster than ours, we should try as much as possible to benefit from them. In particular, even though we don't use the cuts generated by CPLEX in our separation LP, our procedure can benefit from CPLEX cutting planes because the point to cut should be closer to P_{e^*} after several rounds of cutting planes. For this reason, we try to call our procedure at the end of CPLEX's cut generation loop. As far as we know, there is no direct way of knowing if CPLEX has decided that it has exhausted its own cut generators. We attempt to detect it as follows: each time CPLEX calls our callback, we record the number of cuts generated for each class of cuts that CPLEX has; if from one iteration to the next none of those numbers changed, we launch our P_{e^*} procedure. At the end of the procedure, all the cuts found that are binding at the new optimum are returned to CPLEX.

Note that many decisions that could impact the efficiency of our procedure in a branch-and-cut can be made. For example, cut generation procedures usually have policies to reject cuts that are too dense or deemed numerically unstable. Also, our termination criterion and the tolerances we use may be too tight for practical purposes. In this preliminary experiment, we do not intend to settle these issues. First, we want to keep the results in line with those of the previous sections. Second, we believe that those issues should be the subject of further research. Our goal is limited to a preliminary view of the practical behaviour.

Table 8 Summary of results with branch-and-cut

Method	Root gap (%)	Time	No of nodes	Final gap closed
Instances in **set A**				
CPLEX-BAC	73.16	0.13	80.66	100
CPLEX+P_{e^*}-10	85.94	0.33	44.66	100
Instances in **set B**				
CPLEX-BAC	32.76	40.89	11,723.61	100
CPLEX+P_{e^*}-10	46.59	118.61	11,248.55	100
CPLEX+P_{e^*}-1800	59.30	304.87	5,058.73	100
Instances in **set C**				
CPLEX-BAC	23.20	>10,800	>38,139.95	45.81
CPLEX+P_{e^*}-10	25.02	>10,800	>24,889.09	44.45
CPLEX+P_{e^*}-1800	28.85	>10,800	>11,516.78	41.76
danoint				
CPLEX-BAC	2.92	7,364.8	1,119,451	100
CPLEX+P_{e^*}-10	3.60	>10,800	>548,803	40.91
CPLEX+P_{e^*}-1800	5.74	>10,800	>114,323	27.84
nsrand-ipx				
CPLEX-BAC	50.03	>10,800	>902,202	84.25
CPLEX+P_{e^*}-10	79.48	749.55	38,648	100

To test the procedure, we run CPLEX with two settings: CPLEX's default branch-and-cut algorithm[2] (denoted CPLEX-BAC), and CPLEX's branch-and-cut algorithm augmented with our cut callback. The latter is run with two different time limits for the P_{e^*} procedure: 10 s (denoted CPLEX+P_{e^*}-10), and 1800 s (denoted CPLEX+P_{e^*}-1800). In all settings, CPLEX is run on 1 thread and a global time limit of 3 h of CPU time is set. The test set consists of the MIPLIB 3.0 and MIPLIB 2003 instances used in Sect. 6.3. Note that the P_{e^*} procedure is now applied to the model preprocessed by CPLEX. To present the results, we group the instances in three sets: **set A** consists of 37 instances that are solved with both CPLEX-BAC and CPLEX+P_{e^*} in less than 10 s, **set B** contains 16 instances that are not in set A but are solved by all our setups, finally **set C** are 8 instances that could not be solved by any setup. Two instances do not belong to any of these sets and are reported separately: danoint is solved by CPLEX-BAC but not by CPLEX+P_{e^*}, nsrand-ipx is solved by CPLEX+P_{e^*} but not by CPLEX-BAC.

We report the results in Table 8. For each set of instances, we report the average gap closed at the root, the geometric means of CPU time and number of nodes processed and the average final gaps.

An interesting figure in Table 8 is that we are still able to generate violated rank 1 cuts after CPLEX has generated all its cuts. Our methods allows to close 12.78% more gap at the root for instances in set A, 13.83% for instances in set B with a time limit

[2] Dynamic search is not used because it cannot be used with a cut callback.

of 10 s and 26.64% with a time limit of 1800 s. For instances in set C, the difference is less significant: 1.82 and 5.65% with a time limit of 10 and 1800 s respectively. Unfortunately, the improvement in terms of gap closed at the root does not translate to faster solution times. For instances in set A the CPU time is about 3 times bigger. For instances in set B it is about 3 times bigger with CPLEX+P_{e^*}-10 and 7 times bigger for CPLEX+P_{e^*}-1800. For instances in set C, the gap closed after 3 h is on average smaller with CPLEX+P_{e^*}-10 or CPLEX+P_{e^*}-1800. The node reduction is limited with CPLEX+P_{e^*}-10, but with CPLEX+P_{e^*}-1800 we are able to divide the total number of nodes by a factor of more than 2 on set B.

