From SAT to SAT4J

Providing efficient SAT solvers for the Java platform

Daniel Le Berre

CRIL-CNRS FRE 2499, Université d'Artois, Lens, FRANCE
leberre@cril.univ-artois.fr
http://www.sat4.org/

Sophia Antipolis - December 4^{th}, 2006

Agenda

The SAT problem

Definition

Input : A set of clauses built from a propositional language with n variables.
Output : Is there an assignment of the n variables that satisfies all those clauses ?

The SAT problem

Definition

Input : A set of clauses built from a propositional language with n variables.

Output : Is there an assignment of the n variables that satisfies all those clauses ?

Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For C_1, the answer is yes, for C_2 the answer is no

$$C_1 \models \neg(a \wedge \neg c) = \neg a \vee c$$

Where are clauses coming from ?

Suppose :

a *I like free software*

b *I should start a free software project*

c *I should use a free software language*

Then C_1 could represent the beliefs :

- $a \implies b$: *If I like free software, then I should start a free software project.*
- $b \implies c$: *If I start a free software project, then I should use a free software language.*

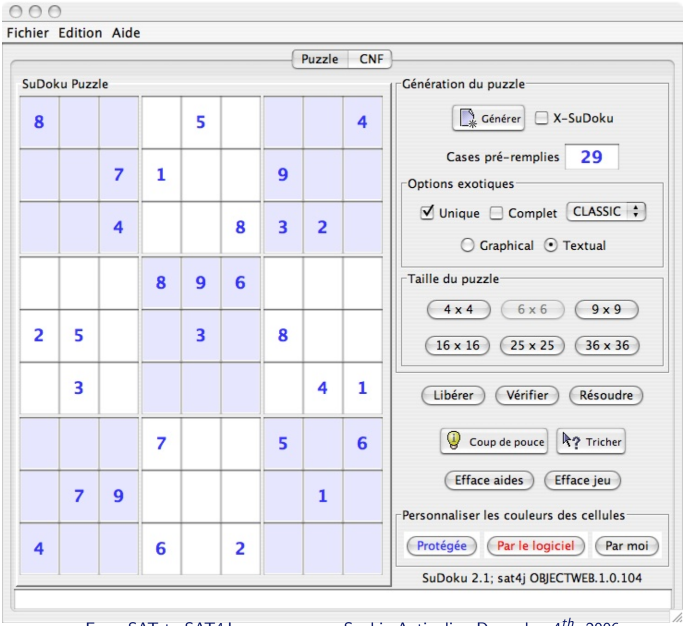
What happens if I like free software and I do not use a free software language ($a \wedge \neg c$) ? This is inconsistent with my beliefs. From C_1 I can deduce $a \implies c$: *If I like free software, then I should use a free software language.*

C_2 SAT instance read by SAT4J (DIMACS)

```
p cnf 3 4
-1 2 0
-2 3 0
1 0
-3 0
```

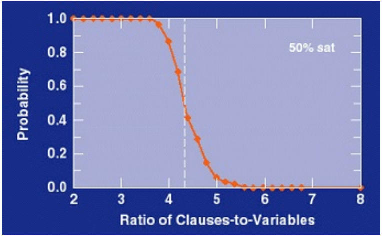
Not really fun !

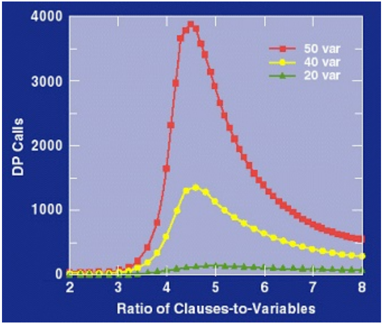
SAT can be fun !



SAT is important in theory ...

- Canonical NP-Complete problem (Cook, 1971)
- Threshold phenomenon on randomly generated k-SAT instances (Mitchell,Selman,Levesque, 1992)





source : http ://www.isi.edu/ szekely/antsebook/ebook/modeling-tools-and-techniques.htm

... and in practice !

- Many problems can be solved using a reduction into SAT :

  1996- Planning (SATPLAN,Blackbox)
  1998- Software Specification (NitPick, Alloy)
  1999- Bounded Model Checking, Equivalence checking, Formal Verification, etc.
  2005- Pseudo Boolean constraints
  2005- Constraints Satisfaction Problems

- SAT solvers are currently being used in production environments : Microsoft, Intel, IBM, Cadence, Synopsys, Valiosys, etc.
- Some people have fun with SAT : SuDoKu, Crosswords, Clue, etc.
- SAT technology is emerging in software engineering

Examples of user applications

- The impact of satisfiability for Linux users

  package dependencies EDOS project, Opium

  bug finder e.g. SATURN

- The impact of satisfiability in software engineering

  Software specification Alloy4, Kodkod

  Feature modeling AHEAD

  Requirements analysis OpenOME

  Many more ...

- SAT solving can also be useful for solving security related applications (e.g. cryptanalysis or access control)!

The SAT conference : www.satisfiability.org

- Workshops from Theoretical Computer Science or Artificial Intelligence

  1996 Siena, Italy (TCS)
  1998 Schloß Eringerfeld, Germany (TCS)
  2000 Renesse, Netherlands (TCS)
  2001 Boston, United States (IA)

- Yearly conference since 2002

  2002 Cincinnati, United States
  2003 Portofino, Italy
  2004 Vancouver, Canada
  2005 St Andrews, Scotland
  2006 Seattle, USA

- Approximately 100 persons attend the conference each year
- SAT'07 will take place in Lisbon, Portugal

The SAT Competition www.satcompetition.org

- The first competitions took place in the 90s :

1992	Paderborn, Germany
1993	2nd Dimacs challenge, United States
1996	Beijing, China

- Since 2002, it is a yearly event ! Numerous participants :

2002	27 solvers
2003	30 solvers
2004	55 solvers
2005	43 solvers

- In 2006, there was a SAT Race (industrial friendly), not a SAT competition !
- Other competitions created after the SAT competition :

2003,2004	QBF
2005	QBF, PB, CSP, SMT, ...
2006	QBF, PB, CSP, SMT, MAX-SAT, ...

