MIP aspects of OR-tools CP-SAT solver.

Frederic Didier (fdid@google.com)

OR-tools

Open source library for discrete optimization

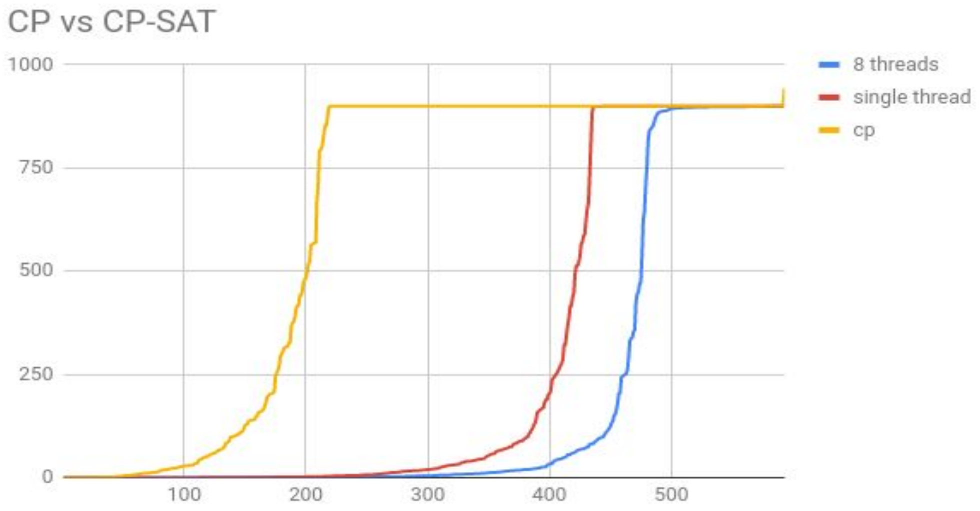
https://github.com/google/or-tools

- Small utilities / algorithms
- Graph algorithms (symmetry, max-flow, min-cost-flow, perfect matching, …)
- LP solver (Glop for simplex, PDLP for first order)
- Common MIP interface (to SCIP, GUROBI, CP-SAT, etc…)
- Vehicle routing solver
- CP-SAT

A Breakthrough: CP vs CP-SAT

Exploiting SAT solvers recent progress: Conflict Directed Clause Learning (CDLC)

Inspired by **Chuffed** and by Peter Stuckey's “Search is dead, long live proof”



Now more like CP-SAT-LP-…

Constraint Programming:

- Rich modeling layer (structure is not lost in the solver)
- Advanced deduction algorithms (mainly for scheduling and routing)

SAT & MaxSAT:

- Core based search
- Model reductions
- Clause Learning

Linear Integer Programming:

- Linear Relaxation + Cuts
- Presolve

Primal heuristics:

- Large Neighborhood Search (LNS)
- Violation based Local Search (from MIP feasibility jump)

Model + blackbox solver

Input: “CpModel” protocol buffer
https://github.com/google/or-tools/blob/stable/ortools/sat/cp_model.proto

Integer variables: $X_i \in [lb_i, ub_i]$, $X_i \in$ int64

Constraints:

- Boolean constraints (on variable are in [0, 1]).
- Linear constraints (with half-reification aka indicator constraints)
- Min/Max, Product, Division, Modulo, Boolean XOR
- Routing: Circuit/Route
- Scheduling: No-Overlap (1d/2d) and cumulative

Difference vs MIP

Main one: Only bounded Integer variables.

Linear constraint and objective with integer coefficients.

Scaling floating point constraint/objective is “easy”:

- CP-SAT can do that automatically given a primal-tolerance
- It is done at the beginning, with clear errors if precision cannot be reached using int64_t coefficients.

Scaling variables automatically not so much…
You can just assume integrality and get feasible solution though.

A simple presolve example

Coefficient strengthening

$A \in [0,10]$ $B \in [0,10]$ $C \in [0,1]$

- Given a linear constraint $3A + 7B + 5C >= 4$

Coefficient strengthening

$A \in [0,10]$ $B \in [0,10]$ $C \in [0,1]$

- Given a linear constraint $3A + 7B + 5C >= 4$
- Because all variables >=0 $3A + 4B + 4C >= 4$

This helps make the LP relaxation stronger

“Big-M” coefficient reduction.

