Cutting to the Chase
Solving Linear Integer Arithmetic

Dejan Jovanović[1] and Leonardo de Moura[2]

[1] New York University
[2] Microsoft Research

Abstract. We describe a new algorithm for solving linear integer programming problems. The algorithm performs a DPLL style search for a feasible assignment, while using a novel cut procedure to guide the search away from the conflicting states.

1 Introduction

One of the most impressive success stories of computer science in industrial applications was the advent of linear programming algorithms. Linear programming (LP) became feasible with the introduction of Dantzig simplex algorithm. Although the original simplex algorithm targets problems over the rational numbers, in 1958 Gomory [12] introduced an elegant extension to the integer case (ILP). He noticed that, whenever the simplex algorithm encounters a non-integer solution, one can eliminate this solution by deriving a plane, that is implied by the original problem, but does not satisfy the current assignment. Incrementally adding these *cutting planes*, until an integer solution is found, yields an algorithm for solving linear programs over the integers. Cutting planes have been studied thoroughly both as an abstract proof system [4], and as a practical preprocessing step for hard structured problems. For such problems, one can exploit the structure by adding cuts tailored for the problem, such as the clique cuts, or the covering cuts [20], which can reduce the search space dramatically.

The main idea behind the algorithm of Gomory, i.e to combine a model searching procedure, with a conflict resolution procedure – a procedure that can derive new facts in order to eliminate a conflicting candidate solution – is in fact quite general. Somewhat later, for example, in the field of automated reasoning, there was a similar development with equally impressive end results. Solvers for the Boolean satisfiability problem (SAT), although a canonical NP-complete problem, have seen a steady improvement over the years, culminating in thrilling advances in the last 10 years. It has become a matter of routine to use a SAT solver on problems with millions of variables and constraints. Of course, there are many ingredients that make a modern SAT solver efficient, but one of the most appealing ones, is the combination of two different approaches to solving a problem. One is a backtracking search for a satisfying assignment, in the style of DPLL [7]. The other, is a search for a resolution refutation of the problem, as in the DP algorithm [8]. To combine these two [19] first noticed that, once a conflict has been encountered, we can derive a clause that explains the conflict, i.e. the *search is guiding the resolution.* As with the Gomory cuts, the explanation clause eliminates the current assignment, forcing a backtrack, and eliminating an (often substantial) part of the search tree. In the other direction, [17] introduced the so called VSIDS heuristic that adjusts the variable selection heuristic so that it prefers the variables involved in the resolution of conflicts, i.e. *the resolution is guiding the search.* This approach to solving SAT problems is commonly called Conflict-Directed Clause Learning (CDCL) and is employed by most modern SAT solvers. Apart from CDCL, there are many other important techniques that have become standard such as fast restarts[17], and fast indexing schemes for unit propagation [17].

In this paper, we propose a new CDCL-like procedure for solving arbitrary ILP problems. Our procedure, inspired by recent algorithms for solving linear real arithmetic [16, 14, 6], has all the important theoretical and practical ingredients that have made CDCL based SAT solvers so successful, including: model search complemented with the generation of resolvents explaining the conflicts; propagation rules enabling reduction of the search space and early detection of conflicts; resolvents learned during analysis of conflicts enable non-chronological backtracking; all resolvents generated during the search are valid, i.e. implied by the input formula, and not conditioned by any decisions; decisions (case-splits) are not based on a fixed variable order, thus enabling dynamic reordering heuristics; and cutting-plane inequalities (resolvents) learned during the search can be removed, allowing for flexible memory management by keeping the constraint database limited.

Another contribution of our paper is a that our procedure *guarantees termination.* We describe two arguments that imply termination. First, we propose a simple heuristic for deciding when a cutting-planes based approach does not terminate, recognizing variables contributing to the divergence. Then, we show that, in such a case, one can isolate a finite number of small cores that are inconsistent with the corresponding current partial models. These cores comprise of two inequalities and at most one divisibility constraint. Finally, we apply Cooper's quantifier elimination procedure to derive a resolvent that will *block* a particular core from ever happening again, which in turn implies termination. And, as a matter of practical importance, our resolvents do not involve disjunctions and are expressed only with valid inequalities and divisibility constraints.

2 Preliminaries

As usual, we denote the set of integers as $\mathbb{Z}$. We assume a finite set of variables X ranging over $\mathbb{Z}$. We use x, y, z, k to denote variables in X, a, b, c, d to denote coefficients in $\mathbb{Z}$, and p, q, r and s for linear polynomials over X with coefficients in $\mathbb{Z}$. In the work that follows, all polynomials are assumed to be in sum-of-monomials normal form $a_1x_1 + \cdots + a_nx_n + c$. Given a polynomial $p = a_1x_1 + \ldots + a_nx_n + c$, and a coefficient b, we use bp to denote the polynomial $(a_1b)x_1 + \ldots + (a_nb)x_n + (bc)$.

Inequalities. We use I and J to denote inequalities $a_n x_n + \cdots + a_1 x_1 + c \leq 0$. We rewrite $p < 0$ as $p + 1 \leq 0$, and $p = 0$ as $p \leq 0 \wedge -p \leq 0$. We use $\mathsf{coeff}(p, x)$ ($\mathsf{coeff}(I, x)$) to denote the coefficient of x in the linear polynomial p (inequality I), where $\mathsf{coeff}(p, x) = 0$ if x does not occur in p (I). We say an inequality I is *tightly-propagating* for a variable x if $\mathsf{coeff}(I, x) \in \{-1, 1\}$.

Divisibility Constraints. In addition to inequalities, we also consider divisibility constraints of the form $d \mid a_1x_1 + \cdots + a_nx_n + c$, where d is a non-zero integer constant. We denote divisibility constraints with the (possibly subscripted) letter D.

