Pseudo-Boolean solving with Sat4j

solving software dependency management problems

Daniel Le Berre

CRIL-CNRS UMR 8188 - Université d'Artois

SAT/SMT Summer school @ MIT, June 15, 2011

Outline

Motivating example : software dependency management

Scientific context : Pseudo-Boolean Optimization

Solving PBO using CDCL architecture

Implementation in the open source Sat4j library

Conclusion

Outline

Motivating example : software dependency management

Scientific context : Pseudo-Boolean Optimization

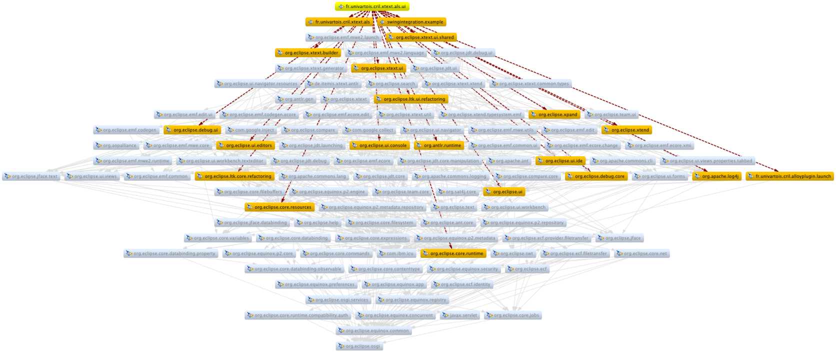
Solving PBO using CDCL architecture

Implementation in the open source Sat4j library

Conclusion

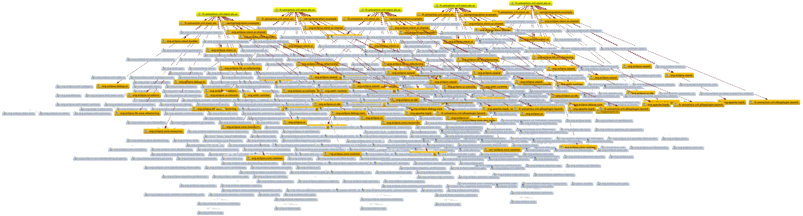
Current softwares are composite !

Alloy 4 Eclipse dependencies : 20 direct / 108 total



Current softwares are composite !

Problems occur when installing several plugins !



Current softwares are composite !

- Linux distributions : made of packages (Debian >50K packages)
- Component based software/platform (Eclipse ecosystem >3K bundles)
- Any complex software : made of libraries (Maven universe >200K libraries)
- There are requirements between the diverse components
  - capabilities can be provided by several components (disjunction)
  - some components cannot be installed together (conflicts)

Dependency Management Problem : formal definition

P a set of packages

$$P = \{smt_1, pico_1, pico_2, crypto_1, crypto_2, lp\_solve_1, glpk_1\}$$

depends $P \to 2^{2^P}$ requirement constraints

$$P = \{smt_1 \to \{\{pico_1, pico_2, crypto_1, crypto_2\}, \{lp\_solve_1, glpk_1\}\}\}$$

conflicts $P \to 2^P$impossible configurations

$P = \{pico_1 \to \{pico_2, crypto_1, crypto_2\},$
$pico_2 \to \{pico_1, crypto_1, crypto_2\}\}$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, depends, conflicts)$ iff
$\forall q \in Q, (\forall dep \in depends(q), dep \cap Q \neq \emptyset) \wedge (conflicts(q) \cap Q = \emptyset)$.

$Q_1 = \{smt_1, picosat_2, glpk_1\} Q_2 = \{smt_1, crypto_1, crypto_2, lp\_solve_1\}$

Dependency Management Problem : formal definition

P a set of packages

$$P = \{smt_1, pico_1, pico_2, crypto_1, crypto_2, lp\_solve_1, glpk_1\}$$

depends $P \rightarrow 2^{2^P}$ requirement constraints

$$P = \{smt_1 \rightarrow \{\{pico_1, pico_2, crypto_1, crypto_2\}, \{lp\_solve_1, glpk_1\}\}\}$$

conflicts $P \rightarrow 2^P$ impossible configurations

$P = \{pico_1 \rightarrow \{pico_2, crypto_1, crypto_2\},$
$pico_2 \rightarrow \{pico_1, crypto_1, crypto_2\}\}$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, depends, conflicts)$ iff
$\forall q \in Q, (\forall dep \in depends(q), dep \cap Q \neq \emptyset) \wedge (conflicts(q) \cap Q = \emptyset)$.

What is the complexity of finding if a Q containing a specific package exists ?

UNIVERSITÉ D'ARTOIS

Just as hard as SAT : NP-complete !

