Introduction to SAT

History, Algorithms, Practical considerations

Daniel Le Berre [1]

CNRS - Université d'Artois

SAT-SMT summer school
Semmering, Austria, July 10-12, 2014

1. Contains material provided by Joao Marques Silva, Armin Biere, Takehide Soh

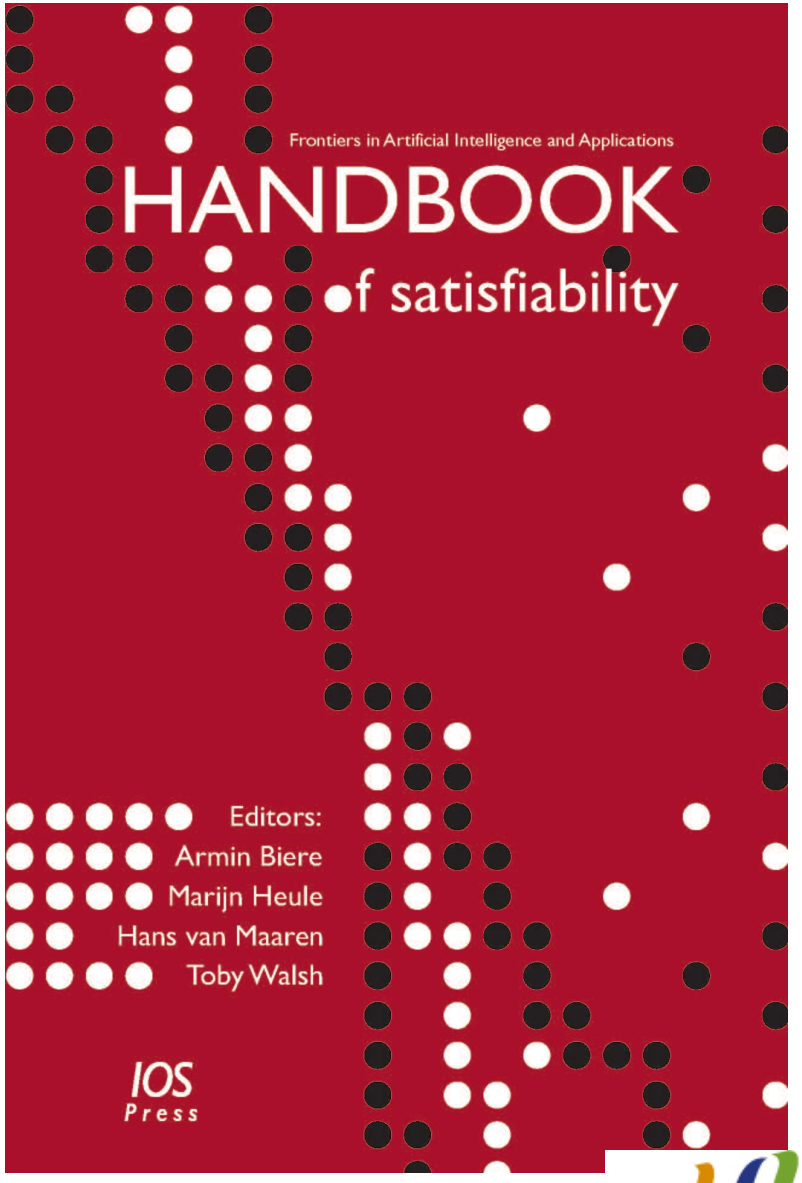
UNIVERSITÉ D'ARTOIS

Agenda

- Introduction to SAT
- A bit of history (DP, DPLL)
- The CDCL framework (CDCL is not DPLL)
  - Grasp
  - From Grasp to Chaff
  - Chaff
  - Anatomy of a modern CDCL SAT solver
- Nearby SAT
  - MaxSat
  - Pseudo-Boolean Optimization
  - MUS
- SAT in practice : working with CNF

Disclaimer

- Not a complete view of the subject
- Limited to one branch of SAT research (CDCL solvers)
- From an AI background point of view
- From a SAT solver designer
- For a broader picture of the area, see the handbook edited in 2009 by the community



UNIVERSITÉ D'ARTOIS

Disclaimer : continued

- Remember that the best solvers for practical SAT solving in the 90's where based on local search or randomized DPLL
- This decade has been the one of Conflict Driven Clause Learning solvers.
- The next one may rise a new kind of solvers (parallel architectures) ...

Agenda

Context : SAT receives much attention since a decade

Why are we all here today ?

- Most companies doing software or hardware verification are now using SAT solvers.
- SAT technology indirectly reaches our everyday life :
  - Intel core I7 processor designed with the help of SAT solvers [Kaivola et al, CAV 2009]
  - Windows 7 device drivers verified using SAT related technology (Z3, SMT solver) [De Moura and Bjorner, IJCAR 2010]
  - The Eclipse open platform uses SAT technology for solving dependencies between components [Le Berre and Rapicault, IWOCE 2009]
- 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).
- The consequence of a new kind of SAT solver designed in 2001 (Chaff)

The SAT problem : theoretical point of view

Definition

Input : A set of clauses C 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 : theoretical point of view

Definition

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

Example

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

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

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

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

The SAT ~~problem~~ solver : practical point of view

Definition

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

Output : If there is an assignment of the n variables that satisfies all those clauses, provide such assignment, else provide a subset of C which cannot be satisfied.

The SAT ~~problem~~ solver : practical point of view

Definition

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

Output : If there is an assignment of the n variables that satisfies all those clauses, provide such assignment, else provide a subset of C which cannot be satisfied.

Example

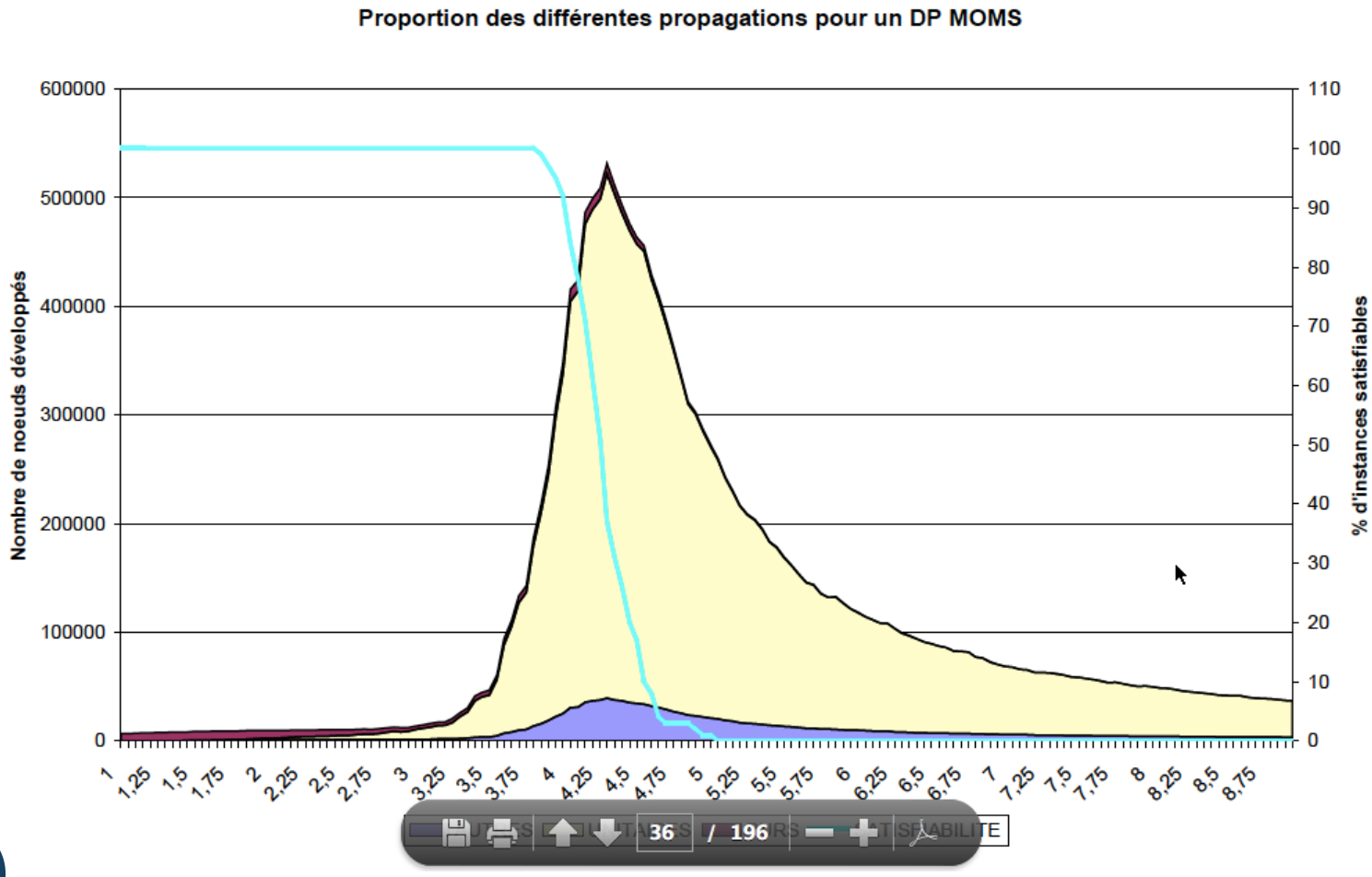
$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

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

For C_1, one answer is $\{a, b, c\}$, for C_2 the answer is C_2

SAT is important in theory ...

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



UNIVERSITÉ D'ARTOIS

... in practice : Computer Aided Verification Award 2009

awarded to

Conor F. Madigan
Sharad Malik
Joao Marques-Silva
Matthew Moskewicz
Karem Sakallah
Lintao Zhang
Ying Zhao

for

fundamental contributions to the development of high-performance Boolean satisfiability solvers.

Authors of GRASP SAT solver
Authors of CHAFF SAT solver

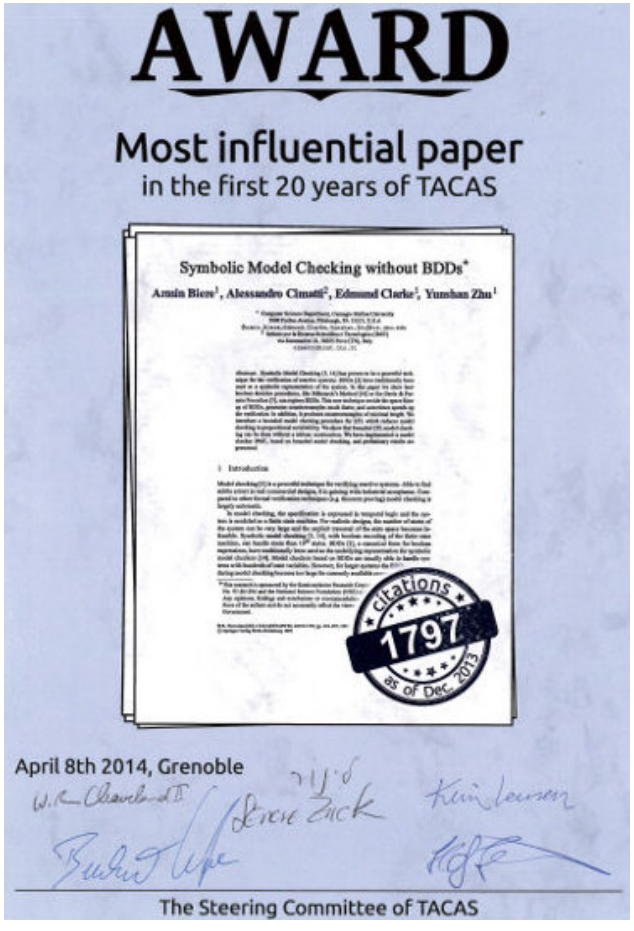
... TACAS 2014 most influential paper in the first 20 years

awarded to

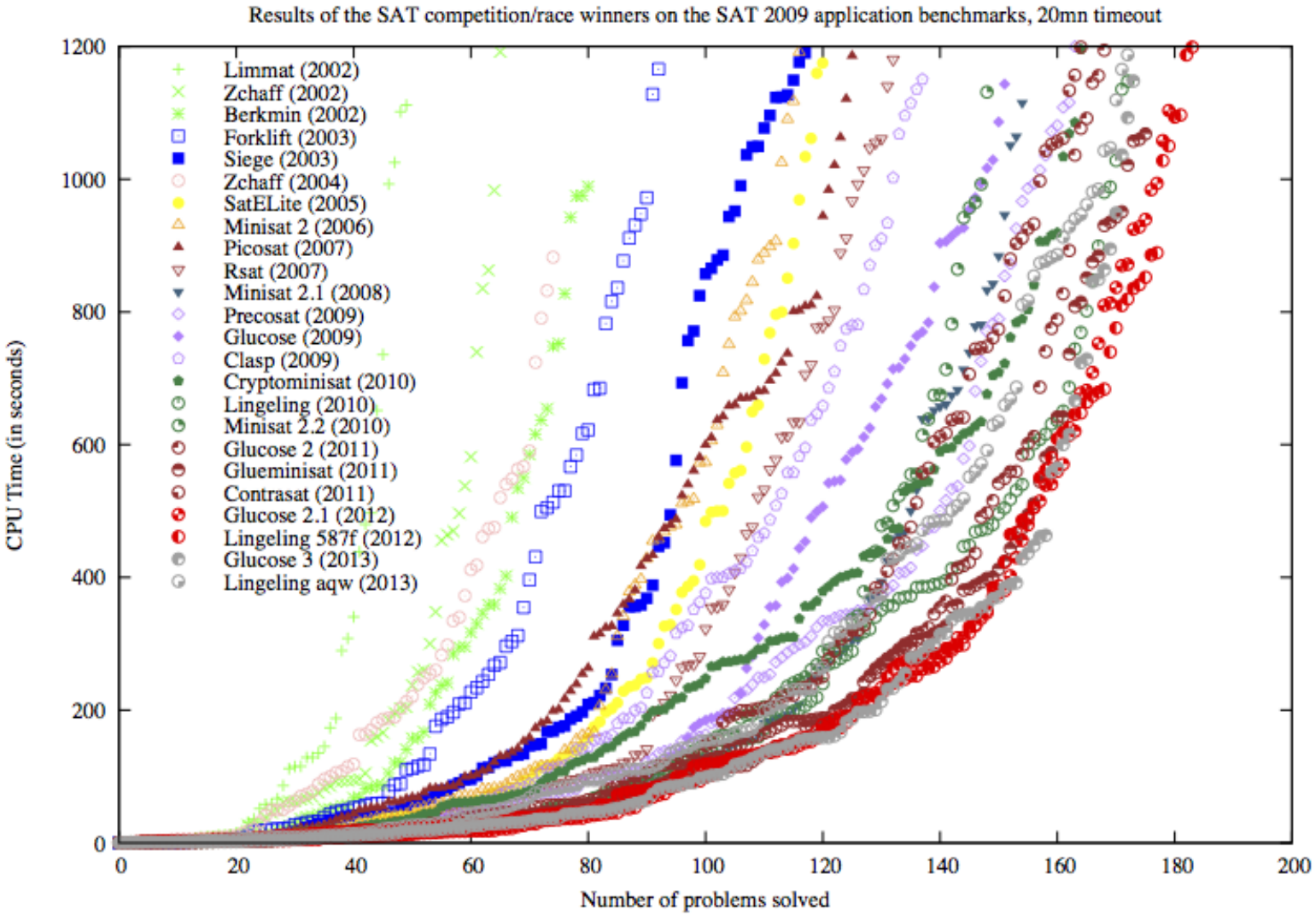
A. Biere
A. Cimatti
E. Clarke
Y. Zhu

for

Symbolic Model Checking without BDDs



Evolution of the performance of some SAT solvers



UNIVERSITÉ D'ARTOIS

Where can we find SAT technology today ?