Finally, given a set of constraints C and a constraint I, we use $C \vdash_{\mathbb{Z}} I$ to denote that I is implied by C in the theory of linear integer arithmetic.

3 The Abstract Search Procedure

We describe our procedure as an abstract transition system in the spirit of Abstract DPLL [18, 15]. The states are pairs of the form $\langle M, C \rangle$, where M is a sequence of *bound refinements*, and C is a set of constraints. We use $[\![]\!]$ to denote the empty sequence. In this section we assume that all constraints in C are inequalities. Bound refinements can be either *decisions* or *implied bounds*. Decided lower and upper bounds are decisions we make during the search, and we represent them in M as $x \geq b$ and $x \leq b$. On the other hand, lower and upper bounds that are implied in the current state by an inequality I, are represented as $x \geq_I b$ and $x \leq_I b$. We say a sequence M is *non-redundant* if, for all variables x, the bound refinements in M are monotone, i.e. all the lower (upper) bounds are increasing (decreasing), and M does not contain the same bound for x, decided or implied.

Let $\mathsf{lower}(x, M)$ and $\mathsf{upper}(x, M)$ denote the best, either decided or implied, lower and upper bounds for x in M, where we assume the usual values of $-\infty$ and ∞, when the corresponding bounds do not exist. A sequence M is *consistent* if there is no x such that $\mathsf{lower}(x, M) > \mathsf{upper}(x, M)$. We lift the best lower and upper bound functions to linear polynomials using identities such as: $\mathsf{lower}(p+q, M)$ is $\mathsf{lower}(p, M)+\mathsf{lower}(q, M)$ when variables in p and q are disjoint[3], $\mathsf{lower}(b, M) = b$, and $\mathsf{lower}(ax, M)$ is $a(\mathsf{lower}(x, M))$ if $a > 0$, and $a(\mathsf{upper}(x, M))$ otherwise.

Definition 1. *We say a sequence M is* well-formed (wf) *with respect to a set of constraints C when M is non-redundant, consistent and M is either an empty sequence or it starts with a wf prefix M', i.e. $M = [\![M', \gamma]\!]$, where the bound refinement γ is either*

- *$x \geq_I b$, with $I \equiv (-x + q \leq 0)$, $C \vdash_{\mathbb{Z}} I$, and $b \leq \mathsf{lower}(q, M')$; or*

[3] In general, when estimating bounds of polynomials, for a consistent sequence M it holds that, if $\mathsf{lower}(p, M)$ and $\mathsf{lower}(q, M)$ are defined, then $\mathsf{lower}(p + q, M) \geq \mathsf{lower}(p, M) + \mathsf{lower}(q, M)$.

– *$x \leq_I b$, with $I \equiv (x - q \leq 0)$, $\mathcal{C} \vdash_{\mathbb{Z}} I$, and $b \geq$ upper(q, M'); or*
– *$x \geq b$, where M' contains $x \leq_I b$ for some I; or*
– *$x \leq b$, where M' contains $x \geq_I b$ for some I.*

Intuitively, in a well-formed sequence, every decision $x \geq b$ ($x \leq b$) amounts to *deciding* a value for x that is equal to the best upper (lower) bound. We say that a state $\langle M, C\rangle$ is well-formed if M is well-formed with respect to C. Note that, when refining a bound, we allow a bound b that is *not necessarily the most precise one* with respect to I. Although going against intuition, the reason for this flexibility will become apparent later.

Given an implied lower (upper) bound refinement $x \geq_I b$ ($x \leq_I b$) and an inequality $ax + p \leq 0$, the function resolve combines (if possible) the tight inequality $I \equiv \pm x + q \leq 0$ with $ax + p \leq 0$. If the combination is not applicable, resolve just returns $p \leq 0$. It is defined as

$$\begin{array}{l}\mathsf{resolve}(x \geq_I b, ax + p \leq 0)\\ \mathsf{resolve}(x \leq_I b, ax + p \leq 0)\end{array} = \begin{cases} |a|q + p \leq 0 & \text{if } a \times \mathsf{coeff}(I, x) < 0\ , \\ ax + p \leq 0 & \text{otherwise}\ . \end{cases}$$

We also define the function bound(I, x, M) that, given an inequality I and a sequence M returns the bound that I implies on x, with respect to M, i.e

$$\mathsf{bound}(ax + p \leq 0, x, M) = \begin{cases} -\left\lceil \frac{\mathsf{lower}(p,M)}{a} \right\rceil & \text{if } a > 0\ , \\ -\left\lfloor \frac{\mathsf{lower}(p,M)}{a} \right\rfloor & \text{if } a < 0\ . \end{cases}$$

Lemma 1. *Given[4] a well-formed state $\langle M, C\rangle$, with $M = [\![M', \gamma]\!]$, such that γ is an implied bound, $p \leq 0$ an inequality, and $q \leq 0 \equiv$ resolve$(\gamma, p \leq 0)$ then*

$$\begin{array}{c} C \vdash_{\mathbb{Z}} (p \leq 0) \quad \textit{implies} \quad C \vdash_{\mathbb{Z}} (q \leq 0)\ , \\ \mathsf{lower}(q, M') \geq \mathsf{lower}(p, M)\ . \end{array}$$

Example 1. In the statement of Lemma 1, we only get to keep lower$(q, M') \geq$ lower(p, M) because all of the implied bounds were justified by tightly-propagating inequalities. If we would allow non-tight justifications, this might not hold. Consider, for example, a state $\langle M, C\rangle$ where

$$C = \{\overbrace{-x \leq 0}^{I},\ \overbrace{-3y + x + 2 \leq 0}^{J}\}\ , \qquad M = [\![x \geq_I 0,\ y \geq_J 1]\!]\ ,$$

and the inequality $1 + 6y \leq 0$. Then, we have that

$$\mathsf{lower}(1 + 6y, M) = 7 \text{ and } \mathsf{resolve}(y \geq_J 1,\ 1 + 6y \leq 0) = 2x + 5 \leq 0\ .$$

So, after performing resolution on y using a non-tight inequality J, the inequality became weaker since i.e lower$(2x + 5, [\![x \geq_I 0]\!]) = 5 \not\geq 7$.

