Math. Prog. Comp. (2013) 5:171–200
DOI 10.1007/s12532-013-0052-9

FULL LENGTH PAPER

Local cuts for mixed-integer programming

Vašek Chvátal · William Cook · Daniel Espinoza

Received: 1 October 2009 / Accepted: 1 March 2013 / Published online: 26 March 2013

Abstract A general framework for cutting-plane generation was proposed by Applegate et al. in the context of the traveling salesman problem. The process considers the image of a problem space under a linear mapping, chosen so that a relaxation of the mapped problem can be solved efficiently. Optimization in the mapped space can be used to find a separating hyperplane, if one exists, and via substitution this gives a cutting plane in the original space. We extend this procedure to general mixed-integer programming problems, obtaining a range of possibilities for new sources of cutting planes. Some of these possibilities are explored computationally, both in floating-point arithmetic and in rational arithmetic.

Mathematics Subject Classification 90C11 · 90C57 · 65K05

Electronic supplementary material The online version of this article (doi:10.1007/s12532-013-0052-9) contains supplementary material, which is available to authorized users.

W. Cook was supported by NSF Grant CMMI-0726370 and ONR Grant N00014-03-1-0040. D. Espinoza was supported by FONDECYT Grant 1110024 and ICM Grant P10-024-F.

V. Chvátal
Department of Computer Science and Software Engineering,
Concordia University, Montreal, Canada

W. Cook
Department of Industrial Engineering,
University of Pittsburgh, Pittsburgh, USA

D. Espinoza (✉)
Department of Industrial Engineering, Universidad de Chile, Santiago, Chile
e-mail: daespino@gmail.com

1 Introduction

Consider a mixed-integer set

$$P_{IP} = \{x \in \mathbb{R}^n : \ Ax = d; \ l \le x \le u; \ x_j \in \mathbb{Z}, \ \forall j \in I\}, \tag{1}$$

where $A \in \mathbb{Q}^{m \times n}$, $d \in \mathbb{Q}^m$, $l \in (\mathbb{Q} \cup \{-\infty\})^n$, $u \in (\mathbb{Q} \cup \{+\infty\})^n$, and $I \subseteq \{1, \ldots, n\}$. As usual, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ denote the integer, rational, and real numbers, respectively. Given an objective vector $c \in \mathbb{Q}^n$, the problem of maximizing $c^T x$ subject to $x \in P_{IP}$ is a *mixed-integer programming* (MIP) problem. A linear relaxation

$$P_{LP} = \{x \in \mathbb{R}^n : \ Ax = d; \ A'x \le d'; \ l \le x \le u\} \tag{2}$$

of P_{IP} is obtained by dropping the integer restrictions on the variables x_j for $j \in I$, and possibly adding a system of inequalities $A'x \le d'$ that is satisfied by all vectors in P_{IP}. The additional inequalities $A'x \le d'$ are called *cutting planes*, or *cuts*.

In the *cutting-plane method*, cuts are added in an iterative fashion, selecting inequalities that are valid for P_{IP}, but violated by an optimal solution x^* to the linear programming (LP) problem $\max(c^T x : x \in P_{LP})$. The cutting-plane method is combined with a branch-and-bound search in a hybrid algorithm, known as branch-and-cut, that is the most successful MIP solution approach to date.

Given a basic optimal solution x^* to $\max(c^T x : x \in P_{LP})$, an elegant technique of Gomory [26] efficiently produces a cutting plane whenever $x^* \notin P_{IP}$. Indeed, the Gomory mixed-integer (GMI) cuts found by this method are one of the most important practical sources of MIP cutting planes, as reported in computational studies by Bixby et al. [14]. In the Bixby et al. study, effectiveness is measured by the improvement in the overall running time of the branch-and-cut algorithm for solving a specific large collection of MIP instances. In other studies, effectiveness is measured as the increase in the lower bound produced by the optimal objective value of the LP relaxation problem. With either measure, the additional value of GMI cuts tends to decrease as more and more cuts are added. The current practical reply to this decreasing performance is to use a variety of techniques for producing cuts in MIP solvers. Classes of MIP cutting planes currently adopted in leading codes include GMI cuts [11], mixed-integer-rounding (MIR) inequalities [34], knapsack covers [23,29], flow covers [28], lift-and-project cuts [10], and $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts [19].

An alternative approach to cutting-plane generation, based on the ability to optimize linear functions over a given relaxation, was proposed by Boyd [15]. His procedure was enhanced by Applegate et al. [5] in their *local-cuts* process for finding cutting planes in the context of the traveling salesman problem (TSP). The local-cuts process uses a mapping to work in a reduced space and thereby exploit fast oracles to solve relaxations of the mapped problem. Local cuts were subsequently adopted by Buchheim et al. [17,18] in the solution of constrained quadratic 0–1 optimization problems and by Althaus et al. [4] in the solution of Steiner-tree problems. Some of the ideas employed in the local-cuts process have also been introduced in the context of disjunctive programming by Balas et al. [9,12,36], where they study the description of cuts derived from general disjunctions over polyhedra, and give a characterization of the valid

inequalities of the convex hull of the disjunctive set, as well as iterative methods to find those cuts.

The goal of the present work is to extend the local-cuts paradigm to general mixed-integer programming. This allow us to (in principle) separate over MIP relaxations for which there is not a known polyhedral description, or where the linear description of the convex hull has exponentially many inequalities. We begin in Sect. 2 by presenting a standard LP-based method for generating MIP cuts using an oracle description of a relaxation of the original problem. In Sect. 3 we show how to obtain facets, or high-dimensional faces, from valid inequalities produced by the cut-generation process, generalizing the tilting procedure of Applegate et al. This is followed, in Sect. 4, by a discussion of linear mappings to simplify the mixed-integer set P_{IP}. In Sect. 5 we present the overall framework for local cuts for general MIP problems. Finally, in Sect. 6 we present threes set of experiments.

2 Separation via optimization

Let $P \subseteq \mathbb{R}^n$ be a rational polyhedron and let $x^* \in \mathbb{Q}^n$. The *separation problem* for (P, x^*) is to find an inequality $a^T x \leq b$ that is valid for P but violated by x^*, if such an inequality exists. A fundamental result in LP theory is the polynomial-time equivalence of separation and optimization for rational polyhedra, via the ellipsoid method. This theorem and its many combinatorial applications are discussed in Grötschel et al. [27]. In this context, polyhedra are represented implicitly. For example, the convex hull of a mixed-integer set P_{IP} can be represented by $Ax = d$, $l \leq x \leq u$ and $x_i \in \mathbb{Z}$, $\forall i \in I \subseteq \{1, \ldots, n\}$, rather than by an explicit linear description of the polyhedron.

We say that *OPT* is an *optimization oracle* for P if, for any $c \in \mathbb{Q}^n$, it asserts that P is the empty set, or provides $x^* \in P \cap \mathbb{Q}^n$ such that $c^T x^* \geq c^T x$ for all $x \in P$, or provides a ray $r^* \in \mathbb{Q}^n$ of P such that $c^T r^* > 0$. The output of the oracle is of the form $(status, \beta, y)$, where *status* is one of `empty`, `unbounded` or `optimal`; β contains the optimal value of $\max(c^T x : x \in P)$ if the problem has an optimal solution; and y contains the optimal solution or an unbounded ray if the status is `optimal` or `unbounded`, respectively.

An optimization oracle can be used as a practical vehicle for solving the separation problem without resorting to the ellipsoid method (which can be overly time-consuming in this context). This idea was championed by Boyd [15,16], who introduced a cutting-plane technique for MIP instances where the variables are restricted to take on 0 or 1 values. Boyd considers single-row relaxations, where his access to the corresponding polytope is via an optimization oracle, implemented as a dynamic-programming algorithm for 0–1 knapsack problems. His separation method is a variant of the simplex algorithm, and he calls the separating inequalities Fenchel cutting planes.

In our work on general MIP instances, we adopt a straightforward approach, based on an LP formulation of the separation problem. Let us write the rational polyhedron P as $P_g + P_r$, where P_g is a non-empty bounded polyhedron having vertices $\{g^i : i \in I_g\} \subset \mathbb{Q}^n$ and P_r is a convex cone generated by rays $\{r^i : i \in I_r\} \subset \mathbb{Q}^n$; here I_g and I_r are finite sets, allowing us to index the vertices and rays. A given $x^* \in \mathbb{Q}^n$ is an element of P if and only if there is a solution (λ^g, λ^r) to the system

$$\sum_{i\in I_g}\lambda_i^g g^i + \sum_{i\in I_r}\lambda_i^r r^i = x^*,\ e^T\lambda^g = 1,\ \lambda^g, \lambda^r \geq 0, \tag{3}$$

where $e \in \mathbb{Q}^n$ denotes the vector of all ones. By duality, system (3) has no solution if and only if there exist $a \in \mathbb{Q}^n$ and $b \in \mathbb{Q}$ such that

$$\begin{aligned} a^T g^i - b &\leq 0, \qquad \forall i \in I_g, \\ a^T r^i &\leq 0, \qquad \forall i \in I_r, \\ a^T x^* - b &> 0. \end{aligned} \tag{4}$$

Any cut that separates x^* from P corresponds to a solution of (4). To choose among these possible cuts, we formulate the problem of maximizing the violation of the resulting inequality, subject to the normalization constraint $\|a\|_1 = 1$, that is, we set the L_1-norm of a to 1. This is a technique described in a number of studies, for example, Balas et al. [10]. The resulting LP model is

$$\begin{aligned} \max\ & a^T x^* - b \\ s.t.\ & a^T g^i - b \leq 0, && \forall i \in I_g, \\ & a^T r^i \leq 0, && \forall i \in I_r, \\ & a - u + v = 0,\ e^T(u+v) = 1,\ u, v \geq 0\ . \end{aligned} \tag{5}$$

Its LP dual can be written as

$$\begin{aligned} \min\ & s \\ s.t.\ & \sum_{i\in I_g}\lambda_i^g g^i + \sum_{i\in I_r}\lambda_i^r r^i + w = x^*, \\ & e^T\lambda^g = 1, \\ & -w + se^T \geq 0,\ w + se^T \geq 0,\ \lambda^r, \lambda^g \geq 0. \end{aligned} \tag{6}$$

The LP problem (6) has $3n+1$ constraints other than bounds on individual variables. If we choose to minimize $\|a\|_1$ subject to the constraint $a^T x^* = b+1$, then we obtain the following formulation for the problem

$$\begin{aligned} \min\ & e^T(u+v) \\ s.t.\ & a^T g^i - b \leq 0 && \forall i \in I_g, \\ & a^T r^i \leq 0 && \forall i \in I_r, \\ & a - v + u = 0,\ a^T x^* - b = 1,\ u, v \geq 0\ , \end{aligned} \tag{7}$$

whose dual is

$$\begin{aligned} \max\ & s \\ s.t.\ & sx^* - \sum_{i\in I_g}\lambda_i^g g^i - \sum_{i\in I_r}\lambda_i^r r^i + w = 0, \\ & -s + e^T\lambda^g = 0, \\ & \lambda^g, \lambda^r \geq 0,\ -e \leq w \leq e. \end{aligned} \tag{8}$$

Note that (8) has only $n + 1$ constraints other than bounds on individual variables. The main advantage of formulation (8) comes from a practical side: Applegate et al. [6] report a relevant difference in running time when solving this formulation against (7), (6) or (5). The reason is that for most LP solvers, the running time grows linearly in the number of constraints (other than bounds on variables) and sub-linear in the number of variables.