Coefficient strengthening

$A \in [0,10]$ $B \in [0,10]$ $C \in [0,1]$

- Given a linear constraint $3A + 7B + 5C >= 4$
- Because all variables >=0 $3A + 4B + 4C >= 4$
- We can go further $1A + 2B + 2C >= 2$

LP Relaxation even stronger.

Harder to code/explain the algo, related to cuts… but need equivalence

Coefficient strengthening

$A \in [0,10]$ $B \in [0,10]$ $C \in [0,1]$

- Given a linear constraint $3A + 7B + 5C >= 4$
- Because all variables >=0 $3A + 4B + 4C >= 4$
- We can go further $1A + 2B + 2C >= 2$
- In CP-SAT we like Booleans $!C \Rightarrow A + 2B >= 2$

Indicator constraint in MIP
or “enforced constraint” and “enforcement literal” in CP-SAT jargon

Coefficient strengthening

$A \in [0,10]$ $B \in [0,10]$ $C \in [0,1]$

- Given a linear constraint $3A + 7B + 5C >= 4$
- Because all variables >=0 $3A + 4B + 4C >= 4$
- We can go further $1A + 2B + 2C >= 2$
- In CP-SAT we like Booleans $!C => A + 2B >= 2$
- If we have in the model $D \in [0,1]$, $D \Leftrightarrow B == 0$

$$!C \ \& \ D => A >= 2$$

(In our internal LP, this will be later re-linearized to $A + 2B + 2(1-D) >=2$)

Exact LP propagation

CP-SAT is an EXACT solver

Even though LP solver is inexact, we just use its output as a "**hint**".

- We use only int64/int128 arithmetic.
- No epsilon! But have to deal with integer overflow.
- **vs MIP**: simplify the code & complexity a lot.

Same for the cuts, we compute everything with integers.

Ingredient 1 : Integer LP

Bounded integer variables: $X_i \in [lb_i, ub_i]$ $X_i \in$ **int64**

Integer linear objective: minimize $\sum_i obj_i X_i$ $obj_i \in$ **int64**

valid integer linear constraints (initial linearization or cuts) :

$$lhs \le \sum_i coeff_i X_i \le rhs \qquad lhs, rhs, coeff_i \in \textbf{int64}$$

Compared to a MIP solver:

- Everything is integer (int64) and bounded.
- integer-overflow precondition: min/max constraint activity fit on int64.
- The constraints do not need to describe the full problem.

Ingredient 2 : Linear combination of constraints

Given any set of floating point constraint multipliers λ_i

Scale them to integer M_i = round(s * λ_i) with a factor s (we use a power of 2) as large as possible and compute exactly:

$$\sum_{\{\text{positive } Mi\}} M_i * (\text{constraint}_i <= \text{rhs}_i)$$
$$+ \sum_{\{\text{negative } Mi\}} M_i * (\text{constraint}_i >= \text{lhs}_i)$$
$$= \text{new\_linear\_terms} <= \text{new\_rhs}$$

Details:

- We compute the new_rhs using int128
- s chosen so that all other coefficients fit on an int64

Ingredient 3: Propagating an integer linear constraint

Canonicalize to: $\sum_i \text{coeff64}_i \, X'_i \text{ <= rhs128}$ $\quad \text{coeff64}_i > 0$

Compute: $\text{min\_activity128} = \sum_i \text{coeff64}_i \, \text{lb64}(X'_i)$

$\text{slack128 = rhs128 - min\_activity128}$

`slack128 < 0:` conflict!

`slack128 >= int64max:` no propagation.

Otherwise, $\forall i$, $\text{new\_ub64}(X'_i) = \text{lb64}(X'_i) + \lfloor \text{slack64} / \text{coeff64}_i \rfloor$

Propagating dual-infeasible LP

- Take dual ray (aka infeasibility proof) as constraint multipliers
- New constraint should give rise to a conflict (i.e. min_activity > rhs).

Note: Even if not conflicting, it will likely have small slack (rhs - min_activity), and can thus propagates bounds...