[4] The proofs of all lemmas and theorems are included in a separate technical report.

The predicate $\mathsf{improves}(I, x, M)$ is true if the inequality $I \equiv ax + p \leq 0$ implies a better bound for x in M, but does not make M inconsistent. It is defined as

$$\mathsf{improves}(I, x, M) = \begin{cases} \mathsf{lower}(x, M) < \mathsf{bound}(I, x, M) \leq \mathsf{upper}(x, M), & \text{if } a < 0, \\ \mathsf{lower}(x, M) \leq \mathsf{bound}(I, x, M) < \mathsf{upper}(x, M), & \text{if } a > 0, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

3.1 Deriving tight inequalities

Since we require that all the implied bound refinements in M are justified by tightly propagating inequalities, we now show, given an inequality $\pm ax + p \leq 0$ such that $\mathsf{improves}(\pm ax + p \leq 0, x, M)$ holds, how to deduce a tightly propagating inequality that can justify the bound implied by $\pm ax + p \leq 0$.

The deduction is described using an auxiliary transition system. The states of this system are tuples of the form

$$\langle M', \pm ax + as \oplus r \rangle \ ,$$

where $a > 0$, s and r are polynomials, M' is a prefix of the initial M, and we keep the invariant that

$$C \vdash_{\mathbb{Z}} \pm ax + as + r \leq 0, \qquad \mathsf{lower}(as + r, M) \geq \mathsf{lower}(p, M) \ .$$

The initial state for tightening $\pm ax + p \leq 0$ is $\langle M, \pm ax \oplus p \rangle$ and the transition rules are as follows.

Consume
$$\langle M, \pm ax + as \oplus aky + r \rangle \quad \Longrightarrow \quad \langle M, \pm ax + as + aky \oplus r \rangle$$
where $x \neq y$.

Resolve-Implied
$$\langle [\![M, \gamma]\!], \pm ax + as \oplus p \rangle \quad \Longrightarrow \quad \langle M, \pm ax + as \oplus q \rangle$$
where γ is an implied bound and $q \leq 0 \equiv \mathsf{resolve}(\gamma, p \leq 0)$

Decided-Lower
$$\langle [\![M, y \geq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as + aky \oplus r + (ak - c)q \rangle$$
where $y \leq_I b$ in M, with $I \equiv y + q \leq 0$, and $k = \lceil c/a \rceil$.

Decided-Lower-Neg
$$\langle [\![M, y \geq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as \oplus cq + r \rangle$$
where $y \leq_I b$ in M, with $I \equiv y - q \leq 0$, and $c < 0$.

Decided-Upper
$$\langle [\![M, y \leq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as + aky \oplus r + (c - ak)q \rangle$$
where $y \geq_I b$ in M, with $I \equiv -y + q \leq 0$, and $k = \lfloor c/a \rfloor$.

Decided-Upper-Pos
$$\langle [\![M, y \leq b]\!], \pm ax + as \oplus cy + r \rangle \implies \langle M, \pm ax + as \oplus cq + r \rangle$$
where $y \geq_I b$ in M, with $I \equiv -y + q \leq 0$, and $c > 0$.

Round (and terminate)
$$\langle M, \pm ax + as \oplus b \rangle \quad \Longrightarrow \quad \pm x + s + \lceil b/a \rceil \leq 0$$

We use $\mathsf{tight}(I, x, M)$ to denote the tightly propagating inequalities derived using some strategy for applying the transition rules above.

Example 2. Given a well-formed state $\langle M_4, C\rangle$, where

$$C = \{\underbrace{-y \le 0}_{I_1}, \underbrace{-x + 2 \le 0}_{I_2}, \underbrace{-y + 7 + x \le 0}_{I_3}, \underbrace{-3z + 2y - 5x \le 0}_{I_4}\}$$
$$M_4 = [\![\ y \ge_{I_1} 0,\ \ x \ge_{I_2} 2,\ \ y \ge_{I_3} 9,\ \ x \le 2\ ]\!]$$

We denote with M_1, M_2, M_3 the prefixes of M_4. In M_4, we have that $\mathsf{bound}(I_4, z, M_4) = 3$, that is, I_4 is implying a lower bound of z in the current state. We now derive a tight inequality that justifies this lower bound.

$\langle M_4, -3z \oplus 2y - 5x\rangle$
$\Longrightarrow$ Decided-Upper-Pos
  $x \le 2$ is a decided bound, M contains implied bound $x \ge_{I_2} 2$.
  We make the coefficient of x divisible by 3 by adding $-x + 2 \le 0$.
$\langle M_3, -3z - 6x \oplus 2y + 2\rangle$
$\Longrightarrow$ Resolve-Implied
  We eliminate y by adding two times $-y + 7 + x \le 0$.
$\langle M_2, -3z - 6x \oplus 2x + 16\rangle$
$\Longrightarrow$ Resolve-Implied
  We eliminate x in $2x + 16$ by adding two times $-x + 2 \le 0$.
$\langle M_1, -3z - 6x \oplus 20\rangle$
$\Longrightarrow$ Round
$-z - 2x + 7 \le 0$

The tightly propagating inequality $-z-2x+7 \le 0$ implies the same lower bound $\mathsf{bound}(-z - 2x + 7 \le 0, z, M) = 3$ for z.