Problem (8) is trivially feasible (the all-zero vector is always a solution); if (8) has an optimal solution, its dual gives us a separating inequality for P and x^*; if (8) is unbounded, then the unbounded ray provides us with a decomposition of x^* into elements of P_g and P_r, thus yielding a proof that $x^* \in P$. Finally, note that problems (5) and (7) are essentially the same, in the sense that any optimal solution of (5) is an optimal solution of (7) (after scaling) and any optimal solution of (7) is an optimal solution for (5) (after scaling).

The large number of variables in problem (8) can be handled by employing a column-generation technique. To begin, we select any (possibly empty) subset of points and rays in P, with index sets I_g and I_r, respectively. We then solve (8) with this new I_g and I_r. If the LP problem is unbounded, then we have a proof that $x^* \in P$. Otherwise, we obtain a tentative inequality $a^T x \leq b$ that is violated by x^*. The optimization oracle can be called to check whether $P \subseteq \{x : a^T x \leq b\}$ by maximizing $a^T x$ over P. If $P \subseteq \{x : a^T x \leq b\}$, we return with the inequality $a^T x \leq b$. Otherwise, we either find a point $g^j \in P$ such that $a^T g^j > b$ and add j to I_g, or we find a ray r^j of P such that $a^T r^j > 0$ and add j to I_r, and repeat the process. A pseudo-code for the method is given in Algorithm 1. We apply Algorithm 1 in our local-cuts procedure, after mapping to a space where an efficient optimization oracle is available.

Algorithm 1 Separation through optimization oracle $\texttt{SEP}(OPT, x^*, I_g, I_r)$

Require: OPT is an optimization oracle for P, x^* is a point to be separated from P, I_g indexes an initial set of points in P, I_r indexes an initial set of rays of P.

loop
  Solve (8) over I_g and I_r.
  if (8) is unbounded **then**
    return $x^* \in P$.
  let a, b be an optimal dual solution to (8)
  $(status, \beta, y) \leftarrow OPT(a)$.
  if $status = \texttt{unbounded}$ **then**
    add y to the set of rays indexed by I_r.
  else if $status = \texttt{optimal}$ and $\beta > b$ **then**
    add y to the set of points indexed by I_g.
  else
    return $x^* \notin P, (a, b)$

3 Obtaining high-dimensional faces

3.1 Building (non) facet certificates

Consider a rational polyhedron $P \neq \emptyset$, a rational vector $x^* \notin P$, and a separating inequality

$$a^T x \leq b, \tag{9}$$

satisfying $P \cap \{x : a^T x = b\} \neq \emptyset$. We describe an algorithm that transforms (9) into a facet-defining inequality for P or identify it as an implicit equality for P. The method extends the tilting algorithm for bounded polyhedra given by Applegate et al. [6] in the context of the TSP. This problem is closely related to the lifting problem and to fractional programming; this relation is explored in detail in Espinoza, Fukasawa and Goycoolea [24].

Let $P^\perp$ be the set of implicit equations for P, that is,

$$P^\perp := \left\{ a \in \mathbb{R}^n : \exists\, c_a \in \mathbb{R}, \forall x \in P\,, a^T x = c_a \right\}.$$

Note that $P^\perp$ is a linear space and $\dim(P) + \dim(P^\perp) = n$. We will make use of the following characterization of facets.

Lemma 1 (Certificates for facets) *Given a polyhedron $P \subseteq \mathbb{R}^n$ and a valid inequality $a^T x \leq b$ for P satisfied at equality by at least one point in P, then $F := \{x : a^T x = b\} \cap P$ is a facet of P if and only if there exist $Q \subseteq P^\perp$, $\overline{x} \in P$ satisfying $a^T \overline{x} < b$, and $P_o \subseteq F$ such that*

$$v^T p_i = 0, \qquad \forall p_i \in Q, \tag{10a}$$
$$w - v^T x^i = 0, \qquad \forall x^i \in P_o, \tag{10b}$$
$$w - v^T \overline{x} = 0, \tag{10c}$$
$$(v, w) \in [-1, 1]^{n+1} \tag{10d}$$

has as unique solution the all-zero vector.

Proof Suppose F is a facet of P. By definition, $\dim(F) = \dim(P) - 1$. Thus, there exists a set $Q \subset P^\perp$ of linearly independent vectors and a set $P_o \subset F$ and $\overline{x} \in P \setminus F$ of affinely independent vectors, satisfying $|Q| + |P_o| = n$. Finally, since all $p \in Q$ and $x \in P_o$ satisfy $p^T(x - \overline{x}) = 0$, we have that the rank of the constraint matrix in (10) is $n + 1$, and the result follows.

For the other direction, take the linearly independent singleton of constraints $B_o := \{(-\overline{x}, 1)\}$. Since the linear system admits a unique solution, we can extend B_o to a basis B of $\mathbb{R}^{n+1}$; since it is a basis, it must contain $n+1$ vectors. It is easy to see that the added vectors of the form $(p_j, 0)$ are linearly independent, and that the added vectors of the form $(-x^j, 1)$ induce a set of vectors $\{x^j\} \in Q$ that are affinely independent; thus proving that $\dim(P^\perp) + \dim(F) \geq |B| - 2 = n - 1$; this, and the fact that $\overline{x} \notin P_o$ proves that F is a facet of P. □

Note that condition (10d) is not really needed, but it is included to make the feasible region of (10) a compact set. The idea is to iteratively build P_o, Q and candidate inequalities using the conditions in (10). For that, we start with P_o as the active points satisfying our current candidate inequality at equality, and with $Q = \emptyset$ (or any previously known subset of $P^\perp$). The facet-finding algorithm ensures that at

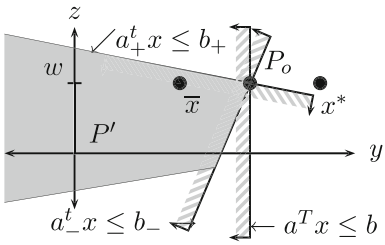

Fig. 1 The *gray area* represents P'; the points $\overline{x}$, P_o and x^* refer to their projection into P'

every iteration either the dimension of the candidate sets P_o or Q is increased, while possibly modifying the current inequality. We do this in such a way that every access to P is through an optimization oracle OPT.

To certify that our current inequality $a^T x \le b$ defines a proper face of P, we maximize $-a^T x$ over P. If the LP problem is unbounded, then we easily find a point $\overline{x} \in P$ such that $a^T \overline{x} < b$; if, on the other hand, the problem has an optimal solution with value different from $-b$, then such an optimal solution provides us with the sought $\overline{x} \in P$; otherwise, the optimal value is $-b$, thus proving that $a^T x = b$ is a valid equation for P that is violated by x^*, and thus proving that the starting inequality is an implicit equation for P separating x^* from P.

Next, using $\overline{x}$, P_o and Q, we check if the only solution to (10) is the all-zero vector. If this is the case, then we have a certificate that $a^T x \le b$ defines a facet of P. Otherwise, $(v, w, \overline{x})$, with $v \ne 0$, certifies that the current inequality is not facet-defining.

Suppose we have a non-facet certificate $(v, w, \overline{x})$. In this case we would like to increase the dimension of P_o or of Q. The idea we use is to *tilt* our current inequality $a^T x \le b$, using as pivot the set P_o, and using as rotating direction the vector (v, w), until we touch the border of P, identifying a new affinely independent point. To illustrate this method, consider $P' := \{(y, z) \in \mathbb{R}^2 : \exists x \in P,\ y = a^T x, z = v^T x\}$, as given in Fig. 1. In this example, rotating $a^T x \le b$ produces the inequalities $a_+^T x \le b_+$ and $a_-^T x \le b_-$, indicated in the figure. If we restrict ourselves to moving in this two-dimensional space, then these two inequalities are the only ones we can obtain by rotating $a^T x \le b$.

We must bear in mind that Fig. 1 is just one possible outcome for P'. Indeed, Fig. 2 shows four ill-behaving cases. Figure 2a is an example where one of the resulting inequalities coincides with $v^T x \le w$; Fig. 2b shows an example where $v^T x = w$ is a valid equation for P, giving us a new vector p to add to Q; Fig. 2c shows an example where one side of the tilting is in fact our original inequality $a^T x \le b$; and Fig. 2d shows an example where both sides of the tilting are our original inequality.

We provide a tilting algorithm in Sect. 3.2, but in the meantime, let us assume that we have a tilting routine with the following characteristics:

Condition 1 (Abstract Tilting Procedure: `TILT`$(a, b, v, w, \overline{x}, P_o, OPT)$).

Input *The input of the algorithm should satisfy all of the following:*
- $a^T x \le b$ *defines a face of* P *and* $P_o \subset P$ *is a set of points satisfying* $a^T x = b$.
- $v^T x \le w$ *is an additional inequality such that* $v^T x = w$ *for all* $x \in P_o$.

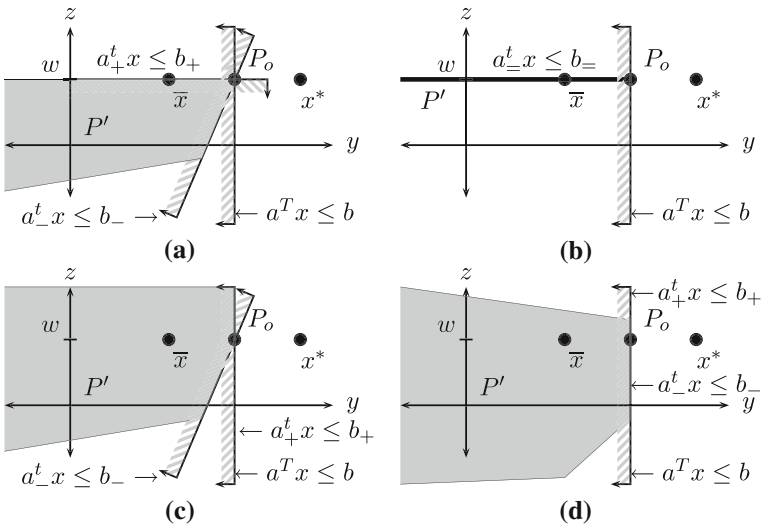

Fig. 2 Possible ill-behaving outcomes for the mapping of P

- $\overline{x} \in P$ *is such that* $a^T\overline{x} < b$ *and* $v^T\overline{x} = w$.
- *OPT is an optimization oracle for* P.

Output *The output should be:*

- $(v', w', \overline{x}')$ *where* (v', w') *is a non-negative combination of* (v, w) *and of* (a, b), $\max(v'^T x : x \in P) = w'$, $\overline{x}' \in P$, $v'^T\overline{x}' = w'$, *and* $\overline{x}'$ *is affinely independent from* P_o.