- Formal methods :
  - Hardware model checking ; Software model checking ; Termination analysis of term-rewrite systems ; Test pattern generation (testing of software & hardware) ; etc.
- Artificial intelligence :
  - Planning ; Knowledge representation ; Games (n-queens, sudoku, social golpher's, etc.)
- Bioinformatics :
  - Haplotype inference ; Pedigree checking ; Analysis of Genetic Regulatory Networks ; etc.
- Design automation :
  - Equivalence checking ; Delay computation ; Fault diagnosis ; Noise analysis ; etc.
- Security :
  - Cryptanalysis ; Inversion attacks on hash functions ; etc.

Where can we find SAT technology today ? II

- Computationally hard problems :
  - Graph coloring ; Traveling salesperson ; etc.
- Mathematical problems :
  - van der Waerden numbers ; Quasigroup open problems ; etc.

- Core engine for other solvers : 0-1 ILP/Pseudo Boolean ; QBF ; #SAT ; SMT ; MAXSAT ; ...
- Integrated into theorem provers : HOL ; Isabelle ; ...
- Integrated into widely used software :
  - Suse 10.1 dependency manager based on a custom SAT solver.
  - Eclipse provisioning system based on a Pseudo Boolean solver.
  - Eiffel language uses Z3 to check contracts.

Agenda

Boolean Formulas

- Boolean formula φ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{T, F\}$
  - Example : $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example : $\varphi(x_1, \ldots, x_3) \equiv$
- A formula φ in disjunctive normal form (DNF) is a disjunction of conjunctions (terms) of literals
  - Example : $\varphi(x_1, \ldots, x_3) \equiv$
- Can encode any Boolean formula into Normal Form

Boolean Formulas

- Boolean formula φ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{T, F\}$
  - Example : $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example : $\varphi(x_1, \ldots, x_3) \equiv (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in disjunctive normal form (DNF) is a disjunction of conjunctions (terms) of literals
  - Example :
    $\varphi(x_1, \ldots, x_3) \equiv$
- Can encode any Boolean formula into Normal Form

Boolean Formulas

- Boolean formula φ is defined over a set of propositional variables $x_1, \ldots, x_n$, using the standard propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and parenthesis
  - The domain of propositional variables is $\{T, F\}$
  - Example : $\varphi(x_1, \ldots, x_3) = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example : $\varphi(x_1, \ldots, x_3) \equiv (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A formula φ in disjunctive normal form (DNF) is a disjunction of conjunctions (terms) of literals
  - Example : $\varphi(x_1, \ldots, x_3) \equiv (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_3 \wedge \neg x_2) \vee x_3$
- Can encode any Boolean formula into Normal Form

The resolution principle and classical simplification rules

John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Communications of the ACM, 5 :23-41, 1965.

$$\textit{resolution:} \quad \frac{x_1 \vee x_2 \vee x_3 \qquad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4}$$

$$\textit{merging:} \quad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

$$\textit{subsumption:} \quad \frac{\alpha \vee \beta \qquad \alpha}{\alpha}$$

The resolution principle and classical simplification rules

John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Communications of the ACM, 5 :23-41, 1965.

$$\textit{resolution:} \quad \frac{x_1 \vee x_2 \vee x_3 \qquad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4}$$

$$\textit{merging:} \quad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

$$\textit{subsumption:} \quad \frac{\alpha \vee \beta \qquad \alpha}{\alpha}$$

What happens if we apply resolution between $\neg x_1 \vee x_2 \vee x_3$ and $x_1 \vee \neg x_2 \vee x_4$?

The resolution principle and classical simplification rules

John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Communications of the ACM, 5 :23-41, 1965.

$$\textit{resolution:}\quad \frac{x_1 \vee x_2 \vee x_3 \qquad x_1 \vee \neg x_2 \vee x_4}{x_1 \vee x_1 \vee x_3 \vee x_4}$$

$$\textit{merging:}\quad \frac{x_1 \vee x_1 \vee x_3 \vee x_4}{x_1 \vee x_3 \vee x_4}$$

$$\textit{subsumption:}\quad \frac{\alpha \vee \beta \qquad \alpha}{\alpha}$$

What happens if we apply resolution between $\neg x_1 \vee x_2 \vee x_3$ and $x_1 \vee \neg x_2 \vee x_4$?

A tautology : $x_2 \vee \neg x_2 \vee x_3 \vee x_4$.

Applying resolution to decide satisfiability

- Apply resolution between clauses with exactly one opposite literal
- possible outcome :
  - a new clause is derived : removed subsumed clauses
  - the resolvent is subsumed by an existing clause
- until empty clause derived or no new clause derived
- Main issues of the approach :
  - In which order should the resolution steps be performed ?
  - huge memory consumption !

The Davis and Putnam procedure : basic idea

Davis, Martin ; Putnam, Hillary (1960). "A Computing Procedure for Quantification Theory". Journal of the ACM 7 (3) : 201-215.

Resolution used for variable elimination : $(A \vee x) \wedge (B \vee \neg x) \wedge R$ is satisfiable iff $(A \vee B) \wedge R$ is satisfiable.

- Iteratively apply the following steps :
  - Select variable x
  - Apply resolution between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
  - Remove all clauses containing either x or $\neg x$
- Terminate when either the empty clause or the empty formula is derived

Proof system : ordered resolution

Variable elimination – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \models$$

Variable elimination – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

Variable elimination – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

Variable elimination – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

Variable elimination – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$x_3 \models$$

Variable elimination – An Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \models$$

$$x_3 \models$$

$$\top$$

- Formula is SAT

The Davis and Putnam procedure : the refinements

Add specific cases to order variable elimination steps

- Iteratively apply the following steps :
  - Apply the pure literal rule and unit propagation
  - Select variable x
  - Apply resolution between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
  - Remove all clauses containing either x or $\neg x$
- Terminate when either the empty clause or the empty formula is derived

Pure Literals

- A literal is pure if only occurs as a positive literal or as a negative literal in a CNF formula
  - Example :
    $\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
  - $\neg x_1$ and x_3 are pure literals
- Pure literal rule : eliminate first pure literals because no resolvant are produced !
- applying a variable elimination step on a pure literal strictly reduces the number of clauses !

Unit Propagation

- Specific case of resolution : only shorten clauses.

$$\textit{unit resolution:} \quad \frac{x_1 \vee x_2 \vee x_3 \qquad \neg x_2}{x_1 \vee x_3}$$

- Since clauses are shortened, new unit clauses may appear. Empty clauses also !
- Unit propagation : apply unit resolution while new unit clauses are produced.

DP60 : The limits

- The approach runs easily out of memory.
- Even recent attempts using a ROBDD representation [Simon and Chatalic 2000] does not scale well.
- The solution : using backtrack search !

DLL62 : Preliminary definitions

- Propositional variables can be assigned value False or True
  - In some contexts variables may be unassigned
- A clause is satisfied if at least one of its literals is assigned value true

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unsatisfied if all of its literals are assigned value false

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A clause is unit if it contains one single unassigned literal and all other literals are assigned value False

  $(x_1 \vee \neg x_2 \vee \neg x_3)$
- A formula is satisfied if all of its clauses are satisfied
- A formula is unsatisfied if at least one of its clauses is unsatisfied

DLL62 : space efficient DP60

Davis, Martin ; Logemann, George, and Loveland, Donald (1962). "A Machine Program for Theorem Proving". Communications of the ACM 5 (7) : 394-397.

- Standard backtrack search
- DPLL(F) :
  - Apply unit propagation
  - If conflict identified, return UNSAT
  - Apply the pure literal rule
  - If F is satisfied (empty), return SAT
  - Select decision variable x
    - If DPLL($F \wedge x$)=SAT return SAT
    - return DPLL($F \wedge \neg x$)

Proof system : tree resolution

Pure Literals in backtrack search

- Pure literal rule :
  Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)
- Example :
  $\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
- The resulting formula becomes :
  $\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

- if l is a pure literal in Σ, then $\Sigma_l \subset \Sigma$
- Preserve satisfiability, not logical equivalency !

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

Unit Propagation in backtrack search

- Unit clause rule in backtrack search :
  Given a unit clause, its only unassigned literal must be assigned value True for the clause to be satisfied
- Example : for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value False
- Unit propagation
  Iterated application of the unit clause rule

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

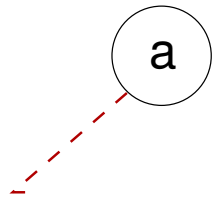
Unit propagation can satisfy clauses but can also falsify clauses (i.e. conflicts)

An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$

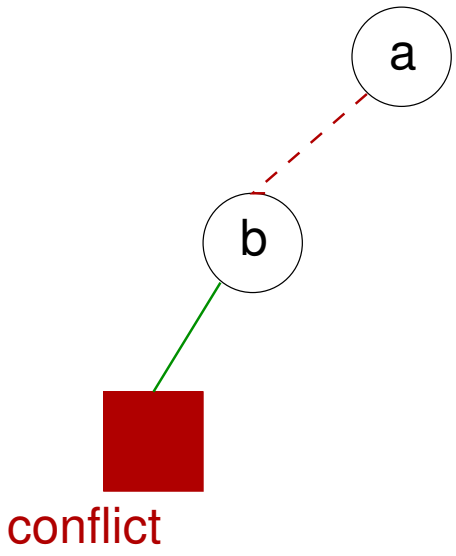
An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



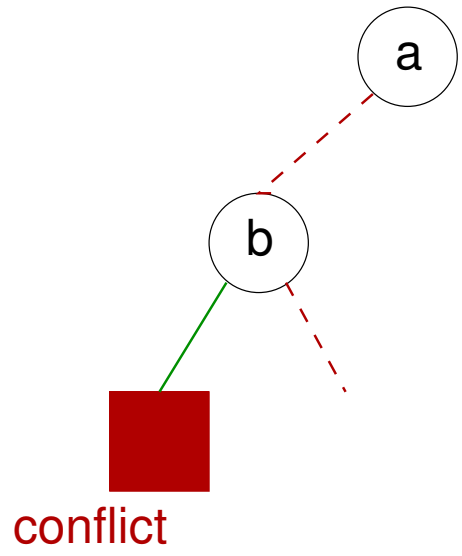
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



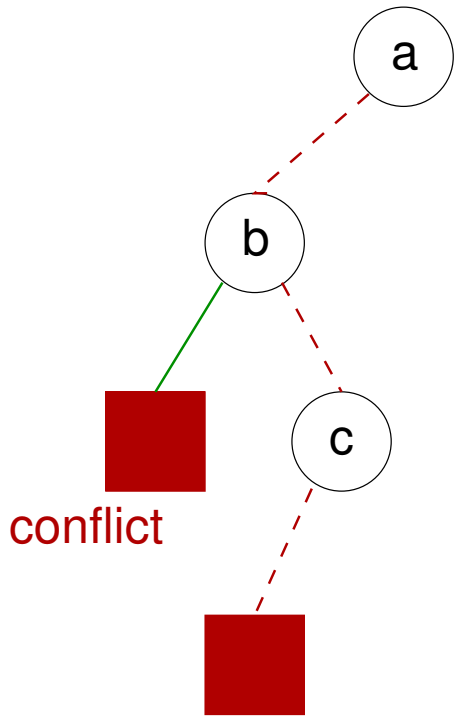
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



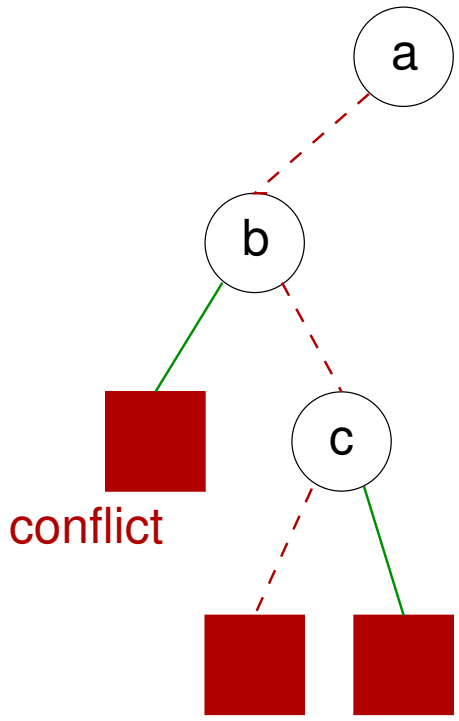
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



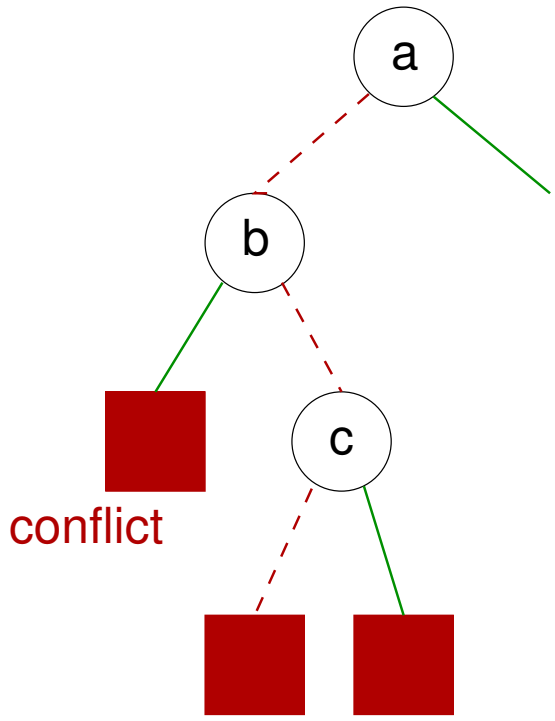
An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



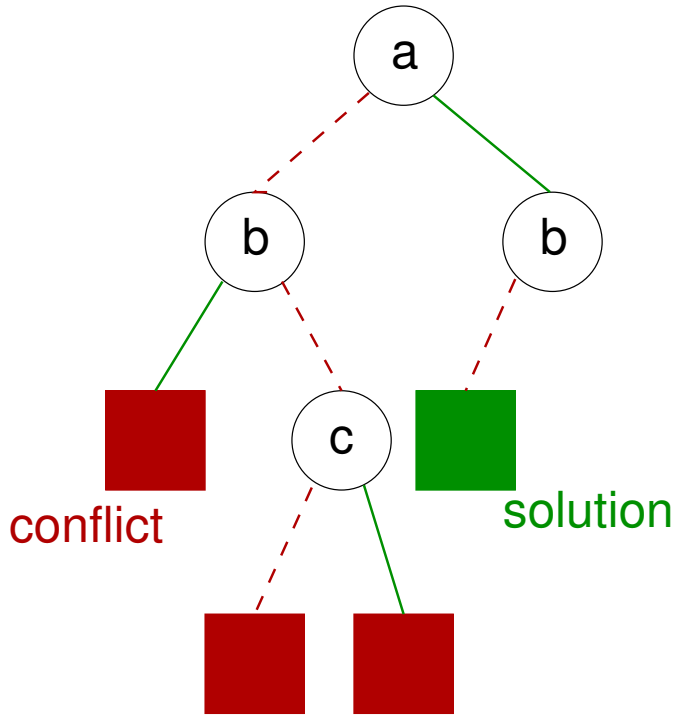
An Example of DPLL

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



An Example of DPLL

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



DP, DLL or DPLL ?