Lemma 2. *Given a well-formed state $\langle M, C\rangle$ and an implied inequality I, i.e. such that $\langle M, C\rangle \vdash_{\mathbb{Z}} I$, and $\mathsf{improves}(I, x, M)$ the procedure for deriving tightly-propagating inequalities terminates with a tight-inequality J such that $\langle M, C\rangle \vdash_{\mathbb{Z}} J$ and*

- *if I improves the lower bound on x, then $\mathsf{bound}(I, x, M) \le \mathsf{bound}(J, x, M)$,*
- *if I improves the upper bound on x, then $\mathsf{bound}(I, x, M) \ge \mathsf{bound}(J, x, M)$.*

Note that in the statement above, it is does not necessarily hold that $\mathsf{improves}(J, x, M)$, as the $\mathsf{improves}$ predicate requires the new bound to be consistent, and the derived inequality might in fact imply a stronger bound.

3.2 Main procedure

We are now ready to define our main transition system: Cutting to the Chase. In the following system of rules, if a propagation rule can derive a new implied bound $x \ge_I b$ or $x \le_I b$, the tightly propagating inequality I is computed eagerly. This simplification clarifies the presentation but, due to the allowance of Definition 1, we can use them as just placeholders and compute them on demand, which is what we do in our implementation.

Decide

$$\langle M, C\rangle \Longrightarrow \langle [\![M, x \geq b]\!], C\rangle \quad \textbf{if } \mathsf{lower}(x, M) < b = \mathsf{upper}(x, M)$$
$$\langle M, C\rangle \Longrightarrow \langle [\![M, x \leq b]\!], C\rangle \quad \textbf{if } \mathsf{lower}(x, M) = b < \mathsf{upper}(x, M)$$

Propagate

$$\langle M, C \cup \{J\}\rangle \Longrightarrow \langle [\![M, x \geq_I b]\!], C \cup \{J\}\rangle \;\textbf{if}\; \begin{cases} \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

$$\langle M, C \cup \{J\}\rangle \Longrightarrow \langle [\![M, x \leq_I b]\!], C \cup \{J\}\rangle \;\textbf{if}\; \begin{cases} \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

Forget

$$\langle M, C \cup \{J\}\rangle \Longrightarrow \langle M, C\rangle \quad \textbf{if } C \vdash_{\mathbb{Z}} J, \text{ and } J \notin C$$

Conflict

$$\langle M, C\rangle \Longrightarrow \langle M, C\rangle \vdash p \leq 0 \quad \textbf{if } p \leq 0 \in C, \mathsf{lower}(p, M) > 0$$

Learn

$$\langle M, C\rangle \vdash I \Longrightarrow \langle M, C \cup I\rangle \vdash I \quad \textbf{if } I \notin C$$

Resolve

$$\langle [\![M, \gamma]\!], C\rangle \vdash I \Longrightarrow \langle M, C\rangle \vdash \mathsf{resolve}(\gamma, I) \quad \textbf{if } \gamma \text{ is an implied bound.}$$

Unsat

$$\langle [\![M, \gamma]\!], C\rangle \vdash b \leq 0 \Longrightarrow \mathsf{unsat} \quad \textbf{if } b > 0$$

Backjump

$$\langle [\![M, \gamma, M']\!], C\rangle \vdash J \Longrightarrow \langle [\![M, x \geq_I b]\!], C\rangle \quad \textbf{if}\; \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

$$\langle [\![M, \gamma, M']\!], C\rangle \vdash J \Longrightarrow \langle [\![M, x \leq_I b]\!], C\rangle \quad \textbf{if}\; \begin{cases} \gamma \text{ is a decided bound} \\ \mathsf{improves}(J, x, M), \\ I = \mathsf{tight}(J, x, M), \\ b = \mathsf{bound}(J, x, M). \end{cases}$$

Theorem 1 (Soundness). *For any derivation sequence $\langle [\![]\!], C_0\rangle \Longrightarrow S_1 \Longrightarrow \cdots \Longrightarrow S_n$, If S_n is of the form $\langle M_n, C_n\rangle$, then C_0 and C_n are equisatisfiable. If S_n is of the form $\langle M_n, C_n\rangle \vdash I$, then C_0 implies I, and C_0 and C_n are equisatisfiable. Moreover, $\langle M_n, C_n\rangle$ is well-formed.*

Example 3. Consider the set of inequalities C

$$\{\underbrace{-x \leq 0}_{I_1}, \underbrace{6x - 3y - 2 \leq 0}_{I_2}, \underbrace{-6x + 3y + 1 \leq 0}_{I_3}\}$$

Now we show C to be unsatisfiable using our abstract transition system.

$\langle [\![]\!], C\rangle$
$\Longrightarrow$ Propagate x using I_1
$\langle [\![x \geq_{I_1} 0]\!], C\rangle$
$\Longrightarrow$ Decide x
$\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], C\rangle$
$\Longrightarrow$ Propagate y using I_3
$\langle [\![x \geq_{I_1} 0,\ x \leq 0,\ y \leq_J -1]\!], C\rangle$, where $J = \mathsf{tight}(I_3, y, [\![x \geq_{I_1} 0, x \leq 0]\!])$
  $\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], 3y \oplus -6x + 1\rangle$
  $\Longrightarrow$ Consume
  $\langle [\![x \geq_{I_1} 0,\ x \leq 0]\!], 3y - 6x \oplus 1\rangle$
  $\Longrightarrow$ Round
  $J \equiv y - 2x + 1 \leq 0$
$\Longrightarrow$ Conflict using I_2
$\langle [\![x \geq_{I_1} 0,\ x \leq 0,\ y \leq_J -1]\!], C\rangle \vdash 6x - 3y - 2 \leq 0$
$\Longrightarrow$ Resolve $\mathsf{resolve}(y \leq_J -1, 6x - 3y - 2 \leq 0) = (3(-2x+1) + 6x - 2 \leq 0)$
$\langle [\![M, x \leq 0]\!], C\rangle \vdash 1 \leq 0$
$\Longrightarrow$ Unsat
unsat

