

Cuts from Proofs: A Complete and Practical Technique for Solving Linear Inequalities over Integers

Isil Dillig · Thomas Dillig · Alex Aiken

Received: date / Accepted: date

Abstract We propose a novel, sound, and complete Simplex-based algorithm for solving linear inequalities over integers. Our algorithm, which can be viewed as a semantic generalization of the *branch-and-bound* technique, systematically discovers and excludes entire subspaces of the solution space containing no integer points. Our main insight is that by focusing on the *defining constraints* of a vertex, we can compute a *proof of unsatisfiability* for the intersection of the defining constraints and use this proof to systematically exclude subspaces of the feasible region with no integer points. We show experimentally that our technique significantly outperforms the top four competitors in the QF-LIA category of the SMT-COMP '08 when solving conjunctions of linear inequalities over integers.

This work was supported by grants from NSF and DARPA (CCF-0430378, CNS-0716695).

Isil Dillig
Department of Computer Science
Stanford University
E-mail: isil@cs.stanford.edu

Thomas Dillig
Department of Computer Science
Stanford University
E-mail: tdillig@cs.stanford.edu

Alex Aiken
Department of Computer Science
Stanford University
E-mail: aiken@cs.stanford.edu

1 Introduction

A quantifier-free system of linear inequalities over integers is defined by $A\mathbf{x} \leq \mathbf{b}$ where A is an $m \times n$ matrix with only integer entries, and \mathbf{b} is a vector in Z^n . This system has a solution if and only if there exists a vector $\mathbf{x}^* \in Z^n$ that satisfies $A\mathbf{x}^* \leq \mathbf{b}$. Determining the satisfiability of such a system of inequalities is a recurring theme in program analysis and verification. For example, array dependence analysis, buffer overrun analysis, and integer overflow checking all rely on solving linear inequalities over integers [1, 2]. Similarly, linear integer inequalities arise in RTL datapath and symbolic timing verification [3, 4]. For this reason, many modern SMT solvers incorporate a dedicated linear arithmetic module for solving this important subclass of constraints [5–9].

While practical algorithms, such as Simplex, exist for solving linear inequalities over the reals [10], solving linear inequalities over integers is known to be an NP-complete problem, and existing algorithms do not scale well in practice. There are three main approaches for solving linear inequalities over integers. One approach first solves the *LP-relaxation* of the problem to obtain a rational solution and adds additional constraints until either an integer solution is found or the LP-relaxation becomes infeasible. The second approach is based on the Omega Test, an extension of the Fourier-Motzkin variable elimination for integers [2]. Yet a third class of algorithms utilize finite-automata theory [24, 11].

The algorithm presented in this paper falls into the first class of techniques described above. Existing algorithms in this class include *branch-and-bound*, *Gomory’s cutting planes* method, or a combination of both, known as *branch-and-cut* [12]. Branch-and-bound searches for an integer solution by solving the two subproblems $A\mathbf{x} \leq \mathbf{b} \cup \{x_i \leq \lfloor f_i \rfloor\}$ and $A\mathbf{x} \leq \mathbf{b} \cup \{x_i \geq \lceil f_i \rceil\}$ when the LP-relaxation yields a solution with fractional component f_i . The original problem has a solution if at least one of the subproblems has an integer solution. Even though upper and lower bounds can be computed for each variable to guarantee termination, this technique is often intractably slow on its own. Gomory’s cutting planes method computes *valid inequalities* that exclude the current fractional solution without excluding feasible integer points from the solution space. Unfortunately, this technique has also proven to be impractical on its own and is often only used in conjunction with branch-and-bound [13].

All of these techniques suffer from a common weakness: While they exclude the current fractional assignment from the solution space, they make no systematic effort to exclude the cause of this fractional assignment. In particular, if the solution of the LP-relaxation lies at the intersection of n planes defined by the initial set of inequalities, and $k \leq n$ of these planes have an intersection that contains no integer points, then it is desirable to exclude at least this entire $n - k$ dimensional subspace. The key insight underlying our approach is to systematically discover and exclude exactly this $n - k$ dimensional subspace rather than individual points that lie on this space. To be concrete, consider the following system with no integer solutions:

$$\begin{aligned} -3x + 3y + z &\leq -1 \\ 3x - 3y + z &\leq 2 \\ z &= 0 \end{aligned} \tag{1}$$

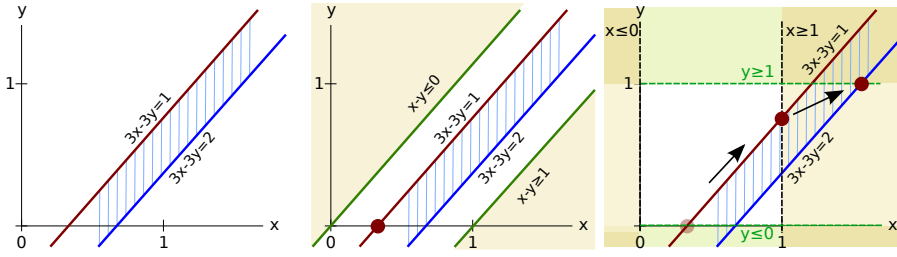


Fig. 1: (a) The projection of Equation 1 onto the xy plane. (b) The green lines indicate the closest lines parallel to the proof of unsatisfiability; the red point marks the solution of the LP-relaxation. (c) Branch-and-bound first adds the planes $x = 0$ and $x = 1$, then the planes $y = 0$ and $y = 1$, and continues to add planes parallel to the coordinate axes.