If we have a non-facet certificate $(v, w, \overline{x})$ for the inequality $a^T x \le b$ with point set P_o and equation set Q, then we can obtain both a_+, b_+ and a_-, b_- with the calls

$$(a_+, b_+, \overline{x}_+) = \texttt{TILT}(a, b, v, w, \overline{x}, P_o, OPT), \quad \text{and}$$
$$(a_-, b_-, \overline{x}_-) = \texttt{TILT}(a, b, -v, -w, \overline{x}, P_o, OPT).$$

With this information, we can finish our facet procedure. If $(v_+, w_+) = (v, w)$, and $(v_-, w_-) = (-v, -w)$, then we are in the situation depicted in Fig. 2b, and we can add v to Q, increasing its dimension by one. In any other situation we have two (possibly identical) inequalities, and we pick the one which is most violated by x^*[1]; assuming that a_+, b_+ is the most violated one, we replace a, b with a_+, b_+ and add $\overline{x}_+$ to P_o. Note that the newly added point increases the dimension of P_o by one.

Since at every step we either increase the dimension of Q or increase the dimension of P_o, the algorithm performs at most n iterations, and in every iteration we produce (or keep) a separating inequality. An outline of the complete method is given in Algorithm 2.

[1] It is easy to see that at least one of them must be violated by x^* if the original $ax \le b$ was violated by x^*.

Algorithm 2 FACET(a, b, x^*, P_o, Q, OPT)

Require: $a^T x \leq b$ is valid for P and $a^T x^* > b$; $\emptyset \neq P_o \subset P$ is such that $a^T x = b$ for all $x \in P_o$; $Q \subset P^\perp$; OPT is an optimization oracle for P.

$x_o \leftarrow x \in P_o$ /* select some point of P_o. */
loop
  /* find proper face certificate */
  $(status, \beta, y) \leftarrow OPT(-a)$.
  if $status =$ optimal and $\beta = -b$ **then**
    return (equation, a, b) /* $P \subseteq \{x : a^T x = b\}$ */
  else if $status =$ unbounded **then**
    $\overline{x} \leftarrow y + x_o$.
  else
    $\overline{x} \leftarrow y$.
  /* get (non-)facet certificate */
  if $(0, 0)$ is the unique solution for Problem (10) **then**
    return (facet, a, b)
  $(v, w) \leftarrow$ a non-trivial solution for Problem (10).
  $(a_+, b_+, \overline{x}_+) \leftarrow$ TILT$(a, b, v, w, \overline{x}, P_o, OPT)$.
  $(a_-, b_-, \overline{x}_-) \leftarrow$ TILT$(a, b, -v, -w, \overline{x}, P_o, OPT)$.
  /* update a, b, P_o, Q */
  if $(a_+, b_+) = (-a_-, -b_-) = (v, w)$ **then**
    $Q \leftarrow Q \cup \{v\}$. /* grow dimension of Q */
  else
    $\lambda_+ \leftarrow \left(a_+^T x^* - b_+\right) / \|a_+\|$.
    $\lambda_- \leftarrow \left(a_-^T x^* - b_-\right) / \|a_-\|$.
    if $\lambda_+ > \lambda_-$ **then**
      $(a, b) \leftarrow (a_+, b_+)$.
      $P_o \leftarrow P_o \cup \{\overline{x}_+\}$. /* grow dimension of P_o */
    else
      $(a, b) \leftarrow (a_-, b_-)$,
      $P_o \leftarrow P_o \cup \{\overline{x}_-\}$. /* grow dimension of P_o */

3.2 Solving the tilting problem

We now describe an algorithm that performs the tilting procedure satisfying Condition 1. We assume that we have a valid inequality $ax \leq b$ for P, a non-empty, finite set $P_o \subset P$ for which every element $x \in P_o$ satisfies $a^T x = b$. We also have another inequality (although it might not be valid for P) $v^T x \leq w$ and a point $\overline{x} \in P$, such that all $x \in P_o$ and $\overline{x}$ satisfy $v^T x = w$ and such that $a^T \overline{x} < b$. We show how to obtain a_+, b_+; the procedure for a_-, b_- is completely analogous.

Our objective is to find a valid inequality $v'^T x \leq w'$ for P and a point $\overline{x}'$ that is affinely independent from P_o and is such that $v'^T \overline{x}' = w'$. The idea is to use $v^T x \leq w$ as our candidate output constraint, and $\overline{x}$ as our candidate for an affinely independent point, thus, we only need to check if $\max(v^T x : x \in P) = w$.

If we maximize $v^T x$ over P and there is an optimal solution with value w, then we are in the situation depicted by Fig. 2a or b. In this case, we return $v^T x \leq w$ as our tilted inequality and report $\overline{x}$ as our new affinely independent point.

If $\max(v^T x : x \in P)$ is unbounded, then we are in the situation depicted by Fig. 2c or by Fig. 1. In this case, we have a ray r of P returned by the optimization oracle. Since $a^T x \leq b$ is a valid inequality for P, we have $a^T r \leq 0$. Let $x' = x + r$, where

Algorithm 3 TILT$(a, b, v_o, w_o, \overline{x}_o, x_o, OPT)$

Require: $\overline{x}_o \in P, a^T\overline{x}_o < b, v_o^T\overline{x}_o = w_o, x_o \in P_o, a^T x_o = b, v_o^T x_o = w_o$.
$a^T x \leq b$ is a valid inequality for P, (a, b) and (v_o, w_o) are linearly independent.
1: $k \leftarrow 0$.
2: **loop**
3: $(status, \beta, y) \leftarrow OPT(v_k)$
4: **if** $status =$ `optimal` and $\beta = w$ **then**
5: **return** $(v_k, w_k, \overline{x}_k)$
6: **if** $status =$ `unbounded` **then**
7: $\overline{x}_{k+1} \leftarrow y + x_o$
8: **else**
9: $\overline{x}_{k+1} \leftarrow y$
10: $\lambda \leftarrow v_k^T\overline{x}_{k+1} - w_k,\ \mu \leftarrow b - a^T\overline{x}_{k+1}$
11: $v_{k+1} \leftarrow \lambda a + \mu v_k,\ w_{k+1} \leftarrow \lambda b + \mu w_k$.
12: $k \leftarrow k + 1$.

x is any point in P_o; note that x' is affinely independent from P_o, since $v^T x = w$ for all points in P_o and $v^T r > 0$. If, on the other hand, $\max(v^T x : x \in P)$ exists, then define x' as the optimal solution to $\max(v^T x : x \in P)$ that is returned by the oracle; in this case, we have again that x' is affinely independent from P_o.

To continue our algorithm we will find a positive combination of $v^T x \leq w$ and $a^T x \leq b$ such that every point in P_o still satisfies it at equality, but such that x' also satisfies it at equality. For this let $\lambda := v^T x' - w > 0$, $\mu := b - a^T x' \geq 0$, and define the inequality

$$(v', w') = \lambda(a, b) + \mu(v, w). \tag{11}$$

Since (11) is a positive combination of (a, b) and (v, w), every point x in P_o satisfies $v'^T x = w'$. Moreover, x' also satisfies (11) at equality. To see this, note that $v'^T x' - w' = \lambda(a^T x' - b) + \mu(v^T x' - w) = -\lambda\mu + \mu\lambda = 0$. Now we replace (v, w) by (v', w') and $\overline{x}$ by x', and we repeat the previous process.

Every step where we redefine our tentative (v, w) inequality is called a *push* step. Figure 3 shows a sequence of push steps that end with the desired inequality.

Algorithm 3 gives an outline of the tilting algorithm, and also defines the output for it. We now show that this algorithm satisfies Condition 1.

Lemma 2 *In every push step, there exists $\lambda_k \geq 0$ and $\mu_k \geq 0$ such that $(v_k, w_k) = \mu_k(v_o, w_o) + \lambda_k(a, b)$ and where $\mu_k + \lambda_k > 0$.*

Proof We proceed by induction, the case $k = 0$ being trivially true with $\lambda_o = 0$, $\mu_o = 1$. We assume now that the result is true for k, and that the algorithm does not stop during this iteration (otherwise we have finished the proof). We have that $\overline{x}_{k+1}$ is such that $\lambda = v_k^T\overline{x}_{k+1} - w_k > 0$ and that $\mu = b - a^T\overline{x}_{k+1} \geq 0$. By definition, $(v_{k+1}, w_{k+1}) = \lambda(a, b) + \mu(v_k, w_k)$, and by assumption $(v_k, w_k) = \lambda_k(a, b) + \mu_k(v_o, w_o)$. Now, by defining $\lambda_{k+1} = \lambda + \lambda_k\mu$ and $\mu_{k+1} = \mu\mu_k$, we obtain our result. □

Lemma 3 *At each step, $\overline{x}_k$ and all $\overline{x} \in P_o$ satisfy $v_k^T x = w_k$, and $\overline{x}_k$ is affinely independent from P_o.*

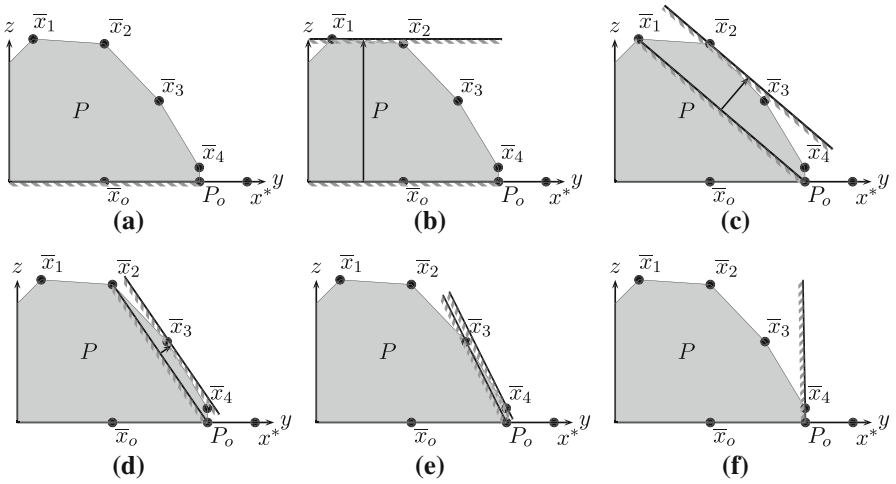

Fig. 3 Sequence of push steps for P

Proof By construction. □

It remains to prove that the algorithm terminates after a finite number of steps. To this end, we need to make the assumption that for a given polyhedron P the oracle will return only one of a finite number of rays and points. This assumption is mild in our context, since any polyhedron with rational data can be represented as a finitely generated cone plus the convex hull of a finite set of feasible points. To verify that the algorithm terminates under this assumption we show that there is a function that strictly increases in every iteration.

Define $f(\lambda, \mu) : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+ \cup \{\infty\}$ as $f(\lambda, \mu) = \frac{\lambda}{\mu}$, where we set $\frac{t}{0} = \infty$, $\forall t \geq 0$.