Slack Introduction. Given a state $S = \langle M, C\rangle$, we say variable x is *unbounded* at S if $\mathsf{lower}(x, M) = -\infty$, $\mathsf{upper}(x, M) = \infty$. We also say x is *stuck* at S if it is unbounded and Propagate cannot be used to deduce a lower or upper bound for x. A state S is *stuck* if all unbounded variables in S are stuck, and no inequality in C is false in M. That is, there is no possible transition for a stuck state S. Before we describe how we avoid stuck states, we make the observation that for every finite set of inequalities C, there is an equisatisfiable set C' such that every variable x in C', $(-x \leq 0) \in C'$. The idea is to replace every occurrence of x in C with $x^+ - x^-$, and add the inequalities $-x^+ \leq 0$ and $-x^- \leq 0$. Instead of using this eager preprocessing step, we use a lazy approach, where *slack variables* are dynamically introduced. Suppose, we are in a stuck state $\langle M, C\rangle$, then we simply select an unbounded variable x, add a fresh *slack* variable $x_s \geq 0$, and add new inequalities to C that "bound" x in the interval $[-x_s, x_s]$. This idea is captured by the following rule:

Slack-Intro

$$\langle M, C\rangle \Longrightarrow \langle M, C \cup \{x - x_s \leq 0, -x - x_s \leq 0, -x_s \leq 0\}\rangle \ \textbf{if} \begin{cases} \langle M, C\rangle \text{ is stuck} \\ x_s \text{ is fresh} \end{cases}$$

Note that it is sound to reuse a slack variable x_s used for "bounding" x, to bound y, and we actually do that in our implementation.

3.3 Termination

We say a set of inequalities C is a *finite problem* if for every variable x in C, there are two integer constants a and b such that $\{x - a \leq 0, -x + b \leq 0\} \subseteq C$. We say a set of inequalities C is an *infinite problem* if it is not finite. That is, there is a variable x in C such that there are no values a and b such that $\{x - a \leq 0, -x + b \leq 0\} \subseteq C$. We say an inequality is *simple* if it is of the form $x - a \leq 0$ or $-x + b \leq 0$. Let Propagate-Simple be a rule such as Propagate, but with an extra condition requiring J to be a simple inequality. We say a strategy for applying the Cutting to the Chase rules is *reasonable* if a rule R different from Propagate-Simple is applied only if Propagate-Simple is not applicable. Informally, a *reasonable* strategy is preventing the generation of derivations where simple inequalities $\{x - a \leq 0, -x + b \leq 0\}$ are ingored and C is essentialy treated as an infinite problem.

Theorem 2 (Termination). *Given a finite problem C, and a* reasonable *strategy, there is no infinite derivation sequence starting from $\langle [\![]\!], C_0 \rangle$.*

3.4 Relevant propagations

Unlike in SAT and Pseudo-Boolean solvers, Propagate rules cannot be applied to exhaustion for infinite problems. If C is unsatisfiable, the propagation rules may remain applicable indefinitely.

Example 4. Consider the followin set of (unsatisfiable) constraints

$$C = \{\overbrace{-x + y + 1 \leq 0}^{I}, \overbrace{-y + x \leq 0}^{J}, \overbrace{-y \leq 0}^{K}\} \ .$$

Starting from the initial state $\langle [\![y \geq 0]\!], C \rangle$, it is possible to generate the following infinite sequence of states by only applying the Propagate rule.

$$\begin{aligned} \langle [\![]\!], C \rangle &\Longrightarrow \langle [\![y \geq_K 0]\!], C \rangle \Longrightarrow \langle [\![y \geq_K 0, x \geq_I 1]\!], C \rangle \\ &\Longrightarrow \langle [\![y \geq_K 0, x \geq_I 1, y \geq_J 1]\!], C \rangle \Longrightarrow \\ &\qquad \langle [\![y \geq_K 0, x \geq_I 1, y \geq_J 1, x \geq_I 2]\!], C \rangle \Longrightarrow \ldots \end{aligned}$$

Let $\mathsf{nb}(x, M)$ denote the number of lower and upper bounds for x in M. Given a state $S = \langle M, C \rangle$, we say a new lower bound $x \geq_I b$ is *δ-relevant* at S if

1. $\mathsf{upper}(x, M) \neq +\infty$, or
2. $\mathsf{lower}(x, M) = -\infty$, or
3. $\mathsf{lower}(x, M) + \delta|\mathsf{lower}(x, M)| < b$ and $\mathsf{nb}(x, M) < \mathsf{Max}$.

If x has a upper bound, then any lower bound is δ-relevant because x becomes bounded, and termination is not an issue for bounded variables. If x does not already have lower bound, then any new lower bound $x \geq_I b$ is relevant. Finally, the third case states that the magnitude of the improvement must be significant and the number of bound improvements for x in M must be smaller than Max. In theory, to prevent non-termination during bound propagation we only need the cutoff Max. The condition $\mathsf{lower}(x, M) + \delta|\mathsf{lower}(x, M)| < b$ is used for pragmatical reasons, and is inspired by an approach used in [1]. The idea is to block any bound improvement for x that is *insignificant* with respect to the already known bound for x.

Even when only δ-relevant propagations are performed, it is still possible to generate an infinite sequence of transitions. The key observation is that Backjump is essentially a propagation rule, that is, it backtracks M, but it also adds a new improved bound for some variable x. It is easy to construct non-terminating examples, where Backjump is used to generate an infinite sequence of non δ-relevant bounds.

We propose a simple heuristic to deal with the termination problem. It is based on the observation that if we generate a non δ-relevant bound for x, then the problem is probably unsatisfiable, and x is in the unsatisfiable core. Thus, when selecting variables for the rule Decide we should give preference to variables that we computed non δ-relevant bounds for.

