Constraint Logic Programming and Integrating Simplex with DPLL($\mathcal{T}$)

Ali Sinan Köksal

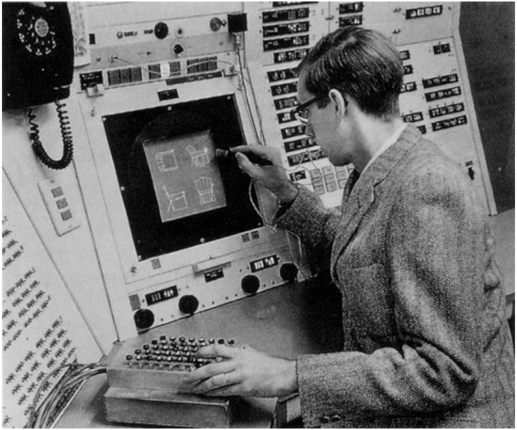
December 3, 2010

Constraint logic programming

- Problem: designing programming systems to reason with and about constraints.
- CLP is a class of programming languages based on:
  - Constraint solving
  - The logic programming paradigm

Constraint programming

- Sketchpad (1963)



Interactive drawing system using static constraints

Logic programming paradigm

An example program in pure Prolog:

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

We can perform the query:

```
?- sibling(sally, erica).
Yes
```

CLP($\mathcal{X}$) framework

- The CLP($\mathcal{X}$) framework [JL87] is a *scheme* where $\mathcal{X}$ can be instantiated with a suitable *domain of discourse*, such as $\mathcal{R}$, the algebraic structure consisting of uninterpreted functors over real numbers [JMSY92].

Structure of CLP($\mathcal{R}$) programs

- Arithmetic terms:
  - Real constants and variables are arithmetic terms
  - If t_1 and t_2 are terms, then $(t_1 + t_2), (t_1 - t_2), (t_1 * t_2)$ are also arithmetic terms
- Terms:
  - Uninterpreted constants, arithmetic terms and variables are terms
  - If f is an n-ary uninterpreted functor and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term
- Constraints:
  - If t_1 and t_2 are arithmetic terms, then $t_1 = t_2, t_1 < t_2$ and $t_1 \leq t_2$ are constraints
  - If not both t_1 and t_2 are arithmetic terms, then only $t_1 = t_2$ is a constraint

Structure of CLP($\mathcal{R}$) programs (2)

- An atom is of the form

$$p(t_1, t_2, \ldots, t_n)$$

  where p is a predicate symbol and $t_1, \ldots, t_n$ are terms.
- A rule is of the form

$$A_0 :- \ \alpha_1, \alpha_2, \ldots, \alpha_k.$$

  where each α_i, $1 \leq i \leq k$ is either a constraint or an atom.
- A CLP($\mathcal{R}$) program is a finite collection of rules.

CLP by example

The following program defines the relation $sumto(n, s)$ where

$$s = \sum_{1 \leq i \leq n} i$$

for natural numbers n.

```
sumto(0,0).
sumto(N,S) :- N >= 1, N <= S, sumto(N-1,S-N).
```

CLP by example (2)

```
sumto(0,0).
sumto(N,S) :- N >= 1, N <= S, sumto(N-1,S-N).
```

- The query `S <= 3, sumto(N, S)` gives rise to three answers:
  $(N = 0, S = 0), (N = 1, S = 1), (N = 2, S = 3)$.
- Computation sequence for $(N = 2, S = 3)$:

$$S \leq 3, sumto(N, S).$$

$$S \leq 3, N = N_1, S = S_1, N_1 \geq 1, N_1 \leq S_1, \\ sumto(N_1 - 1, S_1 - N_1).$$

$$S \leq 3, N = N_1, S = S_1, N_1 \geq 1, N_1 \leq S_1, \\ N_1 - 1 = N_2, S_1 - N_1 = S_2, N_2 \geq 1, N_2 \leq S_2 \\ sumto(N_2 - 1, S_2 - N_2).$$

$$S \leq 3, N = N_1, S = S_1, N_1 \geq 1, N_1 \leq S_1, \\ N_1 - 1 = N_2, S_1 - N_1 = S_2, N_2 \geq 1, N_2 \leq S_2 \\ N_2 - 1 = 0, S_2 - N_2 = 0.$$

Comparison to logic programming

- Can the power of CLP be obtained by making simple changes to LP systems [JM94]?
- In other words, can predicates in LP be regarded as meaningful constraints?

```
add(0, N, N).
add(S(N), M, S(K)) :- add(N, M, K)
```

- The query `add(N, M, K), add(N, M, S(K))` runs forever in a conventional LP system:
  - A global test for the satisfiability of the two *add* constraints is not done by the LP machinery.