Propagating dual-feasible LP

Take negated dual LP values (scaled by s) as constraint multipliers:

new_linear_terms <= new_rhs [from multipliers]
s * objective_linear_term <= s * objective_var [objective definition]

terms + s * minus_objective_var <= new_rhs

Normal CP-SAT propagation of this new constraint:

- Push LP lower bound of objective var
- Or is infeasible if new objective lb > best_known_solution_objective.
- Push upper (or lower) bound of variables (i.e. reduced cost fixing !)

Generic LP Cuts

Goal

Given:

- Integer LP
- Current LP solution for each variables

Derive a new and valid integer linear constraint (linear `terms <= rhs`) that is

- violated by the current LP solution (`LP solution activity > rhs`)
- Has good efficacy = `violation / ‖constraint‖`$_2$

Note that only generic MIP cut here: No TSP, scheduling cuts, etc…

Example: clique cut

X, Y, Z integer variable in [0,1], current LP solution 0.5 for each.

Constraints:

```
X + Y <= 1,
X + Z <= 1,
Y + Z <= 1
```

This could be the optimal to maximize X + Y + Z for instance.

But, given the integrality constraint (ignored by the LP), we can derive

X + Y + Z <= 1, this is a violated cut (violation = 0.5).

General MIP cut framework

Almost all cuts (gomory, MIR, zero half, cover, flow, …) follow this:

1. Aggregate many constraints from the integer lp into one (terms <= rhs)
2. From such single constraint, rewrite it slightly with complementation, implied bounds, and using positive variables.
3. Apply a super-additive function f() to get the cut.
4. Rewrite everything in term of the original variables (i.e. undo step 2).

Aggregation

Same as for explanation, take linear-combination of constraints, but use slacks

$\text{lhs} \leq \sum_i \text{coeff}_i X_i \leq \text{rhs} \quad \Rightarrow \quad \sum_i \text{coeff}_i X_i - S = 0$

new slack integer variable $S \in [\text{lhs}, \text{rhs}]$

Why slacks?

- Final constraint is always tight for LP
- After all steps, it is possible slack still there, it will be substituted back, and that can lead to stronger cut.

Different aggregation heuristics

- No aggregations ! try each cut heuristic on row or -row.
- **MIR** heuristic: combine small number (<= 6) of constraints to eliminate fractional variables.
- **Chvatal-Gomory** (get multipliers λ_i from $\mathbf{B}^{-1}.\mathbf{e}_j$)
  Should lead to single variables not at its bound in aggregated equation!
- **Zero-half** cuts (only use +1/-1 multipliers, heuristic mod 2)
  Idea is to get odd rhs, but even coeff for important variables.
- **Clique** (Weighted Bron Kerbosch max-clique enumeration)

Linear equality rewriting (heuristics too)

Propagate/presolve: minor impact but still useful.

rewrite using positive variables: X' = X - lb   X' ∈ [0, range = ub - lb]

Maybe complement some variables: X = range - X^c

Maybe use some implied bounds (Bool => X >= v):

X = v * B + (X - v * B) = v * B + S, S ∈ [0, range]

This is especially powerful if B already in the constraint!

Make term more "integral": 1 * X[0, 10'000] → 100 * Y[0, 100] → 10'000 * Z[0, 1]

Super-additive function f()

- $f: \mathbb{Z} \to \mathbb{Z}$
- $f(x) + f(y) \le f(x + y)$

This is nice, because:

$$\sum_i \mathrm{coeff64}_i \; X_i = \mathrm{rhs128}, \qquad X_i \text{ positive}$$

$$
\begin{array}{ll}
f(\sum_i \mathrm{coeff64}_i \; X_i) = f(\mathrm{rhs128}) & [\ \le \text{ works too}] \\
\sum_i f(\mathrm{coeff64}_i \; X_i) \le f(\mathrm{rhs128}) & [\ \text{super-additivity}\ ] \\
\sum_i f(\mathrm{coeff64}_i) \; X_i \le f(\mathrm{rhs128}) & [\ X_i \text{ positive integer}\ ]
\end{array}
$$

This is the step that goes from “tight constraint” to “violated constraint”

Clique exemple revisited (a bit artificial)

- Recall X + Y <= 1,
  X + Z <= 1,
  Y + Z <= 1
- We can sum them all: 2X + 2Y + 2Z <= 3
- And apply $f(x) = \lfloor x / 2 \rfloor$ (this is super-additive)
- We get X + Y + Z <= 1

Note that this is the same f() used by zero-half cuts.