7 Conclusions

In this paper we proposed a procedure to approximate the relaxation of a mixed integer linear program given by the lift-and-project closure. Our approach seems competitive at least for strengthening the initial formulation, and is also relatively fast compared to previous similar approaches. These results are another illustration of the strength of rank 1 cuts.

A possible source of improvement for the procedure in terms of CPU time, would be to use the separation oracle in a framework that has better convergence properties than the Kelley cutting plane method. It is well known that the Kelley method has slow convergence. Recently, Fischetti and Salvagnin proposed an in-out scheme for optimizing over P_e which bears many similarities with our approach [29]. The computational results they report do not seem competitive to ours in terms of CPU time but they are able to reduce significantly the number of iterations compared to a Kelley approach. These results give hope that the CPU times to compute P_e could still be significantly reduced (note however that adapting their approach does not seem trivial to us).

Another source of improvement would be to use the Balas Perregaard algorithm to strengthen the cut obtained by solving (MLP). From a theoretical point of view, this is a straightforward thing to do. One just need to give the optimal basis of (MLP) as a starting point of the Balas Perregaard algorithm.

Finally, we would like to emphasize the interest of finding an exact procedure to optimize over P_{e^*}. As noted in Sect. 6.3, none of the heuristics methods to generate rank 1 GMI cuts from the tableaux is dominating the others. Combining the three methods, even in a simple way, should give a better approximation of the optimal value over P_{e^*}. On the MIPLIB 3.0 test set, our respective average strengthening are 58.94% of the integrality gap closed in [24], 60.80% in [30] and 64.77% here. A trivial combination is to take, for each problem, the best result from each method (note that even if we add up the computing times of the three procedures, the CPU time is still short and a small fraction of the time to compute the split closure). If we combine the three methods in this trivial way the average goes up to 70.9%. This certainly shows that the three methods are still far from the optimal value over P_{e^*}. We only know that this value cannot be higher than the one obtained with the split closure which closes 79.2% of the integrality gap for this test set. As stated in introduction, as far as we know, the complexity of optimizing over P_{e^*} is

open. These numbers give a computational motivation for attempting to settle the question.

Acknowledgments Research was supported by ANR grant ANR06-BLAN-0375 and a Google Research Award. The author thanks the two referees for many suggestions that helped improving the paper significantly.