The projection of this system onto the xy plane is shown in Figure 1a. Suppose the LP-relaxation of the problem yields the fractional assignment $(x, y, z) = (\frac{1}{3}, 0, 0)$. The planes

$$\begin{aligned} z &= 0 \\ -3x + 3y + z &= -1 \end{aligned} \quad (2)$$

are the *defining constraints* of this vertex because the point $(\frac{1}{3}, 0, 0)$ lies at the intersection I of these planes. Since I contains no integer points, we would like to exclude exactly I from the solution space. Our technique discovers such intersections with no integer points by computing *proofs of unsatisfiability* for the defining constraints. A proof of unsatisfiability is a single equality that (i) has no integer solutions and (ii) is implied by the defining constraints. In our example, a proof of unsatisfiability for I is $-3x + 3y + 3z = -1$ since it has no integer solutions and is implied by Equation 2. Such proofs can be obtained from the *Hermite normal form* of the matrix representing the defining constraints.

Once we discover a proof of unsatisfiability, our algorithm proceeds as a semantic generalization of branch-and-bound. In particular, instead of branching on a fractional component of the solution, our technique branches around the proof of unsatisfiability, if one exists. In our example, once we discover the equation $-3x + 3y + 3z = -1$ as a proof of unsatisfiability, we construct two new subproblems:

$$\begin{array}{ll} -3x + 3y + z \leq -1 & -3x + 3y + z \leq -1 \\ 3x - 3y + z \leq 2 & 3x - 3y + z \leq 2 \\ z = 0 & z = 0 \\ -x + y + z \leq -1 & -x + y + z \geq 0 \end{array}$$

where $-x + y + z = -1$ and $-x + y + z = 0$ are the closest planes parallel to and on either side of $-3x + 3y + 3z = -1$ containing integer points. As Figure 1b illustrates, neither of these systems have a real-valued solution, and we immediately determine the initial system to be unsatisfiable. In contrast, as shown Figure 1c, branch-and-bound only adds planes parallel to the coordinate axes, repeatedly yielding points that lie on either $3x - 3y = 1$ or $3x - 3y = 2$, neither of which contains integer points. On the other hand, Gomory's cutting planes technique first derives the valid inequality $y \geq 1$

before eventually adding a cut that makes the LP-relaxation infeasible. Unfortunately, this technique becomes much less effective in identifying the cause of unsatisfiability in higher-dimensions.

In this paper, we make the following key contributions:

- We propose a novel, sound, and complete algorithm for solving linear inequalities over integers that systematically excludes subspaces of the feasible region containing no integer points.
- We argue that by focusing on the defining constraints of a vertex, we can quickly home in on the right “cuts” derived from proofs of unsatisfiability of the defining constraints.
- We present a semantic generalization of the branch-and-bound algorithm that utilizes the proofs of unsatisfiability of the defining constraints.
- We show experimentally that the proposed technique significantly outperforms existing state-of-the-art solvers, usually by orders of magnitude. Specifically, we compare Mistral, an implementation of our algorithm, with the top four competitors (by score) in the QF-LIA category of SMT-COMP '08 for solving conjunctions of linear inequalities over integers.
- Our algorithm is easy to implement and does not require extensive tuning to make it perform well in practice. We believe it can be profitably incorporated into existing SMT solvers that reason about linear arithmetic over integers.

2 Technical Background

2.1 Polyhedra, Faces, and Facets

In this section, we review a few standard definitions from polyhedral theory. The interested reader can refer to [13] for an in-depth discussion.

Definition 1 (Convex Polyhedron) The set of (real-valued) solutions satisfying $A\mathbf{x} \leq \mathbf{b}$ describes a convex polyhedron P . The dimension $\dim(P)$ of P is one less than the maximal number of affinely independent points in P .

Definition 2 (Valid Inequality) An inequality $\pi\mathbf{x} \leq \pi_0$ defined by some row vector π and a constant π_0 is a valid inequality for a polyhedron P if it is satisfied by all points in P .

Definition 3 (Faces and Facets) F is a face of polyhedron P if $F = \{\mathbf{x} \in P : \pi\mathbf{x} = \pi_0\}$ for some valid inequality $\pi\mathbf{x} \leq \pi_0$. A facet is a face of dimension $\dim(P) - 1$.

In Figure 2, polyhedron P has dimension 2 because there exist exactly 3 affinely independent points in P . The equation $ax + by \leq c$ is a valid inequality since all points in P satisfy this inequality. The point F is a face with dimension 0 since it is the intersection of P with the valid inequality represented by the dashed line. The line segment G is a facet of P since it is a face of dimension 1.

2.2 Linear Diophantine Equations

Definition 4 (Linear Diophantine Equation) A linear equation of the form $\sum a_i x_i = c$ is diophantine if all coefficients a_i are integers and c is an integer.

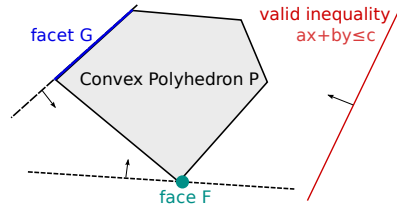


Fig. 2: A convex polyhedron of dimension 2

We state the following well-known result [13]:

Lemma 1 *A linear diophantine equation $\sum a_i x_i = c$ has a solution if and only if c is an integral multiple of the greatest common divisor $\gcd(a_1, \dots, a_n)$.*

Example 1 The equation $3x + 6y = 1$ has no integer solutions since 1 is not evenly divisible by $3 = \gcd(3, 6)$. However, $3x + 6y = 9$ has integer solutions.