- The consequences are sometimes unexpected...

The case of the termination competition

- solvers have one minute to prove that a term or string rewriting system terminates, e.g. :

```
INPUT: ( RULES b c -> a b b , b a -> a c b )
ANSWER: NO
Input system R is not terminating since R admits a
looping reduction from bcaaca to aacabacbcaacabbb
with 10 steps.
```

- huge success of the open source SAT solvers MiniSat and SatELite in the SAT 2005 competition
- Aprove, Jambox and Matchbox used them in 2006
- Results :

Aprove	best for term rewriting systems (except the relative termination subcategory) and for logic programs
Jambox	best for string rewriting systems and relat... termination of term rewriting

Agenda

- What does SAT mean ?
- Why is SAT successful ?
- The SAT4J project
- SAT4J : what about efficiency ?
- Pseudo Boolean Problems
- Constraint Satisfaction Problems
- MAXSAT
- Conclusion and future directions

Why does SAT receive much attention currently ?

- Most companies doing software or hardware verification are now using SAT solvers.
- Many SAT solvers are available from academia or the industry.
- SAT solvers can be used as a black box with a simple input/ouput language (DIMACS).
- A new kind of SAT solver was designed in 2001 (Chaff)
  - algorithmic improvements
  - new complexity/efficiency tradeoff
  - designed with hardware consideration/limitation in mind

Typical application for SAT : reachability analysis

Definition

Given an initial state s_0, a state transition relation ST, a goal state g and a bound k.

Is there a way to reach g from s_0 using ST within k steps ?

Is there a succession of states $s_0, s_1, s_2, ..., s_k = g$ such that $\forall\ 0 \leq i < k\ (s_{i-1}, s_i) \in ST$?

- The problems are generated for increasing k.
- For small k, the problems are usually UNSATISFIABLE
- For larger k, the problems can be either SAT or UNSAT.
- Complete SAT solvers are needed !

1992 - Planning As Satisfiability

$$PAS(S, I, T, G, k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} G(s_i)$$

where :

- **S** the set of possible states s_i
- **I** the initial state
- **T** transitions between states
- **G** goal state
- **k** bound

If the formula is satisfiable, then there is a plan of length k.

1997 - Software Model Analysis

$$SMA(S, op, p) = \exists s, s' \in S \; op(s, s') \wedge p(s) \wedge \neg p(s')$$

where :

- S the set of possible states
- op an operation
- p an invariant

If the formula is satisfiable, then there is an execution of the operation that break the invariant.

Focus on encoding data structures so that the set of states S could be structured

1999 - Bounded Model Checking

$$BMC(S, I, T, p, k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

where :

- **S** the set of possible states s_i
- **I** the initial state
- **T** transitions between states
- **p** is an invariant property
- **k** a bound

If the formula is satisfiable, then there is a counter-example reachable in k steps.

Focus on translating LTL formulas into SAT

Consequence in 2006

- Many Chaff-like solvers available in many languages.
- They can solve problems with millions of variables and clauses.
- SAT solvers are now designed to be embedded in other apps.
- Thanks to its standard input format, it is easy to test and use the latest SAT solvers available.

More and more applications are using SAT

Agenda

The SAT4J project www.sat4j.org

- An open source library of Chaff-like solvers in Java
- Project started late 2003 as an implementation in Java of the MiniSAT specification.
- Library updated continuously with latest SAT technologies
- Efficiency validated during the SAT competitions (2004 and 2005) and the SAT Race 2006.
- Can also handle other kind of constraints :

  cardinality $a + b + c + d \geq 3$

  pseudo boolean $3 * a + 2 * b + 2 * c + d \geq 3$
- Built-in Constraint Satisfaction Problem (CSP) to SAT support (Participated to the First CSP competition in Summer 2005).
- Built in optimization problems support.
- Target easy integration in any Java software !

Some SAT4J Users

Formal verification Kodkod project and Alloy4 (Daniel Jackson @ MIT)

Software engineering
- OpenOME (Yijun Yu et al @ U. Toronto)
- AHEAD (Don Batory @ U. Texas)
- FAMA (David Benavides @ U. Seville)

Semantic web Ontology matching in S-MATCH (Fausto Giunchiglia, Pavel Shvaiko and Mikalai Yatskevich @ U. Trento)

Constraints CONstraints ACQuisition (Christian Bessière, Rémi Coletta et al @ U. Montpellier)

Algorithm configuration Frank Hutter @ UBC

Other
- CROSSWORDS (Andy King and Colin Pigden)
- SUDOKU (Ivor Spence, U. Belfast)
- SAT4SATIN (Ibis group @ Vrije)

Use case 1 : OpenOME

`http://www.cs.toronto.edu/km/openome/`

OpenOME an Eclipse plugin for requirements engineering.

goal model to connect the user's high level requirements with the system's low level configuration items

preferences between goals : one goal is more important than another

expectations a goal needs to be satisfied to a certain degree

- Top-down reasoning propagates the expectations of high level goals downward to obtain the minimal number of low level goals that can fulfill the requirements.
- Top-down reasoning done with SAT4J

Use case 2 : AHEAD tool suite

`http://www.cs.utexas.edu/users/schwartz/ATS.html`

Product line family of programs differentiated by features

Constraint Not all features are compatible

Safe Composition Avoiding type errors in the composed code.

AHEAD theory of software synthesis based on feature composition

SAT used to :

- debug feature models
- perform safe composition

Use case 2 : AHEAD tool suite

`http://www.cs.utexas.edu/users/schwartz/ATS.html`

Product line family of programs differentiated by features

Constraint Not all features are compatible

Safe Composition Avoiding type errors in the composed code.

