The Pseudo-Boolean Evaluation

Second Edition (PB'06)

Vasco MANQUINHO and Olivier ROUSSEL

Ninth International Conference on
Theory and Applications of Satisfiability Testing,
SAT'06

August, 15th 2006

Outline

- linear pseudo-Boolean constraints and optimization problem
- advantages of using pseudo-Boolean constraints
- the big integers problem
- benchmarks and solvers
- evaluation environment
- comparing complete and incomplete solvers
- some results

pseudo-Boolean Constraints

- A **linear** pseudo-Boolean constraint may be defined over boolean variables by

$$\sum_i a_i.l_i \geq d \text{ with } a_i, d \in \mathbb{Z}, l_i \in \{x_i, \neg x_i\}, x_i \in \mathbb{B}$$

  Example: $3a - 3b + 2c + d + f \geq 5$
- It extends both clauses and cardinality constraints
  - cardinalities: all $a_i = 1$ and $d > 1$
  - clauses: all $a_i = 1$ and $d = 1$
- PB constraints are more expressive than clauses (one PB constraint may replace an exponential number of clauses)
- a pseudo-Boolean instance is a conjunction of PB constraints.

Optimization

- Another difference with SAT is that most PB problems contain a linear cost function to optimize. For example,

$$\text{minimize } f = \sum_i c_i.x_i \text{ with } c_i \in \mathbb{Z}, x_i \in \mathbb{B}$$

- Example of pseudo-Boolean Instance

$$\left\{ \begin{array}{ll} \text{minimize} & 5x_1 + x_2 + 8x_3 + 2x_4 + 3x_5 \\ \text{subject to} & x_1 + \bar{x}_2 + x_3 \geq 1 \\ & \bar{x}_1 + x_2 + \bar{x}_3 + x_4 \geq 3 \\ & 2\bar{x}_1 + 4x_2 + 2x_3 + x_4 + 5x_5 \geq 5 \\ & 5x_1 + 4x_2 + 6x_3 + x_4 + 3x_5 \geq 10 \end{array} \right.$$

Optimum: 8

$x_1 = x_2 = x_4 = 1$

$x_3 = x_5 = 0$

Easier problem encodings

Some examples:

- Cardinality constraints:
  at most one pigeon in a hole is encoded as
  p1h1+p2h1+p3h1+...+pNh1$\leq$1
- Adder:
  A+B=C (with A,B,C being n-bits registers) is encoded by one linear contraint
  $\sum_{i=0}^{n-1} 2^i.a_i + \sum_{i=0}^{n-1} 2^i.b_i = \sum_{i=0}^{n-1} 2^i.c_i$
- Multiplier:
  A*B=C (with A,B,C being n-bits registers) is encoded by one non-linear constraint
  $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 2^{i+j}.a_i.b_j = \sum_{i=0}^{n-1} 2^i.c_i$
  Introducing new variables $p_{ij} \Leftrightarrow a_i \wedge b_j$ gives one linear constraint and $3n$ clauses.

More powerful inference rules

- Cutting plane = weighted sum of constraints to eliminate one or more variables
- A few other inference rules
- The pigeon-hole problem encoded as cardinality constraints can be solved polynomially by pseudo-Boolean reasoning.

The Big Integers Problem

Interesting problems will quickly use numbers which are too large to fit in a usual integer variable (e.g. the multiplier problem).
Unfortunately, few solvers use arbitrary precision arithmetic

Without arbitrary precision arithmetic, a solver

- cannot solve all instances
- might give wrong answers caused by integer overflows

Multiprecision computations easily solve the problem (even if it has a cost).

Benchmark categories

Based on the objective function

- **SATUNSAT** No objective function to optimize. The solver must simply find a solution.
- **OPT** An objective function is present. The solver must find a solution with the best possible value of the objective function.