4 Strong Conflict Resolution

In this section, we extend our procedure to be able to handle divisibility constraints, by adding propagation, solving and consistency checking rules into our system. Then we show how to ensure that our procedure terminates even in cases when some variables are unbounded.

Solving divisibility constraints. We will add one proof rule to the proof system, in order to help us keep the divisibility constraints in a normal form. As Cooper originally noticed in [5], given two divisibility constraints, we can always eliminate a variable from one of them, obtaining equivalent constraints.

$$\textsc{div-solve}\ \frac{d_1 \mid a_1 x + p_1,\ d_2 \mid a_2 x + p_2}{\begin{array}{c} d_1 d_2 \mid dx + \alpha(d_2 p_1) + \beta(d_1 p_2) \\ d \mid a_2 p_1 - a_1 p_2 \end{array}}\ \textbf{if}\ \boxed{\begin{array}{c} d = \mathsf{gcd}(a_1 d_2, a_2 d_1) \\ \alpha(a_1 d_2) + \beta(a_2 d_1) = d \end{array}}$$

We use the above proof rule in our transition system to enable such normalization when needed.

Solve-Div

$$\langle M, C\rangle \implies \langle M, C'\rangle \qquad \textbf{if} \begin{cases} D_1, D_2 \in C, \\ (D_1', D_2') = \textsc{div-solve}(D_1, D_2), \\ C' = C \setminus \{D_1, D_2\} \cup \{D_1', D_2'\}. \end{cases}$$

Unsat-Div

$$\langle M, C \cup \{(d \mid a_1 x_1 + \cdots + a_n x_n + c)\}\rangle \implies \mathsf{unsat} \quad \textbf{if}\ gcd(d, a_1, \ldots, a_n) \nmid c$$

Propagation. With divisibility constraints as part of our problem, we can now achieve even more powerful propagation of bounds on variables. We say a variable x is *fixed* in the state $S = \langle M, C\rangle$ if $\mathsf{upper}(x, M) = \mathsf{lower}(x, M)$. Similarly a polynomial p is fixed if all its variables are fixed. To clarify the presentation, for fixed variables and polynomials we write $\mathsf{val}(x, M)$ and $\mathsf{val}(p, M)$ as a shorthand for $\mathsf{lower}(x, M)$ and $\mathsf{lower}(p, M)$.

Let $\langle M, C\rangle$ be a well-formed state, and $D, I \in C$ be a divisibility constraint and a tight inequality

$$D \equiv d \mid ax + p \ , \qquad\qquad I \equiv -x + q \leq 0 \ ,$$

with $a > 0$, $d > 0$, and $x \geq_I b \in M$. Assume, additionally, that p is fixed, i.e. assume that $\mathsf{val}(p, M) = k$.

In order to satisfy the divisibility constraint we then must have an integer z such that $dz = ax+p \geq aq+p$. Since all the variables in $aq+p$ are either assigned or implied, we can now use our system for deriving tight inequalities to deduce $-z + r \leq 0$ that would bound z in this state. Moreover substituting the solution for z, that is on the bound of the inequality, when substituted for x, would also satisfy the divisibility constraint. Using this, since $dz = ax+p$, we can deduce an inequality $-dax - dp + dr \leq 0$ which will guarantee that the bound on x satisfies the divisibility constraint. And, we can also use our procedure to convert this constraint into a tightly propagating one. Similar reasoning can be applied for the upper bound inequalities. We denote, as a shorthand, the result of this whole derivation with $\mathsf{div\text{-}derive}(I, D, x, M)$. We can now use the derivation above to empower propagation driven by divisibility constraints, as summarize below.

Propagate-Div

$$\langle M, C\rangle \implies \langle [\![M, x \geq_I c]\!], C \cup \{I\}\rangle \qquad \textbf{if} \begin{cases} D \equiv d \mid ax + p \in C, \\ x \geq_J b \in M, \\ I = \mathsf{div\text{-}derive}(J, D, x, M) \\ \mathsf{improves}(I, x, M) \\ c = \mathsf{bound}(I, x, M) \end{cases}$$

Eliminating Conflicting Cores. For sets of constraints containing unbounded variables, there is no guarantee that the procedure described in the previous section will terminate, even if learned inequalities (cuts) are not deleted using the Forget rule. In this section, we describe an extension based on Cooper's quantifier elimination procedure that guarantees termination.

Let U be a subset of the variables in X. We say U is the set of *unbounded* variables. Let $\prec$ be a total order over the variables in X such that for all variables $x \in X \setminus U$ and $y \in U$, $x \prec y$. We say a variable x is *maximal* in a constraint C containing x if for all variables y different from x in C, $y \prec x$. For now, we assume U contains all unbounded variables in the set of constraints C, and $\prec$ is fixed. Later, we describe how to dynamically change U and $\prec$ without compromising termination.

A *interval conflicting core* for variable x at state $S = \langle M, C\rangle$ is a set $\{-ax + p \leq 0,\ bx - q \leq 0\}$ such that p and q are fixed at S, and $\mathsf{bound}(-ax + p \leq 0, x, M) > \mathsf{bound}(bx - q \leq 0, x, M)$. A *divisibility conflicting core* for variable x at state S is a set $\{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + s)\}$ such that p, q and s are fixed, and for all values k in the interval $[\mathsf{bound}(-ax + p \leq 0, x, M), \mathsf{bound}(bx - q \leq 0, x, M)]$, $(d \nmid ck + \mathsf{val}(s, M))$. We do not consider cores containing more than one divisibility constraint because rule Solve-Div can be used to eliminate all but one of them. From hereafter, we assume a core is always of the form $\{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + r)\}$, since we can include the redundant divisibility constraint $(1 \mid x)$ to any interval conflicting core. We say x is a *conflicting variable* at state S if there is a interval or divisibility conflicting core for x. The variable x is the *minimal conflicting variable* at S if there is no $y \prec x$ such that y is also a conflicting variable at S. Let x be a minimal conflicting variable at state $S = \langle M, C\rangle$ and $D = \{-ax + p \leq 0,\ bx - q \leq 0,\ (d \mid cx + r)\}$ be a conflicting core for x, then a *strong resolvent* for D is a set R of inequality and divisibility constraints equivalent to

$$\exists x. - ax + p \leq 0 \wedge bx - q \leq 0 \wedge (d \mid cx + r)$$

The key property of R is that in any state $\langle M', C'\rangle$ such that $R \subset C'$, x is not the minimal conflicting variable or D is not a conflicting core.