- DPLL = DP + DLL
- Acknowledge the principles in DP60 and their memory efficient implementation in DP62
- DPLL commonly used to denote complete solvers for SAT : no longer true for modern complete SAT solvers.
- The focus of researchers in the 90's was mainly to improve the heuristics to select the variables to branch on on randomly generated formulas.
- Introduction of non chronological backtracking and learning to solve structured/real world formulas

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (\ a\ \vee b) \wedge (\neg b \vee\ \ c\ \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee\ \ f\ ) \ldots$$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (\ a\ \vee b) \wedge (\neg b \vee\ c\ \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee\ f\ ) \ldots$$

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = False$ and $f = False$
  - Assign $a = False$ and imply assignments
  - A conflict is reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = False$ and $f = False$
  - Assign $a = False$ and imply assignments
  - A conflict is reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg a \wedge \neg c \wedge \neg f \Rightarrow \bot$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments
  - A conflict is reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg a \wedge \neg c \wedge \neg f \Rightarrow \bot$
  - $\varphi \Rightarrow a \vee c \vee f$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = \mathit{False}$ and $f = \mathit{False}$
  - Assign $a = \mathit{False}$ and imply assignments
  - A conflict is reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg a \wedge \neg c \wedge \neg f \Rightarrow \bot$
  - $\varphi \Rightarrow a \vee c \vee f$

  - Learn new clause $(a \vee c \vee f)$

Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = False$ and $f = False$
  - Assign $a = False$ and imply assignments
  - A conflict is reached : $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $\varphi \wedge \neg a \wedge \neg c \wedge \neg f \Rightarrow \bot$
  - $\varphi \Rightarrow a \vee c \vee f$

  - Learn new clause $(a \vee c \vee f)$
  - Next time will propagate a : reveals a missing propagation !

Conflict analysis using resolution

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b,d, e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg d \vee \neg e \vee f)$$

Conflict analysis using resolution

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b, d ,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg b \vee \neg d \vee f)$$

Conflict analysis using resolution

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g, b ,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg b \vee c \vee f)$$

Conflict analysis using resolution

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g ,b,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg g \vee c \vee f)$$

Conflict analysis using resolution

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(\neg a \vee c \vee f)$$

Conflict analysis using resolution

Perform resolution steps in reverse order of the assignments.
Propagations deriving from a : g,b,d,e

$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

Learned : $(a \vee c \vee f)$

$$(c \vee f)$$

Implementation of NCB and Learning for SAT

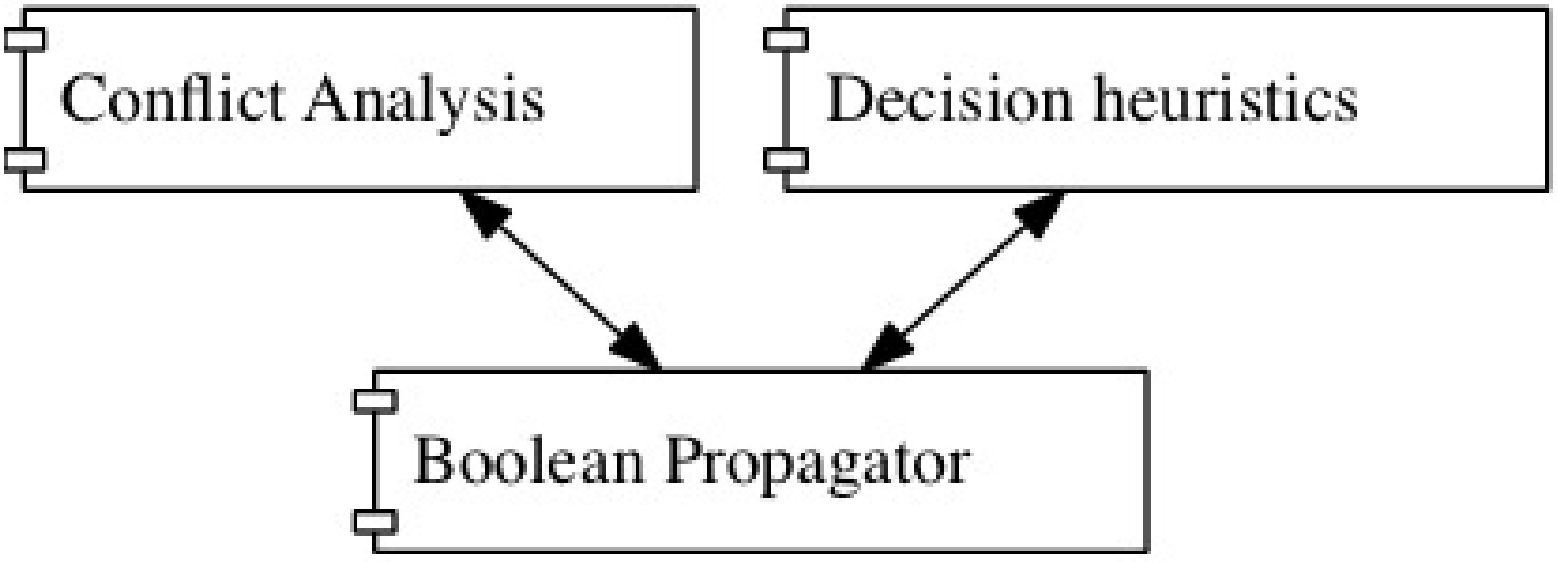
- Two approaches developed independently in two different research communities :
  GRASP/EDA by Marques-Silva and Sakallah (1996)
  - Resolution graph seen as a circuit
  - Conflict analysis thought as detecting faults in a circuit
  - Other sophisticated conflict analysis methods based on truth maintenance systems

  RELSAT/CSP by Bayardo and Schrag (1997)
  - Introduction of CSP based techniques into a SAT solver
  - Conflict Directed Backjumping aka non chronological backtracking [Prosser 93]
  - Size based and relevance based learning schemes
- Main difference : in GRASP's framework, the conflict analysis drives the search, while in RELSAT it is the heuristics (more later).

Agenda

GRASP architecture

João P. Marques Silva, Karem A. Sakallah : GRAPS : A Search Algorithm for Propositional Satisfiability. IEEE Trans. Computers 48(5) : 506-521 (1999)



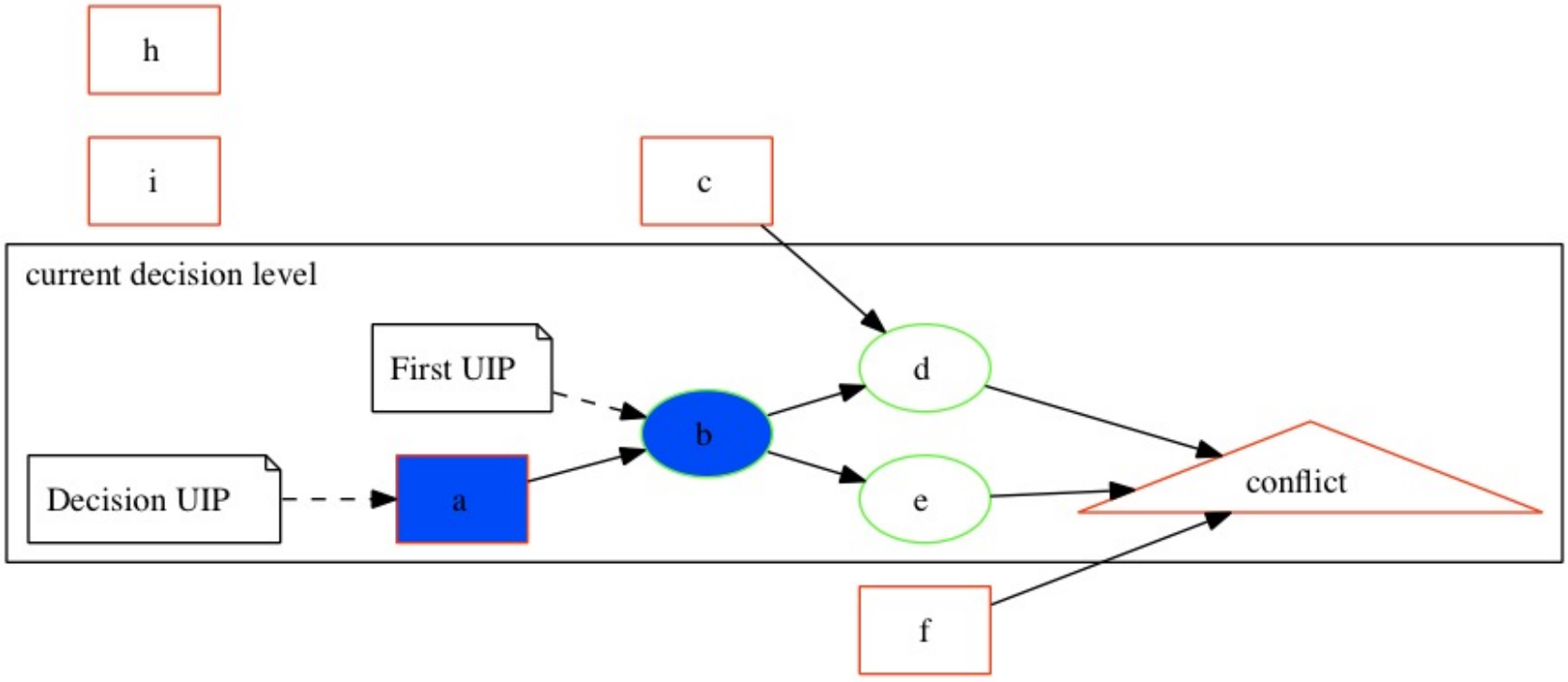
Role of the boolean propagator

- Perform unit propagation on the set of clauses.
- Detect conflicts
- Backtrack according to a specific clause provided by the conflict analyzer

Conflict analyzer

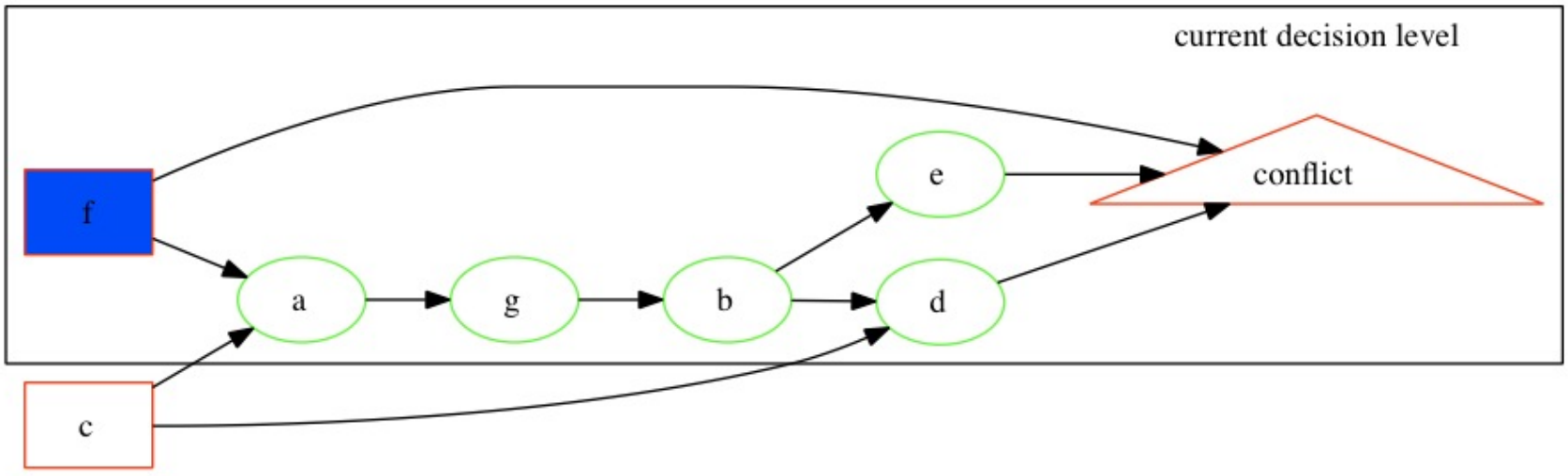
- Must produce a clause that becomes a unit clause after backtracking (asserting clause)
- Introduction of the notion of Unique Implication Point (UIP), as a reference to Unique Sensitization Points in ATPG.
  - Find a literal that need to be propagated before reaching a conflict
  - Based on the notion of decision level, i.e. the number of assumptions made so far.
  - Syntactical : apply resolution until only one literal from current decision level appears in the clause.
  - Decision variables are always UIP : at least one UIP exists for each conflict !
- Backtracking level computed as the lowest decision level of the literals of the clause

Conflict graph for assumption a=False



$$\varphi = (\ a\ \vee b) \wedge (\neg b \vee\ \ c\ \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee\ \ f\ ) \ldots$$

Conflict graph after learning $a \vee c \vee f$ and backjumping



$$\begin{aligned}\varphi \quad = \quad & (a \vee b) \wedge (\neg b \vee \ c \ \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee \ f \ ) \wedge \\ & (a \vee \ c \ \vee \ f \ ) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

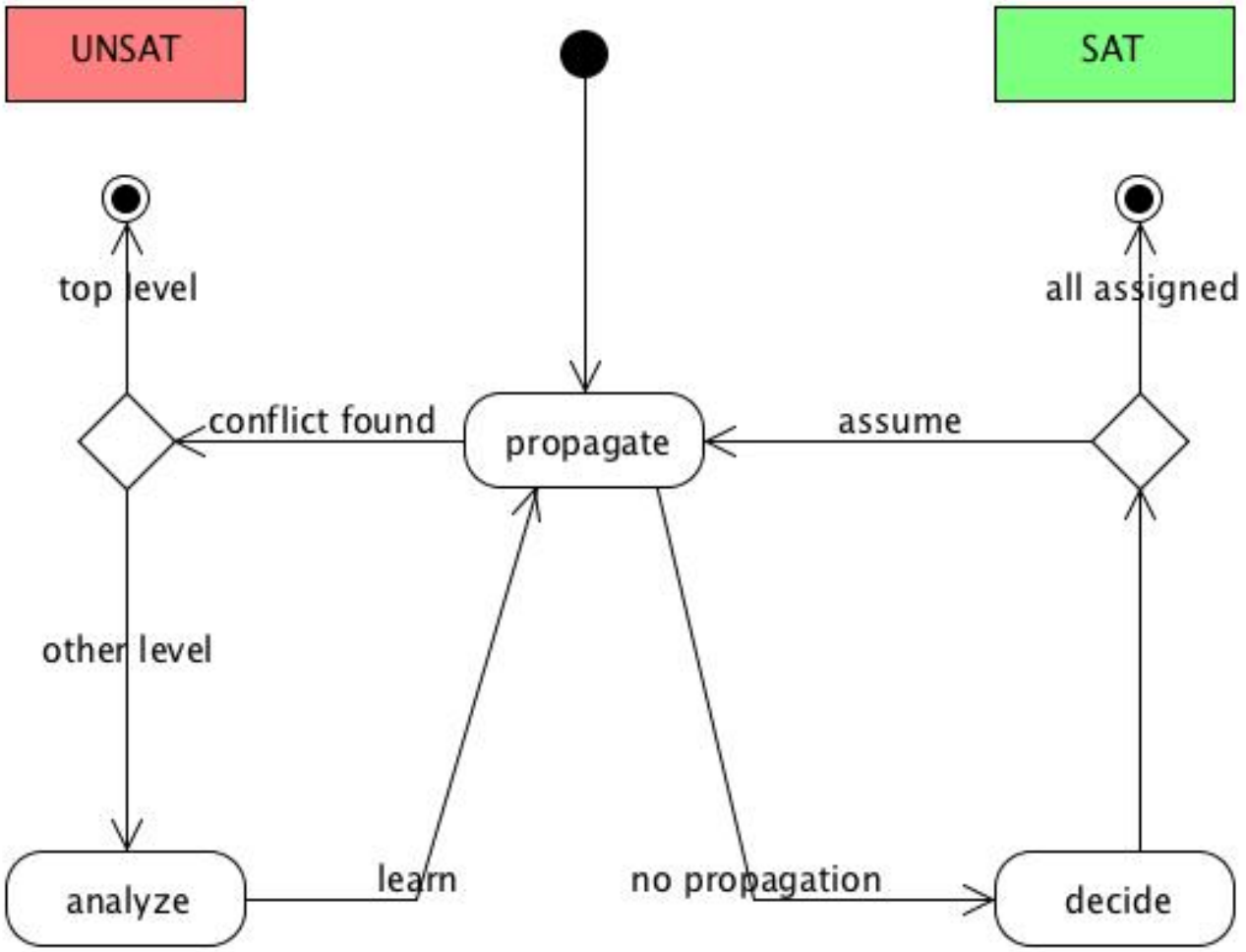
Some remarks about UIPs

- There are many possibilities to derive a clause using UIP
- RELSAT can be seen as applying Decision UIP
- Decision UIP always flip the decision variable truth value : the search is thus driven by the heuristics.
- Using other UIP scheme, the value of any of the literal propagated at the current decision level may be flipped. The search is thus driven by the conflict analysis.
- Generic name for GRASP approach : Conflict Driven Clause Learning (CDCL) solver [Ryan 2004].