AHEAD theory of software synthesis based on feature composition

"Further, the performance of using SAT solvers to prove theorems was encouraging : non-trivial product-lines of programs of respectable size [40+ programs each with 35K Java LOC, ...] could be analyzed and verified in less than 30s."

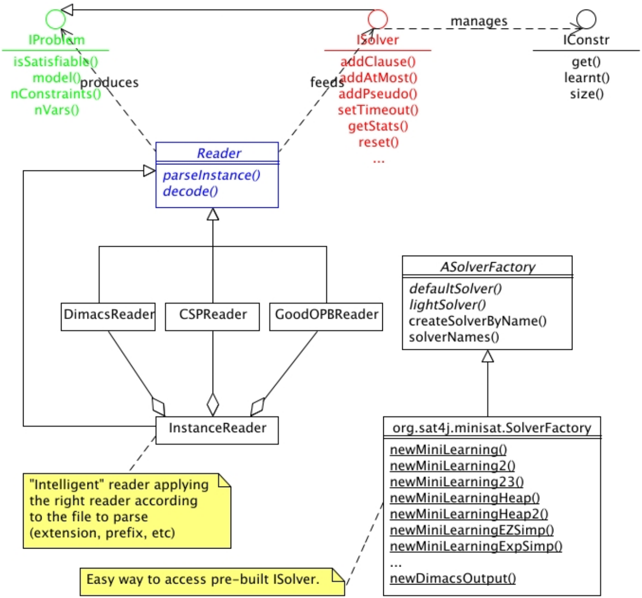
Don Batory and Sahil Thaker, Safe Composition of Product Lines

Use case 3 : Alloy

`http://alloy.mit.edu`

- 10 years old technology (Formerly Nitpick, 96)
- Followed the evolution of SAT solvers :
  - Started with WalkSAT/SATO
  - Then RELSAT/SATZ
  - Took the Chaff wave
  - Now uses MiniSAT
- Take advantage of new features in SAT solvers (e.g. unsat core)
- From the beginning in Java, relying on efficient C/C++ solvers (Java counterparts tried but abandoned)
- SAT4J allows a pure Java tool (still some problems with graph layout)

Overview of SAT4J architecture : End User view



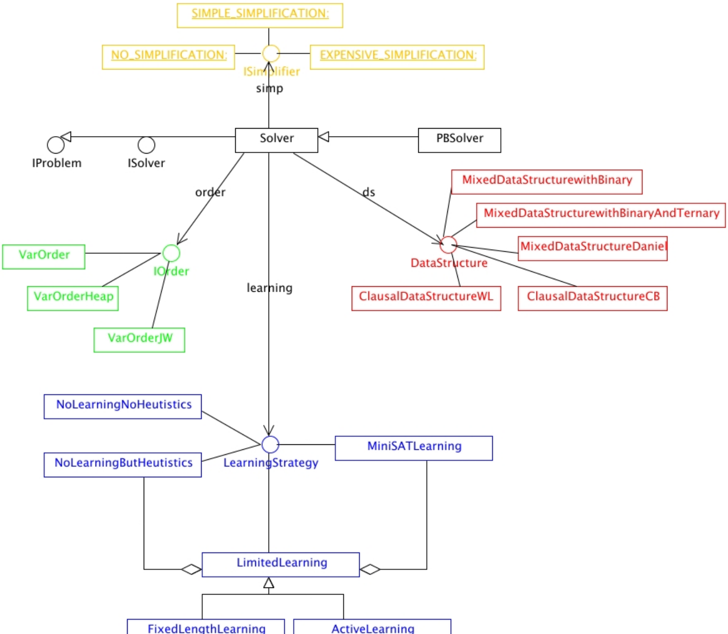
SAT4J for Java programmers : basic use

```java
ISolver solver = SolverFactory.newDefault();
solver.setTimeout(3600); // 1 hour timeout
Reader reader = new DimacsReader(solver);
try { // CNF filename is given on the command line
   IProblem problem = reader.parseInstance(args[0]);
   if (problem.isSatisfiable()) {
       System.out.println("Satisfiable !");
       System.out.println(reader.decode(problem.model()));
    } else {
       System.out.println("Unsatisfiable !");
    }
} catch (FileNotFoundException e) {
} catch (ParseFormatException e) {
} catch (IOException e) {
} catch (ContradictionException e) {
    System.out.println("Unsatisfiable (trivial)!");
} catch (TimeoutException e) {
    System.out.println("Timeout, sorry!");
}
```

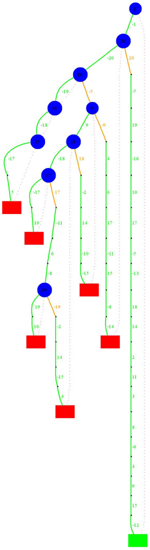
SAT4J for Java programmers : iterate over models

```java
ISolver solver = SolverFactory.newDefault();
ModelIterator mi = new ModelIterator(solver);
solver.setTimeout(3600); // 1 hour timeout
Reader reader = new InstanceReader(mi);
try {// filename is given on the command line
   boolean unsat = true;
   IProblem problem = reader.parseInstance(args[0]);
   while (problem.isSatisfiable()) {
       unsat = false;
       int [] model = reader.decode(problem.model());
       // do something with each model
    }
    if (unsat)
          // do something for unsat case
} catch (FileNotFoundException e) {
   [...]
} catch (ContradictionException e) {
    System.out.println("Unsatisfiable (trivial)!");
} catch (TimeoutException e) {
    System.out.println("Timeout, sorry!");
}
```

