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

Isil Dillig, Thomas Dillig, and Alex Aiken
Computer Science Department
Stanford University

Linear Arithmetic over Integers

- Problem: Given an $m \times n$ matrix A with only integer entries, and a vector $\vec{b} \in \mathbb{Z}^n$, does

$$A\vec{x} \leq \vec{b}$$

have any integer solutions?

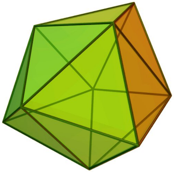
Linear Arithmetic over Integers

- Problem: Given an $m \times n$ matrix A with only integer entries, and a vector $\vec{b} \in \mathbb{Z}^n$, does

$$A\vec{x} \leq \vec{b}$$

have any integer solutions?

- Geometric interpretation: Are there any integer points inside the polyhedron defined by $A\vec{x} \leq \vec{b}$?



Why is This an Important Problem?

- Many applications in software verification, compiler optimizations, and model checking require determining the satisfiability of a system of linear integer inequalities.

Why is This an Important Problem?

- Many applications in software verification, compiler optimizations, and model checking require determining the satisfiability of a system of linear integer inequalities.

  - Verifying buffer accesses: Is integer `i` used as an index in the range of the buffer?

Why is This an Important Problem?

- Many applications in software verification, compiler optimizations, and model checking require determining the satisfiability of a system of linear integer inequalities.

  - Verifying buffer accesses: Is integer `i` used as an index in the range of the buffer?

  - Array dependence analysis: Can `a[i]` and `a[j]` refer to the same memory location?

Why is This an Important Problem?

- Many applications in software verification, compiler optimizations, and model checking require determining the satisfiability of a system of linear integer inequalities.

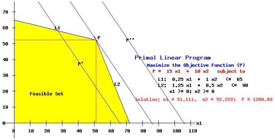
  - Verifying buffer accesses: Is integer `i` used as an index in the range of the buffer?

  - Array dependence analysis: Can a[i] and a[j] refer to the same memory location?

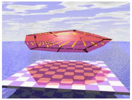
  - Integer overflow checking, RTL-datapath verification, . . .

Existing Techniques

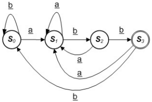
- Simplex-based Approaches:



- The Omega Test:



- Automata-based Approaches:



Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.
- The Omega Test:
  - Extends the Fourier-Motzkin variable elimination technique for reals to integers.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.
- The Omega Test:
  - Extends the Fourier-Motzkin variable elimination technique for reals to integers.
  - Eliminates variables one by one until the problem becomes infeasible or no variables are left.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.
- The Omega Test:
  - Extends the Fourier-Motzkin variable elimination technique for reals to integers.
  - Eliminates variables one by one until the problem becomes infeasible or no variables are left.
- Automata-based Approaches:
  - Encode the linear inequality system as an automaton.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.
- The Omega Test:
  - Extends the Fourier-Motzkin variable elimination technique for reals to integers.
  - Eliminates variables one by one until the problem becomes infeasible or no variables are left.
- Automata-based Approaches:
  - Encode the linear inequality system as an automaton.
  - System is satisfiable if the language accepted by the automaton is non-empty.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.

This Talk

- A new approach for finding better additional constraints to find an integer solution.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.

This Talk

- A new approach for finding better additional constraints to find an integer solution.
- Performs orders of magnitude better than existing approaches.

Existing Techniques

- Simplex-based Approaches:
  - Use Simplex to obtain a real-valued solution
  - No real solution $\Rightarrow$ no integer solution
  - Simplex yields integer solution $\Rightarrow$ integer solution exists
  - Otherwise, add additional constraints and repeat.

This Talk

- A new approach for finding better additional constraints to find an integer solution.
- Performs orders of magnitude better than existing approaches.
- Complete, i.e., guaranteed to find an integer solution if one exists.

Motivating Example

- Consider the system:

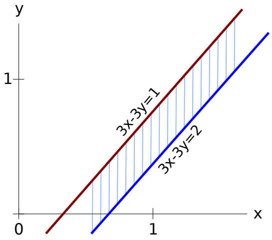
$$
\begin{aligned}
-3x + 3y + z &\leq -1 \\
3x - 3y + z &\leq 2 \\
z &= 0
\end{aligned}
$$

Motivating Example

- Consider the system:

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

Projection of this system onto the xy plane:

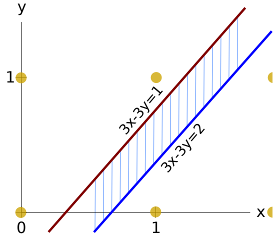


Motivating Example

- Consider the system:

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

Projection of this system onto the xy plane:



- This system has no integer solutions.