Decision heuristics

- Pick an unassigned variable
- Many sophisticated decision heuristics available in the literature for random formulas (MOMS, JW, etc).
- GRASP uses dynamic largest individual sum (DLIS) : select the literal with the maximum occurrences in unresolved clauses.
- Sophisticated heuristics require an exact representation of the state of the CNF after unit propagation !

Putting everything together : the CDCL approach



From GRASP to CHAFF

- Some key insights in the design of SAT solvers were discovered when trying to solve real problems by translation into SAT.
- Huge interest on SAT after the introduction of Bounded Model Checking [Biere et al 99] from the EDA community.
- The design of SAT solver becomes more pragmatic

Application 1 : Planning as satisfiability

Henry A. Kautz, Bart Selman : Planning as Satisfiability. ECAI 1992 : 359-363

- Input : a set of actions, an initial state and a goal state
- Output : a sequence of actions to reach the goal state from the initial state
- One of the first application of SAT in Artificial Intelligence
- A key application for the adoption of SAT in EDA later on
- The instances are supposed to be SAT
- Those instances are too big for complete solvers based on DPLL

1992 - Planning As Satisfiability

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

où :

- 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 at most k.

Greedy SAT (Local Search Scheme for SAT)

```
function GSAT(CNF c, int maxtries, int maxflips) {
    // DIVERSIFICATION STEP
    for (int i =0; i< maxtries ; i++) {
        m = randomAssignment();
        // INTENSIFICATION STEP
        for (int j=0; j<maxflips; j++) {
            if (m satisfies c)
                return SAT;
            flip(m);
        }
    }
    return UNKNOWN;
}
```

Lessons learned from GSAT

- The decision procedure is very simple to implement and very fast !
- Efficiency depends on which literal to flip, and the values of the parameters.
- Problem with local minima : use of Random Walks !
- Main drawback : incomplete, cannot answer UNSAT !
- Lesson 1 : An agile (fast) SAT solver sometimes better than a clever one !

Application 2 : Quasigroup (Latin Square) open problems

- S a set and * a binary operator. $|S|$ is the order of the group.
- a*b=c has a unique solution when fixing any pair of variables.
- equivalent to fill in a $|S| \times |S|$ square with elements of S unique in each row and column.
- Looking for the existence of QG of a given order with additional constraints, e.g. :

$$\text{QG1}\quad x*y=u, z*w=u, v*y=x, v*w=z \Rightarrow x=z, y=w$$

$$\text{QG2}\quad x*y=u, z*w=u, y*v=x, w*v=z \Rightarrow x=z, y=w$$

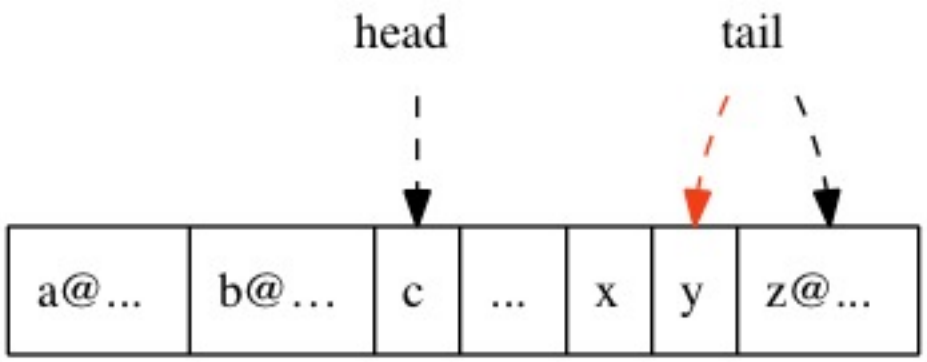
- First open QG problems solved by MGTP (Fujita, Slaney, Benett 93)
- QG2(12) solved by DDPP in 1993.
- QG1(12), QG2(14),QG2(15) solved by SATO in 1996.

SATO head/tail lazy data structure

Zhang, H., Stickel, M. : Implementing Davis-Putnam's method . It appeared as a Technical Report, The University of Iowa, 1994

- CNF resulting for QG problems have a huge amount of clauses : 10K to 150K !
- Encoding of real problems into SAT can lead to very large clauses
- Truth value propagation cost in eager data structure depends on the number of propagation to perform, thus on the size of the clauses
- How to limit the cost of numerous and long clauses during propagation ?
- Answer : use a lazy data structure to detect only unit propagation and falsified literals.

The Head/Tail data structure

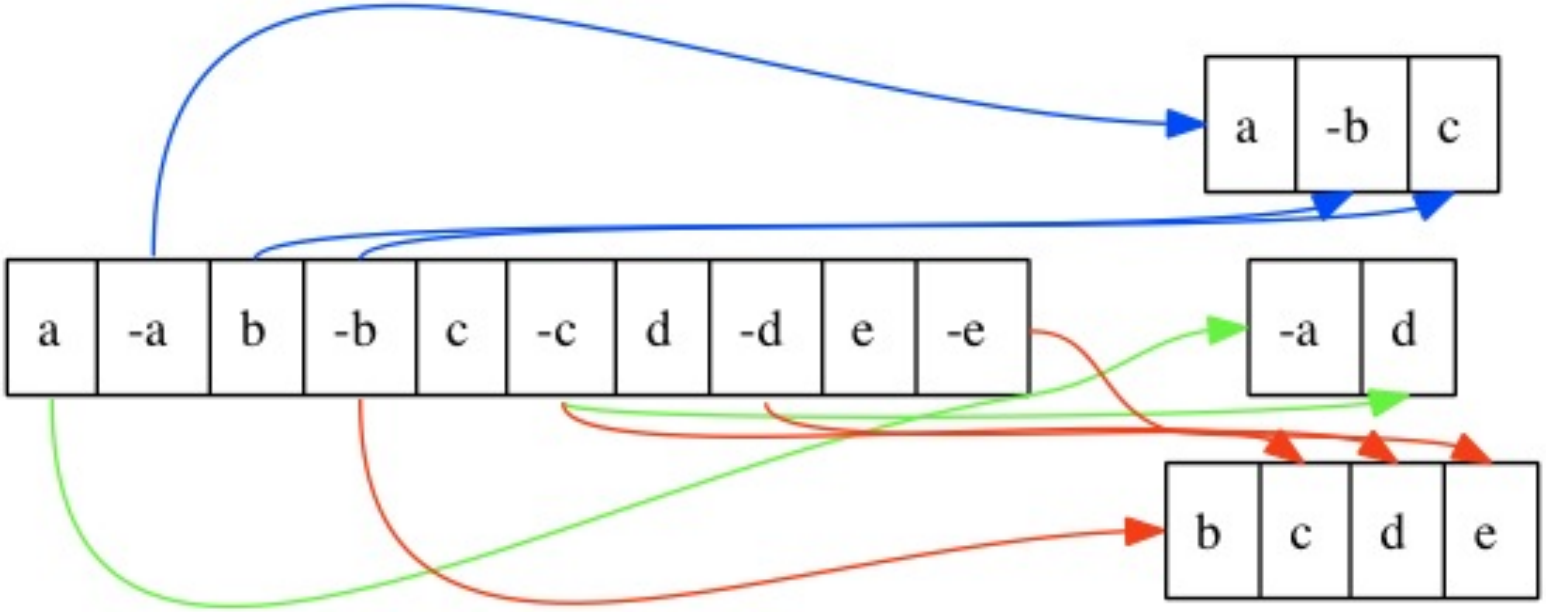


initially put a head (resp. tail) pointer to the first (resp. last) element of the clause

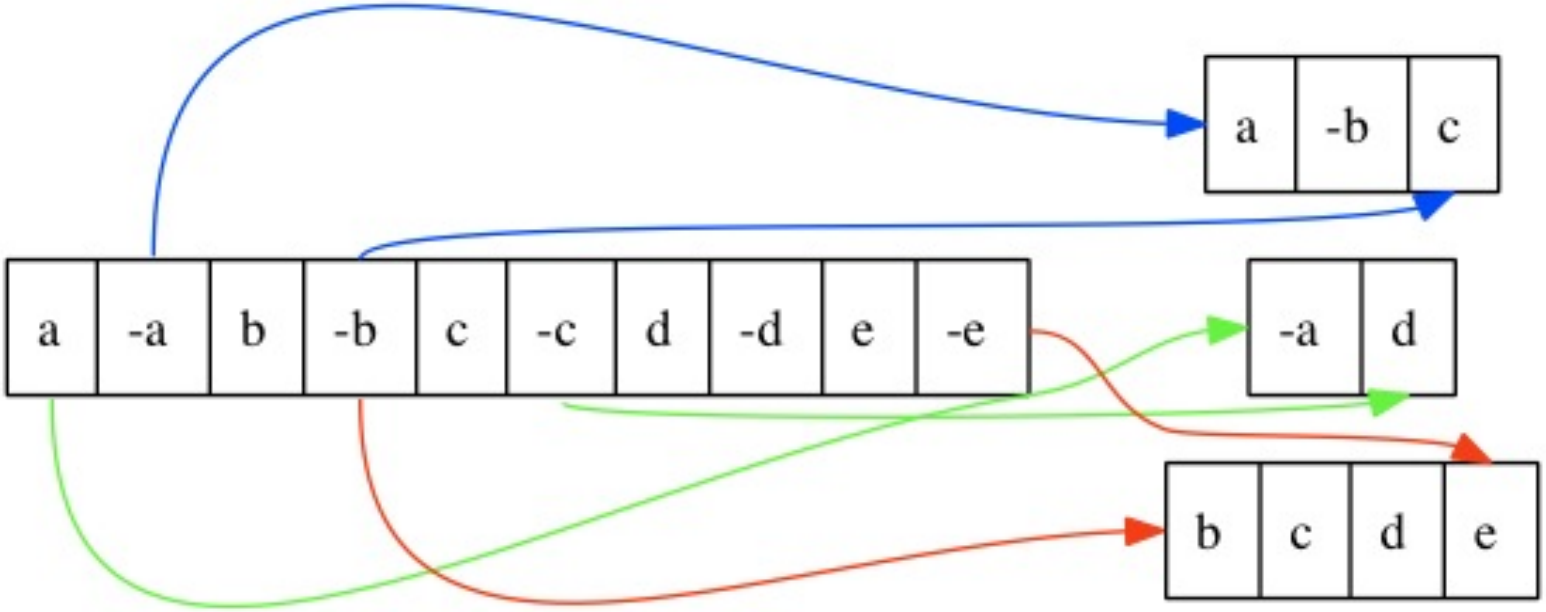
during propagation move heads or tails pointing to the negation of the propagated literal. Easy identification of unit and falsified clauses.

during backtracking move back the pointers to their previous location

Unit propagation with Adjacency lists



Unit propagation with Head /Tail



Pro and Cons of the H/T data structure

advantage reduces the cost of unit propagation

drawback the solver has no longer a complete picture of the reduced CNF !

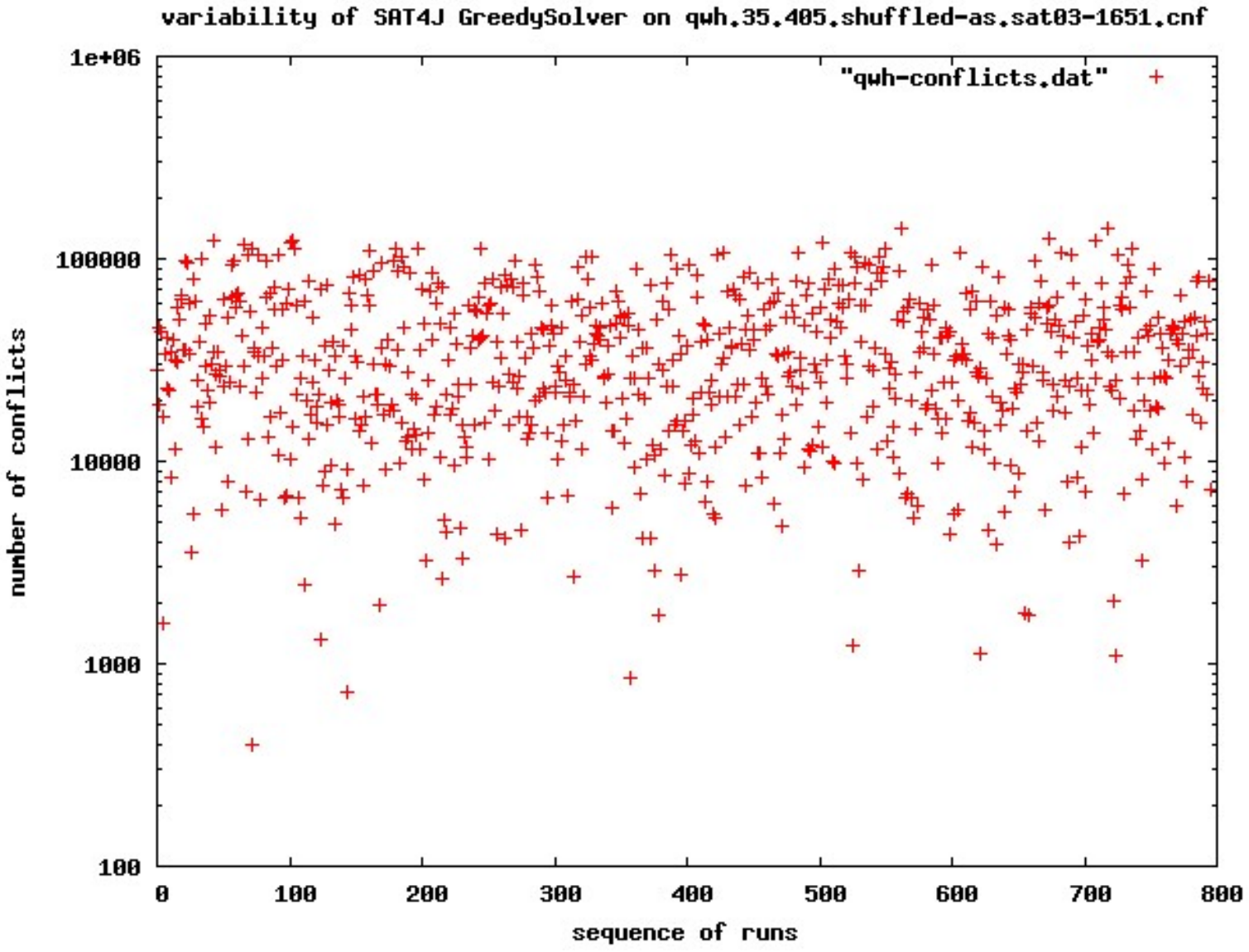
Lesson 2 : data structure matters !

