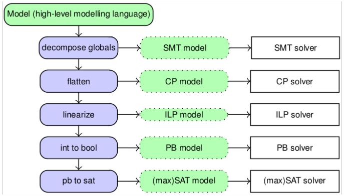
L07: Solving technologies and encodings



Prof. Tias Guns and Dr. Dimos Tsouros

KU LEUVEN

Partly based on slides from Pierre Flener, Uppsala University.

Solvers

You formulated a combinatorial problem in a high-level modeling language...

Now, *which solver* should you use?

Examples (Solving technologies)

With general-purpose solvers, taking model and data as input:

- Boolean satisfiability (SAT)
- Pseudo-Boolean solving (PB)
- (Mixed) Integer Linear Programming (IP and MIP)
- SAT (resp. optimisation) Modulo Theories (SMT and OMT)
- Constraint programming (CP)
- …

Examples (Methodologies, *usually without* modelling and solvers)

- Dynamic programming (DP)
- Greedy algorithms
- Local search (LS)
- Genetic algorithms (GA)
- …

How to Compare Solving Technologies?

Specification language:

- What types of decision variables are available?
- What types of constraints are available?
- Can there be an objective function?

Guarantees:

- Are its solvers exact, given enough time:
  will they prove unsatisfiability? prove optimality? find all solutions?
- If not, is there an approximation ratio for the solution quality?

Features:

- In which application areas has the technology been successfully used?
- Does the solving technology align well with this type of problem?
- Can the modeller influence the search process? If yes, then how?

How Do Solvers Work? (Hooker, 2012)

Definition (Solving = Search + Inference + Relaxation)

- Search: Explore the space of candidate solutions.
- Inference: Reduce the space of candidate solutions.
- Relaxation: Exploit solutions to easier problems.

Definition (Systematic Search)

Progressively build a solution, and backtrack if necessary.
Use inference and relaxation to reduce the search effort.

Systematic search is used in most SMT, CP, ILP/MIP, PB and SAT solvers.

How to model in a specific solvers' input language?

Every solver has their own input language.

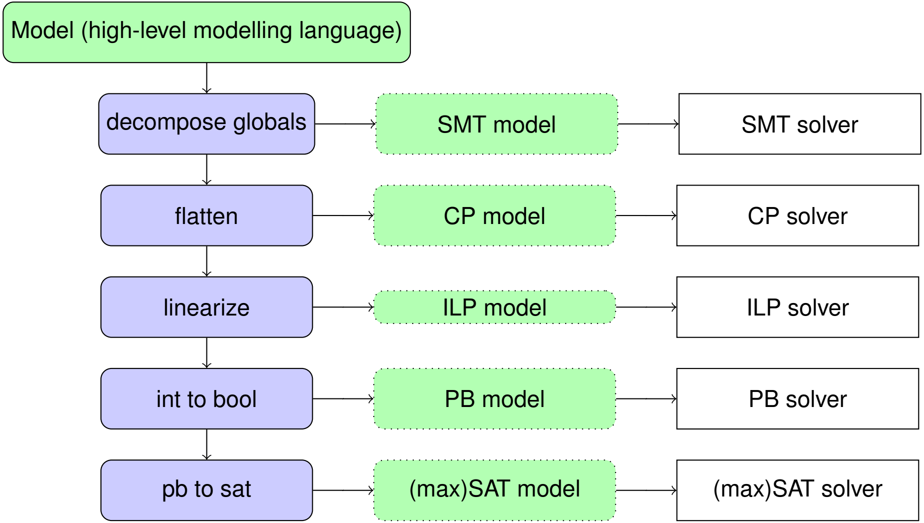
Different communities have different 'standard' input languages.

- SAT: DIMACS format
- Pseudo-Boolean: OPB format
- ILP/MIP: MPS format
- SMT: SMT-LIB format

The CP community does not really have a standard input language (due to the large variety of global constraints possible), BUT it has:
solver-independent modelling languages

How to go from a high-level modelling language to a specific solver input?
Through transformations...

From Model to Model to Solver: transformations

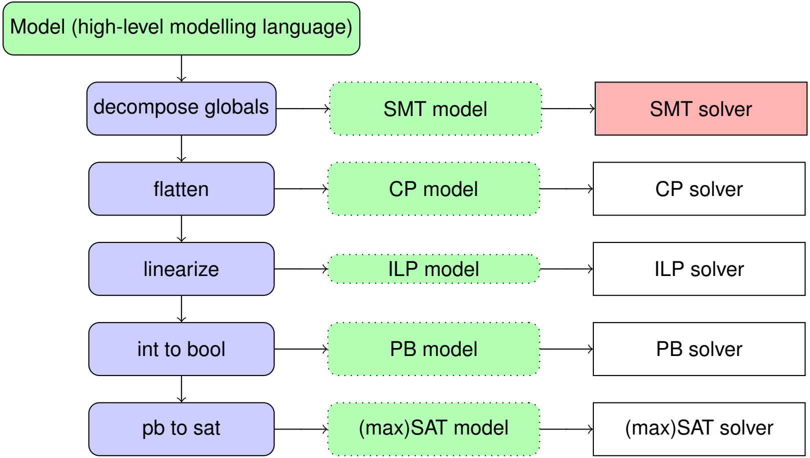


Objectives

An overview of some solving technologies:

- to understand their advantages and limitations;
- to help you choose a technology for a particular model;
- to help you encode and adapt a model to a particular technology.

Overview


Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

SAT Modulo Theories (SMT) and OMT

Modelling Language:

- Language of SAT: Boolean decision variables and clauses.
- Several theories extend the language, such as bit vectors, uninterpreted functions, or linear integer arithmetic.
- SMT is only for satisfaction problems.
- OMT (optimisation modulo theories) extends SMT.

Definition

A theory

- defines types for decision variables and defines constraint predicates;
- is associated with a sub-solver for any conjunction of its predicates.

Different SMT or OMT solvers may have different theories.