Cover cut (or Knapsack cut) example

6X + 4Y + 10Z <= 9 (X=1.0, Y=0.5, Z=0.2, all in [0,1])

X and Y form a "**cover**" (i.e. 6 + 4 > 9)

Lets **complement** the cover: $-6X^c -4Y^c + 10Z \le -1$

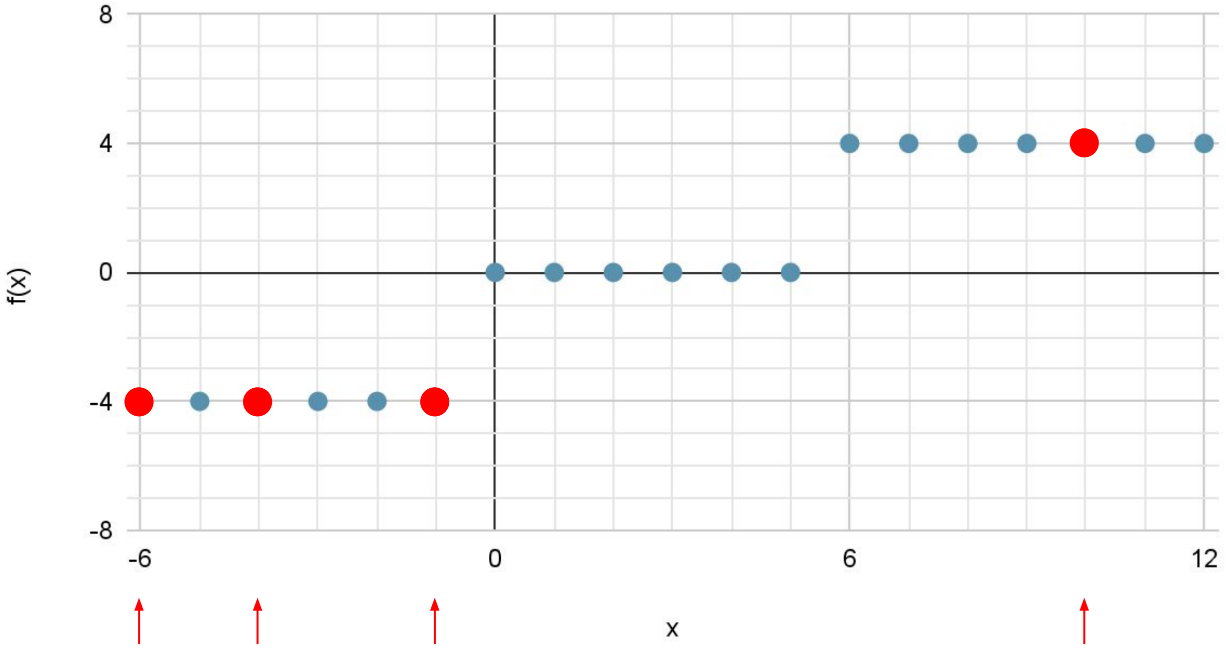
Apply $f(x) = \lfloor x/6 \rfloor$

we get: $-X^c -Y^c + Z \le -1$ (note the lifting of Z)