High variability of SAT solvers runtime !

Heavy-tailed Distributions in Combinatorial Search. Carla Gomes, Bart Selman, and Nuno Crato. In Principles and Practices of Constraint Programming, (CP-97) Lecture Notes in Computer Science 1330, pp 121-135, Linz, Austria., 1997. Springer-Verlag

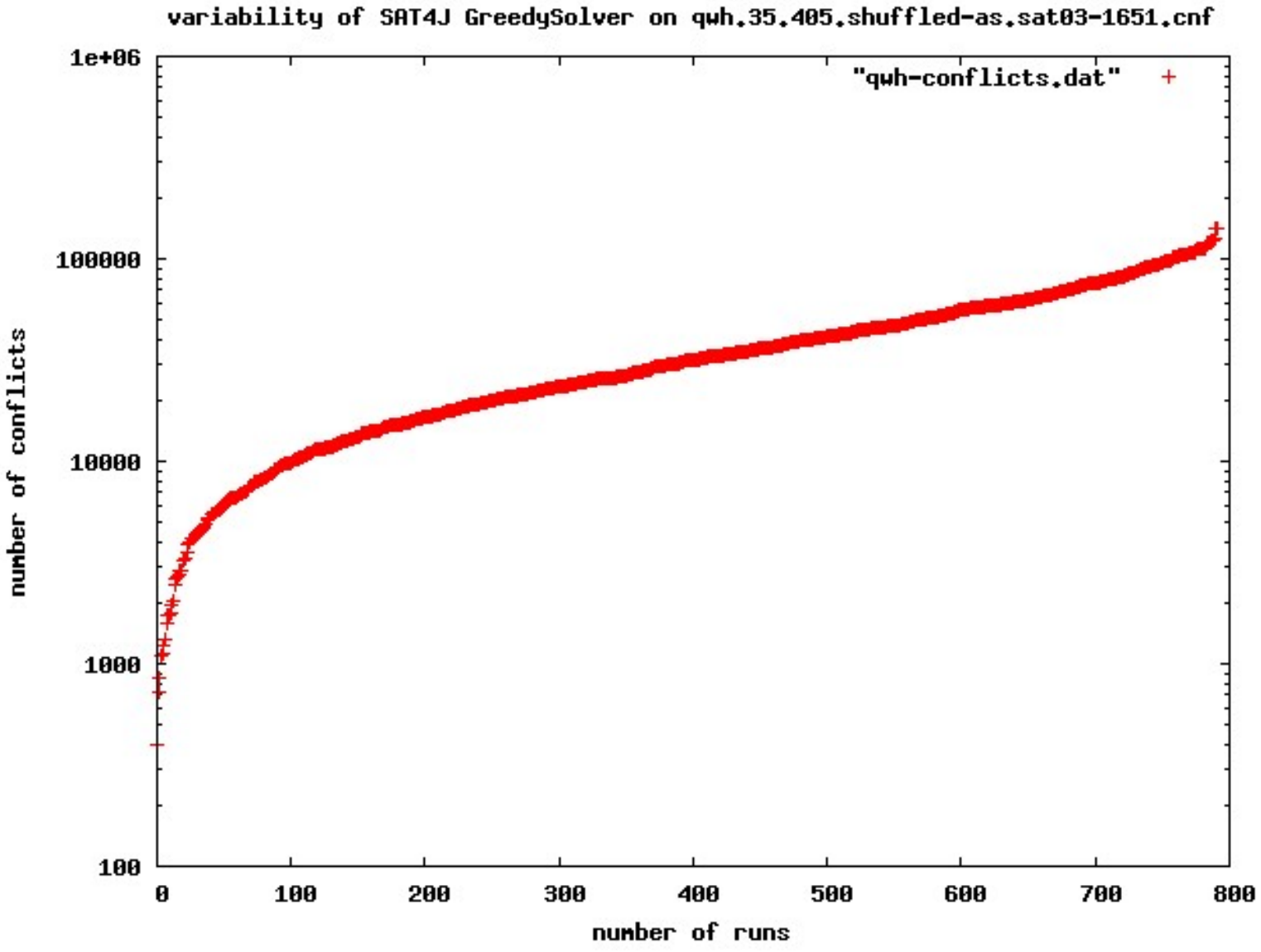
- SAT solvers exhibits on some problems a high runtime variability
- Decision heuristics need to break ties, often randomly
- The solver are sensible to syntactical input changes :
  - Shuffled industrial benchmarks harder than original ones for most solvers
  - The "lisa syndrome" during the SAT 2003 competition
- An explanation : Heavy tailed distribution

Example of variability : SAT4J GreedySolver on QGH



UNIVERSITÉ D'ARTOIS

Example of variability : SAT4J GreedySolver on QGH



Heavy Tailed distribution

- Introduced by the economist Pareto in the context of income distribution
- Widely used in many areas : stock market analysis, weather forecast, earthquake prediction, time delays on the WWW.
- Those distributions have infinite mean and infinite variance
- Some SAT solvers exhibit an Heavy Tailed distribution on Quasigroup Completion with Holes problems.
- What does it mean in practice ?
  - In rare occasion, the solver can get trapped on a very long run
  - while most of the time the run could be short
- the solution : restarts !

Restarting in SAT solvers

- Stop the search after a given number of conflicts/decisions/propagation is achieved (cutoff).
- Start again the search [with increased cutoff to be complete]
- Requires some variability in the solver behavior between two runs
- Problem : how to choose the cutoff value ?
- In theory, an optimal strategy exists [Luby 93].
- Lesson 3 : introduce restarts to make the solver more robusts

The killer app : Bounded Model Checking

A. Biere, A. Cimatti, E. Clarke, M. Fujita, Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In Proc. ACM Design Automation Conf. (DAC'99), ACM 1999.

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

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 within k steps.

SAT vs BDD model checking

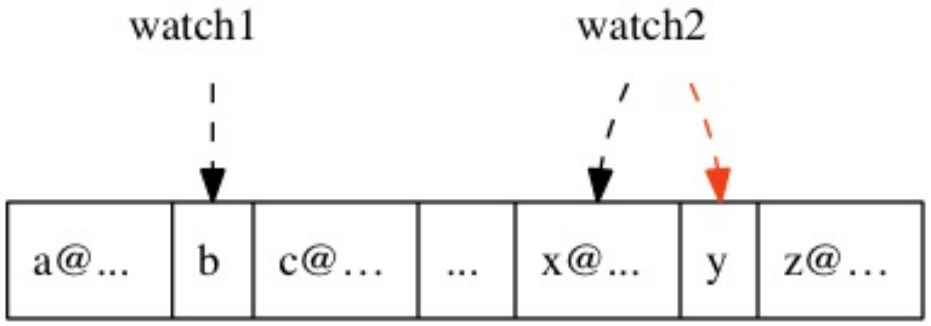
- Some model checking problems out of reach of BDD checkers can be solved thanks to a reduction to SAT
- The behavior of SAT solvers is less dependent of the form of the input than BDD solvers
- But the SAT solvers are not powerful enough yet for industrial use...

The breakthrough : Chaff

Chaff : Engineering an Efficient SAT Solver by M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, 39th Design Automation Conference (DAC 2001), Las Vegas, June 2001.

- 2 order of magnitude speedup on unsat instances compared to existing approaches on BMC (Velev) benchmarks.
- Immediate speedup for SAT based tools : BlackBox "Supercharged with Chaff"
- Based on careful analysis of GRASP internals
- 3 key features :
  - New lazy data structure : Watched literals
  - New adaptative heuristic : Variable State Independent Decaying Sum
  - New conflict analysis approach : First UIP
- Taking into account randomization

The watched literals data structure



initially watch two arbitrary literals in the clause

during propagation move watchers pointers in clauses containing the negation of the propagated literal.

during backtracking do nothing !

advantage cost free data structure when backtracking

issue pointers can move in both directions.

Variable State Independent Decaying Sum

- compatible with Lazy Data Structures
- each literal has a score
- score based on the number of occurrences of the literals in the formula
- score updated whenever a new clause is learned
- pick the unassigned literal with the highest score, tie broken randomly
- regularly (every 256 conflicts), divided the scores by a constant (2)

New Learning Scheme : First UIP

Efficient Conflict Driven Learning in a Boolean Satisfiability Solver by L. Zhang, C. Madigan, M. Moskewicz, S. Malik, Proceedings of ICCAD 2001, San Jose, CA, Nov. 2001

- The idea is to quickly compute a reason for the conflict
- Stop the resolution process as soon as an UIP is detected
- First UIP Shown to be optimal in terms of backtrack level compared to the other possible UIPs [Audemard et al 08].

Chaff : a highly coupled set of features

- Learning does not degrade solver performance because the use of the watched literals
- The VSIDS heuristics does not need a complete picture of the reduced formula, i.e. is compatible with the lazy data structure.
- VSIDS take advantage of the conflict analysis to spot important literals.
- VSIDS provides different orders of literals at each restart
- VSIDS adapt itself to the instance !

The reason of the success ?

- Better engineering (level 2 cache awareness) ?

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?
- Instance-based auto adaptation ?

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?
- Instance-based auto adaptation ?
- ...

The reason of the success ?

- Better engineering (level 2 cache awareness) ?
- Better tradeoff between speed and intelligence ?
- Instance-based auto adaptation ?
- ...

All those reasons are correct. There is a more fundamental reason too ...

CDCL has a better proof system than DPLL !

Proof theory strikes back !

- ... thanks to many others before ...
- Bonet, M. L., & Galesi, N. (2001). Optimality of size-width tradeoffs for resolution. Computational Complexity, 10(4), 261-276.
- Beame, P., Kautz, H., and Sabharwal, A. Towards understanding and harnessing the potential of clause learning. JAIR 22 (2004), 319-351.
- Van Gelder, A. Pool resolution and its relation to regular resolution and dpll with clause learning. In LPAR'05 (2005), pp. 580-594.
- Hertel, P., Bacchus, F., Pitassi, T., and Van Gelder, A. Clause learning can effectively p-simulate general propositional resolution. In Proc. of AAAI-08 (2008), pp. 283-290.
- Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers as resolution engines. Artif. Intell., 175(2) :512–525, 2011
- Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. J. Artif. Intell. Res. (JAIR), 40 :353–373, 2011

CDCL has a better proof system than DPLL !

Proof theory strikes back !

Definition

p-simulation Proof system S p-simulates proof system T , if, for every unsatisfiable formula φ, the shortest refutation proof of φ in S is at most polynomially longer than the shortest refutation proof of φ in T.

CDCL has a better proof system than DPLL !

Proof theory strikes back !

Definition

p-simulation Proof system S p-simulates proof system T , if, for every unsatisfiable formula φ, the shortest refutation proof of φ in S is at most polynomially longer than the shortest refutation proof of φ in T.

Theorem 1 [Pipatsrisawat, Darwiche 09]. CLR with any asserting learning scheme p-simulates general resolution.

Since Chaff ...

- The international SAT competition/SAT race is organized every year
- A huge number of CDCL solvers have been developed, and made available to the community
- SAT has integrated the engineer toolbox to solve combinatorial problems
- Many papers published on the design of efficient SAT solvers

Since Chaff ...

- The international SAT competition/SAT race is organized every year
- A huge number of CDCL solvers have been developed, and made available to the community
- SAT has integrated the engineer toolbox to solve combinatorial problems
- Many papers published on the design of efficient SAT solvers
- ... but a big part of the knowledge still lies in source code !

Minisat : the minimalist CDCL SAT solver

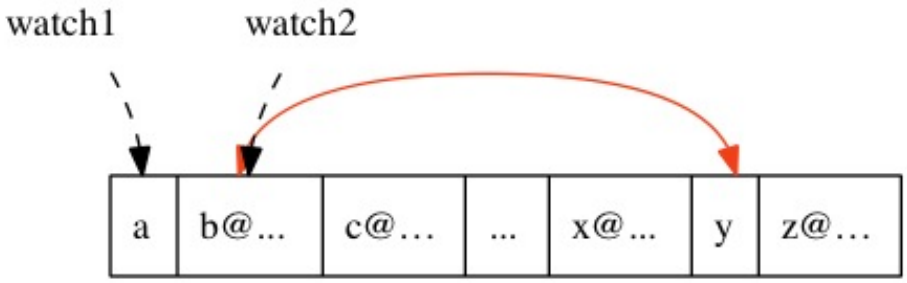
Niklas Eén, Niklas Sörensson : An Extensible SAT-solver. SAT 2003 : 502-518

- very simple implementation of a Chaff-like solver
- resulting from the lessons learned from designing Satzoo (SAT 2003 Winner) and SATnick
- with implementation improvements (Watched Literals, Heuristics, Priority Queue (2005), etc.)
- ready for generic constraints (cardinality, linear pseudo boolean, etc.).
- published description of the design

Reduced the entry level required to experiment with CDCL SAT solvers

The watched literals data structure improved

[mChaff,vanGelder02,Minisat]



initially watch the two first literals in the clause

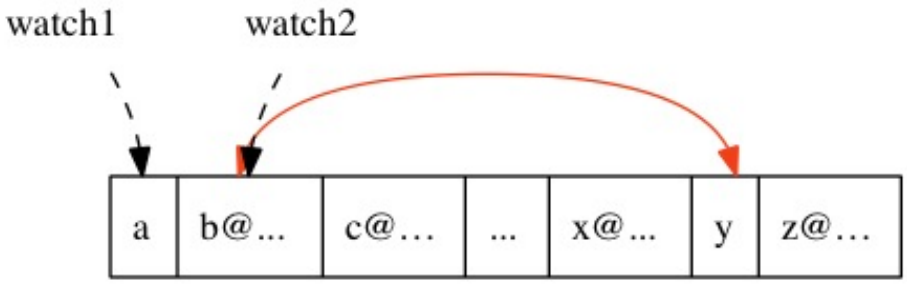
during propagation move falsified literal in second position. Exchange it with an unassigned literal is any. Easy identification of unit and falsified clauses.

during backtracking do nothing !

advantage cost free data structure when backtracking

The watched literals data structure improved

[mChaff,vanGelder02,Minisat]



initially watch the two first literals in the clause

during propagation move falsified literal in second position. Exchange it with an unassigned literal is any. Easy identification of unit and falsified clauses.

during backtracking do nothing !

advantage cost free data structure when backtracking

Moving literals instead of pointers in HT data structure also provides cost free backtracking !