LIA

Example: theory of Linear Integer Arithmetic (LIA) (variables can be unbounded for SMT solvers!)

Mathematical Formulation

$$(x \geq 0)$$
$$(y \leq 0)$$
$$(x = y + 1) \vee (x = 2 \cdot y)$$
$$(x = 2) \vee (y = -2) \vee (x = y)$$

SMT-LIB format

```
(>= x 0)
(<= y 0)
(or (= x (+ y 1)) (= x (* 2 y)))
(or (= x 2) (= y -2) (= x y))
```

Satisfiability Modulo Theories (SMT)

- Determines the satisfiability of a first-order logical formulas over (one or more) background theories
- if SAT: return SAT + a solution to the *theory* problem
  if UNSAT: return UNSAT
- there is a standardized language for many different theories, that all SMT solvers accept: the SMT-LIB language

Typical application areas

- Formal Verification of hardware and software
- Model Checking, and Program Analysis
- Automated Reasoning, Theorem Proving

Boolean abstraction

Separate the theory constraints and create the Boolean skeleton

Example:

$$
\begin{aligned}
&(x \geq 0) \wedge (y \leq 0) \wedge \\
&\big(\,(x = y + 1) \vee (x = 2 \cdot y)\,\big) \wedge \\
&\big(\,(x = 2) \vee (y = -2) \vee (x = y)\,\big)
\end{aligned}
$$

Boolean skeleton: $a \wedge b \wedge \big(c \vee d\big) \wedge \big(e \vee f \vee g\big)$
Theory constraints (each Boolean indicates whether a constraint holds or not):

$$
\begin{aligned}
&a \leftrightarrow (x \geq 0) \ \wedge \ b \leftrightarrow (y \leq 0) \wedge \\
&c \leftrightarrow (x = y + 1) \ \wedge \ d \leftrightarrow (x = 2 * y) \wedge \\
&e \leftrightarrow (x = 2) \ \wedge \ f \leftrightarrow (y = -2) \ \wedge \ g \leftrightarrow (x = y)
\end{aligned}
$$

SMT Solving: DPLL(T)

How it Works (High-Level Overview)
Combines a SAT solver with a theory solver.

- **SAT solver** generates Boolean assignment over the Boolean skeleton;
- **Theory solver** checks consistency of activated theory constraints;
  - if SAT: generate theory-level assignment, return
  - if UNSAT: generate Boolean-level *conflict* between theory constraints, add it to the SAT solver
- repeat.

Theory solvers operate over all (activated) constraints in the theory at once.
Efficient theory solvers are incremental: reuse information from previous checks.

Example SMT solvers: CVC4, Yices 2, Z3, . . .
Example OMT solvers: OptiMathSAT, Z3

SMT/OMT for CP solving

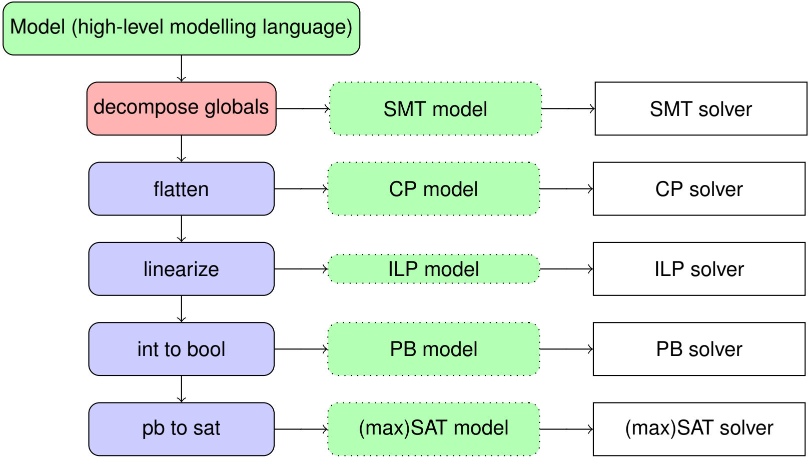
Theory: QF_LIA = ”Quantifier Free, Linear Integer Arithmetic” (and QF_NIA in case of non-linearities)

Supports Bool and Int, as well as logical and arithmetic operators; including nested expressions thereof.

But no global constraints / global functions:

- requires to decompose global constraints

Overview


Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Decomposing global constraints

Rewrite global constraints using (more) primitive constraints.

Example

ALLDIFFERENT$(x_1, \ldots, x_n)$: Its decomposition is a conjunction of $\frac{n\cdot(n-1)}{2}$ disequality constraints:

$$\bigwedge_{i,j \in \{1..n\}, i<j} x_i \neq x_j$$

Example

CUMULATIVE(s,d,r,c): Its time-resource decomposition introduces new Booleans B_{it}, representing if task i (with start time s_i, and duration d_i) is active at time t:

$$\forall t \in \{0..t_{\max} - 1\}, \forall i \in \{1..n\} : \quad B_{it} \leftrightarrow (s_i \leq t) \wedge \neg(s_i \leq t - d_i)$$

The resource constraint at each time t, for n tasks, with r_i being the resource consumption of task i, is expressed as:

$$\forall t \in \{0..t_{\max} - 1\} : \quad \sum_{i \in [1..n]} r_i \cdot B_{it} \leq c$$

Decomposing global functions

The function itself is an integer-valued function. Need to decompose wrt a specific comparison.