Corollary 1 *Let E be a plane defined by $\sum a_i x_i = c$ with no integer solutions and let $g = \gcd(a_1, \dots, a_n)$. Then, the two closest planes parallel to and on either side of E containing integer points are $\lfloor E \rfloor$ and $\lceil E \rceil$, given by $\sum \frac{a_i}{g} x_i = \lfloor c/g \rfloor$ and $\sum \frac{a_i}{g} x_i = \lceil c/g \rceil$ respectively.*

This corollary follows immediately from Lemma 1 and implies that there are no integer points between E and $\lfloor E \rfloor$ as well as between E and $\lceil E \rceil$.

2.3 Proofs of Unsatisfiability and the Hermite Normal Form

Given a system $A\mathbf{x} = \mathbf{b}$ of linear diophantine equations, we can determine in polynomial time whether this system has any integer solutions using the *Hermite normal form* of A .¹ Below we briefly review key properties of the Hermite normal form; the interested reader is referred to [13] for a more in-depth discussion.

Definition 5 (Hermite Normal Form) An $m \times m$ integer matrix H is said to be in Hermite normal form (HNF) if (i) H is lower triangular, (ii) $h_{ii} > 0$ for $0 \leq i < m$, and (iii) $h_{ij} \leq 0$ and $|h_{ij}| < h_{ii}$ for $i > j$.²

Definition 6 (Unimodular Matrix) An $n \times n$ matrix U is *unimodular* if it has only integer entries and $|\det(U)|$ is 1.

We review the following well-known lemmas:

Lemma 2 *For any $m \times n$ matrix A with $\text{rank}(A) = m$, there exists an $n \times n$ unimodular matrix U such that*

$$AU = [H \mid \mathbf{0}]$$

and the matrix H is the unique Hermite normal form of A .

¹ While it is possible to determine the satisfiability of a system of linear diophantine equalities in polynomial time, determining the satisfiability of a system of linear integer *inequalities* is NP-complete.

² There is no agreement in the literature on the exact definition of the Hermite Normal Form. The one given here follows the definition in [13].

While we do not describe the algorithm for computing the Hermite normal form of A , we remark that there exists an efficient polynomial time for computing the Hermite normal form of any matrix A (see [17]). Finally, we also recall the following two well-known results [13]:

Lemma 3 *If H is the Hermite normal form of A , then $H^{-1}A$ contains only integer entries.*

Lemma 4 (Proof of Unsatisfiability) *The system $A\mathbf{x} = \mathbf{b}$ has an integer solution if and only if $H^{-1}\mathbf{b} \in Z^m$. If $A\mathbf{x} = \mathbf{b}$ has no integer solutions, there exists a row vector \mathbf{r}_i of the matrix $H^{-1}A$ such that the corresponding entry $\frac{n_i}{d_i}$ of $H^{-1}\mathbf{b}$ is not an integer. We call the linear diophantine equation $d_i\mathbf{r}_i\mathbf{x} = n_i$ with no integer solutions a proof of unsatisfiability of $A\mathbf{x} = \mathbf{b}$.*

If the equation $d_i\mathbf{r}_i\mathbf{x} = n_i$ is a proof of unsatisfiability of $A\mathbf{x} = \mathbf{b}$, then it is implied by the original system and does not have integer solutions.

Example 2 Consider the defining constraints from the example in Section 1:

$$\begin{aligned} z &= 0 \\ -3x + 3y + z &= -1 \end{aligned}$$

Here, we have:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ -3 & 3 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 \\ -2 & 3 \end{bmatrix} \quad H^{-1}A = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad H^{-1}\mathbf{b} = \begin{bmatrix} 0 \\ -\frac{1}{3} \end{bmatrix}$$

This system does not have an integer solution because $H^{-1}\mathbf{b}$ contains a fractional component, and the equation $-3x + 3y + 3z = -1$ is a proof of unsatisfiability for this system.

3 The Cuts-from-Proofs Algorithm

In this section, we present our algorithm for determining the satisfiability of the system $A\mathbf{x} \leq \mathbf{b}$ over integers. In the presentation of the algorithm, we assume that there is a procedure `lp_solve` that determines the satisfiability of $A\mathbf{x} \leq \mathbf{b}$ over the reals, and if satisfiable returns a vertex v at an extreme point of the polyhedron induced by $A\mathbf{x} \leq \mathbf{b}$. This assumption is fulfilled by standard exterior-point algorithms for linear programming, such as Simplex [10].

Definition 7 (Defining Constraint) An inequality $\pi\mathbf{x} \leq \pi_0$ is a defining constraint of vertex \mathbf{v} of the polyhedron induced by $A\mathbf{x} \leq \mathbf{b}$ if \mathbf{v} satisfies the equality $\pi\mathbf{v} = \pi_0$ where π is a row of A and π_0 is the corresponding entry in \mathbf{b} .

With slight abuse of terminology, we call $\pi\mathbf{x} = \pi_0$ a defining constraint whenever $\pi\mathbf{x} \leq \pi_0$ is a defining constraint.

3.1 Algorithm

Let A be the initial $m \times n$ matrix and let a_{max} be the entry with the maximum absolute value in A . Then, choose any α such that $\alpha \geq n \cdot |a_{max}|$.