How Do Existing Simplex-Based Approaches Deal with this Example?

- The simplest and most common Simplex-based technique is branch and bound.

How Do Existing Simplex-Based Approaches Deal with this Example?

- The simplest and most common Simplex-based technique is branch and bound.

- Since our algorithm can be seen as a generalization of branch and bound, we will first illustrate this technique.

How Do Existing Simplex-Based Approaches Deal with this Example?

- The simplest and most common Simplex-based technique is branch and bound.
- Since our algorithm can be seen as a generalization of branch and bound, we will first illustrate this technique.
- If Simplex yields a solution with fractional component f_i, branch and bound solves two subproblems:

$$A\vec{x} \leq \vec{b} \cup \{x_i \leq \lfloor f_i \rfloor\}$$

How Do Existing Simplex-Based Approaches Deal with this Example?

- The simplest and most common Simplex-based technique is branch and bound.
- Since our algorithm can be seen as a generalization of branch and bound, we will first illustrate this technique.
- If Simplex yields a solution with fractional component f_i, branch and bound solves two subproblems:

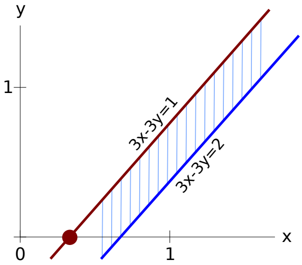
$$A\vec{x} \leq \vec{b} \cup \{x_i \leq \lfloor f_i \rfloor\}$$
$$A\vec{x} \leq \vec{b} \cup \{x_i \geq \lceil f_i \rceil\}$$

Example Using Branch and Bound

- For instance, suppose Simplex yields the solution

$$(x, y, z) = \left(\frac{1}{3}, 0, 0\right)$$

for the previous problem.

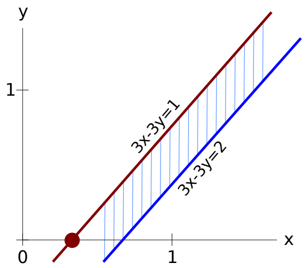


Example Using Branch and Bound

- For instance, suppose Simplex yields the solution

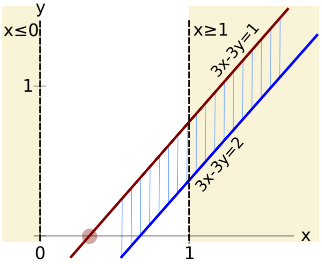
$$(x, y, z) = \left(\frac{1}{3}, 0, 0\right)$$

for the previous problem.



Example Using Branch and Bound

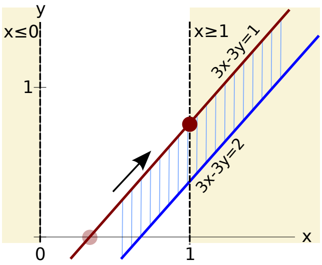
- Branch and bound constructs two subproblems with additional constraints $x \leq 0$ and $x \geq 1$



Example Using Branch and Bound

- For the subproblem where $x \geq 1$, we obtain a new solution

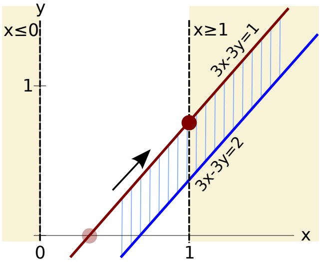
$$(x, y, z) = \left(1, \frac{2}{3}, 0\right)$$



Example Using Branch and Bound

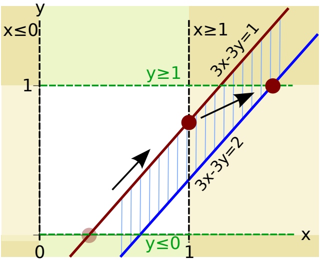
- For the subproblem where $x \geq 1$, we obtain a new solution

$$(x, y, z) = \left(1, \frac{2}{3}, 0\right)$$



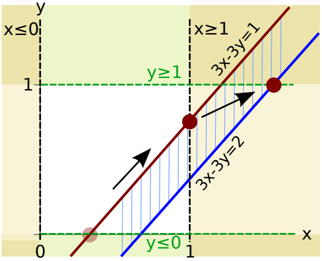
Example Using Branch and Bound

- Now branch and bound constructs another two new subproblems with additional constraints $y \geq 1$ and $y \leq 0$, but the solution is still fractional.