Example

$\textsc{Count}(A, v) == res$ (or $\textsc{CountEq}(A, v, res)$): Its decomposition is a sum constraint over all variables:

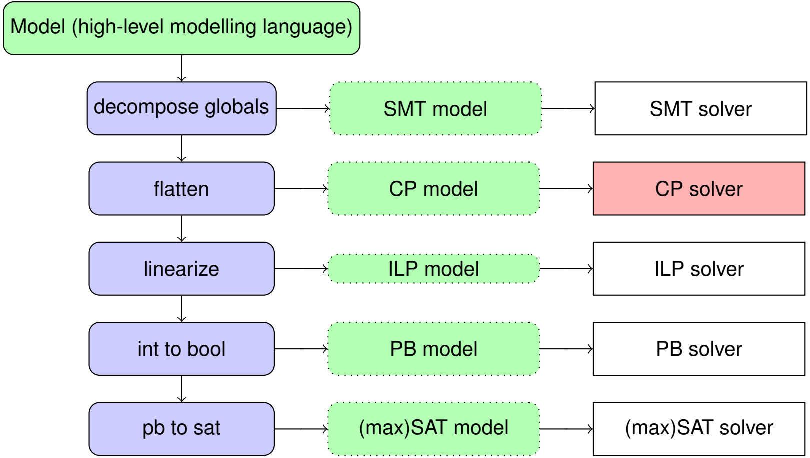
$$\sum_i [A_i = v] = res$$

Example

$\textsc{Element}(Arr, idx) == res$ (or $\textsc{ElementEq}(Arr, idx, res)$, or $Arr[idx] == res$): Its decomposition is a list of implications, specifying that if the index has a given value, then the respective value from the array must be equal to the resulting variable *res*:

$$\forall i \in \{0..n-1\}, \quad (\mathrm{idx} = i) \rightarrow (\mathrm{Arr}_i \ = \ \mathrm{res})$$

Overview



Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Constraint Programming (CP)

- Solves combinatorial optimisation problems with finite-domain variables
- Includes logical, arithmetic and specialised *global constraints*
- Solves both satisfaction and optimisation problems

How it Works (High-Level Overview)

- **Propagation** each constraint reduces the domains of the variables involved as much as possible until no domain can be further reduced;
- **Systematic search** the solver chooses a variable and branches over each of its remaining values

Typical Applications

- Scheduling, Timetabling, Assignment problems
- Routing Problems, Packing problems, esp. with side-constraints
- Puzzles and Games, Configuration Problems

Constraint Programming (CP)

Modelling Language = flat list of (supported) constraints

- Variables: Boolean, integer (finite-domain); a few solvers support sets, floats even graphs
- Logic, arithmetic and **global** constraints
- For satisfaction problems and optimisation problems.

Many solvers There is no standard input format for CP solvers... two things come close:

- XCSP3: an XML format, contrary to most solvers it allows for some form of nesting and many global constraints
- FlatZinc: an intermediary 'flat' predicate list produced by MiniZinc, but creates multiple auxiliary variables and no standard constraint naming (can differ for different solvers)

Domains

Definition

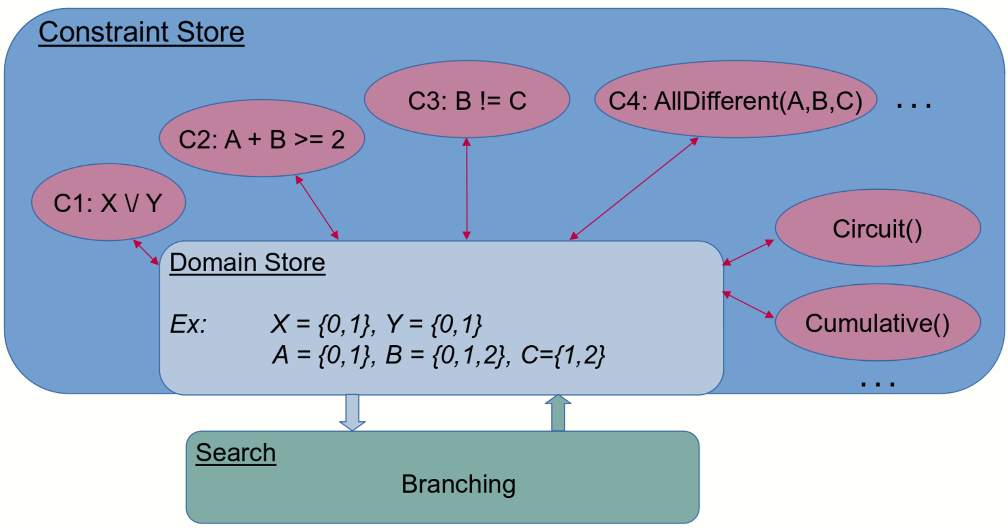
The domain of a decision variable v, denoted here by $\text{dom}(v)$, is the set of values that v can still take during search:

- The domains of the decision variables are reduced by search and by inference (see the next two slides).
- A decision variable is said to be fixed if its domain is a singleton.
- Unsatisfiability occurs if the domain of a decision variable goes empty.

Note the difference between:

- a domain as a technology-independent declarative entity when modelling;
- a domain as a CP-technology procedural data structure when solving.

CP solver structure



CP Solving

Tree Search, upon initialising each domain as in the model:

Satisfaction problem:

1. Perform propagation inference.
2. If the domain of some decision variable is empty, then backtrack.
3. If all decision variables are fixed, then we have a solution.
4. Select a non-fixed decision variable v,
   partition its domain into two parts π_1 and π_2, and make two branches:
   one with $v \in \pi_1$, and the other one with $v \in \pi_2$.
5. Recursively explore each of the two branches.

Optimisation problem: when a feasible solution is found at step 3, first add the constraint that the next solution must be better and then backtrack.

CP Inference

Definition

A propagator for a constraint γ deletes from the domains of the variables of a γ-constraint the values that cannot be in a solution to that constraint.

Examples

- For $x < y$: when $\text{dom}(x) = \{1..4\}$ and $\text{dom}(y) = \{-1..3\}$, delete $\{3, 4\}$ from $\text{dom}(x)$ and $\{-1..1\}$ from $\text{dom}(y)$.
- For $\textsc{AllDifferent}(x, y, z)$: when $\text{dom}(x) = \{1..3\}$ = $\text{dom}(y)$ and $\text{dom}(z) = \{1..4\}$, delete 1 and 3 from $\text{dom}(z)$ so that it becomes the non-range $\{2, 4\}$.