References

1. Andersen, K., Cornuéjols, G., Li, Y.: Reduce-and-split cuts: improving the performance of mixed integer Gomory cuts. Manag. Sci. **50**(11), 1720–1732 (2005)
2. Andersen, K., Cornuéjols, G., Li, Y.: Split closure and intersection cuts. Math. Progr. **102**, 457–493 (2005)
3. Balas, E.: Intersection cuts—a new type of cutting planes for integer programming. Oper. Res. **19**, 19–39 (1971)
4. Balas, E.: Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. SIAM J. Algebraic Discret. Methods **6**(3), 466–486 (1985)
5. Balas, E.: Disjunctive programming: Properties of the convex hull of feasible points. Discret. Appl. Math. **89**, 3–44 (1998) (originally MSRR # 348, Carnegie Mellon University, July 1974)
6. Balas, E., Bonami, P.: Generating lift-and-project cuts from the LP simplex tableau: open source implementation and testing of new variants. Math. Progr. Comput. **1**, 165–199 (2009)
7. Balas, E., Ceria, S., Cornuéjols, G.: A lift-and-project cutting plane algorithm for mixed 0-1 programs. Math. Progr. **58**, 295–324 (1993)
8. Balas, E., Ceria, S., Cornuéjols, G.: Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. Manag. Sci. **42**, 1229–1246 (1996)
9. Balas, E., Jeroslow, R.G.: Strengthening cuts for mixed integer programs. Eur. J. Oper. Res. **4**(4), 224–234 (1980)
10. Balas, E., Perregaard, M.: Lift and project for mixed 0-1 programming: recent progress. Discret. Appl. Math. **123**, 129–154 (2002)
11. Balas, E., Perregaard, M.: A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. Math. Progr. **94**, 221–245 (2003)
12. Balas, E., Saxena, A.: Optimizing over the split closure. Math. Progr. **113**, 219–240 (2008)
13. Balas, E., Zemel, E.: Facets of the knapsack polytope from minimal covers. SIAM J. Appl. Math. **34**, 119–148 (1978)
14. Bixby, R., Ceria, S., McZeal, C., Savelsbergh, M.: Miplib 3.0. http://www.caam.rice.edu/~bixby/miplib/miplib.html (1998)
15. Bixby, R., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: The Sharpest cut, chapter mixed-integer programming: a progress report, pp. 309–326. MPS-SIAM Series on Optimization. SIAM (2004)
16. Bonami, P., Balas, E.: Cgllandp. https://projects.coin-or.org/cgl/wiki/cgllandp. July (2006)
17. Bonami, P., Cornuéjols, G., Dash, S., Fischetti, M., Lodi, A.: Projected Chvátal–Gomory cuts for mixed integer linear programs. Math. Progr. Series A **113**(2), 241–257 (2008)
18. Bonami, P., Minoux, M.: Using rank-1 lift-and-project closures to generate cuts for 0–1 MIPs, a computational investigation. Discret. Optim. **2**(4), 288–307 (2005)
19. Caprara, A., Letchford, A.N.: On the separation of split cuts and related inequalities. Math. Progr. **94**(2–3), 279–294 (2003)
20. Chvátal, V.: Edmonds polytopes and a hierarchy of combinatorial optimization. Discret. Math. **4**, 305–337 (1973)
21. Cook, W., Kannan, R., Schrijver, A.: Chvátal closures for mixed integer programming problems. Math. Progr. **47**, 155–174 (1990)
22. Cornuéjols, G., Li, Y.: Elementary closures for integer programs. Oper. Res. Lett. **28**, 1–8 (2001)
23. Cornuéjols, G., Nannicini, G. Practical strategies for generating rank-1 split cuts in mixed-integer linear programming. Math. Progr. Comput. pp. 1–38. doi:10.1007/s12532-011-0028-6
24. Dash, S., Goycoolea, M.: A heuristic to generate rank-1 gmi cuts. Math. Progr. Comput. **2**, 231–257 (2010)
25. Dash, S., Günlük, O., Lodi, A.: MIR closures of polyhedral sets. Math. Progr. **121**(1), 33–60 (2010)

26. Eisenbrand, F.: On the membership problem for the elementary closure of a polyhedron. Combinatorica **19**(2), (1999)
27. Fischetti, M., Lodi, A.: Optimizing over the first Chvátal closure. Math. Progr. **110**, 3–20 (2007). doi:10.1007/s10107-006-0054-8
28. Fischetti, M., Lodi, A., Tramontani, A.: On the separation of disjunctive cuts. Math. Progr. **128**, 205–230 (2011)
29. Fischetti, M., Salvagnin, D.: An in-out approach to disjunctive optimization. In: Lodi, A., Milano, M., Toth, P., (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 6140, pp. 136–140. Springer, Berlin (2010)
30. Fischetti, M., Salvagnin, D.: A relax-and-cut framework for gomory's mixed-integer cuts. Math. Progr. Comput. **3**, 79–102 (2011)
31. Forrest, J.: CLP. http://www.coin-or.org/ (2004)
32. Gomory, R.: An algorithm for integer solution solutions to linear programming. In: Graves, R.L., Wolfe, P. (eds.) Recent Advances in Mathematical Programming, pp. 269–302. McGraw-Hill, New York (1963)
33. Gomory, R.E.: Solving linear programming problems in integers. In: Bellman R., Hall, M. (eds.) Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics, vol. 10, pp. 211–216. Providence (1960)
34. Kelley, J.E.: The cutting plane method for solving convex programs. J. SIAM **8**(4), 703–712 (1960)
35. Lougee-Heimer, R.: The common optimization interface for operations research. IBM J. Res. Dev. **47**, 57–66 (2003). http://www.coin-or.org
36. Martin, A., Achterberg, T., Koch, T.: MIPLIB 2003. http://miplib.zib.de (2003)
37. Nemhauser, G., Wolsey, L.: A recursive procedure to generate all cuts for 0-1 mixed integer programs. Math. Progr. **46**, 379–390 (1990)
38. Padberg, M., Van Roy, T., Wolsey, L.: Valid linear inequalities for fixed charge problems. Oper. Res. **33**, 842–861 (1985)
39. Roy, T.V., Wolsey, L.: Solving mixed integer programming problems using automatic reformulation. Oper. Res. **35**, 45–57 (1987)