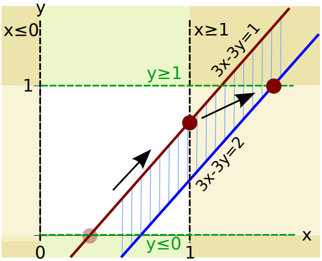
Example Using Branch and Bound

- In fact, by only adding planes parallel to the x and y planes, branch and bound will never exclude the entire space and will keep obtaining more and more fractional solutions.



Example Using Branch and Bound

- While bounds on x and y can be computed to make it terminate, these bounds are extremely large, making branch and bound impractical on its own.



The Problem with Branch and Bound

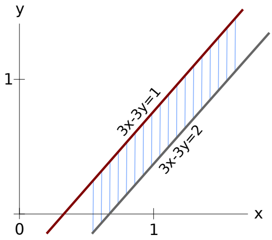
- Branch and bound only excludes a single fractional point from the solution space.

The Problem with Branch and Bound

- Branch and bound only excludes a single fractional point from the solution space.
- But this fractional point might lie on a k-dimensional subspace not containing integer points.

The Problem with Branch and Bound

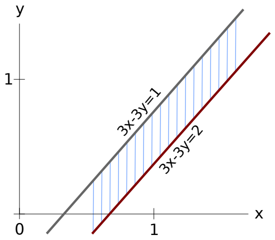
- Branch and bound only excludes a single fractional point from the solution space.
- But this fractional point might lie on a k-dimensional subspace not containing integer points.



The plane $3x - 3y = 1$ does not contain any integer points.

The Problem with Branch and Bound

- Branch and bound only excludes a single fractional point from the solution space.
- But this fractional point might lie on a k-dimensional subspace not containing integer points.



Similarly, $3x - 3y = 2$ also does not contain any integer points.

The Problem with Branch and Bound

- Branch and bound only excludes a single fractional point from the solution space.
- But this fractional point might lie on a k-dimensional subspace not containing integer points.

Insight

- Instead of excluding individual points on this subspace, we would like to exclude exactly this k-dimensional subspace.

The Problem with Branch and Bound

- Branch and bound only excludes a single fractional point from the solution space.
- But this fractional point might lie on a k-dimensional subspace not containing integer points.

Insight

- Instead of excluding individual points on this subspace, we would like to exclude exactly this k-dimensional subspace.
- Our technique systematically identifies and excludes these higher dimensional subspaces containing no integer points.

Outline of the Cuts-from-Proofs Algorithm I

Step 1: When Simplex yields a fractional solution, identify the defining constraints of this vertex.

Outline of the Cuts-from-Proofs Algorithm I

Step 1: When Simplex yields a fractional solution, identify the defining constraints of this vertex.

- Defining constraints of a vertex v are the subset of the inequalities given by $A\vec{x} \leq \vec{b}$ that v satisfies as an equality.

Outline of the Cuts-from-Proofs Algorithm I

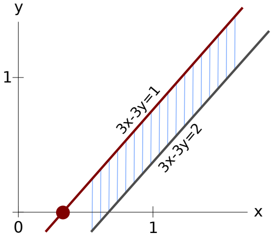
Step 1: When Simplex yields a fractional solution, identify the defining constraints of this vertex.

- Defining constraints of a vertex v are the subset of the inequalities given by $A\vec{x} \leq \vec{b}$ that v satisfies as an equality.
- These exist because Simplex always returns points that lie on the boundary of the polyhedron defined by $A\vec{x} \leq \vec{b}$.

Outline of the Cuts-from-Proofs Algorithm I

Step 1: When Simplex yields a fractional solution, identify the defining constraints of this vertex.

- Defining constraints of a vertex v are the subset of the inequalities given by $A\vec{x} \leq \vec{b}$ that v satisfies as an equality.
- These exist because Simplex always returns points that lie on the boundary of the polyhedron defined by $A\vec{x} \leq \vec{b}$.
- $-3x + 3y + z \leq -1$ is a defining constraint of $(\frac{1}{3}, 0, 0)$ because $-3 \cdot \frac{1}{3} + 3 \cdot 0 + 0 = -1$.



Outline of the Cuts-from-Proofs Algorithm II

Step 2: Determine whether the intersection $A'\vec{x} = \vec{b'}$ of the defining constraints contains any integer points.