Propagation of constraints is executed until **fixed-point**: no constraint can reduce the current domains further.

Strategies and Improvements

Search Strategies:

- On which decision variable to branch next?
- How to partition the domain of the chosen decision variable?
- Which search (depth-first, breadth-first, ...) to use?

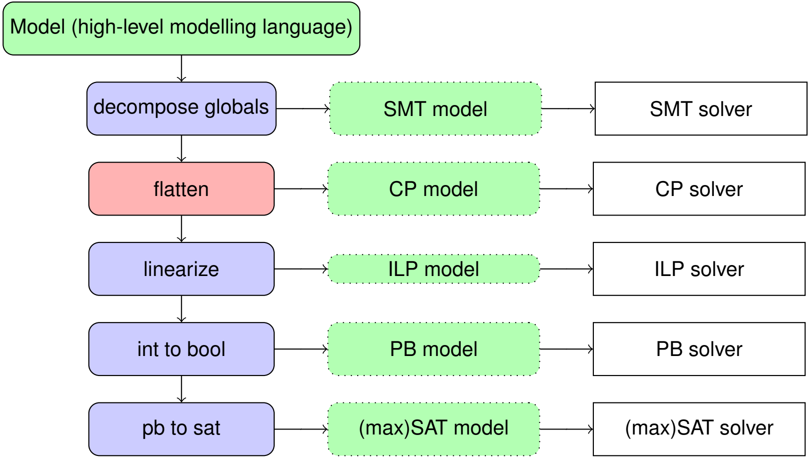
Improvements:

- Propagators, including for global constraints and global functions.
  Not all impossible domain values need to be deleted: there is a compromise between algorithm complexity and achieved inference.
- Partition the chosen domain into at least two parts.
- Domain representations.
- Order in which propagators are executed (and re-actived when a variable changes)
- ...

CP Solving

- Guarantee: exact, given enough time.
- White-box: within a solver one can design one's own propagators and search strategies, or choose among predefined ones.
- Successful application areas:
  - Configuration
  - Scheduling
  - Personnel rostering and timetabling
  - rich vehicle routing
  - …

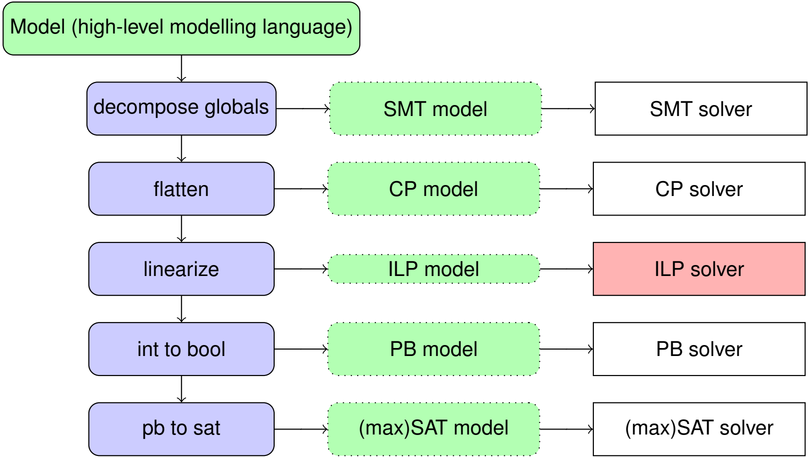
Overview


Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

From CP Language to CP Solver: flattening steps

0. Decompose Unsupported Globals (see previous part)
1. **Push down negation**
   - Simplifies later code by eliminating 'negation' operator, afterwards only in front of Boolean variable
   - Example: $\neg(x \wedge y)$ becomes $(\neg x \vee \neg y)$ and $\neg(a > b)$ becomes $(a \leq b)$
2. **Normalize and simplify expressions**
   - Eliminates unnecessary 'nested' expressions, avoids auxiliary variables later
   - Example: $(x \rightarrow (y \vee z))$ becomes $(\neg x \vee y \vee z)$ and $(a - (b + 2c))$ becomes $(a - b - 2c)$
3. **Unnest arguments using auxiliary variables**
   - Because CP solvers only accept a list of constraints over variables
   - Example: Rewrite $x \vee (a + b \geq 2)$ by:
     - Introduce auxiliary variable w (here: Boolean)
     - Add new constraint to solver: $w = (a + b \geq 2)$
     - Rewrite $(x \vee (a + b \geq 2))$ to $(x \vee w)$
   - Similarly for $(a + (b * c) \geq 0)$: $(w = b * c) \wedge (a + w \geq 0)$

Overview



Model (high-level modelling language)
decompose globals
flatten
linearize
int to bool
pb to sat
SMT model
CP model
ILP model
PB model
(max)SAT model
SMT solver
CP solver
ILP solver
PB solver
(max)SAT solver

Integer Linear Programming (ILP)

- Solves combinatorial optimisation problems where variables are constrained to integer values (including 0/1 variables, e.g. Booleans)
- Formulated using **linear objective functions** and **linear constraints**
- There is also MIP: *Mixed* IP, involving both integer and continuous variables.

How it Works (High-Level Overview)

- **Relaxation** relax the integer constraints to solve a linear program, providing a lower/upper bound
- **Branch-and-Bound** systematically explore branches by dividing the search space and applying bounds to prune infeasible solutions

Typical Applications

- Production planning, Supply chain optimisation
- Vehicle Routing, Network design problems
- Facility location, Scheduling, Workforce allocation

Integer (Linear) Programming (IP = ILP)

Modelling Language:

- Only integer decision variables.
- A set of linear equality and inequality constraints (note: no disequality $\neq$).
- For optimisation problems: linear objective function.