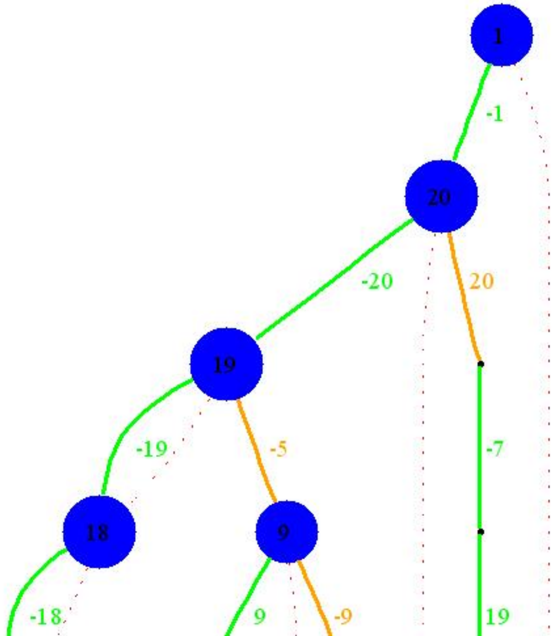
Overview of SAT4J architecture : Power User view



SAT4J search visualization option



SAT4J search visualization option



Agenda

Efficient SAT solving with SAT4J

- Many possible solver configurations available ($>$20)
- No real benchmarking of solvers made for the previous competitions
- SAT Race is special : qualification stage
- Testsets are provided for the race (50 benchmarks)

Efficient SAT solving with SAT4J

- Many possible solver configurations available (>20)
- No real benchmarking of solvers made for the previous competitions
- SAT Race is special : qualification stage
- Testsets are provided for the race (50 benchmarks)

Choosing best configuration for the race

First trial : test everything

Solver	Solved	SAT	UNSAT	Time	Out Of Memory
MiniLearning	36	13	23	319m31.771s	1
MiniLearningHeap	33	10	23	318m25.009s	5
MiniLearningHeapEZSimp	36	12	24	283m8.689s	3
MiniLearning2	33	10	23	389m43.783s	0
MiniLearning2Heap	36	13	23	299m1.987s	0
MiniLearning23	26	12	14	437m10.415s	0
MiniLearningCB	19	8	11	482m26.646s	1
MiniLearningCBWL	27	8	19	402m44.606s	1
MiniLearning2NewOrder	33	13	20	367m30.035s	0
MiniLearningPure	30	8	22	388m40.632s	1
MiniLearningCBWLPure	27	8	19	416m51.705s	1
MiniLearningEZSimp	35	12	23	309m29.826s	1
MiniLearningNoRestarts	31	10	21	379m31.950s	3
ActiveLearning	34	11	23	318m59.267s	1
MiniSAT	33	11	22	333m8.418s	1
MiniSATNoRestarts	30	9	21	377m36.427s	3
MiniSAT2	33	10	23	377m36.427s	0
MiniSAT23	25	11	14	437m44.362s	0
MiniSATHeap	33	10	23	298m31.360s	5
MiniSAT2Heap (default)	36	13	23	297m49.641s	1
MiniSAT23Heap	24	11	13	430m59.189s	2
Relsat	22	6	16	417m22.977s	7
Backjumping	10	7	3	621m9.357s	1

Some competitors

Solver	Solved	SAT	UNSAT	Time
MiniSat 1.14	38	12	26	230m56.139s
zChaff 2004.11.15	34	9	25	368m26.901s
Siege_v4	45	16	29	186m36.902s
SatELite (not GTI)	32	10	22	350m3.909s

Second trial : change memory management

Solver	#	SAT	UNSAT	Time	OOM
MiniLearning	35	12	23	326m10.754s	1
MiniLearningHeap	34	11	23	317m48.771s	5
MiniLearningHeapEZSimp	37	12	25	277m16.920s	4
MiniLearning2	33	9	24	363m50.988s	0
MiniLearning2Heap	37	13	24	279m29.925s	2
MiniLearning2NewOrder	35	12	23	360m2.050s	0
MiniLearningHeap	35	11	24	313m51.323s	1
Activelearning	33	12	23	332m2.813	1
MiniSAT	34	11	23	331m54.472s	1
MiniSAT2	34	10	24	348m58.966s	0
MiniSAT23	33	10	23	354m26.744s	0
MiniSATHeap	35	12	23	291m13.961s	5
MiniSAT2Heap	36	12	24	294m48.496s	3
MiniSATHeapEZSimp	37	12	25	296m24.180s	3

Second trial : change memory management

Solver	#	SAT	UNSAT	Time	OOM
MiniLearning	35	12	23	326m10.754s	1
MiniLearningHeap	34	11	23	317m48.771s	5
MiniLearningHeapEZSimp	37	12	25	277m16.920s	4
MiniLearning2	33	9	24	363m50.988s	0
MiniLearning2Heap	37	13	24	279m29.925s	2
MiniLearning2NewOrder	35	12	23	360m2.050s	0
MiniLearningHeap	35	11	24	313m51.323s	1
Activelearning	33	12	23	332m2.813	1
MiniSAT	34	11	23	331m54.472s	1
MiniSAT2	34	10	24	348m58.966s	0
MiniSAT23	33	10	23	354m26.744s	0
MiniSATHeap	35	12	23	291m13.961s	5
MiniSAT2Heap	36	12	24	294m48.496s	3
MiniSATHeapEZSimp	37	12	25	296m24.180s	3
MiniLearningHeapExpSimp	42	14	28	297m32.545s	0

Second trial : change memory management

Solver	#	SAT	UNSAT	Time	OOM
MiniLearning	35	12	23	326m10.754s	1
MiniLearningHeap	34	11	23	317m48.771s	5
MiniLearningHeapEZSimp	37	12	25	277m16.920s	4
MiniLearning2	33	9	24	363m50.988s	0
MiniLearning2Heap	37	13	24	279m29.925s	2
MiniLearning2NewOrder	35	12	23	360m2.050s	0
MiniLearningHeap	35	11	24	313m51.323s	1
Activelearning	33	12	23	332m2.813	1
MiniSAT	34	11	23	331m54.472s	1
MiniSAT2	34	10	24	348m58.966s	0
MiniSAT23	33	10	23	354m26.744s	0
MiniSATHeap	35	12	23	291m13.961s	5
MiniSAT2Heap	36	12	24	294m48.496s	3
MiniSATHeapEZSimp	37	12	25	296m24.180s	3
MiniLearningHeapExpSimp	42	14	28	297m32.545s	0
Release 1.7, Java 6 RC	42	14	28	276m31.717s	0

Lessons learnt

heap/array Heap based heuristics are definitely better for those benchmarks

2/3 specific binary data structures are helpful in some cases, better for solving satisfiable benchmarks.