Berkmin style heuristic

Evguenii I. Goldberg, Yakov Novikov : BerkMin : A Fast and Robust Sat-Solver. DATE 2002 : 142-149

Ideas :

- force the heuristic to satisfy recently learned clauses to be more reactive than VSIDS
- sophisticated phase selection strategy based on an estimate of the unit propagations to result from the selection (a la SATZ [Li Anbulagan 97]).
- take into account literals met during the conflict analysis

Berkmin performed quite well during SAT 2002 (despite a stupid bug) and it's successor Forklift won in 2003.

First UIP conflict analysis based on Resolution !

Perform resolution steps in reverse order of the assignments.
Suppose
$decisionLevel(f) = x$ and
$decisionLevel(c) = y$ with $x > y$.
Propagations deriving from a : g,b,d, e

Reasons of the propagations :

$$= \quad (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e)$$

Conflicting clause (resolvant) ;

$$(\neg d@x \vee \neg e@x \vee f@x)$$

First UIP conflict analysis based on Resolution !

Perform resolution steps in reverse order of the assignments.
Suppose
$decisionLevel(f) = x$ and
$decisionLevel(c) = y$ with $x > y$.
Propagations deriving from a : g,b, d ,e

Reasons of the propagations :

$$= \quad (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg b \vee c \vee d)$$

Conflicting clause (resolvant) ;

$$(\neg b@x \vee \neg d@x \vee f@x)$$

First UIP conflict analysis based on Resolution !

Perform resolution steps in reverse order of the assignments.
Suppose
$decisionLevel(f) = x$ and
$decisionLevel(c) = y$ with $x > y$.
Propagations deriving from a : g, b ,d,e

Reasons of the propagations :

$$= \quad (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b)$$

Conflicting clause (resolvant) ;

$$(\neg b@x \vee c@y \vee f@x)$$

First UIP conflict analysis based on Resolution !

Perform resolution steps in reverse order of the assignments.
Suppose
$decisionLevel(f) = x$ and
$decisionLevel(c) = y$ with $x > y$.
Propagations deriving from a : g ,b,d,e

Reasons of the propagations :

$$= \quad (a \vee c \vee f) \wedge (\neg a \vee g)$$

Conflicting clause (resolvant) ;

$$(\neg g@x \vee c@y \vee f@x)$$

First UIP conflict analysis based on Resolution !

Perform resolution steps in reverse order of the assignments.
Suppose
$decisionLevel(f) = x$ and
$decisionLevel(c) = y$ with $x > y$.
Propagations deriving from a : g,b,d,e

Reasons of the propagations :

$$= \quad (a \vee c \vee f)$$

Conflicting clause (resolvant) ;

$$(\neg a@x \vee c@y \vee f@x)$$

First UIP conflict analysis based on Resolution !

Perform resolution steps in reverse order of the assignments.
Suppose
$decisionLevel(f) = x$ and
$decisionLevel(c) = y$ with $x > y$.
Propagations deriving from a : g,b,d,e

Reasons of the propagations :
Conflicting clause (resolvant) ;

$$(c@y \vee f@x)$$

First UIP ! only one literal at decision level x left.

Conflict Clause minimization

Minisat 1.13, N. Sörensson, A. Biere. Minimizing Learned Clauses. In Proc. 12th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'09), Lecture Notes in Computer Science (LNCS) vol. 5584, pages 237-243, Springer 2009.

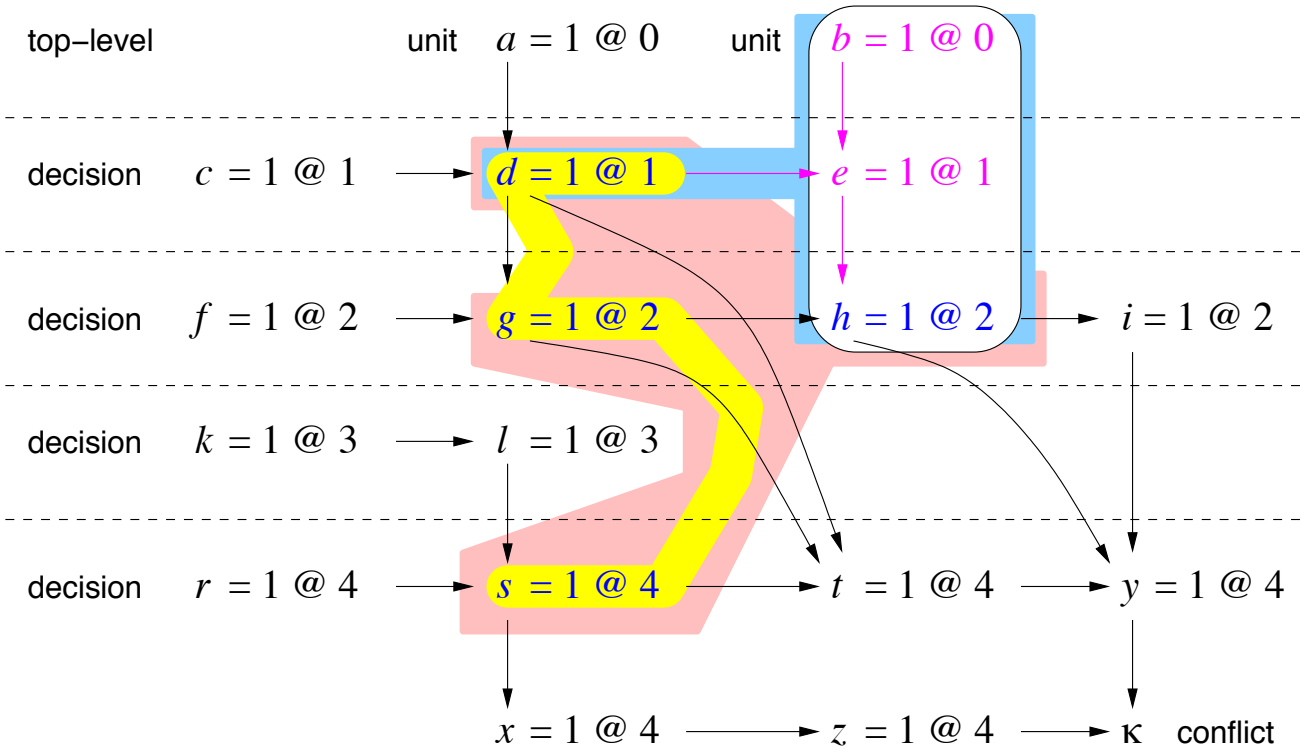
- Clauses generated using the 1st UIP scheme can be simplified
- Using simple direct self subsumption (direct dependencies among the clause's literals outside current decision level) :

self subsumption:

$$\frac{x_1@1 \vee x_2@1 \vee x_3@2 \qquad x_1@1 \vee \neg x_2@1}{x_1@1 \vee x_3@2}$$

- Using a chain of resolution steps

Recursively Minimizing Learned Clause



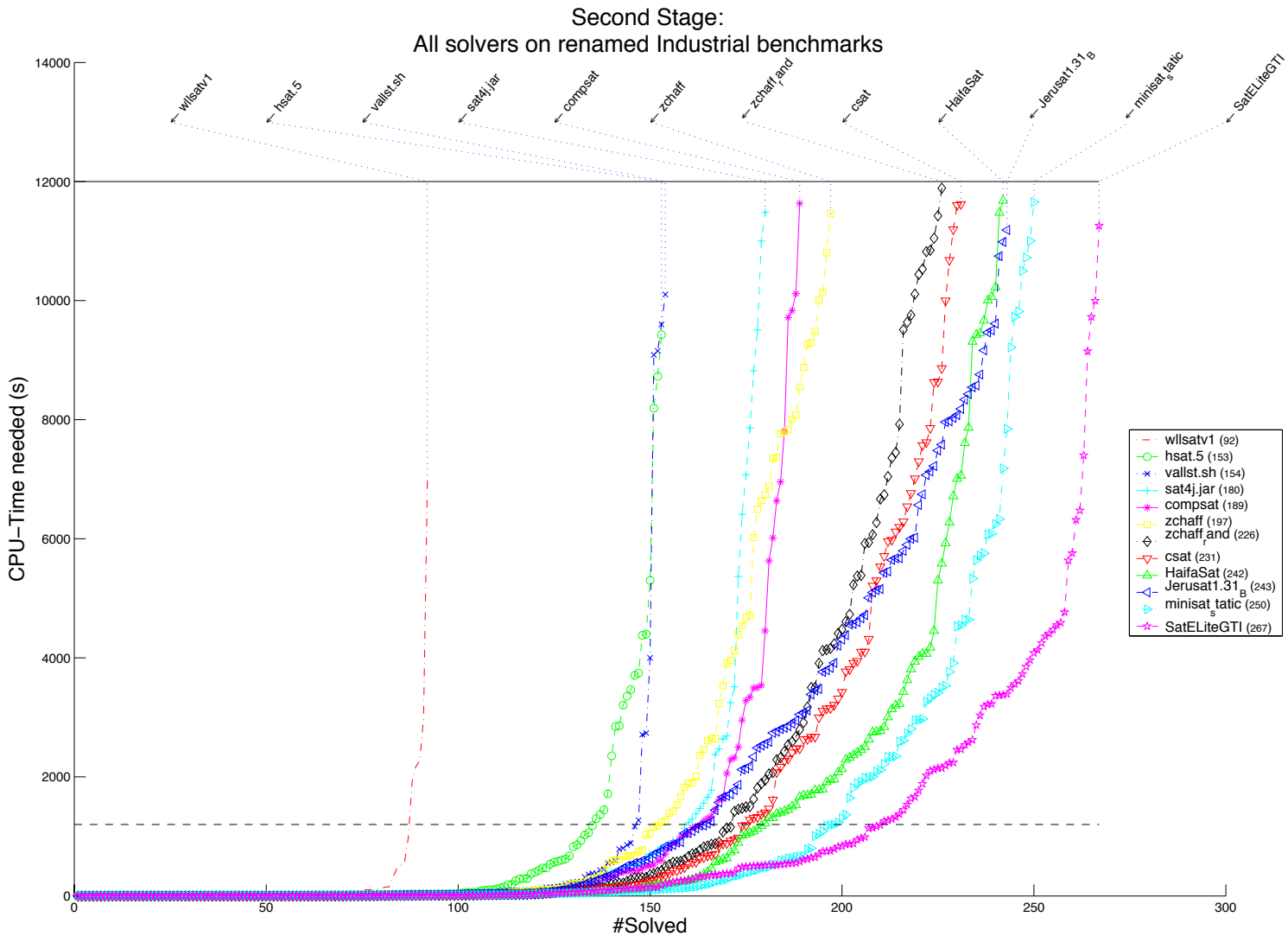
$$\dfrac{(b) \qquad \dfrac{(\bar{d} \vee \bar{b} \vee e) \qquad \dfrac{(\bar{e} \vee \bar{g} \vee h) \quad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h})}{(\bar{e} \vee \bar{d} \vee \bar{g} \vee \bar{s})}}{(\bar{b} \vee \bar{d} \vee \bar{g} \vee \bar{s})}}{(\bar{d} \vee \bar{g} \vee \bar{s})}$$

Preprocessing

Niklas Eén, Armin Biere : Effective Preprocessing in SAT Through Variable and Clause Elimination. SAT 2005 : 61-75

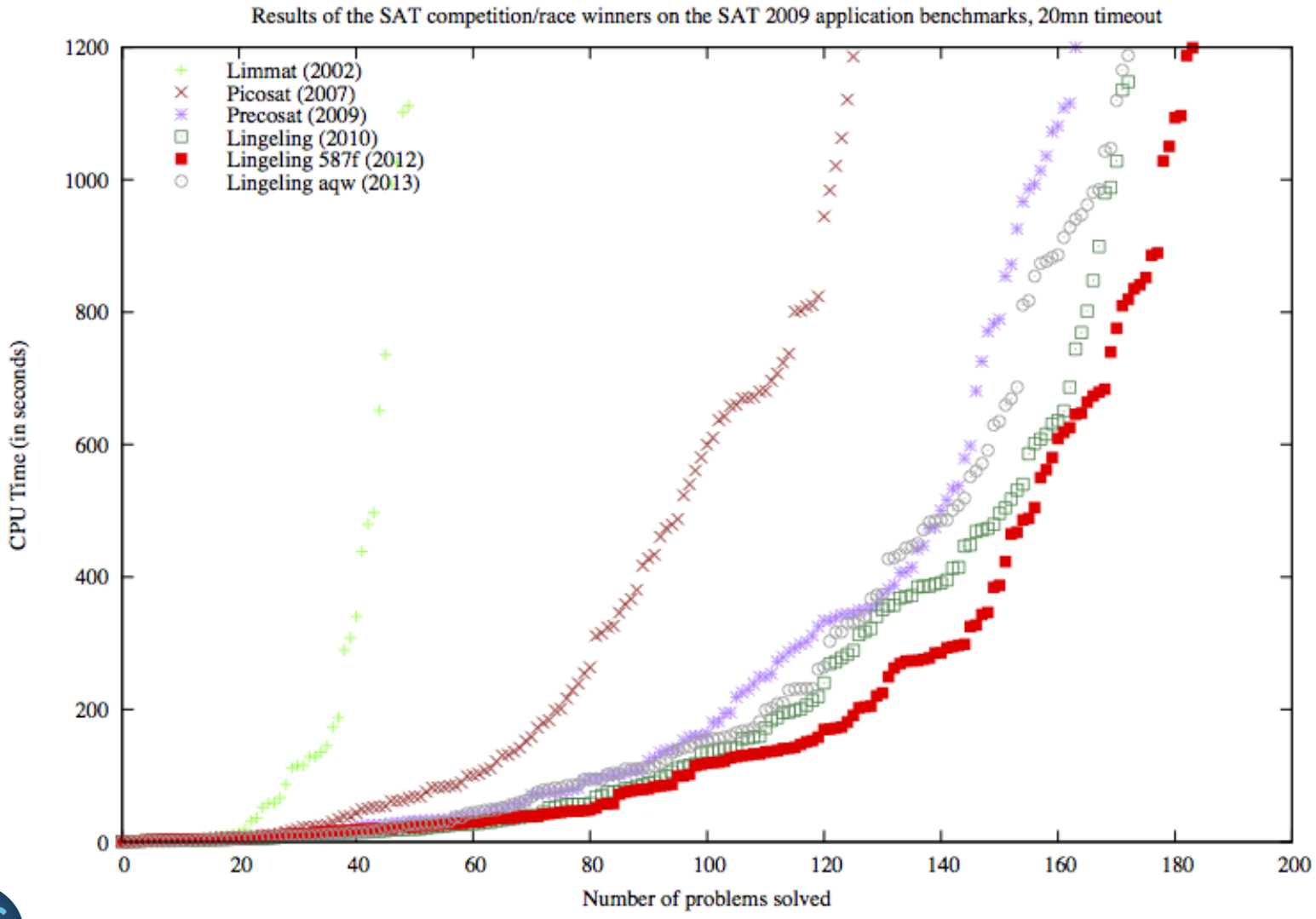
- Variable elimination
  - as in DP60 if the number of clauses does not increase
  - by substitution if a definition such as $x \leftrightarrow y_1 \vee ... \vee y_n$ or $x \leftrightarrow y_1 \wedge ... \wedge y_n$ is detected.
- Clause subsumption
  - self subsumption
  - classical subsumption
- SatELite : de-facto standard pre-processor since 2005
- Included in Minisat 2 (better integration with the SAT solver)
- Still used by SAT solver designers that do not want to implement their own
- But also clause and variable addition
- preprocessing on the fly : in processing in lingeling/cryptominisat

Clause Minimization and Preprocessing @SAT COMP. 2005



UNIVERSITÉ D'ARTOIS

Efficiency of solvers incorporating inprocessing (Armin's solvers)



UNIVERSITÉ D'ARTOIS

Phase Saving

Knot Pipatsrisawat, Adnan Darwiche : A Lightweight Component Caching Scheme for Satisfiability Solvers. SAT 2007 : 294-299

- To concentrate on a single component, keep track of the phase of assigned literals when restarting.
- Always branch first on the recorded phase when taking a decision.
- A small change in the code of the solver, a big improvement in practice (at least for pure SAT :)) !
- Note : RSAT forgets the phase after a while ...