Davis-Putnam-Logemann-Loveland (DPLL)

- DPLL is a decision procedure for the boolean satisfiability problem
- Modern DPLL-based SAT solvers feature:
  - unit propagation
  - heuristics for selecting decision variables
  - 2-literal watching
  - clause learning
  - backjumping

Solvers for quantifier-free theories

Given a quantifier-free theory $\mathcal{T}$, a $\mathcal{T}$-solver decides the satisfiability of finite sets of atoms of $\mathcal{T}$.

Decision procedures for quantifier-free theories

- Decide a boolean combination Φ of atoms of $\mathcal{T}$ by combining a SAT solver with a $\mathcal{T}$-solver.
- Transform Φ into Φ_0 by replacing atoms $\phi_1 \ldots \phi_t$ with propositional variables $p_1 \ldots p_t$
- A valuation b for Φ_0 is a mapping from propositional variables to $\{0, 1\}$
- Define set of atoms Γ_b such that:
  - $\Gamma_b = \{\gamma_1 \ldots \gamma_t\}$
  - $\gamma_i = \phi_i$ if $b(p_i) = 1$
  - $\gamma_i = \neg\phi_i$ if $b(p_i) = 0$
- Φ is satisfiable if there exists b that satisfies Φ_0 and such that Γ_b is consistent in $\mathcal{T}$.

DPLL($\mathcal{T}$)

- DPLL($\mathcal{T}$) is a framework which leverages the DPLL procedure and a $\mathcal{T}$-solver.
- Solver must support:
  - updating the state by asserting new atoms
  - checking consistency of current state
  - backtracking
  - producing explanations for conflicts (an inconsistent subset of atoms asserted in current state)
- Solver can optionally implement theory propagation, but:
  - it must produce an explanation Γ for an implied atom γ, where Γ is a subset of atoms asserted in current state such that $\Gamma \models \gamma$.

DPLL($\mathcal{T}$) example

Consider the following simple example formula Φ in quantifier-free linear arithmetic:

$$(x + y \geq 1 \vee x + y \leq -5) \wedge (x = -1) \wedge (y = -2)$$

Conventions

In the following, we assume that:

- The solver is initialized for a fixed formula Φ
- $\mathcal{A}$ denotes the set of atoms occurring in Φ
- α denotes the set of atoms asserted so far.

Interface for $\mathcal{T}$-solver

We assume that the following API is implemented by the solver:

- `Assert`(γ): assert atom γ in current state.
  - if it returns *ok*, γ is inserted into α
  - if it returns *unsat*(Γ), $\alpha \cup \{\gamma\}$ is inconsistent and $\Gamma \subseteq \alpha$ is an explanation.
- `Check()`: check whether α is consistent
  - if it returns *ok*, α is consistent, and a new checkpoint is created.
  - if it returns *unsat*(Γ), α is inconsistent and $\Gamma \subseteq \alpha$ is an explanation
- `Backtrack()`: backtrack to the last checkpoint
- `Propagate()`: perform theory propagation
  - it returns a set $\{\langle \Gamma_1, \gamma_1 \rangle, \ldots, \langle \Gamma_t, \gamma_t \rangle\}$ where $\Gamma_i \subseteq \alpha$ and $\gamma_i \in \mathcal{A} \setminus \alpha$, such that $\Gamma_i \models \gamma_i$ for $1 \leq i \leq t$.

Remarks on the interface for $\mathcal{T}$-solver

- `Assert`(γ) must be sound but need not be complete: it can return *ok* even if $\alpha \cup \{\gamma\}$ is inconsistent.
- `Check()` must be sound and complete.

$\Longrightarrow$ Several atoms can be asserted in a single "batch"

Quantifier-free linear arithmetic

A quantifier-free linear arithmetic formula is a first-order formula with atoms:

- either propositional variables
- or of the form

  $$a_1 x_1 + \ldots + a_n x_n \bowtie b$$

  where $a_1, \ldots, a_n$ and b are rational numbers, $x_1, \ldots, x_n$ are real (or integer variables), and $\bowtie \in \{=, \leq, <, >, \geq, \neq\}$.