Example

- Integer decision variables: p, q
- Constraints:

$$\begin{aligned} p &\geq 0 \\ q &\geq 0 \\ p + 2 * q &\leq 5 \\ 3 * p + 2 * q &\leq 9 \end{aligned}$$

- Objective: maximize $3 * p + 4 * q$

IP Solving

Basic Idea = Relaxation:

- Polynomial-time algorithms (such as the interior point method and the ellipsoid method) and exponential-time but practical algorithms (such as the simplex method) exist for solving LP models very efficiently.
- Use them for IP by occasionally relaxing an IP model, by dropping its integrality requirement on the decision variables.

Implementations:

- Branch and bound = relaxation + search.
- Cutting-plane algorithms = relaxation + inference.
- Branch and cut = relaxation + search + inference.

Branch and Bound

Tree Search, upon initialising the incumbent (current best solution)'s value to $\pm\infty$:

1. Relax the IP model into an LP model, and solve it.
2. If the LP model is unsatisfiable, then backtrack.
3. If all the decision variables have an integer value in the optimal LP solution: update incumbent to found (coincidentally IP) solution, backtrack.
4. If the objective value of the optimal LP solution is no better than the incumbent, then backtrack.
5. Otherwise, some decision variable v has a non-integer value ρ.
   Make two branches: one with $v \leq \lfloor \rho \rfloor$, and the other one with $v \geq \lceil \rho \rceil$.
6. Create a new search node for each branch, and start exploring one of them

Strategies and Improvements

Search Strategies:

- On which decision variable to branch next?
- Which search node to explore next when backtracking?

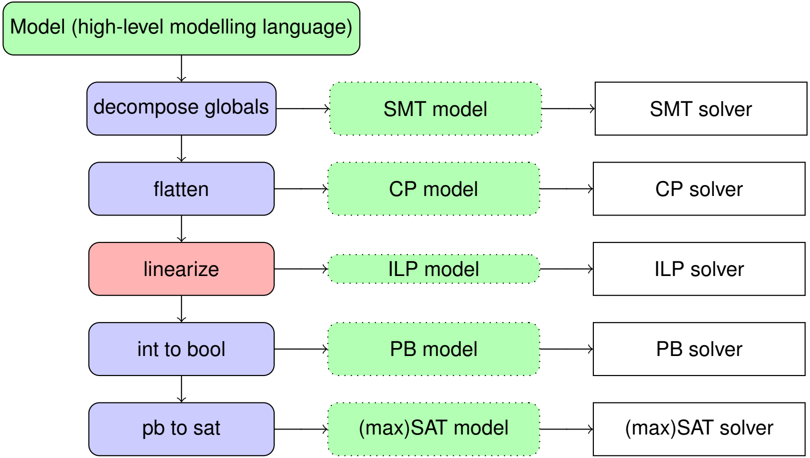
Improvements:

- Cutting planes: Forbid the LP relaxed solution by cutting off a portion of the LP-feasible region that does not contain an integer solution; then compute new LP solution (and bound).
- Decomposition: Split into a master problem and a subproblem, such as by the Benders decomposition.
- Solving the LP relaxation:
  - Primal-dual methods.
  - Efficient algorithms for special cases, such as flows.
- Primal heuristics: getting good feasible solutions quickly (e.g. upper bound)
- …

IP Solving

- Guarantee: exact, given enough time.
- Mainly black-box: limited ways to guide the solving.
- It scales well to thousands of variables.
- Any combinatorial problem can be encoded into IP.
  (but it might require an exponential number of constraints)
- Advantages of ILP solving:
  - Provides both a lower bound and an upper bound on the objective value of optimal solutions, if stopped early.
  - Strong guidance by objective function (through LP solving)
  - Naturally extends to MIP solving.
  - …
- Central method of operations research (OR),
  applied in production planning, linear assignment problems, …

Overview



Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Linearisation

Non-linear constraints have to be 'linearized', e.g. $x \neq y$ and $b \rightarrow x + y \geq 5$. This is possible with modelling idioms that are referred to as big-M formulations; see [Williams, 2013], say.

- The idea is to define a constraint-specific constant M that is big enough, but for performance reasons not too big, and to use such constants in order to implement various logical connectives: see next slide.

'Good' MIP models typically have LP relaxations that are tight: the LP relaxed solution (and hence bounds) are close to the integer solutions.

Adding Big-M constraints typically leads to weaker bounds, to be avoided if possible.

Example (MIP Idiom for Disequality and Disjunction)

How to model $x \neq y$? Rewrite into $(x + 1 \leq y) \vee (y + 1 \leq x)$.
Choose two large constants M_1 and M_2, and introduce a 0-1 variable w so that **the disjunction** is modelled as follows:

$$x + 1 \leq y + M_1 \cdot w$$
$$y + 1 \leq x + M_2 \cdot (1 - w)$$
$$w \in \{0, 1\}$$

- If $w = 0$, then $x + 1 \leq y$ and $y + 1 \leq x + M_2$, which is not constraining if M_2 is large enough.
- If $w = 1$, then $y + 1 \leq x$ and $x + 1 \leq y + M_1$, which is not constraining if M_1 is large enough.

M_1 and M_2 should be large enough to ensure correctness, but as small as possible for performance reasons.

For ILP solvers, decomposing ALLDIFFERENT(X) with inequalities would lead to many Big-M constraints. Can we use a different decomposition?

- *Binary Matrix B*: Define a binary matrix B of shape (n, m), where n is the number of variables and m is the size of the domain $D = \bigcup_{i=1}^{n} \text{dom}(X_i)$.
- Variable B_{ij} indicates whether variable X_i is assigned value D_j. Channel the 2 sets of variables:

$$\sum_{j \in D} (j \cdot B_{ij}) = x_i, \quad \forall i \in \{1..n\}$$

- Each variable can get one value:

$$\sum_{j \in D} B_{ij} = 1, \quad \forall i \in \{1..n\}$$

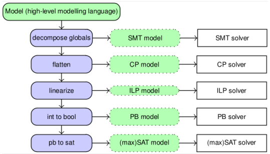
- Each value assigned to *at most one variable*:

$$\sum_{i=1}^{n} B_{ij} \leq 1, \quad \forall j \in D$$

MIP solvers

- SCIP (open-source);
- Cbc (open-source);
- HiGHS (open-source);
- Gurobi Optimizer (commercial: requires a license);
- IBM CPLEX Optimizer (commercial: requires a license);
- FICO Xpress Solver (commercial: requires a license).