Reason Simplification helps for solving UNSAT benchmarks. Expensive reason simplification from MiniSAT 1.14 is the best option for the SAT Race.

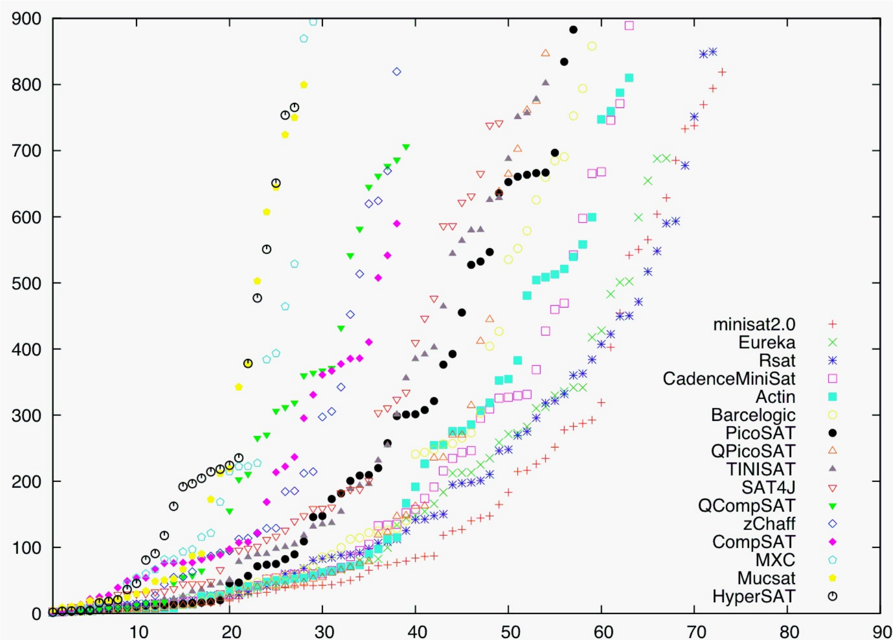
learning Filtering learnt clauses preserve efficiency.

memory management is the weakest part of SAT4J : despite a regular cleanup of the learnt clauses, the solver runs out of memory after a while.

Are solvers in SAT4J state-of-the-art ?

+ MiniSAT is currently the best available open source SAT solver (C++) : SAT4J started as a Java implementation of the original MiniSAT
+ SAT4J is a mature software (almost 3 year old) : core library has been fine tuned over the years
+ SAT4J is updated regularly with latest proven successful techniques
+ Java VMs are more and more powerfull : Java 6 VM will provide 20% speedup for free
- Preprocessing available in MiniSAT 2.0 is not available in SAT4J
- SAT4J is designed for flexibility : fastest SAT solvers reimplement everything from scratch for heavy tuning !

Results of SAT4J during the SAT Race 2006



Agenda

What does SAT mean ?

Why is SAT successful ?

The SAT4J project

SAT4J : what about efficiency ?

Pseudo Boolean Problems

Constraint Satisfaction Problems

MAXSAT

Conclusion and future directions

Linear pseudo boolean constraints : definitions

- boolean variables x_i, truth value $\in \{0, 1\}$.
- $\overline{x_i} = 1 - x_i$.
- General form :

$$\sum_i a_i.x_i \triangleright k$$

  where a_i and k are constants (integer or real) and $\triangleright \in \{=, >, \geq, <, \leq\}$.
- k is called the *degree* of the constraint.
- Example : $3x_1 - 4x_2 + 7\overline{x_3} - x_4 \leq 2$

Specific cases

Clauses and cardinality constraints can be seen as special cases of linear pseudo boolean constraints.

- $x_1 \vee x_2 \vee \ldots x_n$ translates to
  $x_1 + x_2 + \ldots + x_n \geq 1$
- $atleast(k, \{x_1, x_2, \ldots, x_n\})$ translates to
  $x_1 + x_2 + \ldots + x_n \geq k$
- $atmost(k, \{x_1, x_2, \ldots, x_n\})$ translates to
  $\overline{x_1} + \overline{x_2} + \ldots + \overline{x_n} \geq n - k$.

Resolution on clauses = cutting planes on LPBC

cutting planes:
$$\frac{\begin{array}{c}\sum_i a_i.x_i \geq k \\ \sum_i a'_i.x_i \geq k'\end{array}}{\sum_i(\alpha.a_i + \alpha'.a'_i).x_i \geq \alpha.k + \alpha'.k'}$$
with $\alpha > 0$ and $\alpha' > 0$

- we may form a combination which doesn't eliminate any variable.
- one single linear combination may eliminate more than one variable.