See how to decide satisfiability of $(\neg a \lor b \lor c) \land (\neg a \lor \neg b \lor c) \land a \land \neg c$

```
package: a
version: 1
conflicts: a = 2

package: a
version: 2
conflicts: a = 1

package: b
version: 1
conflicts: b = 2

package: b
version: 2
conflicts: b = 1

package: c
version: 1
conflicts: c = 2

package: c
version: 2
conflicts: c = 1
```

```
package: clause
version: 1
depends: a = 2 | b = 1 | c = 1

package: clause
version: 2
depends: a = 2 | b = 2 | c = 1

package: clause
version: 3
depends: a = 1

package: clause
version: 4
depends: c = 2

package: formula
version: 1
depends: clause = 1, clause = 2,
         clause = 3, clause = 4

request: satisfiability
install: formula
```

From dependencies to clauses

- Dependencies can easily be translated into clauses :

```
package: a
version: 1
depends: b = 2 | b = 1, c = 1
```

$$a_1 \rightarrow (b_2 \vee b_1) \wedge c_1$$

$$\neg a_1 \vee b_2 \vee b_1, \neg a_1 \vee c_1$$

- Conflict can easily be translated into binary clauses :

```
package: a
version: 1
conflicts: b = 2, d = 1
```

$$\neg a_1 \vee \neg b_2, \neg a_1 \vee \neg d_1$$

From decision to optimization

The issue is not to find one solution (easy for current SAT solvers), but to find a good solution

- Minimizing the number of installed packages
- Minimizing the size of installed packages
- Ensuring capacity constraints
- Keeping up to date versions of packages
- Preferring most recent packages to older ones
- ...

Can we manage that with a SAT solver ?

Representing optimization criteria with MaxSat ?

$\alpha \equiv \bigwedge_{p_v \in P}(p_v \rightarrow (\bigwedge_{dep \in depends(p_v)} dep), \infty) \wedge$
$\bigwedge_{conf \in conflicts(p_v)}(p_v \rightarrow \neg conf, \infty,) \wedge (q, \infty)$

α denote the formula to satisfy for installing q.

Minimizing the number of installed packages (Partial MaxSat) :

$$\phi \equiv (\bigwedge_{p_v \in P, p_v \neq q} (\neg p_v, k)) \tag{1}$$

Minimizing the size of installed packages (Weighted Partial MaxSat) :

$$\phi \equiv (\bigwedge_{p_v \in P, p_v \neq q} (\neg p_v, size(p_v))) \tag{2}$$

Representing optimization criteria using pseudo-boolean optimization

- Minimizing the number of installed packages :

$$min : \sum_{p_v \in P, p_v \neq q} p_v$$

- Minimizing the size of installed packages :

$$min : \sum_{p_v \in P, p_v \neq q} size(p_v) \times p_v$$

- We can express easily that only one version of package *libnss* can be installed :
$libnss_1 + libnss_2 + libnss_3 + libnss_4 + libnss_5 \leq 1$

Pseudo Boolean Optimization vs Partial Weighted MaxSat

Two approaches to solve the same NP-hard problem

- ϕ a boolean formula built using n variables
- f an evaluation function

$$f : \phi \times \{True, False\}^n \to \mathbb{Z}$$

- Problem : find an assignment I of boolean variables that satisfies ϕ and minimize $f(\phi, I)$

  PBO ϕ made of PB constraints, f linear function on literals

  PWMS ϕ made of clauses, f linear function on clauses

Pseudo Boolean Optimization vs Weighted Partial MaxSat

Translation from PBO to Partial Weighted MaxSat

1. Translate each PB constraint into an equivalent set of hard clauses (see e.g. Minisat+ [10])
2. Translate objective function $min : \sum w_i \times x_i$ into soft weighted unit clauses $(w_i, \neg x_i)$

Translation from Partial Weighted MaxSat to PBO (used in Sat4j)

1. Each hard clause $l_1 \vee l_2 \vee ... \vee l_n$ can be written in pseudo boolean form $\sum_{i=1}^{n} l_i \geq 1$
2. For each soft clause (w_j, c_j), create a new clause $s_j \vee c_j$ to be expressed in PB form.
3. create the optimization function $min : \sum w_j s_j$

Why MaxSat and PBO ?

- PBO nice for capacity constraints, weights attached to literals.
- PWMS nice for relaxing constraints, weights attached to constraints (e.g. optional dependencies in Eclipse)
- Intersection : binate covering problem (CNF + Optimization function)
  - PBO constraints limited to clauses
  - PWMS soft clauses limited to weighted unit clauses
  - Specific case where problems can be easily encoded optimally in both paradigm

Outline

Linear Pseudo-Boolean decision and optimization problems

Linear Pseudo-Boolean constraint

$$-3x_1 + 4x_2 - 7x_3 + x_4 \rhd -5$$

where $\rhd \in \{<, \leq, >, \geq, =\}$.