Rapid Restarts

Jinbo Huang : The Effect of Restarts on the Efficiency of Clause Learning. IJCAI 2007 : 2318-2323

- Restarts bounds usually grow slowly until being large enough to ensure completeness
- Different restart strategies make huge differences depending of the benchmarks
- Rapid Restarts strategies usually a good companion for Phase Saving

Armin Inner/Outer rapid restarts

Armin Biere : PicoSAT Essentials. JSAT 4(2-4) : 75-97 (2008)

```
int inner = 100, outer = 100;
int restarts = 0, conflicts = 0;

for (;;) {
  ... // run SAT core loop for inner conflicts

  restarts++; conflicts += inner;
  if (inner >= outer) {
    outer *= 1.1; inner = 100;
  else
    inner *= 1.1;
}
```

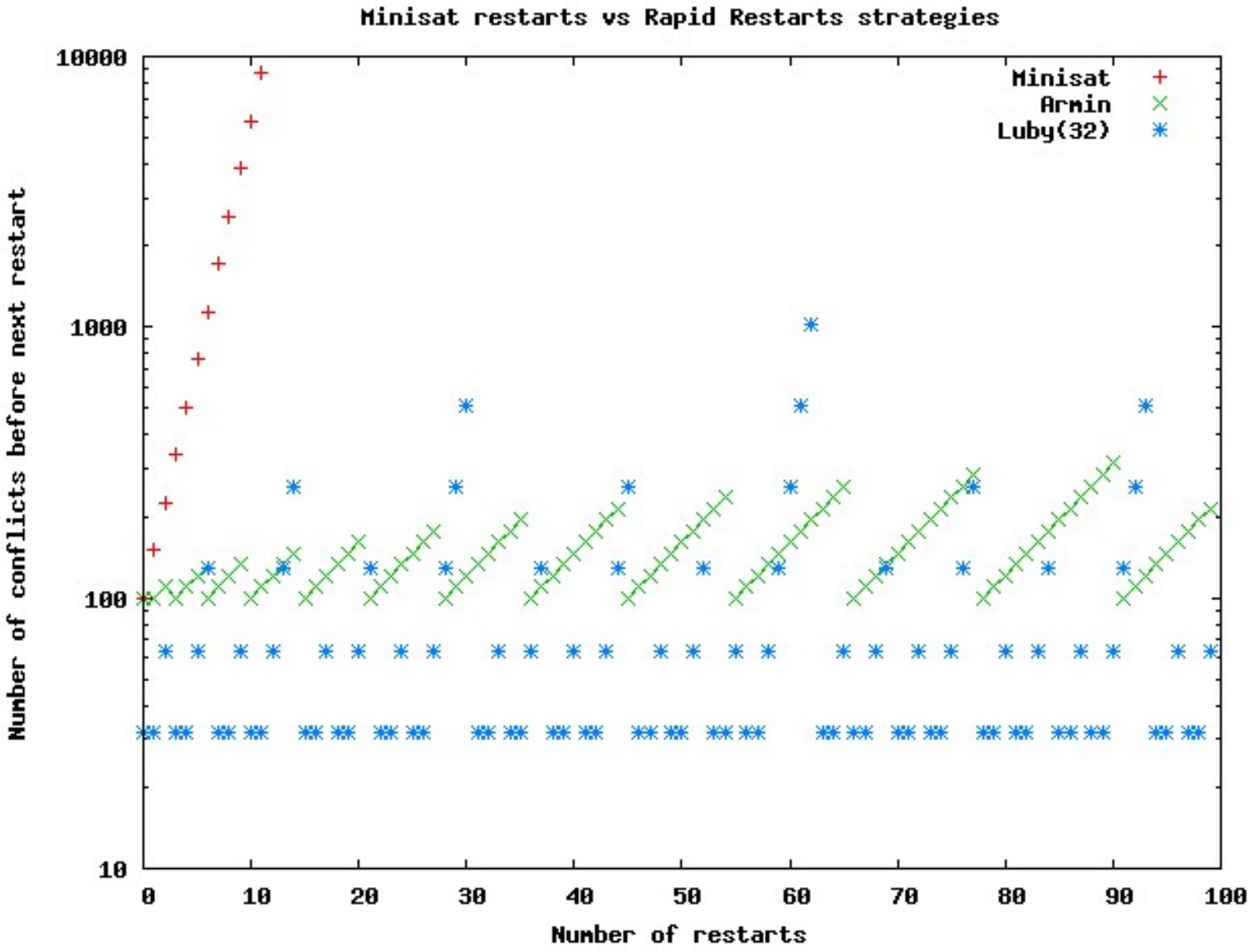
Luby series rapid restarts

Michael Luby, Alistair Sinclair, David Zuckerman : Optimal Speedup of Las Vegas Algorithms. ISTCS 1993 : 128-133

$$t_i = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1; \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \leq i < 2^k - 1. \end{cases}$$

- Used in SATZ_rand and Relsat_rand within Blackbox [Kautz,Selman 99]
- Used in Tinisat and RSAT in 2007 with factor 512.

Comparison of a few different restarts strategies



UNIVERSITÉ D'ARTOIS

Effect of Rapid Restarts in SAT4J

Using the SAT Race 2006 benchmarks set (100 benchmarks), with a timeout of 900 seconds per benchmark :

Configuration	Total	SAT	UNSAT	Time
MiniSAT	58	29	29	835
Luby (factor 32)	59	24	35	790
Luby (factor 512, no PS, no CCM)	48	19	29	947
Luby (factor 512, no CCM)	55	26	29	866
Luby scheme (factor 512)	61	29	32	788
Armin	61	27	34	790

Time is given in minutes, on a PIV 3GHz, 1.5GB of RAM, Java 6 VM under Mandriva Linux 2007.1.

Adaptative restarts during the SAT 2009 competition

picosat A. Biere. Adaptive Restart Control for Conflict Driven SAT Solvers. In Proc. 11th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'08), Lecture Notes in Computer Science (LNCS) vol. 4996, Springer 2008.

Minisat09z Carsten Sinz, Markus Iser : Problem-Sensitive Restart Heuristics for the DPLL Procedure. SAT 2009 : 356-362

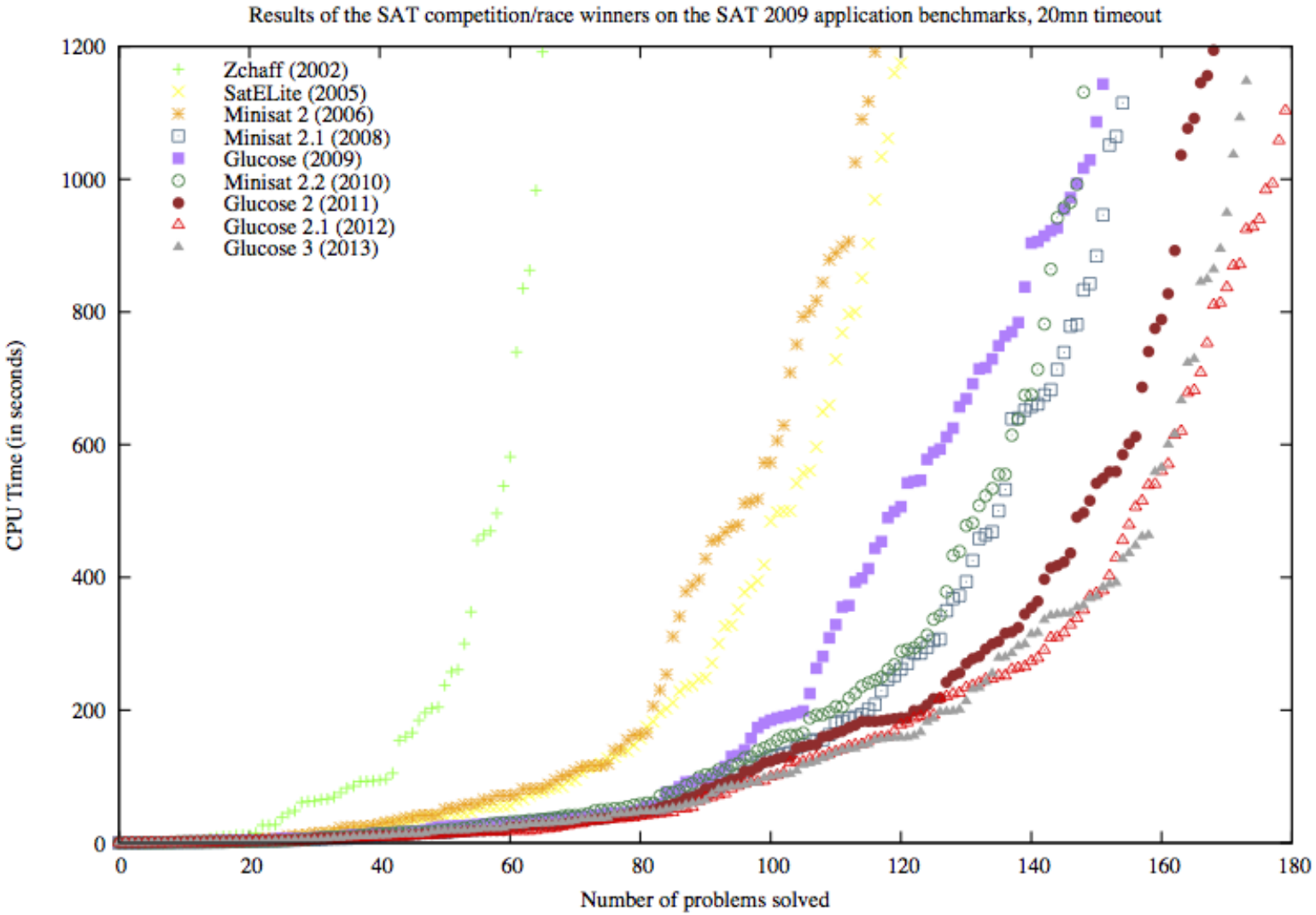
Lysat Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT : a Parallel SAT Solver. Volume 6 (2009), pages 245-262.

glucose Predicting Learnt Clauses Quality in Modern SAT Solver G. Audemard, L. Simon, in Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09), july 2009.

Glucose : agressive learned clause deletion strategies

- Learning a huge amount of clauses reduces the velocity of the solver.
- It would be nice to keep only "important" clauses inside the solver.
- New measure proposed by Glucose : Literal Block Distance (LBD)
  count for each clause the number of different decision level in that clause
  $x_1@1 \vee x_2@3 \vee x_3@1 \vee x_4@2 \vee x_5@1$ LBD $= 3$
- Glucose 1, 2, 2.1, 3.0 : improvement and generalization of the use of LBD inside the solver

Minisat and Glucose



UNIVERSITÉ D'ARTOIS

Agenda

Extending SAT 1 : ~~MaxSat~~ MinUnsat

- Associate to each constraint (clause) a weight (penalty) w_i taken into account if the constraint is violated : Soft constraints ϕ.
- Special weight (∞) for constraints that cannot be violated : hard constraints α
- Find a model I of α that minimizes $weight(I, \phi)$ such that :
  - $weight(I, (c_i, w_i)) = 0$ if I satisfies c_i, else w_i.
  - $weight(I, \phi) = \sum_{wc \in \phi} weight(I, wc)$

Weight	∞	denomination
∞	yes	Sat
k	no	MaxSat
k	yes	Partial MaxSat
$\mathbb{N}$	no	Weighted MaxSat
$\mathbb{N}$	yes	Weighted Partial MaxSat

Partial Max Sat Example : soccer game support

I am French, my family in law is German. Which team should I support when visiting family in law ?

- hard constraint : one should support exactly one team
  $(g \vee f, \infty) \wedge (\neg g \vee \neg f, \infty)$
- soft constraint : supporting Germany (penalty 1 if violated)
  $(g, 1)$
- soft constraint : supporting France (penalty 10 if violated)
  $(f, 10)$

Extending SAT 2 : Pseudo-Boolean problems

Linear Pseudo-Boolean constraint

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

- 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

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

Solving Pseudo Boolean Optimization problems with a SAT solver

- Pseudo-Boolean constraints express a boolean formula $\rightarrow$ that formula can be expressed by a CNF
- One of the best Pseudo-Boolean solver in 2005 was Minisat+, based on that idea : Niklas Eén, Niklas Sörensson : Translating Pseudo-Boolean Constraints into SAT. JSAT 2(1-4) : 1-26 (2006)
- Handling those constraints natively in a CDCL solver isn't hard either (Satire, Satzoo, Minisat, ...) : simplifies the mapping from domain constraints and model constraints, explanations.
- One can easily use a SAT solver to solve an optimization problem using either linear or binary search on the objective function.

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 $\leftarrow$ `isSatisfiable` (setOfConstraints);
if answer *is* UNSAT **then**
  return UNSAT
end
repeat
  model $\leftarrow$ answer ;
  answer $\leftarrow$ `isSatisfiable` (setOfConstraints $\cup$ {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

$$\text{min} : \quad 4x_2 + 2x_3 + x_5$$

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

$< \quad 5$

Optimization algorithm

Formula :

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

Objective function

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

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

$<$ $3 < 5$

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

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

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 : $4x_2 + 2x_3 + x_5$

Objective function value

$<$ $1 < 3$

Optimization algorithm

Formula :

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

Objective function

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

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

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

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.

Extending SAT 3 : Minimally Unsatisfiable Subformula

- Let C be an inconsistent set of clauses.
- $C' \subseteq C$ is an unsat core of C iff C' is inconsistent.
- $C' \subseteq C$ is a MUS of C iff C' is an unsat core of C and no subset of C' is an unsat core of C.

Extending SAT 3 : Minimally Unsatisfiable Subformula

- Let C be an inconsistent set of clauses.
- $C' \subseteq C$ is an unsat core of C iff C' is inconsistent.
- $C' \subseteq C$ is a MUS of C iff C' is an unsat core of C and no subset of C' is an unsat core of C.
- Computing a MUS (set of clauses) is equivalent to computing the set of literals L such that :
  1. L satisfies $\{k_i \vee C_i | C_i \in C\}$
  2. $L \cap K$ is subset minimal

Solvers are available for those problems

Some competitive events are organized for those problems :

- Pseudo Boolean since 2005
- MAX-SAT since 2006
- MUS in 2011
- Certified Unsat track since 2005, successful in 2013 !

As such, a common input format exists, together with a bunch of solvers.

Generalized use of selector variables

The minisat+ syndrom : is a SAT solver sufficient for all our needs ?