✓ Can be done efficiently.

Outline of the Cuts-from-Proofs Algorithm II

Step 2: Determine whether the intersection $A'\vec{x} = \vec{b'}$ of the defining constraints contains any integer points.

✓ Can be done efficiently.

Step 3a: If the intersection does contain integer points, perform conventional branch and bound.

Outline of the Cuts-from-Proofs Algorithm II

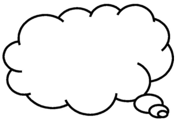
Step 2: Determine whether the intersection $A'\vec{x} = \vec{b'}$ of the defining constraints contains any integer points.

✓ Can be done efficiently.

Step 3a: If the intersection does contain integer points, perform conventional branch and bound.

- There may be integer points within the feasible region that lie on this intersection.

Outline of the Cuts-from-Proofs Algorithm III



Idea:

If the intersection of defining constraints does not contain integer solutions, we want to identify the smallest subset of the defining constraints whose intersection does not contain integer solutions.

Smallest subset

$\Rightarrow$

Highest dimensional subspace

Outline of the Cuts-from-Proofs Algorithm IV

Step 3b: If the intersection of defining constraints does not contain an integer point, compute a proof of unsatisfiability and "branch around" this proof.

Outline of the Cuts-from-Proofs Algorithm V

- A proof of unsatisfiability P for a system of linear equalities $A'\vec{x} = \vec{b'}$ is a plane such that:

Outline of the Cuts-from-Proofs Algorithm V

- A proof of unsatisfiability P for a system of linear equalities $A'\vec{x} = \vec{b'}$ is a plane such that:
    1. it does not contain any integer points

Outline of the Cuts-from-Proofs Algorithm V

- A proof of unsatisfiability P for a system of linear equalities $A'\vec{x} = \vec{b'}$ is a plane such that:
  1. it does not contain any integer points
  2. it is implied by $A'\vec{x} = \vec{b'}$

Outline of the Cuts-from-Proofs Algorithm V

- A proof of unsatisfiability P for a system of linear equalities $A'\vec{x} = \vec{b'}$ is a plane such that:
  1. it does not contain any integer points
  2. it is implied by $A'\vec{x} = \vec{b'}$
- Branching around this proof plane ensures that we exclude at least the intersection of the defining constraints.

Outline of the Cuts-from-Proofs Algorithm V

- A proof of unsatisfiability P for a system of linear equalities $A'\vec{x} = \vec{b'}$ is a plane such that:
  1. it does not contain any integer points
  2. it is implied by $A'\vec{x} = \vec{b'}$

- Branching around this proof plane ensures that we exclude at least the intersection of the defining constraints.

- Result: If there is a smaller subset of the defining constraints whose intersection has no integer solution, we will obtain a proof of unsatisfiability for this higher-dimensional intersection in a finite number of steps.

Hermite Normal Forms

Charles Hermite

(1822-1901)

We can determine whether the defining constraints $A'\vec{x} = \vec{b'}$ have an integer solution and also compute proofs of unsatisfiability efficiently (in polynomial time) by using the Hermite Normal Form of A'.

Determining whether Defining Constraints Have Integer Solutions

- Compute H, the Hermite normal form of A', and H^{-1}.

Determining whether Defining Constraints Have Integer Solutions

- Compute H, the Hermite normal form of A', and H^{-1}.

$$A'\vec{x} = \vec{b'}$$

Determining whether Defining Constraints Have Integer Solutions

- Compute H, the Hermite normal form of A', and H^{-1}.

$$H^{-1}A'\vec{x} = H^{-1}\vec{b'}$$

Determining whether Defining Constraints Have Integer Solutions

- Compute H, the Hermite normal form of A', and H^{-1}.

$$H^{-1}A'\vec{x} = H^{-1}\vec{b'}$$

Important property:

$H^{-1}A'$ is always integral.

Determining whether Defining Constraints Have Integer Solutions

- Compute H, the Hermite normal form of A', and H^{-1}.

$$H^{-1}A'\vec{x} = H^{-1}\vec{b'}$$

Important property:

$A'\vec{x} = \vec{b'}$ has integer solutions

$\Leftrightarrow$

$H^{-1}\vec{b'}$ integral.