We compute the resolvent R using Cooper's left quantifier elimination procedure. It can be summarized by the rule

$$\textsc{Cooper-Left}\ \frac{(d \mid cx + s),\ \ -ax + p \leq 0,\ \ bx - q \leq 0}{\begin{array}{c} 0 \leq k \leq m,\ \ bp - aq + bk \leq 0, \\ a \mid k + p,\ \ ad \mid ck + cp + as \end{array}}$$

where k is a fresh variable and $m = \mathsf{lcm}(a, \frac{ad}{\mathsf{gcd}(ad,c)}) - 1$. The fresh variable k is bounded so it does not need to be included in U. We extend the total order $\prec$ to k by making k the minimal variable. For the special case, where $(d \mid cx + s)$ is $(1 \mid x)$,the rule above simplifies to

$$\frac{-ax + p \leq 0,\ \ bx - q \leq 0}{0 \leq k < a,\ bp - aq + bk \leq 0,\ a \mid p + k}$$

The rule Cooper-Left is biased to lower bounds. We may also define the Cooper-Right rule that is based on Cooper's right quantifier elimination procedure and is biased to upper bounds. We use $\mathsf{cooper}(D)$ to denote a procedure that computes the strong resolvent R for a conflicting core D. Now, we extend our procedure with a new rule for introducing resolvents for minimal conflicting variables.

Resolve-Cooper

$$\langle M, C\rangle \implies \langle M, C \cup \mathsf{cooper}(D)\rangle \quad \textbf{if} \begin{cases} x \in U, \\ x \text{ is the minimal conflicting variable,} \\ D \text{ is a conflicting core for } x. \end{cases}$$

Note in addition to fresh variables, Resolve-Cooper rule also introduces new constraints without resorting to the Learn rule. We will show that this can not happen indefinitely, as the rule can only be applied a finite number of times.

Now we are ready to present and prove a simple and flexible strategy that will guarantee termination of our procedure even in the unbounded case.

Definition 2 (Two-layered strategy). *We say a strategy is two-layered for an initial state $\langle [\![ ]\!], C_0 \rangle$ if*

1. *it is reasonable (i.e., gives preference to the* Propagate-Simple *rules);*
2. *the* Propagate *rules are limited to δ-relevant bound refinements;*
3. *the* Forget *rule is never used to eliminate resolvents introduced by* Resolvent-Cooper*;*
4. *only applies the* Conflict *rule if* Resolve-Cooper *is not applicable.*

Theorem 3 (Termination). *Given a set of constraints C, there is no infinite derivation sequence starting from $S_0 = \langle [\![ ]\!], C \rangle$ that uses a two-layered strategy and U contains all unbounded variables in C.*

As an improvement, we note that we do not need to fix ordering $\prec$ at the beginning. It *can* be modified but, in this case, termination is only guaranteed if we eventually stop modifying it. Moreover, we can start applying the strategy with $U = \emptyset$. Then, far any non-δ-relevant bound refinement $\gamma(x)$, produced by the Backjump rules, we add x to the set U. Moreover, a variable x can be removed from U whenever a lower and upper bound for x can be deduced, and they do not depend on any decided bounds (variable becomes bounded).

5 Experimental Evaluation

We implemented the procedure described in a new solver cutsat. Implementation is a straightforward translation of the presented ideas, with very limited propagation, but includes heuristics from the SAT community such as dynamic ordering based on conflict activity, and Luby restarts. When a variable is to be decided, and we have an option to choose between the upper and lower bound, we choose the value that could satisfy most constraints. The solver source code, binaries used in the experiments, and all the accompanying materials are available at the authors website[5].

In order to evaluate our procedure we took a variety of already available integer problems from the literature, but we also crafted some additional ones. We include the problems that were used in [10] to evaluate their new simplex-based procedure that incorporates a new way of generating cuts to eliminate rational solutions. These problems are generated randomly, with all variables unbounded. This set of problems, which we denote with dillig, was reported hard for modern SMT solvers. We also include a reformulation of these problems, so that all the variables are bounded, by introducing slack variables, which we denote as slack. Next, we include the pure integer problems from the MIPLIB 2003 library [2], and we denote this problem set as miplib2003. The original problems are all very hard optimization instances, but, since we are dealing with the decision problem only, we have removed the optimization constraints and turned them into

[5] `http://cs.nyu.edu/~dejan/cutsat/`

feasibility problems.[6] We include PB problems from the 2010 pseudo-Boolean competition that were submitted and selected in 2010, marked as pb2010, and problems encoding the pigeonhole principle using cardinality constraints, denoted as pigeons. The pigeonhole problems are known to have no polynomial Boolean resolution proofs, and will therefore be hard for any solver that does not use cutting planes. And finally, we include a group of crafted benchmarks encoding a tight n-dimensional cone around the point whose coordinates are the first n prime numbers, denoted as primes. In these benchmarks all the variables are bounded from below by 0. We include the satisfiable versions, and the unsatisfiable versions which exclude points smaller than the prime solution.