Linear-arithmetic solvers for DPLL($\mathcal{T}$)

Common approach: solvers based on incremental versions of the Simplex method

- Implemented in Yices, Simplics, MathSat
- Solver state includes a Simplex tableau derived from assertions
- The tableau can be seen as a set of equalities

$$x_i = b_i + \sum_{x_j \in \mathcal{B}} a_{ij} x_j, \qquad x_i \in \mathcal{N}$$

  where $\mathcal{B}$ and $\mathcal{N}$ are disjoints sets of basic and non-basic variables.
- Additional constraints are imposed, such as non-negativity of slack variables

Incremental Simplex method: pivoting

- `Pivot(`x_r, x_s`)`: swap basic variable x_r and non-basic variable x_s such that $a_{rs} \neq 0$, by replacing

$$x_r = b_r + \sum_{x_j \in \mathcal{N}} a_{rj} x_j$$

with

$$x_s = -\frac{b_r}{a_{rs}} + \frac{x_r}{a_{rs}} - \sum_{x_j \in \mathcal{N} \setminus \{x_s\}} \frac{a_{rj} x_j}{a_{rs}}$$

and eliminating x_s from the rest of tableau by substitution.

Incremental Simplex method operations

- To assert an atom γ of the form $t \geq 0$:
  - Normalize γ by substituting in t basic variables by non-basic ones.
  - Check whether resulting atom $t' \geq 0$ is satisfiable by maximizing t' using the tableau.
- Asserting equalities and strict inequalities follow same principle
- To backtrack:
  - Remove rows from the tableau

Performance issues in incremental Simplex solvers

Asserting and backtracking have significant cost, due to:

- pivoting in assertions
- frequent addition and removal of rows
- frequent creation and deletion of slack variables

Important remarks for performance

- Generating minimal explanations is critical
- Theory propagation must be done cheaply:
  Full propagation is too expensive, heuristic propagation is superior
- Zero detection is expensive
  $\implies$ Convert $t \neq 0$ into $(t > 0) \vee (t < 0)$

A different solver for linear arithmetic

We now proceed to describe a solver for linear arithmetic [DdM06] with the following properties:

- It is still based on the Simplex method
- It reduces the overhead of the incremental Simplex approach

Preprocessing

Idea: avoid incremental Simplex methods by rewriting formula Φ into an equisatisfiable formula $\Phi_A \wedge \Phi'$, where:

- Φ_A is a conjunction of linear equalities
- All atoms of Φ' are *elementary*, i.e. of the form

$$y \bowtie b$$

where y is a variable, b is a rational constant, and $\bowtie \in \{=, \leq, <, >, \geq\}$.

Example transformation

Let Φ be the following formula:

$$\begin{gathered} x \geq 0 \wedge \\ (x + y \leq 2 \vee x + 2y - z \geq 6) \\ \wedge (x + y = 2 \vee x + 2y - z > 4) \end{gathered}$$

Introducing variables s_1 and s_2, it is rewritten to $\Phi_A \wedge \Phi'$ as:

$$\begin{gathered} (s_1 = x + y \wedge s_2 = x + 2y - z) \wedge \\ (x \geq 0 \wedge (s_1 \leq 2 \vee s_2 \geq 6) \wedge (s_1 = 2 \vee s_2 > 4)) \end{gathered}$$

Properties of the rewritten formula

- Formula Φ_A can be written in matrix form as:

  $$Ax = 0$$

  where A is an $m \times n$ matrix with linearly independent rows, and $x \in \mathbb{R}^n$.
- The matrix A is fixed at all times and represents the equations

  $$s_i = \sum_{x_j \in V} c_j x_j$$

  where V is the set of variables of the original formula Φ.

Properties of the rewritten formula (2)

- Checking satisfiability of Φ amounts to finding x such that $Ax = 0$ and x satisfies Φ'.
  $\implies$ It suffices to decide the satisfiability of a set of elementary atoms Γ in linear arithmetic *modulo* the constraints $Ax = 0$.
- If the elementary atoms are only equalities and non-strict inequalities, the problem consists of finding $x \in \mathbb{R}^n$ such that

  $$Ax = 0 \text{ and } l_j \leq x_j \leq u_j \quad \text{for } j = 1, \ldots, n$$

  where l_j is either $-\infty$ or a rational number, and u_j is either $+\infty$ or a rational number.