1. Invoke `lp.solve`. If the result is unsatisfiable, return unsatisfiable. Otherwise, if vertex v returned by `lp.solve` is integral, return v .
2. Identify the defining constraints $A'\mathbf{x}' \leq \mathbf{b}'$ of v .
3. Determine if the system $A'\mathbf{x}' = \mathbf{b}'$ has any integer solutions, and, if not, obtain a proof of unsatisfiability as described in Section 2.3.³
4. There are two cases:

Case 1: (Conventional branch-and-bound) If a proof of unsatisfiability does not exist (i.e., $A'\mathbf{x}' = \mathbf{b}'$ has integer solutions) or if the proof of unsatisfiability contains a coefficient greater than $\alpha \cdot \gcd(a_1, \dots, a_n)$, pick a fractional component f_i of v and solve the two subproblems:

$$\begin{array}{ll} A\mathbf{x} \leq \mathbf{b} & A\mathbf{x} \leq \mathbf{b} \\ v_i \leq \lfloor f_i \rfloor & -v_i \leq -\lceil f_i \rceil \end{array}$$

Case 2: (Branch around proof of unsatisfiability) Otherwise, consider the proof of unsatisfiability $\sum a_i x_i = c$ of $A'\mathbf{x}' = \mathbf{b}'$ and let g be $\gcd(a_1, \dots, a_n)$. The system $A\mathbf{x} \leq \mathbf{b}$ has a solution if either of the two subproblems has a solution:

$$\begin{array}{ll} \begin{bmatrix} A & \\ \frac{a_1}{g} & \dots & \frac{a_n}{g} \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} \mathbf{b} \\ \lfloor \frac{c}{g} \rfloor \end{bmatrix} & \begin{bmatrix} A & \\ -\frac{a_1}{g} & \dots & -\frac{a_n}{g} \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} \mathbf{b} \\ -\lceil \frac{c}{g} \rceil \end{bmatrix} \end{array}$$

3.2 Discussion of the Algorithm

In the above algorithm, if `lp.solve` yields a fractional assignment, then either

- (i) the intersection of the defining constraints does not have an integer solution or
- (ii) the defining constraints do have an integer solution but `lp.solve` did not pick an integer assignment

In the latter case (i.e., (ii)), we simply perform conventional branch-and-bound around any fractional component of the assignment to find an integer point on this intersection. Observe that while the current intersection $A'\mathbf{x}' = \mathbf{b}'$ is guaranteed to contain an integer point, this integer point may or may not lie inside the polyhedron defined by $A\mathbf{x} \leq \mathbf{b}$. Thus, the existence of an integer solution to the system $A'\mathbf{x}' = \mathbf{b}'$ in case (1) of the algorithm does not guarantee the existence of an integer solution to the original system $A\mathbf{x} \leq \mathbf{b}$.

On the other hand (i.e., (i)), if the defining constraints do not admit an integer solution, the algorithm obtains a proof of unsatisfiability with maximum coefficient less than α , if one exists, and constructs two subproblems that exclude this intersection without missing any integer points in the solution space. The constant α ensures that case 2 in step 4 of the algorithm is invoked a finite number of times and guarantees

³ Recall that Lemma 2 defines the Hermite normal form of an $m \times n$ matrix A' when A' has full rank. If A' does not have full rank, observe that we can still compute a proof of unsatisfiability of $A'\mathbf{x}' = \mathbf{b}'$ by dropping redundant rows of the system.

that there is a minimum bound on the volume excluded from the polyhedron at each step of the algorithm. (See Section 3.3 for the relevance of α for termination.)

Branching around the two planes in case 2 of the algorithm guarantees that the intersection $A'\mathbf{x}' = \mathbf{b}'$ of the defining constraints is no longer in the polyhedra defined by the two new subproblems. However, there *may* still exist a strict subset of these defining constraints (i.e., a higher-dimensional subspace) whose intersection contains no integer points but is not excluded from the solution space of the new subproblems. The following example illustrates such a situation.

Example 3 Consider the defining constraints $x + y \leq 1$ and $2x - 2y \leq 1$. Using Hermite normal forms to compute a proof of unsatisfiability for the system

$$\begin{aligned} x + y &= 1 \\ 2x - 2y &= 1 \end{aligned}$$

yields $4x = 3$. While $4x = 3$ is a proof of unsatisfiability for the intersection of $x + y = 1$ and $2x - 2y = 1$, the strict subset $2x - 2y = 1$ has a proof of unsatisfiability on its own (namely itself), and it is not implied by $4x = 3$.

As this example illustrates, the proof of unsatisfiability of a set of constraints does not necessarily imply the proof of unsatisfiability of any subset of these constraints. At first glance, this seems problematic because if the intersection of any subset of the defining constraints contains no integer solutions, we would prefer excluding this larger subspace represented by the smaller set of constraints. Fortunately, as stated by Lemma 7, the algorithm will discover and exclude this higher-dimensional intersection in a finite number of steps. We first prove the following helper lemmas:

Lemma 5 Let $C = \begin{bmatrix} A \\ B \end{bmatrix}$ be an $m \times n$ matrix composed of A and B , and let $\text{HNF}(C) = \begin{bmatrix} H_A & 0 \\ X & Y \end{bmatrix}$. Then, $\text{HNF}(A) = H_A$.

Proof Our proof uses the HNF construction outlined in [13]. Let i be a row that the algorithm is currently working on and let i' be another row such that $i' < i$. Then, by construction, any entry $c_{i'j}$ where $j > i'$ is 0. Since any column operation performed while processing row i adds a multiple of column $k \geq i$ to another column, entry $c_{i'k}$ must be 0. Thus, any column operation is idempotent on row i' .