- variables x_i take their value in $\{0, 1\}$
- $\overline{x_1} = 1 - x_1$
- coefficients and degree are integral constants

Pseudo-Boolean decision problem : NP-complete

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \\ (c) & x_1 + \overline{x_2} + x_5 \geq 1 \end{cases}$$

Plus an objective function : Optimization problem, NP-hard

$$min : 4x_2 + 2x_3 + x_5$$

Rules on LPB constraints : Linear combination

linear combination:

$$\frac{\sum_i a_i.x_i \geq k \qquad \sum_i a'_i.x_i \geq k'}{\sum_i(\alpha.a_i + \alpha'.a'_i).x_i \geq \alpha.k + \alpha'.k'}$$

with $\alpha > 0$ and $\alpha' > 0$

$$\frac{x_1 + x_2 + 3x_3 + x_4 \geq 3 \qquad 2\overline{x_1} + 2\overline{x_2} + x_4 \geq 3}{\begin{array}{c} 2x_1 + 2x_2 + 6x_3 + 2x_4 + 2\overline{x_1} + 2\overline{x_2} + x_4 \geq 2 \times 3 + 3 \\ 2x_1 + 2x_2 + 6x_3 + 2x_4 + 2 - 2x_1 + 2 - 2x_2 + x_4 \geq 9 \\ 6x_3 + 3x_4 \geq 5 \end{array}}$$

Note that $2x + 2\overline{x} = 2$, not 0 !

Note that the coefficients are growing !

Some other inference rules

division One can always reduce a LPB constraint to a clause!

$$\frac{5x_3 + 3x_4 \geq 5}{\lceil 5/5 \rceil x_3 + \lceil 3/5 \rceil x_4 \geq \lceil 5/5 \rceil}$$
$$x_3 + x_4 \geq 1$$

weakening

$$\frac{5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8}{3x_2 + 2x_3 + 2x_4 + x_5 \geq 3}$$

saturation

$$\frac{6x_3 + 3x_4 \geq 5}{5x_3 + 3x_4 \geq 5}$$

Resolution extends Cutting Planes

- Linear combination + division = cutting plane proof system (complete).
- First introduced for linear programming by R. Gomory in 1958
- DPLL proof system is tree resolution [4]
- CDCL proof system is general resolution [3, 9]
- Cutting planes can be seen as a generalization of the resolution (J.N. Hooker, 1988)
- Wish : change Resolution during conflict analysis by Cutting Planes to get a solver with a better proof system than CDCL [5, 6]

Pseudo Boolean Optimization vs Linear Programming

- Boolean variables vs real or integral variables
- Usually part of the constraints are simple clauses
- Reuse successful techniques from SAT (CDCL solvers) in that specific context
- PB solvers complementary to LP solvers : see CPLEX 12.1 results during PB 10 competition
  - Outperforms the PB solvers in optimization problems with small integers, linear constraints category, OPT answers
  - Outperformed by PB solvers on decision problems with small integers, linear constraints

Outline

Minisat 1 generic CDCL architecture

Contract found for generic constraints in Minisat 1.10 :

propagate() propagates newly derived literal or detect contradiction

calcReason() explain propagation and conflicts in terms of literals

Similar to the contract between the SAT solver and the T-solver in modern SMT solvers.

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

- literal x_1 must be propagated to satisfy the constraint because $\sum_{i=2}^{8} x_i = 21 < 25$

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

- literal x_1 must be propagated to satisfy the constraint because $\sum_{i=2}^{8} x_i = 21 < 25$
- Suppose $\neg x_2$,

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

- literal x_1 must be propagated to satisfy the constraint because $\sum_{i=2}^{8} x_i = 21 < 25$
- Suppose $\neg x_2$, then x_3 must be true

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

- literal x_1 must be propagated to satisfy the constraint because $\sum_{i=2}^{8} x_i = 21 < 25$
- Suppose $\neg x_2$, then x_3 must be true
- Suppose now $\neg x_4$ and $\neg x_5$, 6then x_6, x_7, x_8 are propagated.

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

- literal x_1 must be propagated to satisfy the constraint because $\sum_{i=2}^{8} x_i = 21 < 25$
- Suppose $\neg x_2$, then x_3 must be true
- Suppose now $\neg x_4$ and $\neg x_5$, 6then x_6, x_7, x_8 are propagated.

Propagation : the specificity of PB constraints

$$15x_1 + 7x_2 + 7x_3 + 2x_4 + 2x_5 + x_6 + x_7 + x_8 \geq 25$$

Can we propagate something ?

- literal x_1 must be propagated to satisfy the constraint because $\sum_{i=2}^{8} x_i = 21 < 25$
- Suppose $\neg x_2$, then x_3 must be true
- Suppose now $\neg x_4$ and $\neg x_5$, 6then x_6, x_7, x_8 are propagated.