Computing Proofs of Unsatisfiability

$$\underbrace{\begin{bmatrix} \text{———} \; r_1 \; \text{———} \\ \cdots \\ \text{———} \; r_i \; \text{———} \\ \cdots \\ \text{———} \; r_m \; \text{———} \end{bmatrix}}_{H^{-1}A'} \vec{x} = \underbrace{\begin{bmatrix} c_1 \\ \cdots \\ c_i \\ \cdots \\ c_m \end{bmatrix}}_{H^{-1}\vec{b'}}$$

Computing Proofs of Unsatisfiability

$$\underbrace{\begin{bmatrix} \text{———} & r_1 & \text{———} \\ & \dots & \\ {\color{red}a_1} & {\color{red}\dots} & {\color{red}a_n} \\ & \dots & \\ \text{———} & r_m & \text{———} \end{bmatrix}}_{H^{-1}A'} \vec{x} = \underbrace{\begin{bmatrix} c_1 \\ \dots \\ {\color{red}\frac{n_i}{d_i}} \\ \dots \\ c_m \end{bmatrix}}_{H^{-1}\vec{b'}}$$

Computing Proofs of Unsatisfiability

$$\underbrace{\begin{bmatrix} \text{———} & r_1 & \text{———} \\ & \dots & \\ {\color{red}a_1} & {\color{red}\dots} & {\color{red}a_n} \\ & \dots & \\ \text{———} & r_m & \text{———} \end{bmatrix}}_{H^{-1}A'} \vec{x} = \underbrace{\begin{bmatrix} c_1 \\ \dots \\ {\color{red}\frac{n_i}{d_i}} \\ \dots \\ c_m \end{bmatrix}}_{H^{-1}\vec{b'}}$$

Proof of Unsatisfiability

A proof of unsatisfiability of $A'\vec{x} = \vec{b'}$ is:

$$a_1 d_i \cdot x_1 + \ldots + a_n d_i \cdot x_n = n_i$$

Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.

Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.
- Compute the greatest common divisor $g = gcd(a_1, \ldots, a_n)$.

Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.
- Compute the greatest common divisor $g = \mathit{gcd}(a_1, \ldots, a_n)$.
- Then, the closest planes parallel to and on either side of $\Sigma a_i x_i = c_i$ containing integer points are:

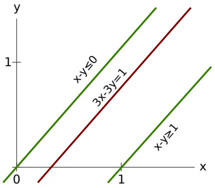
$$\Sigma(a_i/g)x_i = \lfloor c_i/g \rfloor \ \text{ and } \ \Sigma(a_i/g)x_i = \lceil c_i/g \rceil$$

Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.
- Compute the greatest common divisor $g = \mathit{gcd}(a_1, \ldots, a_n)$.
- Then, the closest planes parallel to and on either side of $\Sigma a_i x_i = c_i$ containing integer points are:

$$\Sigma(a_i/g)x_i = \lfloor c_i/g \rfloor \ \text{ and } \ \Sigma(a_i/g)x_i = \lceil c_i/g \rceil$$

Projection of planes containing integer points on either side of $3x - 3y = 1$



Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.
- Compute the greatest common divisor $g = gcd(a_1, \ldots, a_n)$.
- "Branching around" the proof of unsatisfiabilty means solving the two subproblems:

Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.
- Compute the greatest common divisor $g = gcd(a_1, \ldots, a_n)$.

- "Branching around" the proof of unsatisfiabilty means solving the two subproblems:

$$A\vec{x} \leq \vec{b} \cup \{\Sigma(a_i/g)x_i \leq \lfloor c_i/g \rfloor\}$$

Branching around Proofs of Unsatisfiability

- Let $P = \Sigma a_i x_i = c_i$ be a proof of unsatisfiability for the defining constraints of a vertex v.
- Compute the greatest common divisor $g = gcd(a_1, \ldots, a_n)$.

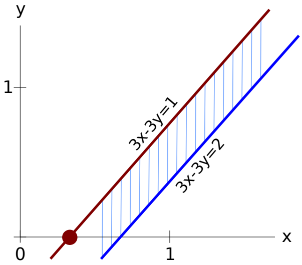
- "Branching around" the proof of unsatisfiabilty means solving the two subproblems:

$$A\vec{x} \leq \vec{b} \cup \{\Sigma(a_i/g)x_i \leq \lfloor c_i/g \rfloor\}$$
$$A\vec{x} \leq \vec{b} \cup \{\Sigma(a_i/g)x_i \geq \lceil c_i/g \rceil\}$$

Cuts-from-Proofs Example

Consider the vertex $(\frac{1}{3}, 0, 0)$ and its defining constraints:

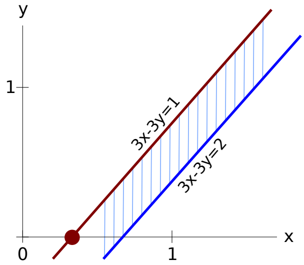
$$\begin{aligned} z &= 0 \\ -3x + 3y + z &= -1 \end{aligned}$$



Cuts-from-Proofs Example

The system $A'\vec{x} = \vec{b'}$ is:

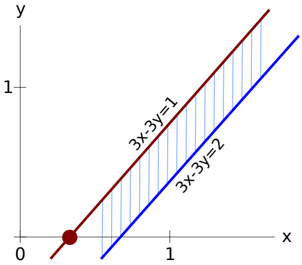
$$\begin{bmatrix} 0 & 0 & 1 \\ -3 & 3 & 1 \end{bmatrix} \vec{x} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$



Cuts-from-Proofs Example

Multiply both sides by H^{-1}:

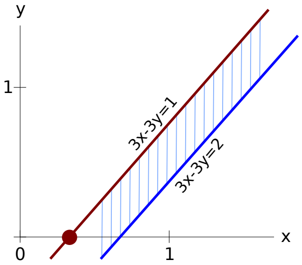
$$\frac{1}{3}\begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 0 & 0 & 1 \\ -3 & 3 & 1 \end{bmatrix}\vec{x} = \frac{1}{3}\begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ -1 \end{bmatrix}$$



Cuts-from-Proofs Example

Here, $H^{-1}A'x = H^{-1}\vec{b}$ is:

$$\begin{bmatrix} 0 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix} x = \begin{bmatrix} 0 \\ -\frac{1}{3} \end{bmatrix}$$

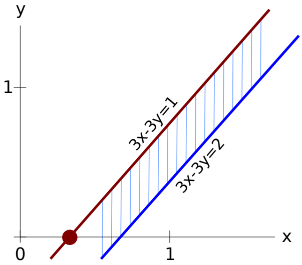


Cuts-from-Proofs Example

Here, $H^{-1}A'x = H^{-1}\vec{b}$ is:

$$\begin{bmatrix} 0 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix} x = \begin{bmatrix} 0 \\ -\frac{1}{3} \end{bmatrix}$$

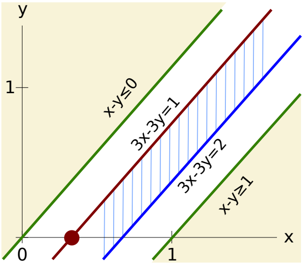
Therefore $-3x + 3y + 3z = -1$ is a proof of unsatisfiability.



Cuts-from-Proofs Example

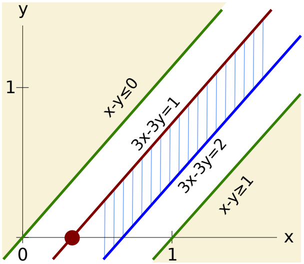
The planes closest to and on either side of the proof plane $-3x + 3y + 3z = -1$ are:

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



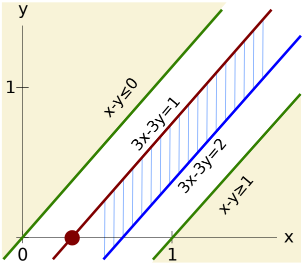
Cuts-from-Proofs Example

Therefore, the Cuts-from-Proofs algorithm solves the two subproblems shown in the figure.



Cuts-from-Proofs Example

Neither subproblem has a real-valued solution, therefore Cuts-from-Proofs terminates in just one step.



Completeness

- To guarantee completeness, it is necessary to restrict the coefficients allowed in the proofs of unsatisfiability to a maximum constant $\alpha \geq n \cdot |a_{max}|$
  - n is the number of variables and $|a_{max}|$ the maximum absolute value of coefficients in the original matrix A.

Completeness

- To guarantee completeness, it is necessary to restrict the coefficients allowed in the proofs of unsatisfiability to a maximum constant $\alpha \geq n \cdot |a_{max}|$
  - n is the number of variables and $|a_{max}|$ the maximum absolute value of coefficients in the original matrix A.
  - This is necessary to prevent the volume "cut" by a proof of unsatisfiability from becoming infinitesimally small over time.