A basic solver

- We first consider a solver that handles only equalities and non-strict inequalities with real variables.
- The solver state includes:
  - A tableau derived from A, which we can represent as:

$$x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j \qquad x_i \in \mathcal{B}$$

  - Lower and upper bounds l_i and u_i for each x_i
  - A mapping β assigning a rational value to each x_i
- Initially, $l_j = -\infty, u_j = +\infty, \beta(x_j) = 0$ for all j.

Invariants for the mapping β

The mapping β always satisfies the following invariants:

- The bounds on non-basic variables are always satisfied, i.e.

$$\forall x_j \in \mathcal{N}, l_j \leq \beta(x_j) \leq u_j$$

- The mapping always satisfies the constraints $Ax = 0$

Main algorithm

- The main procedure is based on the dual Simplex algorithm and uses Bland's pivot-selection rule, which ensures termination.
- It assumes a total order on the problem variables.
- At a given moment, we assume that the invariants on β hold, but the mapping may not satisfy the bound constraints $l_i \leq \beta(x_i) \leq u_i$ for basic variables.
- Procedure `Check()` looks for a new β that satisfies all constraints.

`Check()` procedure

```
loop
    select smallest basic var. $x_i$ s.t. $\beta(x_i) < l_i$ or $\beta(x_i) > u_i$
    if there is no such $x_i$ then
        return SAT
    else if $\beta(x_i) < l_i$ then
        select smallest non-basic var. $x_j$ s.t.
          $(a_{ij} > 0 \wedge \beta(x_j) < u_j) \vee (a_{ij} < 0 \wedge \beta(x_j) > l_j)$
        if there is no such $x_j$ then
            return UNSAT
        else
            PivotAndUpdate($x_i, x_j, l_i$)
        end if
    else if $\beta(x_i) > u_i$ then
        select smallest non-basic var. $x_j$ s.t.
          $(a_{ij} < 0 \wedge \beta(x_j) < u_j) \vee (a_{ij} > 0 \wedge \beta(x_j) > l_j)$
        if there is no such $x_j$ then
            return UNSAT
        else
            PivotAndUpdate($x_i, x_j, u_i$)
        end if
    end if
end loop
```

Termination of `Check()`

Theorem

Procedure `Check()` always terminates.

Proof sketch:

- There is a unique tableau for any set of basic variables $\mathcal{B}$.
- There is a finite number of possible assignments β for base B_t at t-th iteration.
- The state of the solver at iteration t is the pair $\langle \beta_t, B_t \rangle$, and there are finitely many states reachable from S_0.
- If `Check()` does not terminate, the sequence of states must contain a cycle.
- One can show by contradiction that such a cycle cannot occur.

The correctness of the procedure is a consequence of this theorem.

Generating explanations

If an inconsistency is detected (say, at line 8 of `Check()`), then:

- There is a basic variable x_i s.t. $\beta(x_i) < l_i$
- For all non-basic variable x_j, we have:
  $a_{ij} > 0 \implies \beta(x_j) \geq u_j$ and
  $a_{ij} < 0 \implies \beta(x_j) \leq l_j$
- If we define $\mathcal{N}^+ = \{x_j \in \mathcal{N} \mid a_{ij} > 0\}$ and
  $\mathcal{N}^- = \{x_j \in \mathcal{N} \mid a_{ij} < 0\}$, then, by the invariant for β:
  $\beta(x_j) = u_j$ for all $x_j \in \mathcal{N}^+$ and $\beta(x_j) = l_j$ for all $x_j \in \mathcal{N}^-$
- We therefore have:

$$\beta(x_i) = \sum_{x_j \in \mathcal{N}} a_{ij}\beta(x_j) = \sum_{x_j \in \mathcal{N}^+} a_{ij}u_j + \sum_{x_j \in \mathcal{N}^-} a_{ij}l_j$$

Generating explanations (2)

- We have:

$$\beta(x_i) = \sum_{x_j \in \mathcal{N}^+} a_{ij} u_j + \sum_{x_j \in \mathcal{N}^-} a_{ij} l_j$$

- As $x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j$ holds for all x s.t. $Ax = 0$:

$$\beta(x_i) - x_i = \sum_{x_j \in \mathcal{N}^+} a_{ij}(u_j - x_j) + \sum_{x_j \in \mathcal{N}^-} a_{ij}(l_j - x_j)$$