Difference with clauses :

- Can propagate several literals at once
- Can propagate value several times in the same search path

Conflict analysis

Resolution: explain propagation and conflicts in terms of falsified literals in the constraint (PBS, Satzoo)

Cutting Planes: Apply cutting planes inference but

- Need to ensure that resulting constraint is falsified
- No syntactical stopping criteria like UIP for creating assertive constraint
- Beware to growing coefficients

(Galena, PBChaff, Pueblo)

Backtracking

Resolution Usual CDCL backtracking scheme

Cutting Planes A constraint may be assertive at different decision level (including root decision level). Where should we backtrack ?

Backtracking

Resolution Usual CDCL backtracking scheme

Cutting Planes A constraint may be assertive at different decision level (including root decision level). Where should we backtrack ?
Go back to the first decision level where the constraint can propagate values

Lazy data structures

Proposed in [5] and [6].

- General case : vary with degree and max coeff
  Let $M = max(a_i)$
  NbWatch = minimal number of literals x_i such that
  $\sum a_i \geq k + M$.
- Cardinality constraints : vary with degree
  $M = 1$
  $NbWatch = k + 1$
- Clauses : fixed
  $M = 1$
  $k = 1$
  $NbWatch = 2$

What about optimization ?

- SAT solvers are decision engines
- How to reuse them easily in an optimization context
- Want to use the solver as blackbox
- Simplest way : strengthening (linear search)

Optimization using strengthening (linear search)

input : A set of clauses, cardinalities and pseudo-boolean constraints setOfConstraints and an objective function objFct to minimize

output: a model of setOfConstraints, or UNSAT if the problem is unsatisfiable.

```
answer ← isSatisfiable (setOfConstraints);
if answer is UNSAT then
    return UNSAT
end
repeat
    model ← answer ;
    answer ← isSatisfiable (setOfConstraints ∪
                              {objFct < objFct (model)});
until (answer is UNSAT);
return model ;
```

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5$$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Model

$$\overline{x_1}, x_2, \overline{x_3}, x_4, x_5$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5$$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Model

$\overline{x_1}, x_2, \overline{x_3}, x_4, x_5$

Objective function

min : $4x_2 + 2x_3 + x_5$

$<$

Objective function value

5

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5 \qquad < \qquad 5$$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Model

$x_1, \overline{x_2}, x_3, \overline{x_4}, x_5$

Objective function

min : $4x_2 + 2x_3 + x_5 \quad < \quad 5$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Model

$x_1, \overline{x_2}, x_3, \overline{x_4}, x_5$

Objective function

min : $4x_2 + 2x_3 + x_5$

$<$

Objective function value

$3 < 5$

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5 \qquad < \qquad 3$$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Model

$x_1, \overline{x_2}, \overline{x_3}, x_4, x_5$

Objective function

min : $4x_2 + 2x_3 + x_5 \quad < \quad 3$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Model

$$x_1, \overline{x_2}, \overline{x_3}, x_4, x_5$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5$$

$<$

Objective function value

$$1 < 3$$

UNIVERSITÉ D'ARTOIS

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5 \qquad < \qquad 1$$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Objective function

min : $4x_2 + 2x_3 + x_5 \quad < \quad 1$

Optimization algorithm

Formula :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Objective function

$$\min : \quad 4x_2 + 2x_3 + x_5$$

The objective function value 1 is optimal for the formula.

$x_1, \overline{x_2}, \overline{x_3}, x_4, x_5$ is an optimal solution.

Outline

Why Sat4j ? (my vision)

Every programmer deserves to have access to a SAT solver in its preferred language ... even Java programmers.

Providing SAT engines in Java DOES NOT make sense

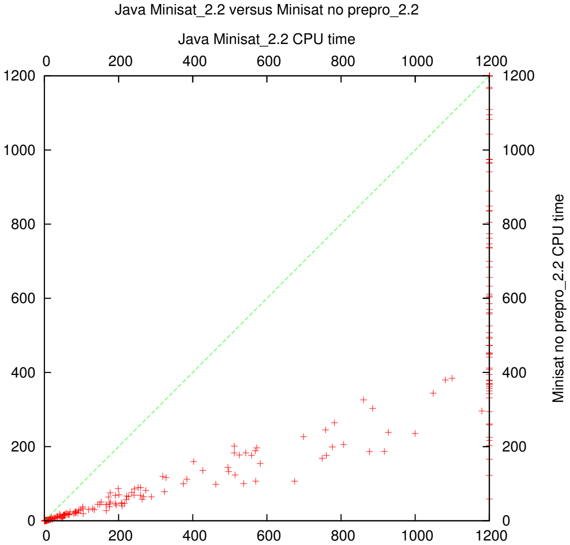
Some quotes from Sat4j users

- *Warning : Alloy4 defaults to SAT4J since it is pure Java and very reliable. For faster performance, go to Options menu and try another solver like MiniSat.*
  Felix Chang, MIT. Message appearing when launching Alloy 4.