Resolution on clauses = cutting planes on LPBC

cutting planes:
$$\frac{\sum_i a_i.x_i \geq k \quad \sum_i a'_i.x_i \geq k'}{\sum_i(\alpha.a_i + \alpha'.a'_i).x_i \geq \alpha.k + \alpha'.k'}$$
$$\text{with } \alpha > 0 \text{ and } \alpha' > 0$$

- we may form a combination which doesn't eliminate any variable.
- one single linear combination may eliminate more than one variable.

cutting planes:
$$\frac{x_1 + x_2 + x_3 \geq 4 \qquad 2\overline{x_1} + 2\overline{x_2} + x_4 \geq 3}{(2x_1 + 2(1 - x_1)) + 2 + 2x_3 + x_4 \geq 8 + 3}$$
$$2x_3 + x_4 \geq 7$$

SAT4JPseudo : Replacing resolution by cutting planes ...

- Using the CDCL framework proposed by GRASP
- With some improvements coming from Chaff (VSIDS, First UIP)
- Cutting planes are used during conflict analysis to generate an assertive constraint.
- Proposed first in Galena (Chai&Kuehlmann 2003) and PBChaff (Dixon 2002/2004).
  - Cardinality approach preferred to Full CP
  - No management of integer overflow
  - Solvers no longer developed

The Pseudo Boolean evaluations

- Organized by Olivier Roussel and Vasco Manquinho in 2005 and 2006
- Uniform input format
- Independent assessment of the PB solvers
- Results freely available in details
- first comprehensive repository of benchmarks
- Various technologies used in 2006

The solvers
Summary of features

	Mini-Sat+	SAT-4J	Pue-blo	PBS	Bsolo	glpPB
Input	Clauses	LPBC	LPBC	LPBC	LPBC	LPBC
Inference	Res.	Full C.P. (boo-lean)	Mixed	Mixed	Mixed	Full C.P. (real)
Optimization	L.S.	L.S.	L.S.	L.S.	B'n'B	Sim-plex

Partial results of the PB05 evaluation

	Mini-Sat+	SAT-4J	Pueblo	PBS	Bsolo	
Decision problems	43 (35)	52 17	61 42	61 28	36 8	UNS SAT
Opt. Small	10 176 (120)	10 120 (226)	10 160 182	10 133 0	10 159 180	UNS OPT SAT
Opt. Medium	0 24 (67)	2 19 107	0 34 74	0 33 0	0 28 82	UNS OPT SAT
Opt. Big	103 26 (64)	85 3 (171)	-	-	90 9 83	UNS OPT SAT

Partial results of the PB06 evaluation

	PB06					Own	
	Mini-Sat+	SAT4J C.P.	Pueblo 1.4	PBS 4.1L	Bsolo	SAT4J Res.	
Decision pbms	172	79	**204**	199	111	165	UNS
	148	92	**153**	144	118	121	SAT
Opt. Small	43	**54**	37	29	40	35	UNS
	405	357	385	352	**409**	367	OPT
	250	303	**323**	0	280	(267)	SAT
Opt. Medium	0 3	0 4	0 4	0 5	0 6	0 5	UNS
	9	9	15	0	7	(9)	OPT
							SAT
Opt. Big	38	37	-	-	30	**40**	UNS
	33	57			14	**72**	OPT
	52	77			69	**96**	SAT

See `http://www.cril.univ-artois.fr/PB06/results/` for details.

Partial detailed results of the PB06 evaluation

	#	MSat+	SAT4J	Pueblo	PBS	Bsolo	glpPB
SAT/UNSAT							
pigeon	20	2	**20**	13	**20**	2	20
queens	100	**100**	18	99	**100**	**100**	**100**
tsp	100	91	20	**100**	85	40	42
fpga	57	35	43	**57**	47	9	26
uclid	50	**47**	30	42	44	38	10
OPT SMALLINT							
minprime	156	**124**	104	118	103	106	52
red.-mps	273	46	**70**	63	27	54	58
OPT BIGINT							
factor.	100	14	**52**	-	-	7	-
Ardal problems (one eq. constraint)							
Ardal_1	12	**10**	2	0	3	2	0

See http://www.cril.univ-artois.fr/PB06/results/ for details.

Good cases for SAT4JPseudo

pigeon hole solvers using resolution cannot solve them. A nice way to check the inference engine of the solvers.

reduced mps Those benchmarks are composed of real LPBC, so solvers with CP capabilities have good results on them.

factorization SAT4J Heuristics was lucky on half of the benchmarks, because of the way it initializes the phase of the variables to branch on according to the objective function.

Bad cases for SAT4JPseudo

TSP and Weighted Queens problems contributed by Gayathri Namasivayam for PB06. Much more clauses than cardinality constraints or PB constraints.

One typical example from the Queens problem :

SAT4J C.P. timeout at 1800 seconds after only 7 restarts for 2338 conflicts at 7 decisions/second

SAT4J Resolution 35 seconds after 16 restarts, 95829 conflicts at 3320 decisions per seconds

The difference lies in the conflict analysis procedure !

Bad cases for SAT4JPseudo

TSP and Weighted Queens problems contributed by Gayathri Namasivayam for PB06. Much more clauses than cardinality constraints or PB constraints.

One typical example from the Queens problem :

SAT4J C.P.
: timeout at 1800 seconds after only 7 restarts for 2338 conflicts at 7 decisions/second

SAT4J Resolution
: 35 seconds after 16 restarts, 95829 conflicts at 3320 decisions per seconds

The difference lies in the conflict analysis procedure !

- Bad results of SAT4JPseudo during the evaluations do not mean Full C.P. approach is wrong : it depends of the implementation (c.f. PB2SAT @ PB05)
- Results heavily depend on the kind of benchmarks : many easy benchmarks make the comparison of solvers difficult.

Agenda

From CSP to SAT

A CSP is a triplet (X,D,C) such that

- **X** $= \{X_1, X_2, ..., X_n\}$ is a set of n variables
- **D** is the domain function that maps to each variable X_i its domain $D(X_i)$, i.e., the set of possible values for X_i.
- **C** $= \{C_1, C_2, ...C_m\}$ is a set of constraints. Each constraint C_j is a relation among the possible values for its variables.