Using blockwise inversion to invert $\text{HNF}(C)$, it can be easily shown that:

$$\text{HNF}(C)^{-1} = \begin{bmatrix} H_A^{-1} & 0 \\ -Y^{-1}XH_A^{-1} & Y^{-1} \end{bmatrix}$$

Thus, it is easy to see that $\text{HNF}(C)^{-1}C = \text{HNF}(C)^{-1}\mathbf{b}'$ implies $\text{HNF}(A)^{-1}A = \text{HNF}(A)^{-1}\mathbf{b}$ if \mathbf{b}' is obtained by adding entries to the bottom of \mathbf{b} . This is the case because both $\text{HNF}(C)^{-1}$ and $\text{HNF}(A)^{-1}$ are lower triangular matrices. Intuitively, this result states that if $A\mathbf{x} = \mathbf{b}$ has a proof of unsatisfiability, we cannot “lose” this proof by adding extra rows at the bottom of A .

Example 4 Consider the constraints from Example 2. Suppose we add the additional constraint $x = 1$ at the bottom of matrix A . Then, we obtain:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ -3 & 3 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad H^{-1}A = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad H^{-1}\mathbf{b} = \begin{bmatrix} 0 \\ -\frac{1}{3} \\ 1 \end{bmatrix}$$

Clearly, $-3x + 3y + 3z = -1$ is still obtained as a proof of unsatisfiability from the second row of $H^{-1}A = H^{-1}\mathbf{b}$.

Lemma 6 Consider any proof of unsatisfiability $\Sigma a_i x_i = c$ of any subset of the initial system $A\mathbf{x} \leq \mathbf{b}$. Then, $\forall i. |a_i| \leq \alpha \cdot \gcd(a_1, \dots, a_n)$.

Proof The coefficients a_i are obtained from the matrix $H^{-1}A'$ where A' is a matrix whose rows are a subset of those of A . Recall from basic linear algebra $H^{-1} = \frac{1}{\det(H)} \text{adj}(H)$ where $\text{adj}(H)$ is the classical adjoint of H . Let the notation $\|A\|$ denote $\max_{i,j} |a_{ij}|$. It is shown in [14] that:

$$\|\text{adj}(H)\| \leq \det(H)$$

for any matrix H in Hermite normal form. Hence any coefficient c in H^{-1} satisfies $|c| \leq 1$, and the entries in $H^{-1}A'$ are therefore bound by $\alpha = n \cdot \|A'\|$. Since the proof of unsatisfiability is some row of $H^{-1}A'$ multiplied by some $d_i > 1$, $d_i \leq \gcd(a_1, \dots, a_n)$ as d_i is a divisor of each a_i . Thus, any coefficient in the proof of unsatisfiability is bound by $\alpha \cdot \gcd(a_1, \dots, a_n)$.

Using the above lemmas, we can now show the following result:

Lemma 7 Let F be a k -dimensional face without integer points of the initial polyhedron P with $\dim(P) = d$. Suppose `lp_solve` repeatedly returns vertices that lie on this face. The algorithm will exclude F from P in a finite number of steps.

Proof Every time `lp_solve` yields a vertex that lies on F , the algorithm excludes from the search space the intersection of the current defining constraints; thus, the next time `lp_solve` yields a vertex, one of these constraints will no longer be defining. At some point, when `lp_solve` returns a vertex on F , its defining constraints will be exactly the $d - k$ of the original constraints defining F , along with new constraints that were added to the bottom of the matrix. By Lemma 5, the additional constraints preserve the proof of unsatisfiability of the original $d - k$ constraints. Furthermore, by Lemma 6, this proof of unsatisfiability will have coefficients with absolute value of at most $\alpha \cdot \gcd(a_1, \dots, a_n)$. Thus, the algorithm will obtain a proof of unsatisfiability for F and exclude all of F from the solution space.

As Lemma 7 elucidates, the Cuts-from-Proofs algorithm discovers any relevant face without integer points on a demand-driven basis without explicitly considering all possible subsets of the initial set of inequalities. This allows the algorithm to add exactly the relevant cuts while staying computationally tractable in practice.

3.3 Soundness and Completeness

It is easy to see that the algorithm given above is correct because it never excludes integer points in the solution space. For arguing termination, we can assume, as standard, that the polyhedron P is finite; if it is not, one can compute maximum and minimum bounds on each variable without affecting the satisfiability of the original problem (see, for example [12, 13]). The key observation is that the volume we cut off the polyhedron cannot become infinitesimally small over time as we add more cuts. To see this, observe that there is a finite set of normal vectors N for the planes added by the Cuts-from-Proofs algorithm. Clearly, this holds for planes added by case 1 of step 4 since all such planes are parallel to one of the coordinate planes. This fact also holds for planes added in case 2 of step 4 since the coefficients of the normal vectors must be less than or equal to α . Since the set N of normal vectors is finite, the algorithm will either terminate or, at some point, it will have to add planes parallel to already existing ones. The following lemma states that these parallel planes are at least some minimal distance ϵ apart:

Lemma 8 (Progress) *Let E be a plane added by the Cuts-from-Proofs algorithm and let E' be another plane parallel to E , also added by the algorithm. Then, E and E' are at least some minimum distance $\epsilon > 0$ apart.*

Proof Let E be defined by $\mathbf{n} \cdot \mathbf{x} = c_1$ and E' be defined by $\mathbf{n} \cdot \mathbf{x} = c_2$. Since c_1 and c_2 are integers and $c_1 \neq c_2$, E and E' are a minimum $d = 1/\sqrt{n_1^2 + \dots + n_k^2}$ apart. Since there are a finite number of non-parallel planes added by the algorithm, choose ϵ to be the minimum such d .

Let $\mathbf{n} \in N$ be any normal vector along which the algorithm must eventually cut. Because P is finite, there is a finite distance δ we can move along \mathbf{n} through P . Since the distance we move along \mathbf{n} is at least ϵ , the algorithm can cut perpendicular to \mathbf{n} at most δ/ϵ times. Hence, the algorithm must terminate.