- *The default SAT Solver used by Forge and JForge is SAT4J, which is pure Java. We highly recommend you use one of the other supported SAT solvers, because they usually exhibit better performance.*
  Greg Dennis, MIT. Quote from Forge web page.

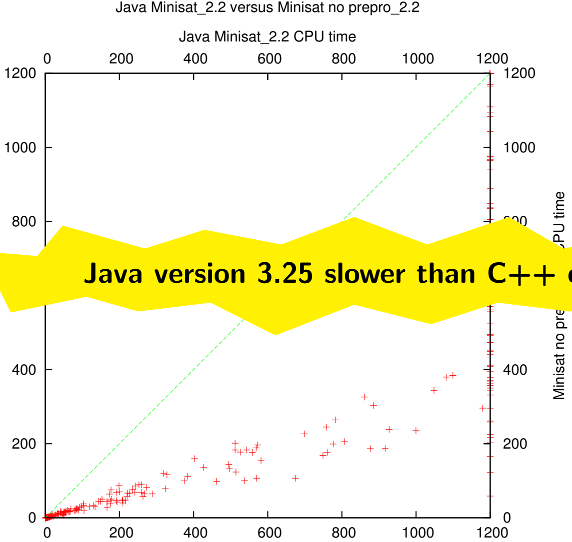
Providing SAT engines in Java DOES NOT make sense

Java vs C++ runtime of Minisat 2.2 without preprocessor (courtesy of Carsten Sinz)



Providing SAT engines in Java DOES NOT make sense

Java vs C++ runtime of Minisat 2.2 without preprocessor (courtesy of Carsten Sinz)



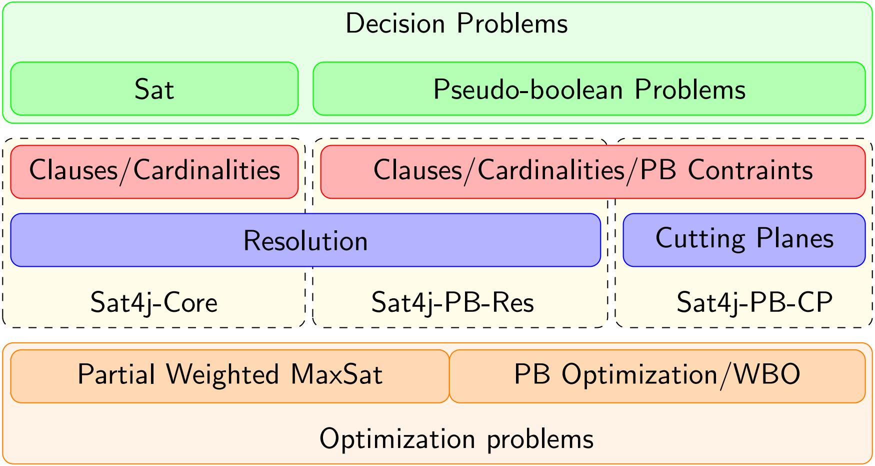
Java version 3.25 slower than C++ one!

Providing SAT engines in Java DOES make sense

Speed is not the only argument

- People use it
- Java is widely used in the Software Engineering community
- Java is widely used by students
- ...

A flexible framework for solving propositional problems



A generic and flexible CDCL solver

Basis Minisat 1.10 specification + conflict minimization from Minisat 1.13

Static Restarts strategies Minisat, Biere, Luby

Generic Conflict minimization None, Simple, Expensive
works with all constraints and data structures

Learning LimitedLearning, LearnAllClauses, NoLearning, ...
learning is not coupled with conflict analysis

Learned clauses deletion Memory based, Glucose

Phase selection Random, Positive, Negative, AppearInLastLearnedClauses, RSAT phase saving

Lazy Data structures Watched Literals, Head/Tail

Default configuration

Pseudo Boolean solvers in SAT4J

SAT4J PB RES learn clauses. takes advantage of the full existing SAT machinery.

SAT4J PB CuttingPlanes learn PB constraints. No lazy data structure for constraints, need arbitrary precision arithmetic for correctness.

- The resolution based PB solver is usually faster than the CP based one.
- Some benchmarks can only be solved using CP solver (e.g. pigeon hole).
- The principles behind each solver are clear : no tweaks to solve a few more benchmarks during the PB evaluations !