Lemma 4 *For all integers $k \geq 0$, $f(\lambda_k, \mu_k) < f(\lambda_{k+1}, \mu_{k+1})$.*

Proof Note that $f(\lambda_o, \mu_o) = \frac{0}{1} = 0$. Moreover, by the proof of Lemma 2 we have that $f(\lambda_{k+1}, \mu_{k+1}) = \frac{\lambda + \lambda_k \mu}{\mu \mu_k} = \frac{\lambda}{\mu \mu_k} + \frac{\lambda_k}{\mu_k}$. Note that if $\mu_k = 0$ then $(v_k, w_k) = \lambda_k(a, b)$, thus ensuring (by hypothesis of the algorithm) that the algorithm will stop in iteration $k + 1$. Then, we may assume that $\mu_k \neq 0$. By the previous equations, we have that $f(\lambda_{k+1}, \mu_{k+1}) = \frac{\lambda}{\mu \mu_k} + f(\lambda_k, \mu_k)$. Since $\lambda > 0$ and $\mu \geq 0$, we have that $\frac{\lambda}{\mu \mu_k} > 0$. This proves our claim. □

Lemma 5 *At every iteration, $f(\lambda_k, \mu_k) = \frac{v_o x_k - w_o}{b - a x_k}$.*

Proof Note that the statement is true for $k = 0$. For any other k we have $f(\lambda_{k+1}, \mu_{k+1}) = \frac{v_k x_{k+1} - w_k + \mu \lambda_k}{\mu \mu_k}$ and $v_k x_{k+1} - w_k + \mu \lambda_k = \lambda_k a x_{k+1} + \mu_k v_o x_{k+1} - \lambda_k b - \mu_k w_o + (b - a x_{k+1}) \lambda_k = \mu_k (v_o x_{k+1} - w_o)$. Since $\mu = b - a x_{k+1}$, we obtain the result. □

Theorem 6 *If the optimization oracle returns only a finite number of distinct feasible points and/or rays, then Algorithm 3 terminates in a finite number of steps.*

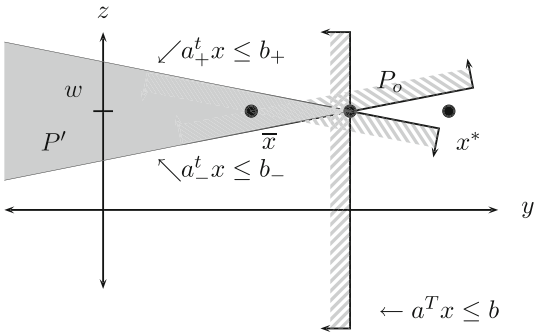

Fig. 4 Rotated inequalities with small violation

Proof First, note that if in step 3 of Algorithm 3 the oracle returns (`unbounded`, r_{k+1}), then $x_{k+1} = r_{k+1} + x_o$, and since $ax_o = b, v_o x_o = w_o$ we obtain that $f(\lambda_{k+1}, \mu_{k+1}) = \frac{v_o r_{k+1}}{-ar_{k+1}}$. On the other hand, if the oracle returns (`optimal`, x_{k+1}), then $f(\lambda_{k+1}, \mu_{k+1}) = \frac{v_o x_{k+1} - w_o}{b - ax_{k+1}}$. This proves that f provides a total order for all possible outputs of the oracle. Since at every iteration we either stop or increase f, and the possible answers of the oracle are finite, the algorithm must terminate. □

3.3 More on the facet procedure

The inequality produced by our facet procedure can have arbitrarily small violation, as illustrated in Fig. 4. However, the distance from x^* to the set of points satisfying both $a_+^T x \le b_+$ and $a_-^T x \le b_-$ is at least the distance from x^* to the set of points satisfying our original constraint $a^T x \le b$. We therefore choose to record both $a_+^T x \le b_+$ and $a_-^T x \le b_-$ in our implementation, but we proceed with the facet procedure only on the most violated of the two inequalities. Note that we might take this approach even further and iterate both inequalities through the facet procedure. The advantage of such a modification is that we find a set of facets of P that ensure the same violation as the original inequality, but the drawback is that it may require many more calls to the oracle.

High-dimensional faces. While the appeal of facets is clear, it may be that the computational cost of the overall procedure is too large. This suggests that we may want to stop the process after a certain number of steps. Another interesting question is how to choose a non-trivial solution v, w from (10) to perform each tilting round. In our implementation we select v, w as the vector satisfying (10), with the largest projection into $x^* - \text{proj}_{\text{conv\_hull}(P_o)}(x^*)$, but many other alternatives exists. This problem can be formulated as shown in (12).

$$\max \quad v^T x^* - w \tag{12a}$$
$$v^T p_i = 0, \qquad \forall p_i \in Q, \tag{12b}$$
$$w - v^T x_i = 0, \qquad \forall x_i \in P_o \cup \{\overline{x}\}, \tag{12c}$$
$$\|v\|_2 \le 1. \tag{12d}$$

A difficulty with (12) is that constraint (12d) is non-linear. However, if we replace it with an L_∞-normalization, we obtain again a linear problem. This is the approach we take in our implementation. At every step of the facet procedure described in Algorithm 2, we choose a non-trivial solution (if one exists) of the problem

$$\begin{aligned} \max \quad & v^T x^* - w && && (13a)\\ & v^T p_i = 0, && \forall p_i \in Q, && (13b)\\ & w - v^T x_i = 0, && \forall x_i \in P_o \cup \{\overline{x}\}, && (13c)\\ & -1 \le v_i \le 1. && && (13d) \end{aligned}$$

This technique can be viewed as a greedy approach for choosing the new tentative $v^T x \le w$ inequality. It would be interesting to find procedures to choose (v, w) such that the final facet satisfies certain properties, for example, that its area is large.

4 The mapping problem

Rather than directly computing inequalities for P_{IP}, the local-cuts procedure works on related sets designed to be easier to handle via optimization methods. Consider a function $\pi : \mathbb{R}^n \to \mathbb{R}^{n'}$ and a set $P'_{IP} \subseteq \mathbb{R}^{n'}$ such that for all points $x \in P_{IP}$ we have $\pi(x) \in P'_{IP}$. We call π a *mapping function* and we call P'_{IP} a *valid mapping* for P_{IP}. We denote by $\overline{P}_{IP}$ and $\overline{P}'_{IP}$ the convex hull of P_{IP} and P'_{IP}, respectively.

If π is a linear (affine) function, then any valid inequality $a^T y \le b$ for $\overline{P}'_{IP}$ yields a potential cutting plane for the original problem, namely $a^T \pi(x) \le b$. Indeed, if we write $\pi(x) = Mx - m_o$, then the cut in the original space can be written as $a^T Mx \le b + a^T m_o$. Note also that any linear mapping function satisfies $\pi(\overline{P}_{IP}) \subset \overline{P}'_{IP}$. We call π a *separating* mapping function if $\pi(x^*) \notin P'_{IP}$.

Many of the well-known techniques for producing general MIP cutting planes can be framed as methods for obtaining valid inequalities in suitably mapped spaces, including GMI cuts, MIR inequalities, and Fenchel cuts. The local-cuts paradigm provides a method for extending and combining these procedures, giving the possibility of strengthening the LP relaxations, at the expense of increased computation in the construction of the cuts.

A general discussion of separating mappings is given in the Ph.D. thesis of Espinoza [25]. Here we illustrate the procedure by considering a simple class called *implicit mappings*.

Implicit mappings. A natural way to obtain a mapping is to define P'_{IP} as the set of points y that are a linear combination of points x satisfying a set of linear combinations of the original constraints. More precisely, we say that a mapping is *implicit* if

$$P'_{IP} = \left\{ y \in \mathbb{R}^{n'} : \exists x \in \mathbb{R}^n,\ l \le x \le u,\ RAx = Rd,\ y = Mx,\ y_i \in \mathbb{Z}, \forall i \in I' \right\}$$

where R and M are matrices and $I' \subseteq \{1, \ldots, n'\}$. In this case the mapping function is just $\pi(x) = Mx$. By considering the system $RAx = Rd$ obtained by taking linear

combinations of $Ax = d$, the class of implicit mappings includes those relaxations created by selecting rows of a simplex tableau for $\max(c^T x : x \in P_{LP})$, such as the GMI cuts. In the case of GMI cuts, the mapping P'_{IP} has a single integer variable, obtained by aggregating the integer variables appearing in the tableau row.

A sufficient condition for an implicit mapping to be valid is to have $M_{ij} = 0, \forall i \in I', j \notin I$, and $M_{ij} \in \mathbb{Z}, \forall i \in I', j \in I$. That is, all integer variables in P'_{IP} must be integer combinations of integer variables in P_{IP}. A necessary condition for an implicit mapping to be separating is that there exists some objective function $\mu \neq 0$ such that $\max(\mu^T y : y \in P'_{LP}) = \mu^T M x^*$, where x^* is the point that we are separating. If we also have that $\pi(x^*)$ is the unique optimal solution to $\max(\mu^T y : y \in P'_{LP})$, and some integer-constrained variable $\pi(x^*)_i = y_i^*, i \in I'$, is fractional, then the mapping is separating.

Simple local cuts. The well-known result of Lenstra [32] provides a polynomial-time algorithm for solving MIP instances having a fixed number of integer variables. This suggests the following implicit mapping. Set M and R as identity matrices, and define $I' \subseteq I$ such that $|I'| \leq k$ for some fixed k. We show in a computational study that this simple process can produce effective cuts, with k as small as four.

Disjunctive mappings. Yet another possibility is to consider as our relaxation the union of the LP relaxations of the active leaves of a partial branch-and-bound tree, and use as mapping space the projection into the set of branching variables used in the partial branch-and-bound tree and the objective function.

To be more precise, let $J = \{i : x_i$ was branched on at some point in the partial branch-and-bound tree $\}$, and let $n' = 1 + |J|$. With this, we define

$$\begin{aligned} \pi : \ & \mathbb{R}^n \to \mathbb{R}^{n'} \\ \pi(x) = & \left(cx, \text{proj}_J(x)\right), \end{aligned} \tag{14}$$

where c is the objective function in the original problem. Also, we implicitly define

$$P'_{IP} = \text{conv\_hull}\left(\bigcup_{P_i \in \mathcal{L}} \pi(P_i)\right),$$

where $\mathcal{L}$ is the set of active leaves in the partial branch-and-bound tree.

In this case, the oracle reduces to optimizing linear functions over a set of LP relaxations, and taking the maximum over all of them.

With this choice of π, whenever we have a partial branch-and-bound tree which improves the lower bound for the problem, we are guaranteed to obtain a separating cut, and recover that bound through cuts.

The resulting cuts are related to the cuts proposed by Balas and Perregaard [36], where they use the set of all disjunctions arising from a set of k arbitrary disjunctions. However, in our case we use only the leaves from the given partial branch-and bound tree, which in most cases contain far fewer than 2^k leaves for a set of k branching variables. Another difference is that we work on a mapped space, whose dimension can be much smaller than the dimension of the original problem, which should require

far fewer calls to the oracle to perform a separation step. Moreover, we can prune many disjunctions, not only by infeasibility, but also by objective value.