- We can then derive the implications:

$$\bigwedge_{x_j \in \mathcal{N}^+} x_j \leq u_j \implies \sum_{x_j \in \mathcal{N}^+} a_{ij}(u_j - x_j) \geq 0$$

and

$$\bigwedge_{x_j \in \mathcal{N}^-} x_j \geq l_j \implies \sum_{x_j \in \mathcal{N}^-} a_{ij}(l_j - x_j) \geq 0$$

Generating explanations (3)

- We have:

$$\bigwedge_{x_j \in \mathcal{N}^+} x_j \leq u_j \implies \sum_{x_j \in \mathcal{N}^+} a_{ij}(u_j - x_j) \geq 0$$

and

$$\bigwedge_{x_j \in \mathcal{N}^-} x_j \geq l_j \implies \sum_{x_j \in \mathcal{N}^-} a_{ij}(l_j - x_j) \geq 0$$

- Finally, we derive:

$$\bigwedge_{x_j \in \mathcal{N}^+} x_j \leq u_j \wedge \bigwedge_{x_j \in \mathcal{N}^-} x_j \geq l_j \implies x_i \leq \beta(x_i)$$

- As we also have $\beta(x_i) < l_i$, this is inconsistent with $l_i \leq x_i$
- Therefore we have the (minimal) explanation:

$$\Gamma = \{x_j \leq u_j \mid x_j \in \mathcal{N}^+\} \cup \{x_j \geq l_j \mid x_j \in \mathcal{N}^-\} \cup \{x_i \geq l_i\}$$

Assertion procedures

The `Assert()` function relies on two functions `AssertUpper(`$x_i \leq c_i$`)` and `AssertLower(`$x_i \geq c_i$`)`:

- `AssertUpper(`$x_i \leq c_i$`)`:

  if $c_i \geq u_i$ **then**
      return `SAT`
  else if $c_i < l_i$ **then**
      return `UNSAT`
  else
      $u_i := c_i$
      if x_i non-basic and $\beta(x_i) > c_i$ **then**
          `Update(`c_i`)`
      end if
      return `OK`
  end if

Backtracking

- We only need to store:
  - the value u_i before it is updated by `AssertUpper`
  - the value l_i before it is updated by `AssertLower`
- In particular, we don't store successive βs on a stack: the last β obtained after a successful `Check()` is a model for all previous checkpoints.

Theory propagation

- *Unate propagation*
  - very cheap to implement
  - if bound $x_i \geq c_i$ is asserted, any unassigned atom $x_i \geq c'$ with $c' < c$ is implied.
  - useful in practice
- *Bound refinement*
  - Given a row of tableau:

$$x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j$$

We can refine currently asserted bounds on x_i using bounds on non-basic variables

Example

- Initial state: $A_0 = \{s_1 = -x + y, s_2 = x + y\}$

Example

- Initial state: $A_0 = \{s_1 = -x + y, s_2 = x + y\}$
- Assert $x \leq 4$

Example

- Initial state: $A_0 = \{s_1 = -x + y, s_2 = x + y\}$
- Assert $x \leq 4$
- Assert $-8 \leq x$

Example

- Initial state: $A_0 = \{s_1 = -x + y, s_2 = x + y\}$
- Assert $x \leq 4$
- Assert $-8 \leq x$
- Assert $s_1 \leq 1$

Handling strict inequalities

Lemma

A set of linear arithmetic literals Γ *containing strict inequalities* $S = \{p_0 > 0, \dots, p_n > 0\}$ *is satisfiable iff there exists a rational number* $\delta > 0$ *such that for all* δ' *such that* $0 < \delta' \leq \delta$, $\Gamma_\delta = (\Gamma \cup S_\delta) \setminus S$ *is satisfiable, where* $S_\delta = \{p_1 \geq \delta, \dots, p_n \geq \delta\}$.

- We can replace strict inequalities by non-strict ones if a small enough δ is known
- We treat δ symbolically instead of computing an explicit value

Handling strict inequalities (2)

- Bounds and assignments range over the set $\mathbb{Q}_\delta$ of pairs of rationals
- $(c, k) \in \mathbb{Q}_\delta$ is denoted by $c + k\delta$
- Define operations:

$$\begin{array}{rcl} (c_1, k_1) + (c_2, k_2) & \equiv & (c_1 + c_2, k_1 + k_2) \\ a \times (c, k) & \equiv & (a \times c, a \times k) \\ (c_1, k_1) \leq (c_2, k_2) & \equiv & (c_1 < c_2) \vee (c_1 = c_2 \wedge k_1 \leq k_2) \end{array}$$

where a is a rational number.