Remarks about the optimization procedure

- No need for an initial upper bound !
- Phase selection strategy takes into account the objective function.
- External to the PB solver : can use any PB solver.
- SAT, SAT, SAT, ..., SAT, UNSAT pattern
- SAT answer usually easier to provide than UNSAT one
- In practice : optimality is often hard to prove for the Resolution based PB solver (pigeon hole ?).
- Ideally, would like to run the CP PB solver to prove optimality at the end.
- Problem : how to detect that we need to prove optimality ?

Remarks about the optimization procedure

- No need for an initial upper bound !
- Phase selection strategy takes into account the objective function.
- External to the PB solver : can use any PB solver.
- SAT, SAT, SAT, ..., SAT, UNSAT pattern
- SAT answer usually easier to provide than UNSAT one
- In practice : optimality is often hard to prove for the Resolution based PB solver (pigeon hole ?).
- Ideally, would like to run the CP PB solver to prove optimality at the end.
- Problem : how to detect that we need to prove optimality ?
- Nice idea suggested by Olivier Roussel submitted to PB 2010 : run the Res and CP PB solvers in parallel !

Optimization with solvers running in parallel

input : A set of clauses, cardinalities and pseudo-boolean constraints setOfConstraints and an objective function objFct to minimize

output: a model of setOfConstraints, or UNSAT if the problem is unsatisfiable.

```
answer ← isSatisfiable (setOfConstraints);
if answer is UNSAT then
    return UNSAT
end
repeat
    model ← answer ;
    answer ← isSatisfiable (setOfConstraints ∪
                            {objFct < objFct (model)});
until (answer is UNSAT);
return model ;
```

logic-synthesis/normalized-jac3.opb @ PB2010

```
% Cutting Planes
1.17/0.78 c #vars        1731
1.17/0.78 c #constraints  1254
1.76/1.03 c SATISFIABLE
1.76/1.03 c OPTIMIZING...
1.76/1.03 o 26
3.40/1.91 o 25
5.93/3.41 o 24
6.97/4.33 o 23
7.49/4.88 o 22
8.44/5.72 o 21
9.00/6.27 o 20
9.62/6.87 o 19
10.44/7.61 o 18
11.54/8.79 o 17
13.03/10.13 o 16
25.34/22.07 o 15
1800.11/1773.42 s SATISFIABLE
```

```
% Resolution
1.17/0.75 c #vars        1731
1.17/0.75 c #constraints  1254
1.57/0.91 c SATISFIABLE
1.57/0.91 c OPTIMIZING...
1.57/0.91 o 26
2.55/1.42 o 23
2.96/1.60 o 22
3.35/1.80 o 21
16.34/14.32 o 20
55.04/52.91 o 19
766.33/763.00 o 18
1800.04/1795.76 s SATISFIABLE
```

logic-synthesis/normalized-jac3.opb @ PB2010

```
% Cutting Planes
1.17/0.78 c #vars     1731
1.17/0.78 c #constraints  1254
1.76/1.03 c SATISFIABLE
1.76/1.03 c OPTIMIZING...
1.76/1.03 o 26
3.40/1.91 o 25
5.93/3.41 o 24
6.97/4.33 o 23
7.49/4.88 o 22
8.44/5.72 o 21
9.00/6.27 o 20
9.62/6.87 o 19
10.44/7.61 o 18
11.54/8.79 o 17
13.03/10.13 o 16
25.34/22.07 o 15
1800.11/1773.42 s SATISFIABLE
```

```
% Res // CP
1.35/0.84 c #vars     1731
1.35/0.84 c #constraints  1254
1.99/1.85 c SATISFIABLE
1.99/1.85 c OPTIMIZING...
1.99/1.85 o 26 (CuttingPlanes)
2.61/2.89 o 25 (Resolution)
3.91/3.92 o 24 (Resolution)
4.12/5.00 o 23 (Resolution)
5.92/6.01 o 22 (Resolution)
7.72/7.04 o 21 (Resolution)
9.63/8.07 o 20 (CuttingPlanes)
13.04/10.09 o 19 (CuttingPlanes)
15.66/12.10 o 18 (CuttingPlanes)
20.27/15.14 o 17 (CuttingPlanes)
70.03/41.35 o 16 (CuttingPlanes)
218.63/118.14 o 15 (CuttingPlanes)
305.11/164.68 s OPTIMUM FOUND
```