5 The local-cuts procedure: Implementation details and settings

We are now in position to describe the full scheme for MIP local cuts. We start with a given MIP problem P_{IP}, a linear relaxation P_{LP}, and a fractional basic optimal solution x^* to P_{LP}. We choose some linear mapping π and an image space P'_{IP}, for which we provide an oracle description $\mathcal{O}_{P'_{IP}}$. We then call Algorithm 1 as $\texttt{SEP}(\mathcal{O}_{P'_{IP}}, \pi(x^*), \emptyset, \emptyset)$. If the algorithm finds a separating inequality, we proceed to call Algorithm 2 using as input the separating inequality found by Algorithm 1, and using as P_o the set of points satisfying the separating inequality at equality. We then take the resulting inequalities and map them back to the original space and add them to our current relaxation.

Also, instead of asking for facets of the mapped problem, we ask only for faces with dimension at least ten, but keeping all intermediate constraints produced by algorithm FACET.

Precision. Any error in the solution of the separation LP (8), or in the solutions provided by the oracle, may lead to invalid inequalities. Applegate et al. [6] deal with this issue by using integer representations of the inequalities and of the LP solutions, and also by using an oracle whose solutions are always integer. Since we are dealing with general mixed integer problems, such an approach is not viable. The problem of numerically safe cuts has been widely studied (see for example [20]). Also, a hybrid approach for solving exactly, in rational arithmetic, general MIP problems was developed by Cook et al. [21] with competitive running times. In this study we take two extreme implementation strategies.

Floating point implementation: To have comparable results with others studies in the literature, we implemented our algorithm using standard *floating-point* number representation, relying on CPLEX 12.4 as our base LP and IP solver. It is important to note that although we did try to make the code as safe and accurate as possible, by its very nature we can not guarantee that the reported values are correct in the sense that every generated inequality does not cut off feasible solutions.

Rational implementation: In order to have numbers where no numerical errors can occur, we implemented a different code using rational representation, and the exact rational LP solver described in [7]. A point to note when working with rational inequalities is that their representation may grow as we go along in the algorithm. Also, from our experiments, we know that even LP instances with simple coefficients can generate solutions that need a long encoding to be represented. It seems that as we allow more and more complicated inequalities, the situation only worsens; this is in line with the experience of other authors [37]. Thus, to keep the running time for this implementation under some control, we choose to accept only cuts where the denominator and numerator of each coefficient can be expressed using at most 320 bits.

Improving oracles. In the tilting procedure described in Algorithm 3, at each iteration we have a candidate inequality and we ask whether or not it is valid for the polyhedron.

Table 1 CPLEX settings for mapping experiments

Local cuts	CPX-root-pre	CPX-root-cut	CPX-bc
	CLIQUES = −1.	CLIQUES = 2.	
	COVERS = −1.	COVERS = 2.	
	CUTSFACTOR = 10.	CUTSFACTOR = 10.	
	DISJCUTS = −1.	DISJCUTS = 2.	
	EPGAP = 10^{-5}.	EPGAP = 10^{-5}.	
CUTSFACTOR = 10.	FLOWCOVERS = −1.	FLOWCOVERS = 2.	
EPGAP = 10^{-5}.	FLOWPATHS = −1.	FLOWPATHS = 2.	
HEURFREQ = −1.	FRACCUTS = −1.	FRACCUTS = 2.	CUTSFACTOR = 10.
NODELIM = **xxx**.	GUBCOVERS = −1.	GUBCOVERS = 2.	EPGAP = 10^{-5}.
PROBE = 3.	HEURFREQ = −1.	HEURFREQ = −1.	LBHEUR = ON.
RINSHEUR = −1.	IMPLBD = −1.	IMPLBD = 2.	MIPEMPHASIS = 3.
STARTALG = 4.	MIPEMPHASIS = 3.	MIPEMPHASIS = 3.	STARTALG = 4.
THREADS = 1.	MIRCUTS = −1.	MIRCUTS = 2.	THREADS = 1.
VARSEL = 3.	NODELIM = 1.	NODELIM = 1.	
	PROBE = 3.	PROBE = 3.	
	RINSHEUR = −1.	RINSHEUR = −1.	
	STARTALG = 4.	STARTALG = 4.	
	THREADS = 1.	THREADS = 1.	
	VARSEL = 4.	VARSEL = 4.	
	ZEROHALFCUTS = −1.	ZEROHALFCUTS = 2.	

All options not reported here were left at their default value The name of the actual CPLEX parameter is the given name with `CPX_PARAM_` prepend to it. Each local-cut configuration changes the corresponding NODELIM parameter

If it is not valid, then it is sufficient to find a point in P that violates the candidate inequality. In particular, an optimal solution is not required. We have seen in practice that working with such an *improving oracle* speeds-up the overall performance of the local-cut procedure.

Basic hardware and software settings. All runs where made using a single thread with an address space limit and data limit of 4Gb. The machines used were running Linux 2.6.18 under the x86_64 architecture, with two quad-core Intel® Xeon® E5620 processors and with 48Gb of RAM. To obtain more accurate timing measures, the machines where configured (in BIOS) with the following technologies disabled: Intel® Turbo Boost Technology, Intel® Hyper-Threading Technology, and Intel® Virtualization Technology (VT-x). These settings allowed us to run up to eight instances on a machine without (much) interference between processes.

CPLEX settings. The CPLEX 12.4 MIP code has 149 parameters configurable by the user. To describe the settings used in our study, we report which CPLEX options were changed from their default values (as declared in the CPLEX 12.4 manual), and to what value they were changed. The changes for each algorithm can be found in Table 1. Note that some further differences in behavior may be expected because our code uses cut,

branch, delete node, and incumbent node callback functions, which may disable other advanced-search mechanisms within the CPLEX MIP solver. Configuration **CPX-bc** is traditional branch-and-cut with emphasis on bound improvement; **CPX-root-pre** is pure pre-processing at the root node (no cutting planes are added); **CPX-root-cut** is as an aggressive cutting strategy limited to the root node; while the **Local Cuts** configuration is a strong-branching strategy with extra space for cutting and without heuristics. Note that all configurations use barrier to solve the starting root LP; in our tests this resulted in slightly better running times than with either primal or dual simplex.

6 Computational experiments, conclusions, and further questions

As described earlier, we performed three sets of experiments. In the first set, we show that the ability to implicitly work on small-dimensional spaces can be key when trying to use this cut-generation scheme. These tests are carried out on the MIPLIB 3.0 [13] set of test instances, with computations performed in floating-point arithmetic. To serve as a reference value for the gap closed by the cutting planes in this first set of experiments, we use the split-closure bound computed by Balas and Saxena [12]. In the second set of experiments, we use disjunctive mappings and test the performance of the resulting cuts on the MIPLIB 3.0 [13], MIPLIB 2003 [3] and MIPLIB 2010 [31] test sets; in this case the comparison is done against CPLEX 12.4. In the third set of experiments, our computations are carried out in full rational arithmetic using *simple mappings*.

6.1 The importance of mappings

In these experiments, our main objective is to show that mapping into small-dimensional spaces is a key ingredient if we want to generate cuts in a reasonable running time.

In this setting we compare relaxations of the form

$$P(\mathcal{L}, J) := \operatorname*{proj}_{\mathbb{R}^J} \left(\text{conv\_hull} \left(\bigcup_{L_i \in \mathcal{L}} L_i \right) \right),$$

where we assume that our original problem is of the form $\min\{z : (z, x) \in P \subseteq \mathbb{R}^n,\ x_i \in \mathbb{Z}, \forall i \in I \subseteq \{1, \ldots, n\}\}$, $J \subseteq \{1, \ldots, n\}$ and $\mathcal{L} = \{L_i\}_{i=1}^k$ is a collection of convex sets whose union contains P. In our particular implementation, we generate $\mathcal{L}$ by letting CPLEX 12.4 branch until it generates up to 2, 8, 32 and 128 nodes in the branch and cut tree using strong branching, and we choose J as either $J = \{1, \ldots, n\}$ (i.e. use the full set of variables), or let $J = \{z\} \cup \{x_k : \text{CPLEX branched at some point using variable } x_k\}$. Note that in the second case, we are using *disjunctive mappings* as defined in (14). In the first case, if x^* is a lexicographically minimal solution, we are using a *separating mapping*, and as such, we know that we can always find a cut. The only difference is the dimension of the relaxation that we work on and the strength

of the resulting cuts. Is easy to see that the bound obtained in the first case must be stronger than in the second case, but the question is how much we lose in terms of closed gap by working on a smaller space and how much faster we are by doing so.

It is important to note that if CPLEX solves the problem using fewer search nodes than the prescribed limit, then, without adding any cuts, the procedure will report the optimal value for the problem as its bound. On the other hand, when $|J| < n$, the mapping may not be pointed, which implies that no cut separating the projected fractional point exists, in which case the procedure will stop without adding any cut and reporting the branch-and-bound value as its final root LP bound. These experiments had a running time limit of one day.

Our algorithm tries to separate the projected LP optimum of the root node using the chosen relaxation. If we succeed in finding a separating inequality, we add it to the original root node relaxation, resolve the LP, and (try to) separate the resulting point with the same relaxation until no separating inequality satisfying $(ax^* - b)/\|a\|_\infty \geq \varepsilon_1$ and $(ax^* - b)/\|a\|_2 \geq \varepsilon_2$ is found. Using this new LP relaxation, we generate a new partial branch-and-bound tree and iterate the previous process. If the current relaxation has not been able to produce a single cut, we stop the cutting process.

To speed up the cut-generation process, we store the previous optimal base for every active leaf in our relaxation. We also use an upper bound (given by the current right hand side of the candidate inequality) for the LP algorithm in every leaf problem. Furthermore, we store a short list (five elements in the current implementation) of the leaves that get the largest objective values in the first optimization round. We then use this list as a *short* pricing list, and once this set does not produce violating points, we do a full pricing and generate (if any) a new hot list.

To have a comparison point, we use the results presented by Balas and Saxena in [12] over the MIPLIB 3.0 instances. We also use the LP lower bounds used in [12]; these configurations will be called **SC**, and **LB** respectively. For our mappings, we call **B2-f**, **B8-f**, **B32-f** and **B128-f** the configurations using up to 2, 8, 32 and 128 nodes respectively, and where the cuts have a full support, and we will call **B2-s**, **B8-s**, **B32-s** and **B128-s** the configurations using up to 2, 8, 32 and 128 nodes respectively, and where the cuts have a reduced support as described above. To compare the strength of the resulting cuts, we compare the closed root LP gap for each instance p, defined as

$$CG_{conf,p} := 100 \cdot \left(1 - \frac{UB_p - z_{conf,p}}{UB_p - LB_p}\right), \tag{15}$$

where UB_p is the optimal value/best known bound for problem p, LB_p is the lower bound used in [12], and $z_{conf,p}$ is the LP bound obtained with algorithm $conf$. Note that instances `enigma` and `dsbmip` have a root gap of zero, and as such, the closed gap is 100 % on all configurations. Note that the definition given by (15) is the one used in [12].