In order to compare to the state-of-the art we compare to three different types of solvers. We compare to the current best integer SMT solvers, i.e yices 1.0.29 [11], z3 2.15 [9], mathsat5 [13] and mathsat5+cfp that simulates the algorithm from [10]. On all 0-1 problems in our benchmark suite, we also compare to the sat4j [3] PB solver, one of the top solvers from the PB competition, and a version sat4j+cp that is based on cutting planes. And, as last, we compare with the two top commercial MIP solvers, namely, gurobi 4.0.1 and cplex 12.2, and the open source MIP solver glpk 4.38. The MIP solvers have largely been ignored in the theorem-proving community, as it is claimed that, due to the use of floating point arithmetic, they are not sound.

Table 1. Experimental results.

problems	miplib2003 (16)		pb2010 (81)		dillig (250)		slacks (250)		pigeons (19)		primes (37)	
cutsat	722.78	12	1322.61	46	4012.65	223	2722.19	152	0.15	19	5.08	37
smt solvers	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved	time(s)	solved
mathsat5+cfp	575.20	11	2295.60	33	**2357.18**	**250**	160.67	98	0.23	19	1.26	37
mathsat5	**89.49**	**11**	1224.91	38	3053.19	245	**3243.77**	**177**	0.30	19	**1.03**	**37**
yices	226.23	8	57.12	37	5707.46	159	7125.60	134	**0.07**	**19**	0.64	32
z3	532.09	9	**168.04**	**38**	885.66	171	589.30	115	0.27	19	11.19	23
pb solvers												
sat4j	**22.34**	**10**	**798.38**	**67**	0.00	0	0.00	0	110.81	8	0.00	0
sat4j+cp	28.56	10	349.15	60	0.00	0	0.00	0	**4.85**	**19**	0.00	0
mip solvers												
glpk	242.67	12	1866.52	46	4.50	248	0.08	10	**0.09**	**19**	**0.44**	**37**
cplex	53.86	15	1512.36	58	8.65	250	8.76	248	0.51	19	3.47	37
gurobi	**28.96**	**15**	**1332.53**	**58**	**5.48**	**250**	**8.12**	**248**	0.21	19	0.80	37

All tests were conducted on an Intel Pentium E2220 2.4 GHz processor, with individual runs limited to 2GB of memory and 600 seconds. The results of our experimental evaluation are presented in Table 1. The rows are associated with the individual solvers, and columns separate the problem sets. For each problem

[6] All of the problems have a significant Boolean part, and 13 (out of 16) problems are pure PB problems

set we write the number of problems that the solver managed to solve within 600 seconds, and the cumulative time for the solved problems. We mark with bold the results that are best in a group of solvers, and we underline the results that are best among all solvers.

Compared to the SMT solvers, cutsat performs surprisingly strong, particularly being a prototype implementation. It outperforms or is the same as other smt solvers, except mathsat5 on all problem sets. Most importantly, it outperforms even mathsat5 on the real-world miplib2003 and pb2010 problem sets. The random dillig problems seem to be attainable by the solvers that implement the procedure from [10], but the same solvers surprisingly fail to solve the same problems with the slack reformulation (slacks).

Also very noticeable, the commercial MIP solvers outperform all the SMT solvers and cutsat by a big margin.

6 Conclusion

We proposed a new approach for solving ILP problems. It has all key ingredients that made CDCL-based SAT solver successful. Our solver justifies propagation steps using tightly-propagating inequalities that guarantee that any conflict detected by the search procedure can be resolved using only inequalities. We presented an approach to integrate Cooper's quantifier elimination algorithm in a model guided search procedure. Our first prototype is already producing encouraging results.

We see many possible improvements and extensions to our procedure. A solver for Mixed Integer-Real problems is the first natural extension. One basic idea would be to make the real variables bigger than the integer variables in the variable order $\prec$, and use Fourier-Moztkin resolution (instead of Cooper's procedure) to explain conflicts on rational variables. Integrating our solver with a Simplex-based procedure is another promising possibility. The idea is to use Simplex to check whether the current state or the search is feasible in the rational numbers or not. In principle, our solver can be integrated with a SMT solver based on DPLL(T). For example, it is straightforward to extract proofs/lemmas from unsatisfiable problems. On the other hand, there are many technical problems that need to be addressed. One radical, but appealing possibility, would be to use our solver instead of a SAT solver as the main search engine in a SMT solver.

Acknowledgements. We would like to thank Ken McMillan for reading an early draft and providing useful feedback, and Alberto Griggio for providing us with a custom version of mathsat5.

References

1. T. Achterberg. *SCIP: Solving constraint integer programs.* PhD thesis, TU Berlin, 2007.

2. T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.
3. D. L. Berre and A. Parrain. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
4. V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, April 1973.
5. D. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99):300, 1972.
6. S. Cotton. Natural domain SMT: A preliminary assessment. In *FORMATS*, 2010.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):397, 1962.
8. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.
9. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS 2008, Budapest, Hungary*, volume 4963 of *LNCS*, page 337. Springer, 2008.
10. I. Dillig, T. Dillig, and A. Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *CAV*, 2009.
11. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, LNCS, pages 81–94, 2006.
12. R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.
13. A. Griggio. A practical approach to SMT(LA(Z)). *SMT workshop*, 2010.
14. K. Korovin, N. Tsiskaridze, and A. Voronkov. Conflict resolution. In *Principles and Practice of Constraint Programming*, 2009.
15. S. Krstic and A. Goel. Architecting Solvers for SAT Modulo Theories: Nelson-Oppen with DPLL. In *FroCos*, 2007.
16. K. L. McMillan, A. Kuehlmann, and M. Sagiv. Generalizing DPLL to richer logics. In *CAV*, 2009.
17. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *DAC*, 2001.
18. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract DPLL procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.
19. J. P. M. Silva and K. A. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD*, 1997.
20. L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*. Wiley New York, 1999.