Completeness

- To guarantee completeness, it is necessary to restrict the coefficients allowed in the proofs of unsatisfiability to a maximum constant $\alpha \geq n \cdot |a_{max}|$
  - n is the number of variables and $|a_{max}|$ the maximum absolute value of coefficients in the original matrix A.
  - This is necessary to prevent the volume "cut" by a proof of unsatisfiability from becoming infinitesimally small over time.
  - The constant $n \cdot |a_{max}|$ ensures that if all the proofs of unsatisfiability with coefficients less than or equal to $n \cdot |a_{max}|$ are added, the system will either become infeasible or it contains integer points.

Experiments

- We compare the performance of the Cuts-from-Proofs algorithm against the top four competitors of SMT-COMP'08: Z3, Yices, MathSAT, and CVC3.

Experiments

- We compare the performance of the Cuts-from-Proofs algorithm against the top four competitors of SMT-COMP'08: Z3, Yices, MathSAT, and CVC3.
- Among these tools,
  - Z3 and Yices use the Simplex-based branch-and-cut algorithm, which is a combination of branch and bound and Gomory's cutting planes method.

Experiments

- We compare the performance of the Cuts-from-Proofs algorithm against the top four competitors of SMT-COMP'08: Z3, Yices, MathSAT, and CVC3.
- Among these tools,
  - Z3 and Yices use the Simplex-based branch-and-cut algorithm, which is a combination of branch and bound and Gomory's cutting planes method.
  - CVC3 uses the Omega Test.

Experiments

- We compare the performance of the Cuts-from-Proofs algorithm against the top four competitors of SMT-COMP'08: Z3, Yices, MathSAT, and CVC3.
- Among these tools,
  - Z3 and Yices use the Simplex-based branch-and-cut algorithm, which is a combination of branch and bound and Gomory's cutting planes method.
  - CVC3 uses the Omega Test.
  - MathSAT uses a combination of branch-and-cut and the Omega test.

Experiments

- We compare the performance of the Cuts-from-Proofs algorithm against the top four competitors of SMT-COMP'08: Z3, Yices, MathSAT, and CVC3.
- Among these tools,
  - Z3 and Yices use the Simplex-based branch-and-cut algorithm, which is a combination of branch and bound and Gomory's cutting planes method.
  - CVC3 uses the Omega Test.
  - MathSAT uses a combination of branch-and-cut and the Omega test.
- We did not compare against tools specialized in mixed integer-linear programming, such as CPLEX and GLPK
  - because they do not support infinite precision arithmetic and yield unsound results.

Implementation

- Cuts-from-Proofs is implemented as part of the Mistral constraint solver.

Implementation

- Cuts-from-Proofs is implemented as part of the Mistral constraint solver.
  - Mistral implements the combined theory of linear integer arithmetic and uninterpreted functions.

Implementation

- Cuts-from-Proofs is implemented as part of the Mistral constraint solver.
  - Mistral implements the combined theory of linear integer arithmetic and uninterpreted functions.
  - Mistral is used to solve large arithmetic constraints that arise from analyzing unbounded data structures like arrays.

Implementation

- Cuts-from-Proofs is implemented as part of the Mistral constraint solver.
  - Mistral implements the combined theory of linear integer arithmetic and uninterpreted functions.
  - Mistral is used to solve large arithmetic constraints that arise from analyzing unbounded data structures like arrays.

- Implementation utilizes an infinite precision arithmetic library based on GNU MP

Implementation

- Cuts-from-Proofs is implemented as part of the Mistral constraint solver.
  - Mistral implements the combined theory of linear integer arithmetic and uninterpreted functions.
  - Mistral is used to solve large arithmetic constraints that arise from analyzing unbounded data structures like arrays.

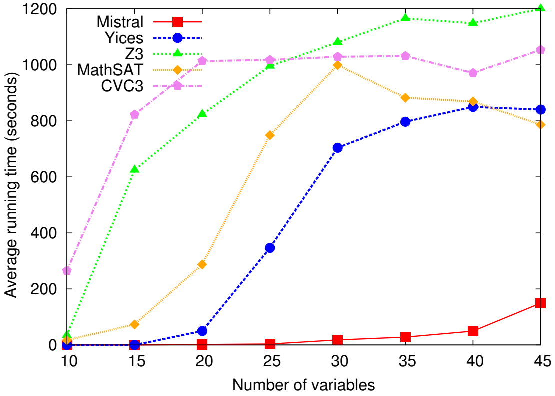
- Implementation utilizes an infinite precision arithmetic library based on GNU MP
  - Performs computation natively on 64-bit values

Implementation

- Cuts-from-Proofs is implemented as part of the Mistral constraint solver.
  - Mistral implements the combined theory of linear integer arithmetic and uninterpreted functions.
  - Mistral is used to solve large arithmetic constraints that arise from analyzing unbounded data structures like arrays.