Why are we doing this again?

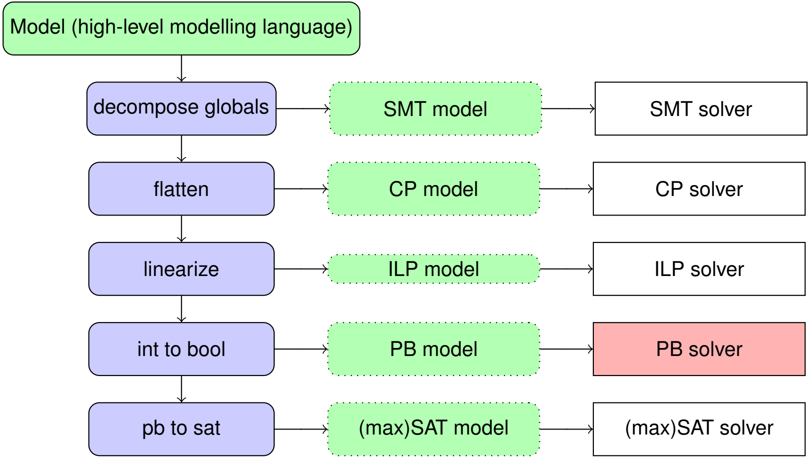


Better understand what a solver CAN do, and what it is GOOD at (or bad).

- CP: recommended if you have many global constraints
- ILP: recommended if many linear constraints and optimizing the 'continuous relaxation' is meaningful
- SMT: recommended if many disjunctive constraints (SAT reasoning + LIA reasoning)

(but there is no rulebook, recommended to try multiple solvers)

Overview



Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Pseudo-Boolean (PB) Optimization

- **Pseudo-Boolean Constraints**: $\sum_{i=1}^{n} a_i x_i \leq b$, where a_i, b are integers and x_i are Boolean variables.
- Solves combinatorial optimization problems where the objective function and constraints are **linear** and only involve **Boolean variables** (0-1 variables)
- Extends SAT by allowing linear inequalities over Boolean variables

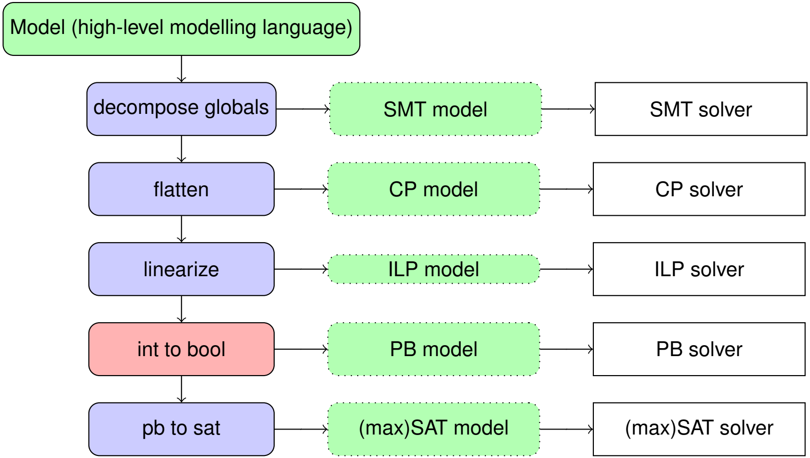
How it Works (High-Level Overview)

1. Either encode into SAT (see next part);
2. Or use native pseudo-boolean cutting plane solving (not covered in lecture)

Typical Applications

- Hardware/software verification, Circuit design
- Resource allocation, Packing problems
- Scheduling, Timetabling, Logistics

Overview


Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Encoding into (pseudo)Boolean constraints

Challenges:

- How to encode an integer variable into a collection of Boolean variables?
- How to encode a constraint on integer variables into (a collection of) constraints on Boolean variables?

Considerations:

- We want few variables.
- We want few constraints, or short constraints.

As usual, there are many possibilities and it is not always clear what the best choice is.

Encoding an Integer Variable

Well-known encodings, described on the next slides:

- Direct (or sparse or one-hot) encoding:
  a Boolean ’equality’ variable for each value in the domain.
- Order encoding:
  a Boolean ’inequality’ variable for each value in the domain.
- Bit (or binary or log) encoding:
  a Boolean variable for each bit in the base-2 representation of the largest domain value.

Direct Encoding of an Integer Variable

Consider an integer variable x with domain $1..n$:

- Create a Boolean variable $b_{[x=k]}$ for all $k \in \{1..n\}$
- The variable $b_{[x=k]}$ is **true** if and only if $x = k$ holds
- Consistency constraint:
  - Variable x has exactly one value from domain: $\sum_{k \in \{1..n\}}(b_{[x=k]}) = 1$
- Example encodings of simple constraints:
  - The constraint $x \neq k$ is encoded as $\neg b_{[x=k]}$.
  - The constraint $x < k$ is encoded as $\bigwedge_{j \in k..n} \neg b_{[x=j]}$.
  - In any constraint, can replace x by $\sum_k k * b_{[x=k]}$

Order Encoding of an Integer Variable

Consider an integer variable x with domain $1..n$:

- Create a Boolean variable $b_{[x \geq k]}$ for all k in $1..(n+1)$.
- The variable $b_{[x \geq k]}$ is **true** if and only if $x \geq k$ holds.
- Consistency constraints:
  - Order: $\bigwedge_{k \in 1..n} \left(b_{[x \geq k+1]} \rightarrow b_{[x \geq k]}\right)$
  - Bounds of domain: $b_{[x \geq 1]} \wedge \neg b_{[x \geq n+1]}$
- Example encodings of simple constraints:
  - The constraint $x = k$ is encoded as $b_{[x \geq k]} \wedge \neg b_{[x \geq k+1]}$.
  - The constraint $x \neq k$ is encoded as $\neg b_{[x \geq k]} \vee b_{[x \geq k+1]}$.
  - In any constraint, can replace x by $\sum_k b_{[x \geq k]}$.