Another common metric of gap is the closed relative gap CRG_p. In order to have a meaningful value on all instances, we define CRG_p as

$$\mathrm{CRG}_{conf,p} = 100 \cdot \left(1 - \frac{UB_p - z_{conf,p}}{\max\{1, |UB_p|, UB_p - LB_p\}}\right). \tag{16}$$

Table 2 Average results on MIPLIB 3.0 instances where at least one cut was added in each configuration

Configuration	T_{conf}	Average *CG*	Average *CRG*
LB	–	0.00	80.26
CPX-root-pre	1.41	17.99	85.94
CPX-root-cut	2.54	53.92	92.69
SC	–	72.36	95.42
B2-s	2.99	59.27	92.95
B2-f	13,146.04	62.93	93.54
B8-s	3.68	61.81	94.01
B8-f	20,901.01	67.39	94.41
B32-s	7.07	64.61	94.10
B32-f	13,105.19	70.52	95.01
B128-s	14.27	69.16	94.57
B128-f	5,879.10	72.74	95.19
CPX-bc	19.83	99.44	99.98

Note that this expression is always well defined, it always give values between 0 and 100 %, and is relative on the actual optimal value for the problem.

To compare the computational effort, we measure the total running time in seconds for each configuration $conf$ on each problem p as $t_{conf,p}$. To avoid problems measuring small running times, we define $\bar{t}_{conf,p} = \max\{1, t_{conf,p}\}$. With this, we define the time performance T_{conf} for configuration $conf$, as the geometric average of $\bar{t}_{conf,p}$.

In this experiment we set $\varepsilon_1 = 2^{-10}$ and $\varepsilon_2 = 2^{-12}$. If our code encountered any problem on a given instance, then it stops execution and reports the (valid) bound found thus far. This is the case on instances `arki001, misc06, pk1, pp08a, pp08aCUTS`, and `swath` where the separation process needed to deal with unbounded sub-problems during the column-generation procedure, which at this stage is not implemented. For instances `danoint, fast0507` and `blend2`, when using the identity mapping, our code could not find a globally valid cut after 2048 oracle calls, and stopped at that point. Also, on instances `cap6000, gesa2, gesa2-o, mas76, mod011, modglob, khb05250` and `rentacar` our code detected numerical instabilities and stopped normal execution before the time limit was reached. Table 2 has the average results over instances where all local-cut configurations added at least one cut. The first column has the name of the configuration; the second column has the geometric average of running time, and a $-$ for those cases where we do not have data; the third column has the average of CG_p; finally, the fourth column has the average CRG_p. In Table 2, row **CPX-root-pre** and **CPX-root-cut** represent the bound obtained by CPLEX with the settings defined in Table 1.

We see a speed-up factor between 400 and 5,500 when working on cuts with small support rather than with cuts with full support, and the trade-off in terms of gap is relatively small. The average percentage of root LP gap closed by **SC**, **B2-s**, **B8-s**, **B32-s** and **B128-s** is 72.74, 59.27, 61.81, 64.61 and 69.16 % respectively.

Tables 3 and 4 present our results for disjunctive mappings in detail; the first column has the instance name, the next two columns present the value of the the lower bound

Table 3 Floating point results in MIPLIB 3.0 instances, part I

Problem	LB	SC	B2-s		B8-s		B32-s		B128-s	
			lb	#	lb	#	lb	#	lb	#
10teams	924	924	924	0	924	0	924	0	924	0
air03	3.4016×10^5	3.4016×10^5	3.4016×10^5	0	3.4016×10^5	0	3.4016×10^5	0	3.4016×10^5	0
air04	5.6137×10^4	5.6002×10^4	5.5821×10^4	7	5.5840×10^4	11	5.5947×10^4	33	5.6116×10^4	29
air05	2.6374×10^4	2.6185×10^4	2.6055×10^4	6	2.6100×10^4	19	2.6122×10^4	18	2.6220×10^4	114
arki001	7.5807×10^6	7.5806×10^6	7.5801×10^6	1	7.5801×10^6	1	7.5801×10^6	1	7.5801×10^6	2
bell3a	8.7843×10^5	8.7293×10^5	8.7379×10^5	1	8.7403×10^5	8	8.7404×10^5	6	8.7418×10^5	4
bell5	8.9664×10^6	8.9214×10^6	8.9583×10^6	5	8.9569×10^6	3	8.9569×10^6	3	8.9646×10^6	9
blend2	7.59898	7.05612	7.30613	12	7.27803	13	7.3405	114	7.29903	141
cap6000	-2.4513×10^6	-2.4514×10^6	-2.4514×10^6	1	-2.4514×10^6	1	-2.4514×10^6	5	-2.4514×10^6	10
dano3mip	578.574	576.232	578.574	16	578.466^t	30	578.197^t	15	578.135^t	5
danoint	65.5738	62.8857	62.7343	2	62.8117	11	62.8013	47	62.8794	19
dcmulti	1.8818×10^5	1.8818×10^5	1.8741×10^5	3	1.8792×10^5	6	1.8781×10^5	18	1.8818×10^5	0
dsbmip	−305.198	−305.198	−305.198	0	−305.198	0	−305.198	0	−305.198	0
egout	568.101	568.101	568.101	0	568.101	0	568.101	0	568.101	0
enigma	0	0	0	0	0	0	0	1	0	5
fast0507	174	172.499	173	1	173	13	173	14	173	4
fiber	4.0593×10^5	4.0512×10^5	3.9364×10^5	9	3.9400×10^5	31	3.9604×10^5	126	4.0593×10^5	0
fixnet6	3.9830×10^3	3.9761×10^3	3.8775×10^3	5	3.9272×10^3	8	3.9119×10^3	21	3.9830×10^3	0
flugpl	1.2015×10^6	1.2015×10^6	1.1734×10^6	16	1.1738×10^6	9	1.1751×10^6	4	1.1757×10^6	7
gen	1.1231×10^5	1.1231×10^5	1.1231×10^5	0	1.1231×10^5	0	1.1231×10^5	0	1.1231×10^5	0
gesa2	2.5779×10^7	2.5775×10^7	2.5779×10^7	9	2.5779×10^7	6	2.5779×10^7	16	2.5779×10^7	0
gesa2-o	2.5779×10^7	2.5779×10^7	2.5779×10^7	21	2.5777×10^7	12	2.5779×10^7	50	2.5779×10^7	33

Table 3 continued

Problem	LB	SC	B2-s		B8-s		B32-s		B128-s	
			lb	#	lb	#	lb	#	lb	#
gesa3	2.7991×10^7	2.7984×10^7	2.7972×10^7	6	2.7956×10^7	3	2.7978×10^7	30	2.7991×10^7	14
gesa3_o	2.7991×10^7	2.7983×10^7	2.7961×10^7	16	2.7970×10^7	5	2.7980×10^7	40	2.7991×10^7	0
gt2	2.1166×10^4	2.1040×10^4	2.1156×10^4	3	2.1166×10^4	1	2.1166×10^4	0	2.1166×10^4	0
harp2	-7.3899×10^7	-7.4140×10^7	-7.4086×10^7	3	-7.4073×10^7	8	-7.4081×10^7	25	-7.4055×10^7	122
khb05250	1.0694×10^8	1.0694×10^8	1.0693×10^8	4	1.0693×10^8	4	1.0694×10^8	0	1.0694×10^8	0
l152lav	4.7220×10^3	4.7188×10^3	4.6720×10^3	21	4.6680×10^3	3	4.6870×10^3	133	4.6970×10^3	37
lseu	1.1200×10^3	1.1021×10^3	1.0540×10^3	3	1.0660×10^3	19	1.0810×10^3	20	1.0920×10^3	30
markshare1	0	0	0	0	0	0	0	0	0	2
markshare2	0	0	0	0	0	0	0	0	0	2
mas74	1.1801×10^4	1.0667×10^4	1.0584×10^4	9	1.0612×10^4	11	1.0619×10^4	16	1.0623×10^4	9

Values with a *t* superscript indicate instances that reached the running time limit

Table 4 Floating point results in MIPLIB 3.0 instances, part II

Problem	LB	SC	B2-s		B8-s		B32-s		B128-s	
			lb	#	lb	#	lb	#	lb	#
mas76	4.0005×10^4	3.9188×10^4	3.9020×10^4	16	3.9017×10^4	17	3.9013×10^4	23	3.9090×10^4	12
misc03	3.3600×10^3	2.6596×10^3	2.2920×10^3	6	2.3790×10^3	15	2.5990×10^3	11	3.3600×10^3	0
misc06	1.2850×10^4	1.2850×10^4	1.2850×10^4	2	1.2850×10^4	5	1.2850×10^4	0	1.2850×10^4	0
misc07	2.8100×10^3	1.6955×10^3	1.5160×10^3	7	1.6120×10^3	25	1.6140×10^3	26	1.7480×10^3	98
mitre	1.1515×10^5	1.1515×10^5	1.1515×10^5	0	1.1515×10^5	0	1.1515×10^5	0	1.1515×10^5	0
mkc	−564.549	−594.492	−592.83	1	−577.664	3	−577.707	35	−580.673	64
mod008	307	306.997	304	6	305	3	304	22	304	31
mod010	6.5480×10^3	6.5480×10^3	6.5480×10^3	0	6.5480×10^3	0	6.5480×10^3	0	6.5480×10^3	0
mod011	-5.4558×10^7	-5.6014×10^7	-5.4589×10^7	7	-5.4580×10^7	14	-5.4559×10^7	37	-5.4558×10^7	49
modglob	2.0740×10^7	2.0729×10^7	2.0740×10^7	7	2.0740×10^7	4	2.0740×10^7	10	2.0740×10^7	0
noswot	−43	−43	−43	0	−43	0	−43	0	−43	4
nw04	1.6862×10^4	1.6862×10^4	1.6862×10^4	1	1.6862×10^4	2	1.6862×10^4	0	1.6862×10^4	0
p0033	3.0890×10^3	3.0175×10^3	3.0755×10^3	1	3.0890×10^3	2	3.0890×10^3	0	3.0890×10^3	0
p0201	7.6150×10^3	7.4295×10^3	7.5190×10^3	13	7.5830×10^3	31	7.5440×10^3	39	7.6150×10^3	0
p0282	2.5841×10^5	2.5839×10^5	2.5740×10^5	32	2.5728×10^5	5	2.5631×10^5	12	2.5721×10^5	19
p0548	8.6910×10^3	8.6424×10^3	8.6910×10^3	3	8.6910×10^3	2	8.6900×10^3	6	8.6910×10^3	0
p2756	3.1240×10^3	3.1235×10^3	3.1210×10^3	5	3.1240×10^3	6	3.1240×10^3	16	3.1240×10^3	16
pk1	11	0	0	0	0	0	0	3	0	4
pp08a	7.3500×10^3	7.2133×10^3	7.2701×10^3	11	7.3010×10^3	23	7.2872×10^3	14	7.3228×10^3	22
pp08aCUTS	7.3500×10^3	7.2717×10^3	7.2663×10^3	22	7.2954×10^3	20	7.2796×10^3	33	7.2831×10^3	20
qiu	−132.873	−312.552	−807.47	14	−698.341	30	−697.017	54	−667.257	27
qnet1	1.6029×10^4	1.6029×10^4	1.6014×10^4	6	1.6029×10^4	6	1.6029×10^4	3	1.6029×10^4	0

Table 4 continued

Problem	LB	SC	B2-s		B8-s		B32-s		B128-s	
			lb	#	lb	#	lb	#	lb	#
qnet1_o	1.6029×10^4	1.6029×10^4	1.5993×10^4	4	1.6029×10^4	9	1.6029×10^4	0	1.6029×10^4	0
rentacar	3.0356×10^7	2.8806×10^7	2.9770×10^7	3	3.0321×10^7	10	3.0356×10^7	0	3.0356×10^7	0
rgn	82.2	82.2	82.2	0	82.2	0	82.2	0	82.2	0
rout	1.0775×10^3	1.0495×10^3	994.133	11	995.244	40	998.96	83	1.0043×10^3	199
set1ch	5.4537×10^4	∞	5.4535×10^4	8	5.4531×10^4	5	5.4537×10^4	13	5.4537×10^4	36
seymour	420.225	415.63	414	3	414	5	414	11	414	12
stein27	18	13	13	1	15	18	15	61	15	9
stein45	30	22	22	0	23	30	23	75	23	35
swath	426.022	372.39	380.922	2	381.766	15	382.662	20	382.215	249
vpm1	20	20	20	0	20	0	20	0	20	0
vpm2	13.75	13.0183	13.25	2	13.25	10	13.25	52	13.5	67

and the root LP relaxation for the split closure as reported by Balas and Saxena [12]; finally, the next four pairs of columns present the root LP value and the number of added cuts for **B2-s**, **B8-s**, **B32-s** and **B128-s** configurations.