Defining δ

If $(c_1, k_1) \leq (c_2, k_2)$ holds in $\mathbb{Q}_\delta$, then we can find $\delta_0 > 0$ such that

$$c_1 + k_1\varepsilon \leq c_2 + k_2\varepsilon$$

is satisfied by all positive $\varepsilon \leq \delta_0$. Define it as:

$$\begin{array}{lcl} \delta_0 = & \frac{c_2 - c_1}{k_1 - k_2} & \text{if } c_1 < c_2 \text{ and } k_1 > k_2 \\ \delta_0 = & 1 & \text{otherwise} \end{array}$$

Defining δ for the general case

More generally, assume we have $2m$ elements of $\mathbb{Q}_\delta$, $v_i = (c_i, k_i), w_i = (d_i, h_i)$ for $1 \leq i \leq m$. If the m inequalities $v_i \leq w_i$ hold in $\mathbb{Q}_\delta$, then there exists $\delta_0 > 0$ such that

$$\begin{array}{rcl} c_1 + k_1\varepsilon & \leq & d_1 + h_1\varepsilon \\ & \vdots & \\ c_m + k_m\varepsilon & \leq & d_m + h_m\varepsilon \end{array}$$

are satisfied by all positive $\varepsilon \leq \delta_0$. We can define:

$$\delta_0 = \min\left\{ \frac{d_i - c_i}{k_i - h_i} \mid c_i < d_i \text{ and } k_i > h_i \right\}$$

Problem and solution conversion

- A problem with strict inequalities can be converted into another without strict inequalities
- Convert $x_i > l_i$ into $x_i \geq l_i + \delta = l'_i$
- Convert $x_i < u_i$ into $x_i \leq u_i - \delta = u'_i$
- The basic solver described previously will give an assignment β' mapping variables to elements of $\mathbb{Q}_\delta$, if the problem is satisfiable
- If $l'_j = (c_j, k_j), u'_j = (d_j, h_j), \beta'(x_j) = (p_j, q_j)$, we already know that there exists $\delta_0 > 0$ such that

$$c_j + k_j\varepsilon \leq p_j + q_j\varepsilon \leq d_j + h_j\varepsilon \qquad \text{for } 1 \leq j \leq n$$

  holds for all positive $\varepsilon \leq \delta_0$.
- Define satisfying assignment $\beta(x_j) = p_j + q_j\delta_0$ for original problem

Integer and mixed integer problems

- The previously described algorithm is not complete if some variables must be integers.
- A *branch and cut* strategy is used to be complete for the integer case. It is the combination of:
  - the branch and bound algorithm
  - a cutting plane generation algorithm

Branch and bound

Consider the problem

$$Ax = 0$$
$$l_j \le x_j \le u_j \text{ for } 1 \le j \le n$$

with the additional condition that x_i is an integer variable for $i \in I \subseteq \{1, \dots, n\}$.

Branch and bound (2)

- Solve the *linear programming relaxation*, i.e. search for a solution in reals
- If relaxation is infeasible, the problem is infeasible too.
- If an assignment β is found that satisfies all integer constraints, we are done.
- If there exists $i \in I$ such that $\beta(x_i) \notin \mathbb{Z}$, then solve (recursively) the two subproblems:

$$S_0 : \quad \begin{cases} Ax = 0 \\ l_j \leq x_j \leq u_j & \text{for } 1 \leq j \leq n \text{ and } j \neq i \\ l_i \leq x_i \leq \lfloor \beta(x_i) \rfloor \end{cases}$$

$$S_1 : \quad \begin{cases} Ax = 0 \\ l_j \leq x_j \leq u_j & \text{for } 1 \leq j \leq n \text{ and } j \neq i \\ \lfloor \beta(x_i) \rfloor + 1 \leq x_i \leq u_i \end{cases}$$

The need for a cutting plane generation algorithm

- If not all integer variables have an upper and a lower bound, branch and bound may not terminate.
- Example:

$$1 \leq 3x - 3y \leq 2$$

  This constraint is unsatisfiable if x and y are integers. A naïve branch and bound algorithm loops on this input.
- W.l.o.g. we assume that all integer variables are bounded.
- The bounds are typically too large, and cutting plane algorithms are needed to accelerate convergence.