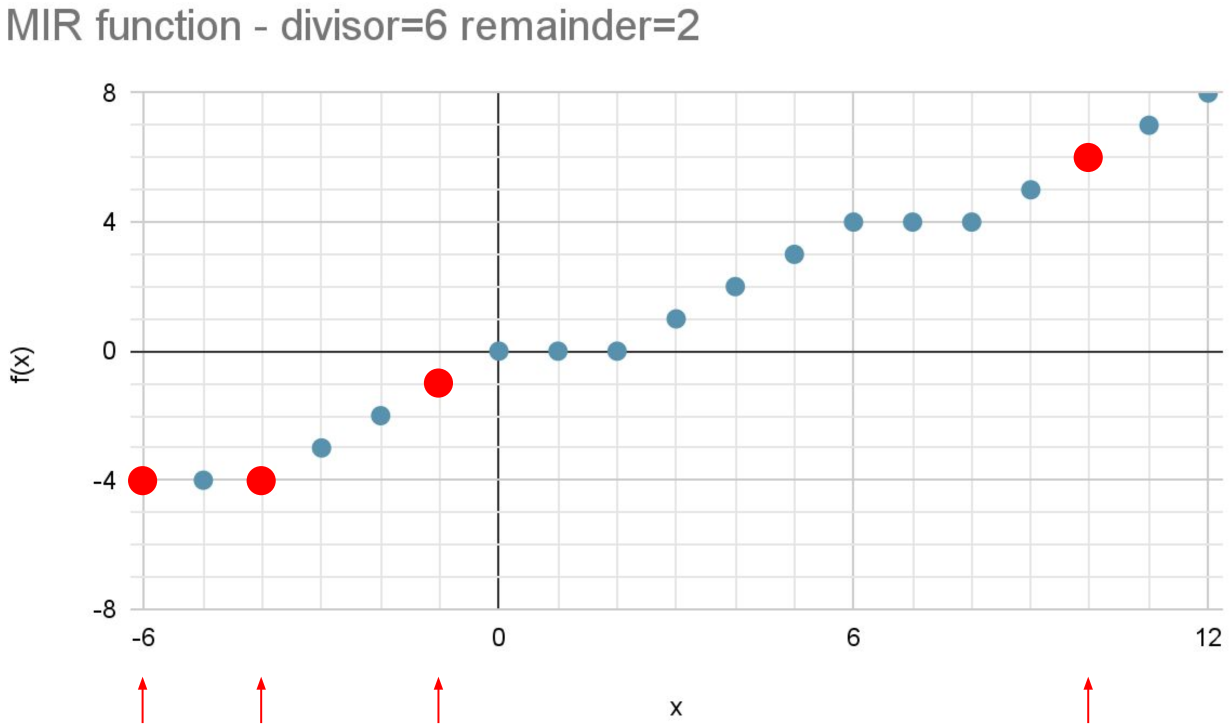
substituting back: X + Y + Z <= 1 (violation = 0.7 !)

Various choices for f()

Division / 6 (rescaled)



Various choices for f()



MIR cuts

For **cover**, we complement so that all coeff positive.

For **MIR**, we complement so that lp value of each term is smallest.

Then, try to take as divisor for f(), coefficients of terms with large lp values

We define **remainder = rhs % divisor** and **scale = divisor - remainder**

We use MIR super-additive function:

$$f(x) = scale * \lfloor x/div \rfloor + \max(0, (x \% div) - remainder)$$

Various other tricks

- When choosing f() we want final coeff to be small. So we use slightly more complex MIR function so that `f(divisor) = scale` stay small.
- Once f() chosen, we can try other possible complementation
- We can drop small terms rather than applying f() to them.
- We can also try to use different implied bounds once f() is chosen.
- …

Probably still a lot of room for improvement in that code !!
And still other heuristic to write (path mixing cuts)

Real example from log on beasleyC2.mps

```
INPUT:
coeff=    1454428938435252 lp=0.888889 range=9
coeff=   11635431507481974 lp=0.888889 range=9
coeff=    1454428938435255 lp=0.496158 range=1
coeff=    1454428938435254 lp=0.503842 range=1
coeff=    1454428938435254 lp=0.496158 range=1
coeff=    1454428938435255 lp=0.547352 range=1
coeff=    1454428938435254 lp=0.315313 range=1
coeff=    1454428938435254 lp=0.728197 range=1
coeff=  -11635431507481978 lp=0.111111 range=1
coeff=   -93083452059856042 lp=0.111111 range=1
coeff=-1060278696119297370 lp=0.098765 range=1
coeff=    2908857876870509 lp=0.95649  range=1
…
coeff=  -34906294522445990 lp=0          range=9
coeff=                 -16 lp=0          range=9
…
<=      -97446738875161748
```

```
f():  Div      1060278696119297370
      Rem       962831957244135622
      scale                     8

CUT:  coeff=-1 lp=0.111111 range=1
      coeff=-7 lp=0.111111 range=1
      coeff=-8 lp=0.098765 range=1
      …
      coeff=-3 lp=0        range=9
      coeff=-1 lp=0        range=9
      …
      <= -8
```

Questions?