Note however that this comparison is not always fair; while the split closure is constrained to work on rank-1 inequalities, our procedure iterates the process and also uses the full machinery available in the CPLEX branch-and-bound implementation. On the other hand, the split-closure procedure did not have a precise time-limit. Moreover, most instances in MIPLIB 3.0 are very small; this can be seen in that for 14 instances a single branch is able to prove optimality, and 26 instances can be solved in less than 128 branch-and-bound nodes. So, these results must be taken with a grain of salt. For this reason, we take these experiments a step further in the next section.

6.2 Full MIPLIB experiments

For these experiments, we use the full MIPLIB 3.0 [13], MIPLIB 2003 [3] and MIPLIB 2010 [31] collections as our test bed; a total of 488 instances. We compare five configurations:

CPX-root-pre: LP bounds obtained by CPLEX as defined in Table 1.
CPX-root-cut: LP bounds obtained by CPLEX as defined in Table 1.
CPX-bc: final bounds obtained by CPLEX as defined in Table 1.
B128-s: LP bounds obtained with our disjunctive mapping experiments using branch-and-bound trees with up to 128 nodes, $\varepsilon_1 = 2^{-3}$, $\varepsilon_2 = 2^{-10}$, and with cuts with a sparse support; CPLEX options as defined in Table 1.
B512-s: LP bounds obtained with our disjunctive mapping experiments using branch-and-bound trees with up to 512 nodes, $\varepsilon_1 = 2^{-3}$, $\varepsilon_2 = 2^{-10}$, and with cuts with a sparse support; CPLEX options as defined in Table 1.

All runs have a memory limit of 4Gb and a total running time limit of 8 hours, all used a single thread during execution, and the separation step in the local-cuts procedure was limited to up to 8192 oracle calls. The online supplement to this paper contains the results for all 488 instances.

Instances `ns2122603`, `neos-1112787`, and `transportmoment` from the unstable set of instances of MIPLIB 2010 produced conflicting bounds and their results where not considered in our averages. Also, in instances `hawaiiv10-130, in, ivu06-big, ns1663818, rmine21, rmine25, splan1, zib01`, and `zib02` from the extra large data set of MIPLIB2010 and instance `bab3` from the challenge data set of MIPLIB2010, all configurations failed to report a single valid bound.

To compute the *average* performance, we consider instances where all configurations could produce at least one valid bound, and where both configurations **B128-s** and **B512-s** added at least one cut. This leaves us with 295 instances and Table 5 presents the average result of each configuration on this set of instances.

Figure 5 shows the percentage of closed relative gap (in increasing order) on instances where all five configurations produced at least one valid bound. The horizontal axis gives the percentage of test instances and the vertical axis gives the closed

Table 5 Average results in instances where both **B128-s** and **B512-s** generated at least one cut

Configuration	T_{conf}	Average *CG*	Average *CRG*
CPX-root-pre	3.88	5.24	66.97
CPX-root-cut	18.63	30.81	74.49
B128-s	850.30	38.02	76.74
B512-s	2239.00	40.02	77.55
CPX-bc	3915.00	70.79	86.53

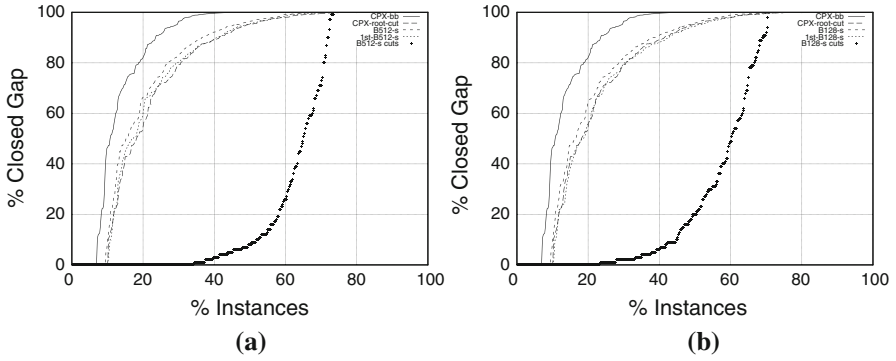

Fig. 5 Relative closed gap with floating-point experiments on full MIPLIB test-set. **a** Performance Profile for B512-s. **b** Performance Profile for B128-s

relative gap; here, a point (p, q) in the graph means that the worst p percent of the instances closed less than q percent of the relative gap.

It is not surprising to see that our procedure gets better bounds than aggressive cuts at the root node, or that by iterating the branching procedure we improve over the initial partial branch-and-bound run. What is interesting is that we could summarize part of the branch-and-bound information with cuts having a sparse support, which in turn allows the procedure to run in a reasonable amount of time.

Unfortunately, one issue keeps showing up in these floating-point tests; namely, how accurate or safe are the cuts that we generate? This is not a minor issue; as we declared above, there were some instances where our implementation stopped execution due to numerical problems. Even in the results provided by Balas and Saxena, although all closure LP's give correct bounds when solved with commercial solvers, there are two instances that are reported as infeasible by exact LP solvers, that is, the cutting procedure was so aggressive as to make the LP infeasible when solved in exact rational arithmetic.

6.3 Rational arithmetic tests

In these tests our procedure is fully rational, and, as such, there are no rounding errors or problems caused by cutting off feasible solutions. However, this advantage comes at a price: our computations in rational arithmetic can be very time consuming. For this reason we use as test bed a set of small instances obtained from the MIPLIB

3.0 [13], MIBLIB 2003 [3], and the Mittleman [35] collections. From this full set, we removed any instance that could not be solved in under five minutes with CPLEX 10 [30]. Furthermore, to restrict to instances where cutting is needed to obtain a good relaxation, we also eliminated the instances having an integrality gap of less than 1 %, that is, the difference between the MIP optimal value and the optimal value of the original LP relaxation is less than 1 % of the MIP value. The resulting test set consists of 36 instances for the tests performed in rational arithmetic. For these experiments we implemented simple mappings as described in Sect. 4.

Recall that simple mappings select a small number k of integer variables and relax the integrality conditions for all other x_i, $i \in I$. To carry out the local-cuts process, we need to create an optimization oracle for these relaxations. For this purpose, we implemented an exact rational branch-and-cut MIP code on top of an exact LP solver. The implementation includes a form of pseudo-cost branching as discussed in [1,2,33], K-cuts as explained in [22], GMI cuts derived from the aggregation of two tableau rows, and super additive lifted cover inequalities as in both [8,29].

To test the effectiveness of the local cuts, we first computed default LP relaxations by repeatedly applying the cutting-plane routines listed above. A great advantage of working in rational arithmetic is that we can run these routines until no violated cuts are produced, rather than terminating them after several rounds in an effort to limit numerical difficulties. Each round of cutting works by sequentially looking for lifted cover inequalities, then K-cuts, and then Gomory cuts derived from the aggregation of two tableau rows. If one of the procedures is successful, then we add up to 500 of the cuts, re-solve the resulting LP, and start the process anew.

When testing local cuts, we also use our default cutting-plane-generation methods, that is, the local cuts are generated after default cuts have failed. If acceptable local cuts are found, our default routines are first tried again during the next round of cutting, until none of the cutting-plane schemes can produce acceptable cuts.

To select the k variables in the simple mappings, we order the integer variables x_i according to the distance of x_i^* to the nearest integer value, where x^* is the current optimal LP solution. The first mapping uses the k most fractional variables, and each subsequent mapping is obtained by randomly choosing one of the k variables and replacing it by a random choice among the remaining fractional variables.

We tested the following three configurations.

LO : Our default set of cutting planes.
LS_4 : Local cuts from simple mappings with $k = 4$ integer variables.
LS_6 : Local cuts from simple mappings with $k = 6$ integer variables.

For each configuration and all 36 test instances, we computed LP relaxations (no branching allowed) using up to 25 hours of computing time for each instance and recording the final LP bound. If the configuration did run to termination, then we recorded the LP value found at the end of the 25-hour period. These tests were carried out on computer nodes equipped with a 2.33 GHz Intel Xeon E5345 processor and 4Gb of random-access memory.