Cuts

Assume β is a solution to the LP relaxation P of problem S, but not to S itself. A *cut* is a linear inequality

$$a_1 x_1 + \ldots + a_n x_n \leq b$$

that is not satisfied by β but is satisfied by any element in the convex hull of S.
The cut can be added as a new constraint to S, yielding a problem S'

- that has the same solutions as S
- but whose LP relaxation P' is strictly more constrained than P.

Deriving Gomory cuts

We have:

$$\begin{aligned} x_i - \beta(x_i) &= \sum_{j \in J} a_{ij}(x_j - l_j) - \sum_{j \in K} a_{ij}(u_j - x_j) \\ x_i - \lfloor \beta(x_i) \rfloor &= f_0 + \sum_{j \in J} a_{ij}(x_j - l_j) - \sum_{j \in K} a_{ij}(u_j - x_j) \end{aligned}$$

where

$$\begin{aligned} J &= \{ j \in I \mid x_j \in \mathcal{N}' \wedge \beta(x_j) = l_j \} \\ K &= \{ j \in I \mid x_j \in \mathcal{N}' \wedge \beta(x_j) = u_j \} \\ \mathcal{N}' &= \mathcal{N} \cap \{ x_j \mid l_j < u_j \} \end{aligned}$$

Deriving Gomory cuts (2)

We have:

$$x_i - \lfloor \beta(x_i) \rfloor \quad = \quad f_0 + \sum_{j \in J} a_{ij}(x_j - l_j) - \sum_{j \in K} a_{ij}(u_j - x_j)$$

which holds for all x that satisfies the problem S. Furthermore, for any such x, $x_i - \lfloor \beta(x_i) \rfloor$ is an integer and the following also hold:

$$\begin{aligned} x_j - l_j \geq 0 &\quad \text{for all } j \in J \\ u_j - x_j \geq 0 &\quad \text{for all } j \in K \end{aligned}$$

Deriving Gomory cuts (3)

We consider two cases:

- If $\sum_{j \in J} a_{ij}(x_j - l_j) - \sum_{j \in K} a_{ij}(u_j - x_j) \geq 0$, then:

$$f_0 + \sum_{j \in J} a_{ij}(x_j - l_j) - \sum_{j \in K} a_{ij}(u_j - x_j) \geq 1$$

as $f_0 > 0$ and the left-hand side is an integer. Then we have:

$$\sum_{j \in J^+} a_{ij}(x_j - l_j) - \sum_{j \in K^-} a_{ij}(u_j - x_j) \geq 1 - f_0$$

where $J^+ = \{j \in J \mid a_{ij} \geq 0\}$ and $K^- = \{j \in K \mid a_{ij} < 0\}$.
Equivalently:

$$\sum_{j \in J^+} \frac{a_{ij}}{1 - f_0}(x_j - l_j) + \sum_{j \in K^-} \frac{-a_{ij}}{1 - f_0}(u_j - x_j) \geq 1$$

Deriving Gomory cuts (4)

We apply the same procedure for the other case, and combining the two cases, we obtain:

$$\begin{aligned}\sum_{j\in J^+}\frac{a_{ij}}{1-f_0}(x_j-l_j)+\sum_{j\in J^-}\frac{-a_{ij}}{f_0}(x_j-l_j) & \quad + \\ \sum_{j\in K^+}\frac{a_{ij}}{f_0}(u_j-x_j)+\sum_{j\in K^-}\frac{-a_{ij}}{1-f_0}(u_j-x_j) & \quad \geq \quad 1\end{aligned}$$

which is a *mixed-integer Gomory cut*: it is satisfied by any x that satisfies S, but it is not satisfied by the assignment β (as the left-hand side is equal to 0 in that case).

References

Bruno Dutertre and Leonardo de Moura.
Integrating Simplex with DPLL(T).
Technical Report SRI-CSL-06-01, SRI International, 2006.

Joxan Jaffar and Jean-Louis Lassez.
Constraint logic programming.
In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '87, pages 111–119, New York, NY, USA, 1987. ACM.

Joxan Jaffar and Michael J. Maher.
Constraint Logic Programming: A Survey.
J. Log. Program., 19/20:503–581, 1994.

Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey, and Roland H. C. Yap.
The CLP(R) language and system.
ACM Trans. Program. Lang. Syst., 14:339–395, May 1992.