logic-synthesis/normalized-jac3.opb @ PB2010

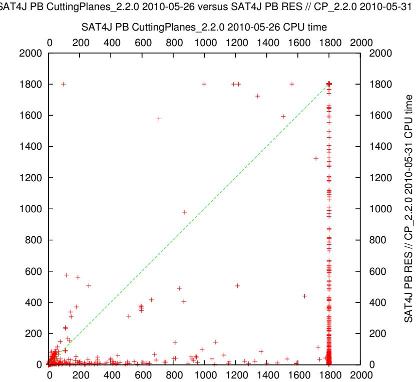
Cutting Planes

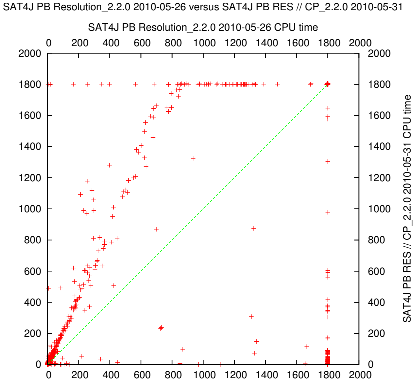
```
1800.11/1773.42 s SATISFIABLE
1800.11/1773.41 c learnt clauses : 2618
1800.11/1773.42 c speed (assignments/second) : 226
```

Res // CP

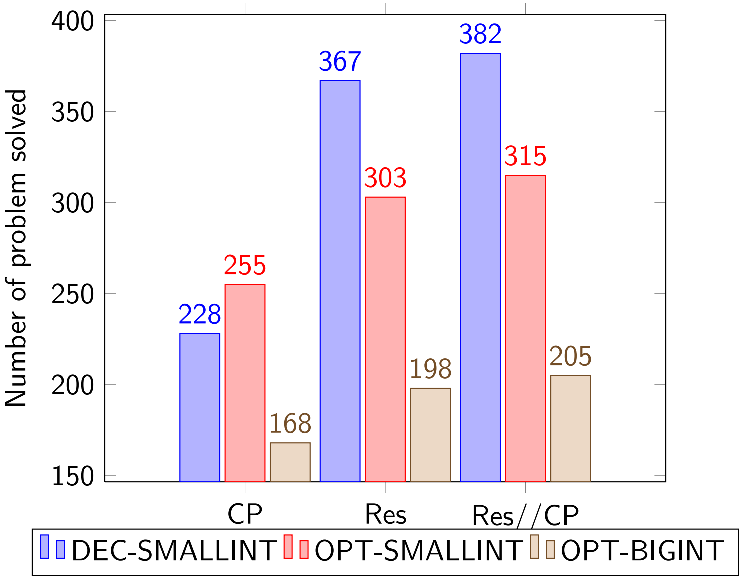
```
305.11/164.68 s OPTIMUM FOUND
305.11/164.68 c learnt clauses : 1318
305.11/164.68 c speed (assignments/second) : 3927
```

Scatter plots Res // CP vs CP, Resolution





Pseudo Boolean Competition 2010 results



Regarding the idea to run the two solvers in //

- Res // CP globally better than Res or CP solver during PB 2010 in number of benchmarks solved.
- Res // CP twice as slow as Res on many benchmarks.
- Decision problems : solves the union of the benchmarks solved by Res and CP in half the timeout (CPU time taken into account, not wall clock time).
- Optimization problems : "cooperation" of solvers allow to solve new benchmarks !

Selector variables + assumptions = explanation

- From the beginning in Minisat 1.12
- Add a new selector variable per constraint
- Check for satisfiability assuming that the selector variables are falsified
- if UNSAT, analyze the final root conflict to keep only selector variables involved in the inconsistency
- Apply a minimization algorithm afterward to compute a minimal explanation (QuickXplain, Insertion, Deletion)
- Advantages :
  - no changes needed in the SAT solver internals
  - works for any kind of constraints !
- See in action during the MUS/HLMUS track of the SAT competition next week !

Selector variables for various constraints

Selector variable principle : satisfying the selector variable should satisfy the selected constraint.

clause simply add a new variable

$$\bigvee l_i \quad \Rightarrow \quad s \vee \bigvee l_i$$

cardinality add a new weighted variable

$$\sum l_i \geq d \quad \Rightarrow \quad d \times s + \sum l_i \geq d$$

The new constraints is PB, no longer a cardinality !

pseudo add a new weighted variable

$$\sum w_i \times l_i \geq d \quad \Rightarrow \quad d \times s + \sum w_i \times l_i \geq d$$