The full collection of results is presented in Table 6; for comparison we also include CPLEX-10 (floating-point) LP bounds obtained by restricting CPLEX's MIP solver to the same class of cutting planes as those implemented in our rational solver. The

Table 6 Rational results and comparison with CPLEX-10

Problem	LP bound				Running time		
	LS_6	LS_4	L0	CPX	LS_6	LS_4	L0
aflow30a	1026.54	1027.23	1026.54	1086.35	25.0 h	25.0 h	4.3 min
bc1	0.787	0.787	0.783	0.7854	8.1 h	8.4 h	1.1 min
blend2	7.083	7.086	7.052	7.0397	25.0 h	25.0 h	7.1 s
fiber	378957	378957	378957	388376	25.0 h	25.0 h	2.4 min
fixnet6	2887.43	2942.13	2887.43	1968.06	25.0 h	25.0 h	1.2 min
flugpl	1.20147e+06	1.17838e+06	1.17196e+06	1.17188e+06	26.3 min	7.3 min	0.5 s
gt2	21102.3	21102.3	21102.3	20872.1	3.2 h	2.9 h	8.6 s
l152lav	4681.75	4681.86	4681.57	4656.36	25.0 h	25.0 h	12.2 min
lseu	1058.18	1057.44	1056.37	1031.82	25.0 h	25.0 h	3.5 s
mas76	38986.7	38959.2	38893.9	38986.9	1.5 h	1.0 h	1.4 s
misc03	2212.42	2215.21	2185	2135	10.9 min	6.0 min	27.6 s
misc07	1442.36	1436.52	1425	1425	33.4 min	21.1 min	43.0 s
mod008	292.463	292.463	292.463	295.058	25.0 h	25.0 h	5.3 s
neos10	−1207.76	−1207.76	−1207.76	−1183.65	25.0 h	25.0 h	25.0 h
neos1	13.381	13.381	13.381	5.6	25.0 h	25.0 h	25.0 h
neos20	−474.929	−474.929	−474.927	−474.997	25.0 h	25.0 h	48.9 min
neos21	2.979	2.764	2.44	2.7498	2.4 h	1.3 h	2.2 min
neos7	488493	488493	488493	580658	2.2 h	6.3 h	5.9 min
p0033	3084.91	3074.31	2994.08	3007.49	12.7 min	59.0 min	1.4 s
p0201	7339.35	7335.06	7334.62	7182.57	25.0 h	25.0 h	58.8 s
p0282	252445	252415	252408	255248	25.0 h	25.0 h	9.7 s
pk1	0	0	0	0	21.6 h	18.2 h	4.6 s
pp08a	6409.54	6409.54	6409.54	7104.06	8.0 min	7.8 min	34.7 s
pp08aCUTS	6730.15	6730.15	6730.15	6971.47	1.1 h	44.8 min	1.3 min
qiu	−894.038	−894.038	−894.038	−893.308	6.7 h	25.0 h	1.0 h
qnet1	14601.3	14601.3	14601.3	15548.6	25.0 h	25.0 h	9.5 min
qnet1_o	14135.1	14135.1	14134.1	15523.1	25.0 h	25.0 h	20.4 min
rentacar	2.93286e+07	2.93286e+07	2.93821e+07	2.90153e+07	25.0 h	25.0 h	4.3 min
rgn	70.02	62.235	60.54	76.9636	25.0 h	25.0 h	5.5 s
rout	994.872	992.107	984.47	982.173	25.0 h	24.2 h	1.9 min
set1ch	50338.7	50338.7	50342.7	49667.5	25.0 h	25.0 h	3.2 min
stein27	13.578	13	13	13	6.3 h	9.4 h	3.9 s
stein45	22	22	22	22	1.5 h	1.3 h	10.4 s
swath1	340.766	340.766	340.766	340.775	25.0 h	25.0 h	5.3 min
swath2	341.242	341.242	341.242	345.042	25.0 h	25.0 h	11.1 min
vpm2	11.615	11.447	11.362	12.7515	24.2 h	25.0 h	36.6 s

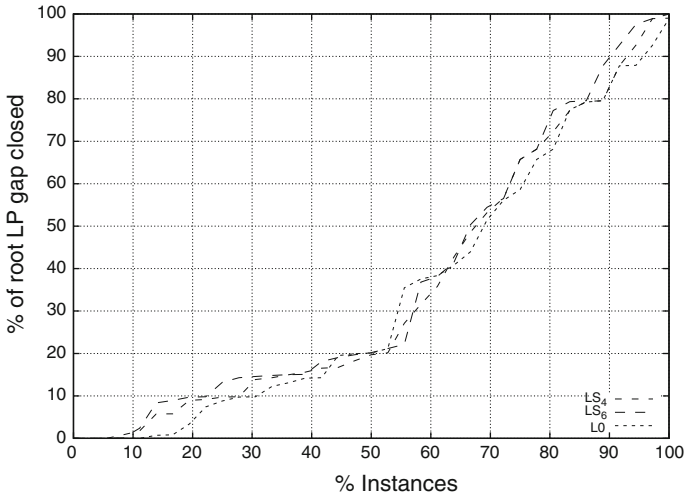

Fig. 6 Closed relative gap on rational arithmetic tests

CPLEX bounds could be expected to be slightly worse than the L0 bounds, since CPLEX is likely limiting the number of rounds of cutting planes (due to the accumulation of floating-point error) to a greater extent than we have limited the L0 computations. However, CPLEX performed significantly better than the L0 configuration in a number of instances. The reason for this appears to be due to the poor numerical properties of some of the test instances after adding Gomory cuts, where most cuts found in the rational solver are rejected by the 320-bit limit for expressing rational numbers in cuts. For example, increasing the limit to 3,200 bits raised the L0 bound for `mod008` to 296.9, before the run exceeded the available 4Gb of memory. However, trying to run our code with this extended precision in all instances was beyond our memory limit.

To compare results from the L0, LS_4, and LS_6 configurations, we plotted, in Fig. 6, the final closed relative gap for each configuration on our 36 problems. In Fig. 6, the horizontal axis gives the percentage of test instances and the vertical axis gives the closed relative gap; here, a point (p, q) in the graph means that the worst p percent of the instances closed less than q percent of the relative gap.

Although the three plots do not differ widely, in average L0 finished with 33.54 % of closed relative gap, LS_4 finished with 34.69 % of closed relative gap, while LS_6 finished with 36.08 % of closed relative gap; showing a small advantage for LS_4 and LS_6 over L0. However, these tests are far from conclusive. Still, the floating-point results seem promising, and there are many open possibilities to explore. We hope that the reported study serves as a starting point in this direction.

Acknowledgments We would like to thank the anonymous referees for their comments and suggestions, which greatly improved the readability and quality of this manuscript. We would also like to give special thanks to Egon Balas and Anureet Saxena for making available their detailed results and actual split cuts for all MIPLIB 3.0 instances, as presented in [12].

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**, 1–41 (2009)
2. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**, 42–54 (2005)
3. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. **34**, 1–12 (2006)
4. Althaus, E., Polzin, T., Daneshmand, S.V.: Improving linear programming approaches for the Steiner tree problem. In: Jansen, K., Margraf, M., Mastrolilli, M., Rolim, J.D.P. (eds.) Experimental and Efficient Algorithms, Second International Workshop, WEA 2003, pp. 1–14. Springer, Berlin (2003)
5. Applegate, D., Bixby, R.E., Chvátal, V., Cook, W.: TSP cuts which do not conform to the template paradigm. In: Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School], pp. 261–304. Springer, London (2001)
6. Applegate, D., Bixby, R.E., Chvátal, V., Cook, W.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2006)
7. Applegate, D., Cook, W., Dash, S., Espinoza, D.: Exact solutions to linear programming problems. Oper. Res. Lett. **35**, 693–699 (2007)
8. Atamtürk, A.: Sequence independent lifting for mixed-integer programming. Oper. Res. **52**, 487–490 (2004)
9. Balas, E.: Disjunctive programming: Properties of the convex hull of feasible points. Discr. Appl. Math. **89**, 3–44 (1998)
10. Balas, E., Ceria, S., Cornuéjols, G.: A lift-and-project cutting plane algorithm for mixed 0–1 programs. Math. Program. **58**, 295–324 (1993)
11. Balas, E., Ceria, S., Cornuéjols, G., Natraj, N.: Gomory cuts revisited. Oper. Res. Lett. **19**, 1–9 (1996)
12. Balas, E., Saxena, A.: Optimizing over the split closure. Math. Program. **113**, 219–240 (2008)
13. Bixby, R.E., Boyd, E.A., Indovina, R.R.: MIPLIB: a test set of mixed integer programming problems. SIAM News **25**, 16 (1992)
14. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: Mixed-integer programming: a progress report. In: Grötschel, M. (ed.) The Sharpest Cut: The Impact of Manfred Padberg and His Work, pp. 309–325. SIAM, Philadelphia (2004)
15. Boyd, E.A.: Generating Fenchel cutting planes for knapsack polyhedra. SIAM J. Optim. **3**, 734–750 (1993)
16. Boyd, E.A.: Fenchel cutting planes for integer programs. Oper. Res. **42**, 53–64 (1994)
17. Buchheim, C., Liers, F., Oswald, M.: Local cuts revisited. Oper. Res. Lett. **36**, 430–433 (2008)
18. Buchheim, C., Liers, F., Oswald, M.: Speeding up IP-based algorithms for constrained quadratic 0–1 optimization. Tech. Rep. zaik2008-578, Zentrum für Angewandte Informatik Köln, Germany (2008)
19. Caprara, A., Fischetti, M.: 0,1/2-Chvátal-Gomory cuts. Math. Program. **74**, 221–235 (1996)
20. Cook, W., Dash, S., Fukasawa, R., Goycoolea, M.: Numerically safe gomory mixed-integer cuts. INFORMS J. Comput. **21**, 641–649 (2009)
21. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: An exact rational mixed-integer programming solver. In: Integer Programming and Combinatorial Optimization, 15th International IPCO Conference, IBM T. J. Watson Research Center, Yorktown Heights, New York, NY, USA, Lecture Notes in Computer Science, pp. 104–116. Springer, Berlin (2011)
22. Cornuéjols, G., Li, Y., Vandenbussche, D.: K-cuts: a variation of Gomory mixed integer cuts from the LP tableau. INFORMS J. Comput. **15**, 385–396 (2003)
23. Crowder, H.P., Johnson, E.L., Padberg, M.W.: Solving large-scale zero-one linear programming problems. Oper. Res. **31**, 803–834 (1983)
24. Espinoza, D., Fukasawa, R., Goycoolea, M.: Lifting, tilting and fractional programming revisited. Oper. Res. Lett. 559–563 (2010)
25. Espinoza, D.G.: On Linear Programming, Integer Programming and Cutting Planes, PhD thesis, School of Industrial and Systems Engineering. Georgia Institute of Technology (2006)
26. Gomory, R.E.: An algorithm for the mixed integer problem. Tech. Rep. RM-2597, RAND Corporation (1960)
27. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd edn. Springer, Berlin (1993)
28. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Lifted cover inequalities for 0–1 integer programs. Math. Program. **85**, 437–467 (1999)
29. Gu, Z., Nemhauser, G.L., Savelsbergh, M.W.P.: Sequence independent lifting in mixed integer programming. J. Combinator. Optim. **4**, 109–129 (2000)

30. ILOG: User's Manual, ILOG CPLEX 10.0, ILOG CPLEX Division, Incline Village, Nevada (2006)
31. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., Wolter, K.: MIPLIB 2010. Math. Program. Comput. **3**(2011), 103–163 (2011). doi:10.1007/s12532-011-0025-9
32. Lenstra H.W Jr.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**, 538–548 (1983)
33. Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of search strategies for mixed integer programming. INFORMS J. Comput. **11**, 173–187 (1999)
34. Marchand, H., Wolsey, L.A.: Aggregation and mixed integer rounding to solve MIPs. Oper. Res. **49**, 363–371 (2003)
35. Mittelmann, H.: Mixed integer linear programming benchmark (free codes). http://plato.asu.edu/ftp/milpf.html (2008)
36. Perregaard, M., Balas, E.: Generating cuts from multiple-term disjunctions. In: IntegerProgramming and Combinatorial Optimization, 8th International IPCO Conference, Utrecht, The Netherlands, pp. 348–360. Springer, Berlin (2001)
37. Zanette, A., Fischetti, M., Balas, E.: Lexicography and degeneracy: can a pure cutting plane algorithm work? Math. Program. 1–24 (2009). doi:10.1007/s10107-009-0335-0