Based on the size of coefficients

- **SMALLINT** small integers: no constraint with a sum of coefficients greater than 2^{20} (20 bits)
- **MEDINT** medium integers: a constraint with a sum of coeff. with more than 20 bits but no constraint with a sum greater than 30 bits
- **BIGINT** big integers: at least one constraint with a sum of coefficients greater than 2^{30} (30 bits)

Additional categorization into Handmade, Random and Industrial benchmarks has been made.

Benchmarks

1753 benchmarks (almost 1GB).

- Submitted by contestants to PB05 or PB06
- Found on the web (OPB)
- Found on the web and translated (MPS format from linear programming)

For each instance with big integers, another instance with reduced coefficients was created (by dividing all coefficients by a common number). This doesn't preserve the semantics but ensures that we have at least as much instances in the SMALLINT category as in the BIGINT category.

Submitted solvers (1/2)

9 submitted solvers (and a few more versions)

absconPseudo Fred Hemery & Christophe Lecoutre
a CSP based solver

bsolo J. Marques-Silva & V. Manquinho
integrates SAT-based techniques with estimation procedures on the value of the cost function

glpPB Hossein Sheini & K. Sakallah
simple use of an integer linear programming toolkit

minisat+ Niklas Een & Niklas Sörensson
translates PB constraints to SAT

PB-smodels Gayathri Namasivayam
translates pseudo-Boolean constraints into a logic program accepted by *smodels* that solves search problems encoded as logic programs

Submitted solvers (2/2)

PBS
: Bashar AlRawi & Fadi Aloul
an extension of the *zchaff 2004* SAT solver to handle pseudo-Boolean constraints

Pueblo
: Hossein Sheini & K. Sakallah
an extension of the *minisat* SAT solver to handle pseudo-Boolean constraints; uses a general pseudo-Boolean learning mechanism

SAT4JPSEUDO
: Daniel Le Berre & Anne Parrain
a *Galena* like CDCL (Constraint Driven Constraint Learning) solver written in Java

wildcat-*
: Lengning Liu & Miroslaw Truszczynski
local search solver based on *wsat* generalized for pseudo-Boolean constraints

Only 3 solvers have support for big integers (bsolo, minisat+, SAT4JPSEUDO).

Evaluation environment

- Solvers must answer OPTIMUM FOUND, SATISFIABLE, UNSATISFIABLE or UNKNOWN. When answering OPTIMUM FOUND or SATISFIABLE solvers must output a certificate that is checked offline.
- Solvers were encouraged to output a line each time they found a better solution. These lines were timestamped and give information on the progression toward the best solution.
- On timeout, a solver receives a signal and can output the best solution it found. This year, we had a good way to stop multi-processes solvers on timeout.
- A preliminary test phase to detect bugs and other problems. In the final phase, buggy solvers were eliminated. No solver was found incorrect in the final phase (but OPTIMUM and UNSATISFIABLE answers cannot be completely verified for efficiency reasons)