if the weights are positive, else use

$$(d + \sum_{w_i<0} |w_i|) \times s + \sum w_i \times l_i \geq d$$

Outline

Motivating example : software dependency management

Scientific context : Pseudo-Boolean Optimization

Solving PBO using CDCL architecture

Implementation in the open source Sat4j library

Conclusion

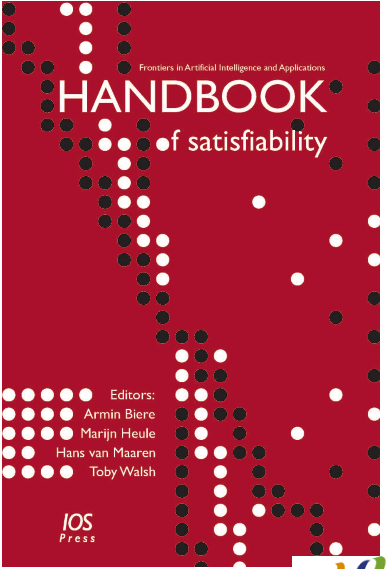
SAT4J today

- SAT4J MAXSAT considered state-of-the-art on Partial [Weighted] MaxSAT application benchmarks (2009).
- SAT4J PB (Res, CP) are not very efficient, but correct (arbitrary precision arithmetic).
- SAT4J SAT solvers can be found in various software from academia (Alloy 4, Forge, ...) to commercial applications (GNA.sim).
- SAT4J PB Res solves Eclipse plugin dependencies since June 2008 (Eclipse 3.4, Ganymede) [8]
  - SAT4J ships with every product based on the Eclipse platform (around 13 millions downloads per year on Eclipse.org since June 2008)
  - SAT4J helps to build Eclipse products daily (e.g. nightly builds on Eclipse.org, IBM, SAP, etc)
  - SAT4J helps to update Eclipse products worldwide daily

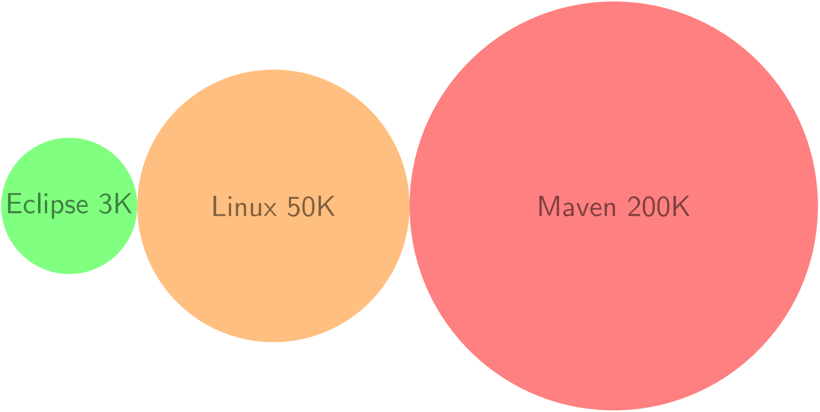
Pseudo-Boolean Optimization

- For more details, see the chapter on PBO by Vasco Manquinho and Olivier Roussel in the Handbook of satisfiability

- See the PB competition web site for latest results, benchmarks and software : `http://www.cril.fr/PB11/`



Scaling the dependency problem in an interactive setting



See http://www.mancoosi.org/misc/ for benchmarks and solvers for Linux dependency problems.

See http://wiki.eclipse.org/Equinox/p2/CUDFResolver for our Eclipse solution adapted to Linux settings [2]

Thanks for your attention

Questions ?

Mancoosi : managing software complexity [online].
2008.
Available from : `http://www.mancoosi.org` [cited September 2010].

Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques Silva, and Pascal Rapicault.
Solving Linux Upgradeability Problems Using Boolean Optimization.
CoRR, abs/1007.1021, 2010.

Paul Beame, Henry A. Kautz, and Ashish Sabharwal.
Towards understanding and harnessing the potential of clause learning.
J. Artif. Intell. Res. (JAIR), 22 :319–351, 2004.

Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen.

On the relative complexity of resolution refinements and cutting planes proof systems.
SIAM J. Comput., 30(5) :1462–1484, 2000.

Donald Chai and Andreas Kuehlmann.
A fast pseudo-boolean constraint solver.
In *ACM/IEEE Design Automation Conference (DAC'03)*, pages 830–835, Anaheim, CA, 2003.

Heidi Dixon.
Automated Pseudo-Boolean Inference within the DPLL framework.
PhD thesis, University of Oregon, 2004.

Environment for the development and distribution of open source software (edos) fp6-ist-004312 [online].
2005.
Available from : http://www.edos-project.org.

Daniel Le Berre and Pascal Rapicault.
Dependency management for the eclipse ecosystem.
In *Proceedings of IWOCE2009 - Open Component Ecosystems International Workshop*, pages 21–30, August 2009.

Knot Pipatsrisawat and Adnan Darwiche.
On the power of clause-learning sat solvers as resolution engines.
Artif. Intell., 175(2) :512–525, 2011.

Niklas Eén Niklas Sörensson.
Translating pseudo-boolean constraints into sat.
JSAT, 2 :1–26, 2006.

Tommi Syrjänen.
A rule-based formal model for software configuration.
Master's thesis, Helsinki University of Technology, 1999.

Chris Tucker, David Shuffelton, Ranjit Jhala, and Sorin Lerner.
Opium : Optimal package install/uninstall manager.
In *ICSE*, pages 178–188. IEEE Computer Society, 2007.