From CSP to SAT : naive version (Walsh,2000)

Variables for each variable X_i, and each value $d_j \in D(X_i)$, a new propositional variable$p_{i,j}$ is created.

Domains for each variable X_i, a cardinality constraint specify that a single value can be selected from the domain : $\sum_x p_{i,x} = 1$.

Forbidden Tuples (nogoods) Each forbidden tuple $(x_1, x_2, ..., x_k)$ is represented by a clause of length k containing the negated proposition variables representing the values x_i.

Authorized Tuples (supports) Compute the complementary forbidden tuples and proceed as above.

Example : 3-queens

- $X = \{X_1, X_2, X_3\}$
- $D(X_i) = \{1, 2, 3\} \forall i$
- Création des variables propositionnelles

  v_{01}, v_{02}, v_{03}

  v_{10}, v_{12}, v_{23}

  v_{20}, v_{12}, v_{23}
- Relation 1 (nogood) R1 :

$$(1,1)(1,2)(2,2)(2,1)(2,3)(3,3)(3,2)$$

- Relation 2 (nogood) R2 :

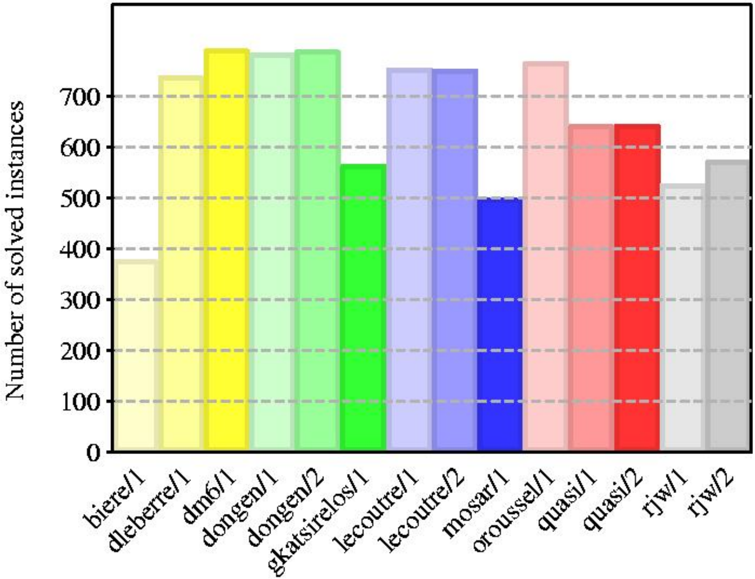
$$(1,1)(1,3)(2,2)(3,3)(3,1)$$

- $C = \{C_1 = R1(X_1, X_2), C_2 = R2(X_1, X_2), C_3 = R1(X_2, X_3)\}$

Constraints produced

- The domain definitions produce 3 cardinality constraints $v_{01} + v_{02} + v_{03} = 1, v_{11} + v_{12} + v_{13} = 1, v_{21} + v_{22} + v_{23} = 1.$
- C_1 produces 9 binary clauses : $\neg v_{01} \vee \neg v_{11}, \neg v_{01} \vee \neg v_{12}, \neg v_{02} \vee \neg v_{12}, \neg v_{02} \vee \neg v_{11}, \neg v_{02} \vee \neg v_{13}, \neg v_{03} \vee \neg v_{13}, \neg v_{03} \vee \neg v_{12}$
- C_2 produces 5 binary clauses : $\neg v_{01} \vee \neg v_{11}, \neg v_{01} \vee \neg v_{13}, \neg v_{02} \vee \neg v_{12}, \neg v_{03} \vee \neg v_{13}, \neg v_{03} \vee \neg v_{11}$
- C_3 produces 9 binary clauses : $\neg v_{11} \vee \neg v_{21}, \neg v_{11} \vee \neg v_{22}, \neg v_{12} \vee \neg v_{22}, \neg v_{12} \vee \neg v_{21}, \neg v_{12} \vee \neg v_{23}, \neg v_{13} \vee \neg v_{23}, \neg v_{13} \vee \neg v_{22}$

Results of the first CSP competition (binary constraints)



source : http ://cpai.ucc.ie/05/CallForSolvers.html

binary/n-ary

	SAT-based			Dedicated		
	biere	dleberre	roussel	dm6	dongen	lecoutre
non binary constraints (147 benchmarks)						
Solved	26	52	50	-	70	97
Time	262	2425	1952	-	2337	8031
binary constraints (922 benchmarks)						
Solved	377	739	769	822	818	759
Time	19894	15859	8070	12679	13642	18460

Selection of solvers that participated to the CSP05 .

source : http ://cpai.ucc.ie/05/CallForSolvers.html

From CSP to SAT : support version (Gent,2002)

Replace the translation of authorized binary tuples by constraints preserving arc consistency.

- For each set of authorized binary tuple like $C = \{(a, b_1), (a, b_2), ..., (a, b_n)\}$
- Create a clause $\neg a \vee b_1 \vee b_2 \vee ... \vee b_n$
- Needed in both directions : for a, but also for b_i.
- For values not appearing in the constraints, unit negative clause !

Advantage Forbidden tuple computation no longer needed !