4 Implementation

In Section 4.1, we first discuss improvements over the basic algorithm presented in Section 3; then, in Section 4.2, we discuss the details of our implementation.

4.1 Improvements and Empirical Observations

An improvement over the basic algorithm described in Section 3 can be achieved by selectively choosing the proofs of unsatisfiability that the algorithm branches on. In particular, recall from Lemma 7 that if `lp_solve` repeatedly returns vertices on the same face with no integer points, the algorithm will also repeatedly obtain the same proof of unsatisfiability. Thus, in practice, it is beneficial to delay branching on a proof until the same proof is obtained at least twice. This can be achieved by using case 1 in step 4 of the algorithm instead of case 2 each time a new proof is discovered. Since few of these proofs appear repeatedly, this easy modification often allows the algorithm to exclude only the highest-dimensional intersection with no integer points

without having to branch around additional intermediate proofs. In our experience, this optimization can improve running time up to a factor of 3 on some examples.

An important empirical observation about the algorithm is that the overwhelming majority ($> 99\%$) of the proofs of unsatisfiability do not result in true branching. In practice, one of the planes parallel to the proof of unsatisfiability often turns out to be a valid inequality, while the other parallel plane lies outside the feasible region, making its LP-relaxation immediately unsatisfiable. Thus, in practice, the algorithm only branches around fractional components of an assignment.

4.2 Implementation Details

Our implementation of the Cuts-from-Proofs algorithm is written in C++ and consists of approximately 5000 lines of code, including modules to perform various matrix operations as well as support for infinite precision arithmetic. The Cuts-from-Proofs algorithm is a key component of the Mistral constraint solver, which implements the decision procedure for the combined theory of integer linear arithmetic and uninterpreted functions. Mistral is used in the Compass program analysis system (under development) to solve large real-world constraints that arise from modeling contents of unbounded data structures, such as arrays and linked lists.

Our Simplex implementation, used as the `lp.solve` procedure in the Cuts-from-Proofs algorithm, uses *Bland's rule* for pivot selection [12]. Mistral utilizes a custom-built infinite precision arithmetic library based on the GNU MP Bignum Library (GMP) [15]. Our library performs computation natively on 64-bit values until an overflow is detected, and then switches to GNU bignums. If no overflow is detected, our implementation results in less than 25% slow down over native word-level arithmetic. We also found the selective use of hand-coded SIMD instructions to improve performance of Simplex by approximately a factor of 2.

Our implementation for Hermite normal form conversion is based on the algorithm given in [16]. This algorithm uses the modulo reduction technique of [17] to control the number of required bits in any intermediate computation. In practice, the Hermite normal form conversion takes less than 5% of the overall running time and is not a bottleneck.

The implementation of the core Cuts-from-Proofs algorithm takes only about 250 lines of C++ code and does not require any features beyond what is discussed in this paper. In our implementation, α was chosen to be $10n \cdot |a_{max}|$, and we have not observed the coefficients in the computed proofs of unsatisfiability to exceed this limit. In practice, the coefficients stay reasonably small.

5 Experimental Results

To evaluate the effectiveness of the Cuts-from-Proofs algorithm, we compared Mistral with the four leading competitors (by score) in the QF-LIA category of SMT-COMP '08, namely Yices 1.0.16, Z3.2, MathSAT 4.2, and CVC3 1.5 obtained from [18]. We did not compare Mistral against (mixed) integer linear programming tools specialized for optimization problems. Existing tools such as GLPK [19], lp-solve [20], and CPLEX [21] all use floating point numbers instead of infinite precision arithmetic and yield unsound results for determining satisfiability even on small systems

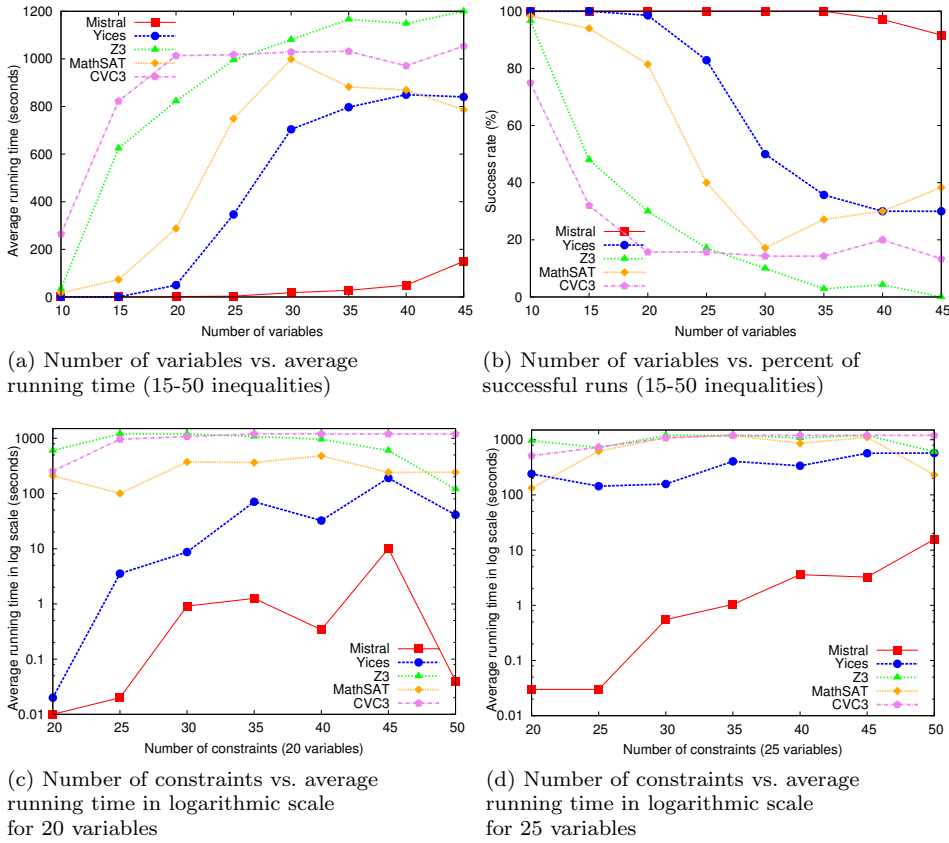


Fig. 3: Experimental Results (fixed coefficient)