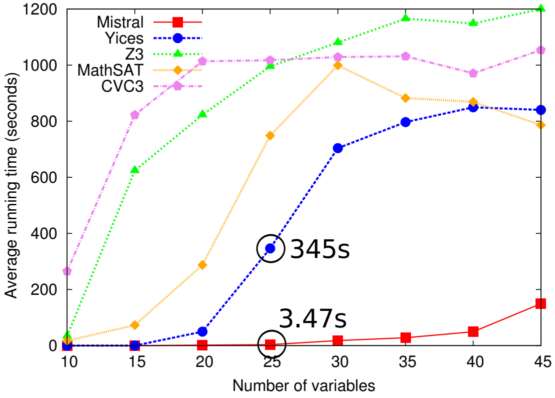
- Implementation utilizes an infinite precision arithmetic library based on GNU MP
  - Performs computation natively on 64-bit values
  - But switches to infinite precision representation when overflow is detected.

Experiments



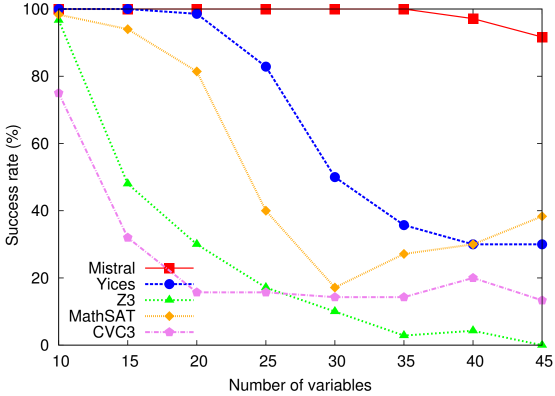
Number of variables vs. average running time. All systems are randomly generated inequalities with fixed coefficient size.

Experiments



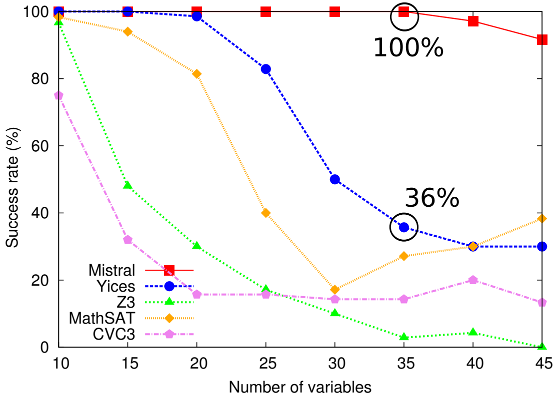
Number of variables vs. average running time. All systems are randomly generated inequalities with fixed coefficient size.

Experiments



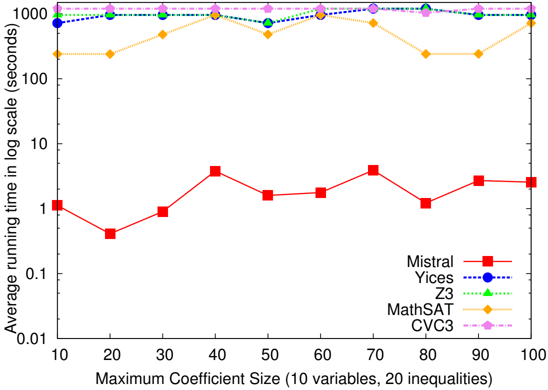
Number of variables vs. percent of successful runs. All systems are randomly generated inequalities with fixed coefficient size.

Experiments



Number of variables vs. percent of successful runs. All systems are randomly generated inequalities with fixed coefficient size.

Experiments



Maximum coefficient vs. average running time for a 10x20 system.

Any Questions?

Related Work

Pugh, W.:
The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependence Analysis.
Communications of the ACM (1992)

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

Nemhauser, G.L., Wolsey, L.:
Integer and Combinatorial Optimization.
John Wiley & Sons (1988)

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

Jain, H., Clarke, E., Grumberg, O.:
Efficient Craig Interpolation for Linear Diophantine (Dis)equations and Linear Modular Equations.
In: CAV '08, Berlin, Heidelberg, Springer-Verlag (2008) 254–267