Log Encoding of an Integer Variable

Consider an integer variable x with domain $1..n$:

- Encode the value of x in binary, using just $\lceil \log_2(n) \rceil$ Boolean variables.
- Let b_i be a Boolean variable representing the i-th bit in the binary encoding of x.
- In any constraint, can replace x by (note that the lowest value for x is 1):

$$x = 1 + \sum_{i=0}^{\lceil \log_2(n) \rceil - 1} b_i \cdot 2^i$$

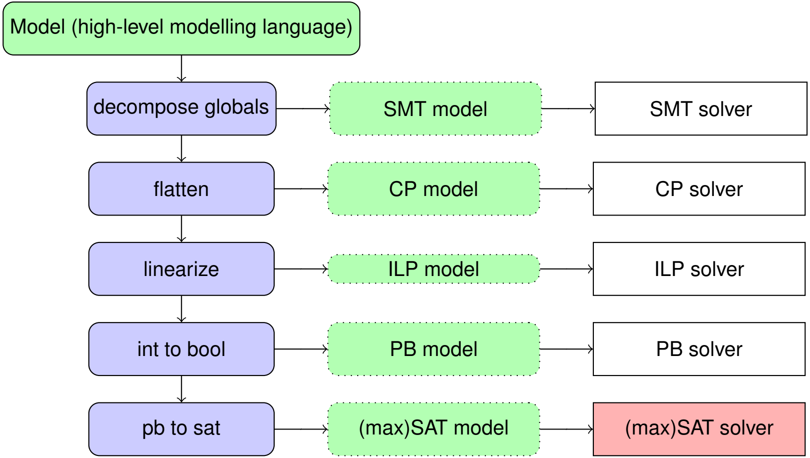
  where $b_i \in \{0, 1\}$.
- Consistency constraint:
  - Upper bound of domain: $1 + \sum_{i=0}^{\lceil \log_2(n) \rceil - 1} b_i \cdot 2^i \leq n$ (in case n is not a power of 2)

Encodings: example with overview

	$E^{\mathbb{D}}(x)$				$E^{\mathbb{O}}(x)$			$E^{\mathbb{B}}(x)$	
$\mathcal{A}(x)$	$[\![x = 0]\!]$	$[\![x = 1]\!]$	$[\![x = 2]\!]$	$[\![x = 3]\!]$	$[\![x \geq 1]\!]$	$[\![x \geq 2]\!]$	$[\![x \geq 3]\!]$	$[\![\mathrm{bit}^0(x, 1)]\!]$	$[\![\mathrm{bit}^0(x, 0)]\!]$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	1
2	0	0	1	0	1	1	0	1	0
3	0	0	0	1	1	1	1	1	1

Tab. 2.1: Solutions for $x \in [0..3]$ and corresponding solutions of the encoding variables of $E^{\mathbb{D}}(x)$, $E^{\mathbb{O}}(x)$ and $E^{\mathbb{B}}(x)$

Overview



Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Boolean Satisfiability (SAT) / Max-SAT

- Determines the satisfiability of a propositional logic formula over Boolean variables, typically expressed in conjunctive normal form (CNF)
- A solution is a complete assignment that satisfies the formula
- Max-SAT extends SAT by maximizing the number of satisfied clauses in cases where not all clauses can be satisfied

How it Works (High-Level Overview)

- **DPLL Algorithm** systematically explores variable assignments and backtracks upon conflicts
- **Conflict-Driven Clause Learning (CDCL)** captures conflicts to avoid repeating mistakes in future search

Typical Applications

- Hardware/software verification, Model checking
- Planning and Resource allocation

Boolean Satisfiability Solving (SAT)

Modelling Language:

- Only Boolean decision variables.
- CNF is a conjunction ($\wedge$) of clauses. A clause is a disjunction ($\vee$) of literals. A literal is a Boolean decision variable (x) or its negation ($\neg x$).
- Only for satisfaction problems; else: MaxSAT.

Example

- Boolean decision variables: w, x, y, z
- Clauses:

$$(\neg w \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg w \vee x \vee \neg z) \\ \wedge (x \vee y \vee z) \wedge (w \vee \neg z)$$

- A solution: $w = \text{False}$, $x = \text{True}$, $y = \text{True}$, $z = \text{False}$

Boolean Satisfiability Solving (SAT) – CPMpy

Modelling Language:

- Only Boolean decision variables.
- CNF is a conjunction ($\wedge$) of clauses. A clause is a disjunction ($\vee$) of literals. A literal is a Boolean decision variable (x) or its negation ($\neg x$).
- Only for satisfaction problems; else: MaxSAT.

Example (in CPMpy)

- Decision variables: `w, x, y, z = cp.boolvar(shape=4)`
- Clauses:

```
model += (~w | ~y) & (~x | y) & (~w | x | ~z) \
         & (x | y | z) & (w | ~z)
```

- A solution: `w=False, x=True, y=True, z=False`

The SAT Problem

Given a clause set, find an assignment, that is, Boolean values for all the decision variables, so that all the clauses are satisfied.

- The decision version of this problem is NP-complete.
- Any combinatorial problem can be encoded into SAT.
  Careful: “encoded into” is not “reduced from”, but “reduced to”.
  It might require an exponential number of constraints...
- There has been intensive research on SAT solving since the 1960s, and still very active with yearly competitions.
- Most modern SMT/CP/PB solvers are built on/include a SAT solver.