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

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

From Weighted Partial Max SAT to PBO

Once cardinality constraints, pseudo boolean constraints and objective functions are managed in a solver, one can easily build a weighted partial Max SAT solver

- Add a selector variable s_i per soft clause C_i : $s_i \vee C_i$
- Objective function : minimize $\sum s_i$
- Partial MAX SAT : no selector variables for hard clauses
- Weighted MAXSAT : use a weighted sum instead of a sum.
  Special case : do not add new variables for unit weighted clauses $w_k l_k$
  Ignore the constraint and add simply $w_k \times \overline{l_k}$ to the objective function.

Selector variables + assumptions = explanation (MUS)

- Assumptions available from the beginning in Minisat 1.12 (incremental SAT)
- 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
- Advantages :
  - no changes needed in the SAT solver internals
  - works for any kind of constraints !
- Approach used in Sat4j and Picosat

From Unsat Core computation to MaxSat : MSU

Z. Fu and S. Malik, On solving the partial MAX-SAT problem, in International Conference on Theory and Applications of Satisfiability Testing, August 2006, pp. 252-265.

Recent advances in practical Max Sat solving rely on unsat core computation [Fu and Malik 2006] :

- Compute one unsat core C' of the formula C
- Relax it by replacing C' by $\{ r_i \vee C_i | C_i \in C' \}$
- Add the constraint $\sum r_i \leq 1$ to C
- Repeat until the formula is satisfiable
- If $MinUnsat(C) = k$, requires k loops.

Many improvement since then (PM1, PM2, MsUncore, etc) : works for Weighted Max Sat, reduction of the number of relaxation variables, etc.

Fu&Malik's Algorithm : msu1.0

x_6, x_2 $\quad \neg x_6, x_2$ $\quad \neg x_2, x_1$ $\quad \neg x_1$

$\neg x_6, x_8$ $\quad x_6, \neg x_8$ $\quad x_2, x_4$ $\quad \neg x_4, x_5$

x_7, x_5 $\quad \neg x_7, x_5$ $\quad \neg x_5, x_3$ $\quad \neg x_3$

Example CNF formula

Fu&Malik's Algorithm : msu1.0

x_6, x_2	$\neg x_6, x_2$	$\neg x_2, x_1$	$\neg x_1$
$\neg x_6, x_8$	$x_6, \neg x_8$	x_2, x_4	$\neg x_4, x_5$
x_7, x_5	$\neg x_7, x_5$	$\neg x_5, x_3$	$\neg x_3$

Formula is UNSAT ; Get unsat core

Fu&Malik's Algorithm : msu1.0

x_6, x_2 $\quad \neg x_6, x_2$ $\quad \neg x_2, x_1, b_1$ $\quad \neg x_1, b_2$

$\neg x_6, x_8$ $\quad x_6, \neg x_8$ $\quad x_2, x_4, b_3$ $\quad \neg x_4, x_5, b_4$

x_7, x_5 $\quad \neg x_7, x_5$ $\quad \neg x_5, x_3, b_5$ $\quad \neg x_3, b_6$

$\sum_{i=1}^{6} b_i \leq 1$

Add blocking variables and AtMost1 constraint

Fu&Malik's Algorithm : msu1.0

x_6, x_2	$\neg x_6, x_2$	$\neg x_2, x_1, b_1$	$\neg x_1, b_2$
$\neg x_6, x_8$	$x_6, \neg x_8$	x_2, x_4, b_3	$\neg x_4, x_5, b_4$
x_7, x_5	$\neg x_7, x_5$	$\neg x_5, x_3, b_5$	$\neg x_3, b_6$
$\sum_{i=1}^{6} b_i \leq 1$			

Formula is (again) UNSAT ; Get unsat core

Fu&Malik's Algorithm : msu1.0

x_6, x_2, b_7 $\quad \neg x_6, x_2, b_8$ $\quad \neg x_2, x_1, b_1, b_9$ $\quad \neg x_1, b_2, b_{10}$

$\neg x_6, x_8$ $\quad x_6, \neg x_8$ $\quad x_2, x_4, b_3$ $\quad \neg x_4, x_5, b_4$

x_7, x_5, b_{11} $\quad \neg x_7, x_5, b_{12}$ $\quad \neg x_5, x_3, b_5, b_{13}$ $\quad \neg x_3, b_6, b_{14}$

$\sum_{i=1}^{6} b_i \leq 1$ $\quad \sum_{i=7}^{14} b_i \leq 1$

Add new blocking variables and AtMost1 constraint

Fu&Malik's Algorithm : msu1.0

x_6, x_2, b_7 $\quad \neg x_6, x_2, b_8$ $\quad \neg x_2, x_1, b_1, b_9$ $\quad \neg x_1, b_2, b_{10}$

$\neg x_6, x_8$ $\quad x_6, \neg x_8$ $\quad x_2, x_4, b_3$ $\quad \neg x_4, x_5, b_4$

x_7, x_5, b_{11} $\quad \neg x_7, x_5, b_{12}$ $\quad \neg x_5, x_3, b_5, b_{13}$ $\quad \neg x_3, b_6, b_{14}$

$\sum_{i=1}^{6} b_i \leq 1$ $\quad \sum_{i=7}^{14} b_i \leq 1$

Instance is now SAT

Fu&Malik's Algorithm : msu1.0

x_6, x_2, b_7 $\quad \neg x_6, x_2, b_8$ $\quad \neg x_2, x_1, b_1, b_9$ $\quad \neg x_1, b_2, b_{10}$

$\neg x_6, x_8$ $\quad x_6, \neg x_8$ $\quad x_2, x_4, b_3$ $\quad \neg x_4, x_5, b_4$

x_7, x_5, b_{11} $\quad \neg x_7, x_5, b_{12}$ $\quad \neg x_5, x_3, b_5, b_{13}$ $\quad \neg x_3, b_6, b_{14}$

$\sum_{i=1}^{6} b_i \leq 1$ $\quad \sum_{i=7}^{14} b_i \leq 1$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

Organization of msu1.0

[Fu&Malik'06]

- Clauses characterized as :
  - Initial : derived from clauses in φ
  - Auxiliary : added during execution of algorithm
    - E.g. clauses from cardinality constraints
- While exist unsatisfiable cores
  - Add fresh set B of blocking variables to non-auxiliary soft clauses in core
  - Add new AtMost1 constraint

$$\sum_{b_i \in B} b_i \leq 1$$

    - At most 1 blocking variable from set B can take value 1
- MaxSAT solution is $|\varphi| - \mathcal{I}$, where $\mathcal{I}$ is number of iterations

Main interest of the approach

- Takes advantage of unsat core computation
- Works well in practice on real MAXSAT problems
- Completely orthogonal to "reasoning-based" MAX SAT approaches.

Agenda

Real problems are not in CNF !

Using SAT technology is hard because

- Efficient encodings are not trivial
- Input format for solvers is not meant for end users
- Reasoning at the boolean level is error prone

Requires some abstraction

Copris

http ://bach.istc.kobe-u.ac.jp/copris/

- CSP DSL in Scala
- Front end to award winning Sugar
  uses the Order Encoding for domain constraints
- Translates CSP into SAT (Dimacs) SMT (SMTLIB 2.0) or CSP (XCSP 2.0, JSR331)
- All-in-one jar with Sat4j as default backend

Scarab

http ://kix.istc.kobe-u.ac.jp/ soh/scarab/

- CSP DSL in Scala
- Full Order Encoding in Scala
- Designed to work intimately with Sat4j
  - Native constraints
  - Incremental SAT
  - Library of predefined solvers
  - ...
- Everything runs in a JVM

Designing Constraint Models in Scarab

Pandiagonal Latin Square $PLS(n)$ is a problem of placing different n numbers into $n \times n$ matrix such that each number is occurring exactly once for each row, column, diagonally down right, and diagonally up right.

- alldiff Model
  - One uses alldiff constraint, which is one of the best known and most studied global constraints in constraint programming.
  - The constraint alldiff$(a_1, \ldots, a_n)$ ensures that the values assigned to the variable $a_1, \ldots, a_n$ must be pairwise distinct.
- Boolean Cardinality Model
  - One uses Boolean cardinality constraint.

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)

alldiff Model

Pandiagonal Latin Square $PLS(5)$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

1	2	3	4	5
3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3

- $x_{ij} \in \{1, 2, 3, 4, 5\}$
- alldiff in each row (5 rows)
- alldiff in each column (5 columns)
- alldiff in each pandiagonal (10 pandiagonals)
- $PLS(5)$ is satisfiable.

Scarab Program for *alldiff* Model

```
 1:  import jp.kobe_u.scarab.csp._
 2:  import jp.kobe_u.scarab.solver._
 3:  import jp.kobe_u.scarab.sapp._
 4:
 5:  val n = args(0).toInt
 6:
 7:  for (i <- 1 to n; j <- 1 to n)  int('x(i,j),1,n)
 8:  for (i <- 1 to n) {
 9:   add(alldiff((1 to n).map(j => 'x(i,j))))
10:   add(alldiff((1 to n).map(j => 'x(j,i))))
11:   add(alldiff((1 to n).map(j => 'x(j,(i+j-1)%n+1))))
12:   add(alldiff((1 to n).map(j => 'x(j,(i+(j-1)*(n-1))%n+1))))
13:  }
14:
15:  if (find)  println(solution)
```

Implementing alldiff in Scarab

- In Scarab, all we have to do for implementing global constraints is just decomposing them into simple arithmetic constraints [Bessiere et al. '09].

In the case of alldiff$(a_1, \ldots, a_n)$,

It is decomposed into pairwise not-equal constraints

$$\bigwedge_{1 \leq i < j \leq n} (a_i \neq a_j)$$

.

- It is also known that some extra constraints improves performance in computation.

Extra Constraints for $alldiff(a_1, \ldots, a_n)$

- In Pandiagonal Latin Square $PLS(n)$, all integer variables $a_1, \ldots, a_n$ have the same domain $\{1, \ldots, n\}$.
- Then, we can add the following extra constraints.
- Permutation constraints :

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} (a_j = i)$$

  - It represents that one of $a_1, \ldots, a_n$ must be assigned to i.
- Pigeon hole constraint :

$$\neg \bigwedge_{i=1}^{n} (a_i < n) \wedge \neg \bigwedge_{i=1}^{n} (a_i > 1)$$

  - It represents that mutually different n variables cannot be assigned within the interval of the size $n - 1$.

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad$ $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad$ $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad$ $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals)

    $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad$ $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals)

    $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad$ $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals)

    $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

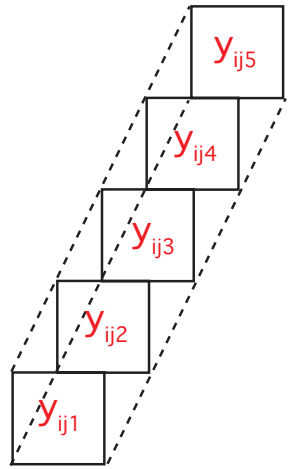
y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}

- $y_{ijk} \in \{0, 1\}$ $\quad$ $y_{ijk} = 1 \Leftrightarrow k$ is placed at (i, j)
- for each value (5 values)
  - for each row (5 rows) $\quad$ $y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad$ $y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals)

    $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$

Boolean Cardinality Model

y_{11k}	y_{12k}	y_{13k}	y_{14k}	y_{15k}
y_{21k}	y_{22k}	y_{23k}	y_{24k}	y_{25k}
y_{31k}	y_{32k}	y_{33k}	y_{34k}	y_{35k}
y_{41k}	y_{42k}	y_{43k}	y_{44k}	y_{45k}
y_{51k}	y_{52k}	y_{53k}	y_{54k}	y_{55k}



- $y_{ijk} \in \{0,1\}$ $\quad y_{ijk} = 1 \Leftrightarrow k$ is placed at (i,j)
- for each value (5 values)
  - for each row (5 rows) $\quad y_{i1k} + y_{i2k} + y_{i3k} + y_{i4k} + y_{i5k} = 1$
  - for each column (5 columns) $\quad y_{1jk} + y_{2jk} + y_{3jk} + y_{4jk} + y_{5jk} = 1$
  - for each pandiagonal (10 pandiagonals)

    $y_{11k} + y_{22k} + y_{33k} + y_{44k} + y_{55k} = 1$
- for each (i,j) position (25 positions) $\quad y_{ij1} + y_{ij2} + y_{ij3} + y_{ij4} + y_{ij5} = 1$

Scarab Program for Boolean Cardinality Model

```
 1:  import jp.kobe_u.scarab.csp._
 2:  import jp.kobe_u.scarab.solver._
 3:  import jp.kobe_u.scarab.sapp._
 4:
 5:  for (i <- 1 to n; j <- 1 to n; num <- 1 to n)
 6:    int('y(i,j,num),0,1)
 7:
 8:  for (num <- 1 to n) {
 9:    for (i <- 1 to n) {
10:     add(BC((1 to n).map(j => 'y(i,j,num)))===1)
11:     add(BC((1 to n).map(j => 'y(j,i,num)))===1)
12:     add(BC((1 to n).map(j => 'y(j,(i+j-1)%n+1,num))) === 1)
13:     add(BC((1 to n).map(j => 'y(j,(i+(j-1)*(n-1))%n+1,num))) === 1)
14:   }
15:  }
16:
17:  for (i <- 1 to n; j <- 1 to n)
18:    add(BC((1 to n).map(k => 'y(i,j,k))) === 1)
19:
20:  if (find)  println(solution)
```

“Real” problems have to be encoded into a CNF

- Finding the right encoding is as important as finding the right solver
- Good SAT encodings typically increase the number of variables
- Powerful SAT encodings are designed to favor unit propagation

“Real” problems have to be encoded into a CNF

- Finding the right encoding is as important as finding the right solver
- Good SAT encodings typically increase the number of variables
- Powerful SAT encodings are designed to favor unit propagation

Example : solving pandiagonal latin square with Scarab

2	3	5	1	4
5	1	4	2	3
4	2	3	5	1
3	5	1	4	2
1	4	2	3	5

Encoding and solving times using different approaches

- Each approach encodes differently cardinality constraint $\sum x_i \leq 1$
- Native means specific handling (no encoding)

n	S/U	Pairwise		Totalizer		Seq. Counter		Native BC	
		Enc.	Sol.	Enc.	Sol.	Enc.	Sol.	Enc.	Sol.
7	S	0.772	0.007	0.088	0.003	0.336	0.012	0.042	0.001
8	U	0.392	0.045	0.134	0.016	0.325	0.026	0.044	0.012
9	U	0.696	0.038	0.191	0.048	0.369	0.063	0.048	0.019
10	U	1.204	0.046	0.258	0.175	0.475	0.137	0.054	0.024
11	S	2.702	0.149	0.341	0.180	0.635	0.109	0.063	0.023
12	U	6.165	0.150	0.443	0.633	0.876	0.731	0.080	0.137
14	U	34.345	3.719	0.712	12.521	1.567	10.806	0.104	3.856
15	U	80.502	221.028	0.890	415.011	2.026	262.875	0.126	215.593
16	U	185.215	190.803	1.096	T.O.	2.591	363.120	0.143	202.636
Total		727.978		>1032.739		647.078		423.004	

To bring back home

- Modern SAT solvers architecture is called CDCL
- CDCL $\neq$ DPLL
- CDCL solvers designed for "application benchmarks"
- See Christophe's talk this afternoon for Parallel SAT solving
- See invited talk by Armin Biere at Pragmatics of SAT (VSL) for lingeling (inprocessing) details