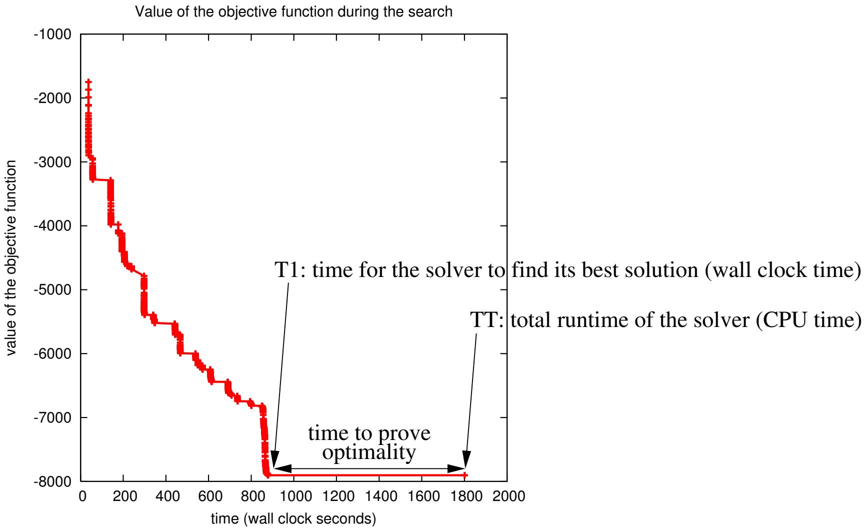
Evaluation environment

- Cluster of bi-Xeon 3 GHz, 2MB cache, 2GB RAM (but all solvers were run in 32 bits mode)
  kindly provided by the CRIL, University of Artois, France
- Each solver was given a time limit of 30 minutes (1800s) and a memory limit of 1800 MB (to avoid swapping).
- 237 days of CPU time used in the final phase

Complete and incomplete solvers

- Complete solvers are able to prove unsatisfiability (and therefore are able to prove optimality).
- Incomplete solvers can't prove unsatisfiability because they never stop (they don't know if they have gone through all the search space).
- TT (Total Time) is the CPU time used by a solver until completion. Useful to compare complete solvers. Useless for incomplete solvers (TT=timeout).
- T1 is the time needed by the solver to find its best solution. TT-T1 is the time needed to prove optimality (when the solver doesn't time out). For efficiency reasons, T1 is currently wall clock time and not CPU time. T1 is used to compare both incomplete and complete solvers (from an incomplete solver point of view)

Complete and incomplete solvers

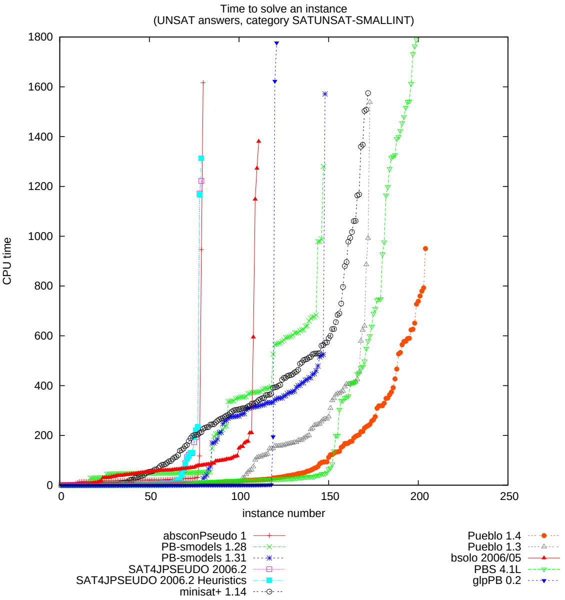


Comparing the solvers

Several ways with different point of views

- number of instances they solve completely (UNSAT or OPT answers)
- number of instances they solve partially (timeout, but a solution found)
- number of best solutions found
- number of times they are the fastest to give the best solution
- comparison of execution time
- ...

#instances solved in a given amount of time



Top solvers

A first approach on the number of solved instances

Category	UNSAT answers	SAT/OPT answers	Both answers
SATUNSAT-SMALLINT	Pueblo 1.4 PBS 4.1L	wildcat-skc wildcat-rnp	Pueblo 1.4 PBS 4.1L
OPT-SMALLINT	SAT4J Heur. SAT4J	bsolo minisat+	bsolo minisat+
OPT-BIGINT	SAT4J minisat+	SAT4J Heur. (*) minisat+	SAT4J Heur. (*) minisat+

(*) On a number of instances in these categories, it is known that the solver first tried to falsify variables and was therefore immediately very close to the best solution.

See the results by yourself

- See the poster to get a sample of the results !
- All the results are publicly available at http://www.cril.univ-artois.fr/PB06
- Experimentations are over but a more complete analysis of the results should be available in a few months.

Directions for next evaluation

- larger benchmark set (please contribute !) with trivial instances removed
- more solvers (write you own right now!)
- more tools to compare solvers
- merge the MEDIUM INTEGER category with BIG INTEGER
- possible extension to non-linear pseudo-Boolean constraints