DPLL [Davis-Putnam-Logemann-Loveland, 1962]

Inference:

- Unit propagation: If all the literals in a clause evaluate to `false`, except one whose decision variable has no value yet, then that literal is made to evaluate to `true` so that the clause becomes satisfied.
  Example: $x \vee \neg y \vee \neg z$ with $x = F$ and $y = T$ propagates $z = \ldots$

Tree Search: (start with empty valuation)

1. Perform inference, e.g. unit propagation
2. If some clause is unsatisfied, then backtrack.
3. If all decision variables have a value, then we have a solution.
4. Select an unvalued decision variable b and make two branches: one with $b =$ `true`, and the other one with $b =$ `false`.
5. Recursively explore each of the two branches.

Strategies and Improvements over DPLL

Search Strategies:

- On which decision variable to branch next?
- Which branch to explore next?
- Which search (depth-first, breadth-first, . . .) to use?

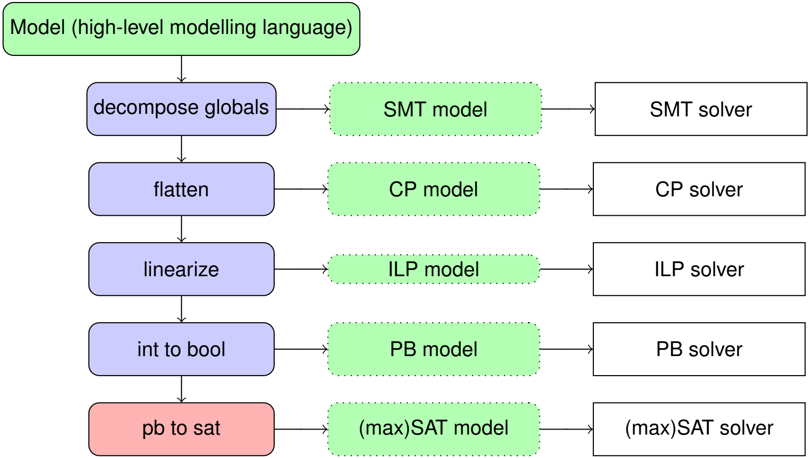
Improvements:

- Clause learning: on failure, analyse the conflict and learn a new clause from it
- Backjumping: backtrack multiple levels at once (based on learned clause)
- Restarts: backjump to root node (valid due to learned clauses)
- A lot of implementation details, e.g. data structures
- . . .

SAT Solving

- Guarantee: exact, given enough time.
- Mainly black-box: there are limited ways to guide the solving.
- It can scale to millions of decision variables and clauses.
- Encoding a problem can yield a huge SAT model.
- For debugging and explanation purposes, solvers can extract an unsatisfiable core, that is a subset of the clauses that make the model unsatisfiable.
- It is mainly applied in hardware verification and software verification.

Overview


Model (high-level modelling language)
decompose globals
SMT model
SMT solver
flatten
CP model
CP solver
linearize
ILP model
ILP solver
int to bool
PB model
PB solver
pb to sat
(max)SAT model
(max)SAT solver

Encoding PB to SAT

Example: $\sum_{k\in\{1..n\}} b_k = 1$ (as used e.g. in the direct encoding of integers)

$$\sum_{k\in\{1..n\}} b_k = 1 \Leftrightarrow \sum_{k\in\{1..n\}} b_k \geq 1 \;\wedge\; \sum_{k\in\{1..n\}} b_k \leq 1 \tag{1}$$

$$\sum_{k\in\{1..n\}} b_k \geq 1 \Leftrightarrow \bigvee_{k\in 1..n} b_k \tag{2}$$

$$\sum_{k\in\{1..n\}} b_k \leq 1 \Leftrightarrow \bigwedge_{i,j\in 1..n,\; i<j} \left(b_i + b_j \leq 1\right) \tag{3}$$

$$\Leftrightarrow \bigwedge_{i,j\in 1..n,\; i<j} \left(\neg b_i \vee \neg b_j\right) \tag{4}$$

From 1 PB constraint over m variables,
to 1 clause over m variables + $m * (m-1)/2$ binary clauses.

Encoding any PB Constraint to SAT

Pseudo-Boolean Constraints: $\sum_{i=1}^{n} a_i x_i \leq b$, with a_i, b integers, x_i Boolean variables.
Many encoding techniques exist, e.g.: (details beyond the scope of this course)

- **Adder Networks:**
  - Use a network of binary adder circuits (like in hardware) to encode the sum.
  - Network is polynomial in size, but has weaker propagation (not *arc consistent*).
- **Sorting Networks:**
  - Use a network of comparators to sort inputs, enforcing PB constraints on sums.
  - Strong propagation & works well for small/medium n, but often exponential size.
- **Totalizer Encoding:**
  - Introduces auxiliary variables for cumulative sums.
  - Useful for incremental solving; strong propagation but at worst exponential size.
- **Binary Decision Diagrams (BDD):**
  - Use a BDD over the x_is to compactly represent the allowed values.
  - Strong propagation, but building BDDs can be computationally expensive & worst-case exponential size.

Choosing a Solver Technology

- Do you need guarantees that a found solution is optimal,
  that all solutions are found, and that unsatisfiability is provable?
- What types of decision variables are in your model?
- What constraint predicates are in your model?
- Does your problem look like a well-known problem?
- How do backends perform on easy problem instances?
- What is your favourite technology or backend?

Some Caveats

- Each problem can be modelled in many different ways.
- Different models of the same problem are better suited for different backends.
- Performance on small instances does not always scale to larger instances.
- Sometimes, a good search strategy is more important than a good model (see next lecture).
- Not all backends of the same technology have comparable performance.
- Some pure problems can be solved by specialist solvers, such as Concorde for the travelling salesperson problem, but real-life side constraints often make them inapplicable.
- Some problems are maybe even solvable in polynomial time and space.

Take-Home Message:

- There are many solving technologies and backends.
- It is useful to highlight the commonalities and differences.
- No solving technology or backend can be universally better than all the others, unless $\mathrm{P} = \mathrm{NP}$.

☞ With solver-independent frameworks: can try multiple ones!

To go further:

John N. Hooker.
Integrated Methods for Optimization.
2nd edition, Springer, 2012.