due to rounding errors. Furthermore, we did not use the QF-LIA benchmarks from SMT-COMP because they contain arbitrary boolean combinations of linear integer inequalities and equalities, making them unsuitable for comparing different algorithms to solve integer linear programs. The full set of test inputs and running times for each tool is available from <http://www.stanford.edu/~isil/benchmarks.tar.gz>. All experiments were performed on an 8 core 2.66 GHz Xeon workstation with 24 GB of memory. (All the tools, including Mistral, are single-threaded applications.) Each tool was given a maximum running time of 1200 seconds as well as 4 GB of memory. Any run exceeding the time or memory limit was aborted and marked as failure. If a run was aborted, its running time was assumed to be 1200 seconds for computing average running times.

In the experiments, presented in Figure 3, we randomly generated more than 500 systems of linear inequalities, containing between 10 and 45 variables and between 15 and 50 inequalities per system with a fixed maximum coefficient size of 5. Figure 3a plots the number of variables against the average running time over all sizes of constraints, ranging from 15 to 50. As is evident from this figure, the Cuts-from-Proofs algorithm results in a dramatic improvement over all existing tools. For instance, for

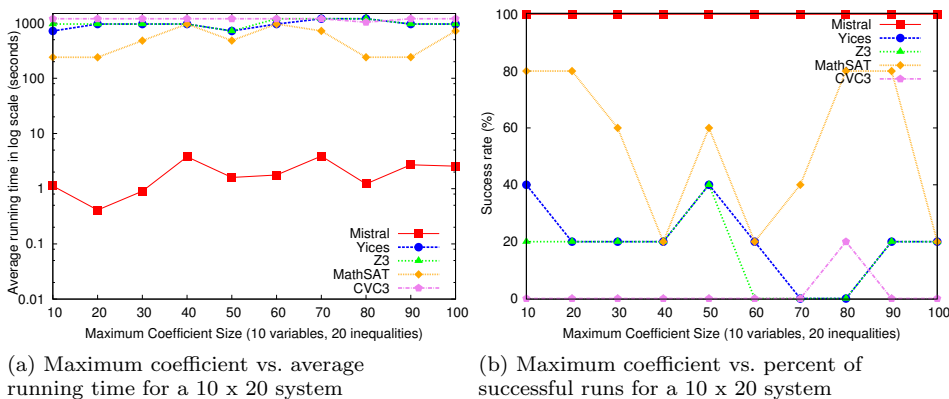


Fig. 4: Experimental Results (fixed dimensions)

25 variables, Yices, Mistral’s closest competitor, takes on average 347 seconds while Mistral takes only 3.45 seconds. This trend is even more pronounced in Figure 3b, which plots number of variables against the percentage of successful runs. For example, for 35 variables, Yices has a success rate of 36% while Mistral successfully completes 100% of its runs, taking an average of only 28.11 seconds.

Figures 3c and 3d plot the number of inequalities per system against average running time on a logarithmic scale for 20 and 25 variables, respectively. We chose not to present detailed breakouts for larger numbers of variables since such systems trigger time-out rates over 50% for all tools other than Mistral. These graphs demonstrate that the Cuts-from-Proofs algorithm reliably performs significantly, and usually at least an order of magnitude, better than any of the other tools, regardless of the number of inequalities per system.

To evaluate the sensitivity of different algorithms to maximum coefficient size, we also compared the running time of different tools for coefficients ranging from 10 to 100 for systems with 10 variables and 20 inequalities. As shown in Figure 4, Mistral is less sensitive to coefficient size than the other tools. For example, for maximum coefficient 50, Mistral’s closest competitor, MathSAT, takes an average of 482 seconds with a success rate of 60% while Mistral takes an average of 1.6 seconds with a 100% success rate.

Among the tools we compared, Yices and Z3 use a Simplex-based branch-and-cut approach, while CVC3 implements the Omega test. MathSAT mainly uses a Simplex-based algorithm augmented with the Omega test as a fallback mechanism. In our experience, one of the main differences between Simplex-based and Omega test based algorithms is that the former run out of time, while the latter run out of memory. On average, Simplex-based tools seem to perform better than tools using the Omega test.

We believe these experimental results demonstrate that the Cuts-from-Proofs algorithm outperforms leading implementations of existing techniques by orders of magnitude and significantly increases the size and complexity of integer linear programs that can be solved. Furthermore, our algorithm is easy to implement and does not require extensive tuning to make it perform well. We believe that the Cuts-from-Proofs

algorithm can be profitably incorporated into existing SMT solvers that integrate the theory of linear integer arithmetic.