Drawback Produced clauses are no longer binary
Limited to binary constraints

Naive/support comparison

On some benchmarks, the difference is obvious :

	Naïve (s)	Translation time (s)	Support (s)
hanoi3	1	<1	1
hanoi4	18	1	2
hanoi5	731	33	2
hanoi6	-	1840	7
hanoi7	-	-	22

qk1 benchmarks (18 instances)

- naïve: no instance solved with 10mn TO each
- support: all solved (UNSAT) in less than 2mn

January 2006 results (CRIL internal)

	SAT4J	Dedicated 1	Dedicated 2
non binary constraints (186 benchmarks)			
UNSAT	27	-	28
SAT	61	-	125
binary constraints (2031 benchmarks)			
UNSAT	842	1004	995
SAT	760	840	827

January 2006 results (CRIL internal) : non random

	SAT4J	Dedicated 1	Dedicated 2
non binary constraints (150 benchmarks)			
UNSAT	27	-	28
SAT	48	-	108
binary constraints (1041 benchmarks)			
UNSAT	400	386	396
SAT	560	536	536

September 2006 : Second CSP competition, first stage

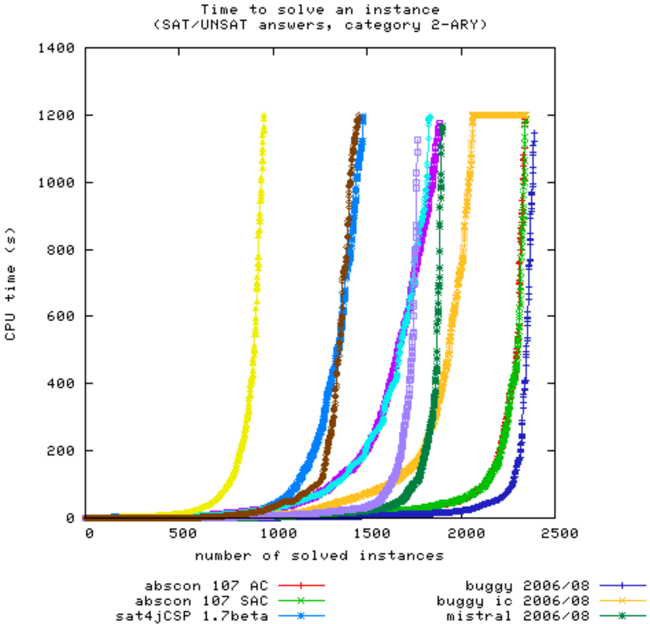
	SAT4J	Abscon	BProlog	Buggy
non binary constraints (978 benchmarks)				
UNSAT	68	77	46	-
SAT	273	429	379	-
Total	341	506	425	-
binary constraints (2673 benchmarks)				
UNSAT	614	1053	598	1066
SAT	864	1290	858	1322
Total	1478	2343	1456	2388

September 2006 : Second CSP competition, first stage

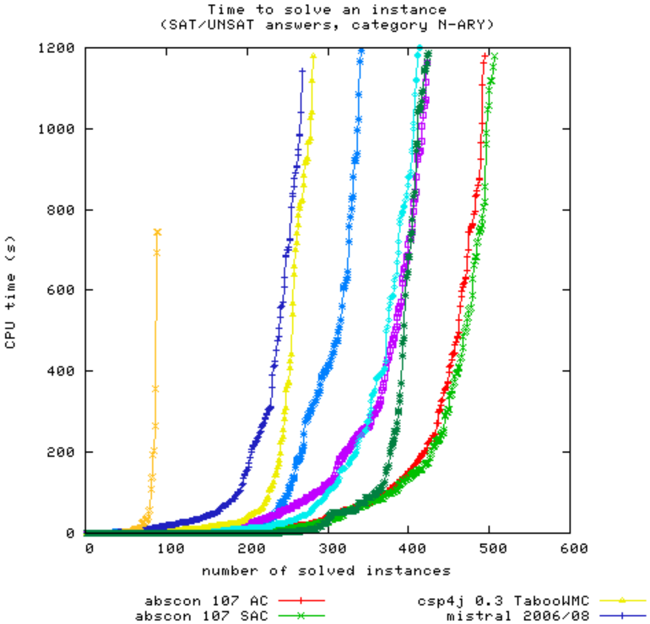
	SAT4J	Abscon	BProlog	Buggy
non binary constraints (978 benchmarks)				
UNSAT	68	77	46	-
SAT	273	429	379	-
Total	341	506	425	-
binary constraints (2673 benchmarks)				
UNSAT	614	1053	598	1066
SAT	864	1290	858	1322
Total	1478	2343	1456	2388

- For the first competition, constraints were given in extension.
- For the second competition, they can be given in intention.
- SAT-based encoding requires extensional form : it is sometimes impossible to generate it from the intensional form.

Results of the second CSP competition (binary constraints)



Results of the second CSP competition (n-ary constraints)



Agenda

MAXSAT and the optimization framework

- Can use a linear search to solve optimization problems :
  1. Find a solution
  2. Evaluate its cost function
  3. Add a new constraint to limit the search to better solutions
  4. Repeat until no more solutions : latest one is optimal
- Allow solving MAXSAT by adding one selector variable per clause
- MAXSAT solver submitted to the first MAXSAT evaluation
- Results where pretty bad for MAXSAT (underlying SAT solver might not be appropriate). Binary Search and Linear Search solvers based on zChaff confirmed those bad results.
- Good results on one class of benchmarks in the weighted MAX-SAT category.

Agenda

What does SAT mean ?

Why is SAT successful ?

The SAT4J project

SAT4J : what about efficiency ?

Pseudo Boolean Problems

Constraint Satisfaction Problems

MAXSAT

Conclusion and future directions

Conclusion

- SAT4J is a mature library of SAT solvers in Java
- The library allows easy integration of SAT technology into Java programs
- Additional features are provided :
  - Pseudo Boolean solving
  - CSP to SAT translation
  - Optimization framework
- SAT4J evolves with SAT technology : new state-of-the-art features are integrated regularly.

Future directions

- Improving Pseudo Boolean Solving
- Allowing reasoning on new And-Inverter Graph input
- Allowing manipulation of CSP constraints without grounding them
- Adding some MiniSAT 2.0 preprocessing techniques
- Improving user documentation and tutorials
- Separation of core SAT/PB/CSP code in next major release 2.0
- Release 1.7 is the first community driven release of SAT4J : more user-oriented features expected in the future
- Grid/Distributed Computing (Ibis and ProActive)