6 Related Work

As discussed in Section 1, there are three major approaches for solving linear inequalities over integers. LP-based approaches include branch-and-bound, Gomory’s cutting planes method, and various combinations of the two [13, 12]. The cutting planes method derives valid inequalities from the final Simplex tableau. More abstractly, a Gomory cut can be viewed as the proof of unsatisfiability of a single inequality obtained from a linear combination of the original set of inequalities. This is in contrast with our Cuts-from-Proofs algorithm which obtains a proof from the set of defining constraints, rather than from a single inequality in the final Simplex tableau. Unfortunately, the number of cuts added by Gomory’s cutting planes technique is usually very large, and few of these cuts ultimately prove helpful in obtaining an integer solution [12]. Branch-and-cut techniques that combine branch-and-bound and variations on cutting planes techniques have proven more successful and are used by many state-of-the-art SMT solvers [5, 6, 8]. However, the algorithm proposed in this paper significantly outperforms leading implementations of the branch-and-cut technique.

Another technique for solving linear integer inequalities is the Omega test, an extension of the Fourier-Motzkin variable elimination for integers [2]. A drawback of this approach is that it can consume gigabytes of memory even on moderately sized inputs, causing it to perform worse in practice than Simplex-based techniques.

A third approach for solving linear arithmetic over integers is based on finite automata theory [11]. Unfortunately, while complete, automata-based approaches perform significantly worse than all of the aforementioned techniques. The authors are not aware of any tools based on this approach that are currently under active development.

Another proposal [22] for solving linear arithmetic over integers is to translate the formula into an equisatisfiable boolean formula, whose satisfiability can then be checked using a standard boolean SAT solver. This technique is mainly targeted for special classes of ILP problems that arise frequently in verification where most of the constraints are difference constraints and each of the remaining non-difference constraints contains few variables.

Hermite normal forms are a well-studied topic in number theory, and efficient polynomial-time algorithms exist for computing Hermite normal forms [16, 14]. Their application to solving systems of linear diophantine equations is discussed, for example, in [13, 12]. Jain et al. study the application of Hermite normal forms to computing interpolants of systems of linear diophantine equalities and disequalities [23]. We adopt the term “proof of unsatisfiability” from the literature on Craig interpolation [26, 27].

Conclusion

We have presented a novel, sound, and complete algorithm called Cuts-from-Proofs for solving linear inequalities over integers and demonstrated experimentally that this algorithm significantly outperforms leading implementations of existing approaches.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments and feedback. We would also like to thank David Dill for his useful suggestions and Suhabe Bugrara for his comments on a draft of this paper.

References

1. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program, ACM Press (1978) 84–97
2. Pugh, W.: The Omega test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM* (1992)
3. Brinkmann, R., Drechsler, R.: RTL-datapath verification using integer linear programming. In: *VLSI Design*. (2002) 741–746
4. Amon, T., Borriello, G., Hu, T., Liu, J.: Symbolic timing verification of timing diagrams using presburger formulas. In: *DAC '97: Proceedings of the 34th annual conference on Design automation*, New York, NY, USA, ACM (1997) 226–231
5. Dutertre, B., De Moura, L.: The Yices SMT solver. Technical report, SRI International (2006)
6. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems* (April 2008) 337–340
7. Barrett, C., Tinelli, C.: CVC3. In Damm, W., Hermanns, H., eds.: *Proceedings of the 19th International Conference on Computer Aided Verification*. Volume 4590 of *Lecture Notes in Computer Science*, Springer-Verlag (July 2007) 298–302 Berlin, Germany.
8. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT4 SMT solver. In: *Proceedings of the 20th international conference on Computer Aided Verification*, Berlin, Heidelberg, Springer-Verlag (2008) 299–303
9. Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: The Barcelogic smt solver. In: *Proceedings of the 20th international conference on Computer Aided Verification*, Berlin, Heidelberg, Springer-Verlag (2008) 294–298
10. Dantzig, G.: *Linear Programming and Extensions*. Princeton U. Press (1963)
11. Ganesh, V., Berezin, S., Dill, D.: Deciding Presburger arithmetic by model checking and comparisons with other methods. In: *FMCAD '02: Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, London, UK, Springer-Verlag (2002) 171–186
12. Schrijver, A.: *Theory of Linear and Integer Programming*. J. Wiley & Sons (1986)
13. Nemhauser, G.L., Wolsey, L.: *Integer and Combinatorial Optimization*. John Wiley & Sons (1988)
14. Storjohann, A., Labahn, G.: Asymptotically fast computation of hermite normal forms of integer matrices. In: *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '96*, ACM Press (1996) 259–266
15. <http://gmplib.org/>: Gnu mp bignum library
16. Cohen, H.: *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer-Verlag (1993)
17. Domich, P., Kannan, R., L. Trotter, J.: Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research* **12**(1) (February 1987) 50–59
18. <http://www.smtcomp.org>: Smt-comp'08
19. <http://www.gnu.org/software/glpk/>: Glpk (gnu linear programming kit)
20. <http://lpsolve.sourceforge.net/5.5/>: lp_solve reference guide
21. <http://www.ilog.com/products/cplex/>: Cplex
22. Seshia, S., Bryant, R.: Deciding quantifier-free Presburger formulas using parameterized solution bounds. (2004)
23. Jain, H., Clarke, E., Grumberg, O.: Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. In: *Proceedings of the 20th international conference on Computer Aided Verification*, Berlin, Heidelberg, Springer-Verlag (2008) 254–267
24. Wolper, P. and Boigelot, B.: An automata-theoretic approach to Presburger arithmetic constraints. In: *Proceedings of the Symposium on Static Analysis (SAS)* (1995) 21–32

25. Golub, G.H., Van Loan, C.F.: *Matrix Computations*. JHU Press (1996)
26. Craig W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory, *Journal of Symbolic Logic* (1957) 269–285
27. McMillan, KL: Applications of Craig interpolants in model checking. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2005*, 1–12