Satisfiability: Algorithms, Applications and Extensions

Javier Larrosa[1] **Inês Lynce**[2] Joao Marques-Silva[3]

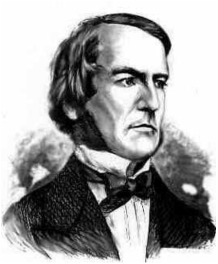
[1]Universitat Politécnica de Catalunya, Spain

[2]**Technical University of Lisbon, Portugal**

[3]University College Dublin, Ireland

SAC 2010

SAT: A Simple Example



- Boolean Satisfiability (SAT) in a short sentence:
  - SAT is the problem of deciding (requires a yes/no answer) if there is an assignment to the variables of a Boolean formula such that the formula is satisfied
- Consider the formula $(a \vee b) \wedge (\neg a \vee \neg c)$
  - The assignment $b = \mathit{True}$ and $c = \mathit{False}$ satisfies the formula!

SAT: A Practical Example

- Consider the following constraints:
  - John can only meet either on Monday, Wednesday or Thursday
  - Catherine cannot meet on Wednesday
  - Anne cannot meet on Friday
  - Peter cannot meet neither on Tuesday nor on Thursday
  - QUESTION: When can the meeting take place?
- Encode then into the following Boolean formula:
  $(\mathit{Mon} \vee \mathit{Wed} \vee \mathit{Thu}) \wedge (\neg \mathit{Wed}) \wedge (\neg \mathit{Fri}) \wedge (\neg \mathit{Tue} \wedge \neg \mathit{Thu})$
  - The meeting *must* take place on *Monday*

Outline

Outline

Outline

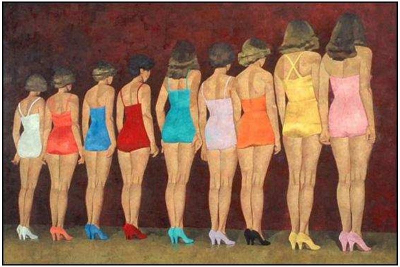
Motivation - Why SAT?

- Boolean Satisfiability (SAT) has seen significant improvements in recent years
  - At the beginning is was *simply* the first known NP-complete problem [Stephen Cook, 1971]
  - After that mostly theoretical contributions followed
  - In the 90's practical algorithms were developed and made available
  - Currently, SAT founds many practical applications
  - SAT extensions found even more applications

Motivation - Some lessons from SAT I

- Time is everything
  - Good ideas are not enough, you have to be fast!
  - One thing is the algorithm, another thing is the implementation
  - Make your source code available
    - Otherwise people will have to wait for years before realising what you have done
    - At least provide an executable!

Motivation - Some lessons from SAT II



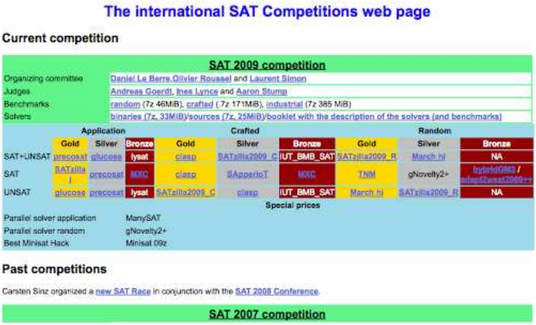
- Competitions are essential
  - To check the state-of-the-art of SAT solvers
  - To keep the community alive (for almost a decade now)
  - To get students involved
- Part of the credibility of a community comes from the correctness and robustness of the tools made available

Motivation - Some lessons from SAT III

- There is no perfect solver!
  - Do not expect your solver to beat all the other solvers on all problem instances
- What makes a good solver?
  - Correctness and robustness for sure...
  - Being most often the best for its category: industrial, handmade or random
  - Being able to solve instances from different problems

www.satcompetition.org

- Get all the info from the SAT competition web page
  - Organizers, judges, benchmarks, executables, source code
  - Winners
    - Industrial, Handmade and Random benchmarks
    - SAT+UNSAT, SAT and UNSAT categories
    - Gold, Silver and Bronze medals



The international SAT Competitions web page

Current competition

SAT 2009 competition

Organizing committee	Daniel Le Berre, Olivier Roussel and Laurent Simon
Judges	Andreas Goerdt, Ines Lynce and Aaron Stump
Benchmarks	random (7z 46MiB), crafted (.7z 171MiB), industrial (7z 385 MiB)
Solvers	binaries (7z, 33MiB)/sources (7z, 25MiB)/booklet with the description of the solvers (and benchmarks)

	Application			Crafted			Random		
	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
SAT+UNSAT	precosat	glucose	lysat	clasp	SATzilla2009_C	IUT_BMB_SAT	SATzilla2009_R	March hi	NA
SAT	SATzilla I	precosat	MXC	clasp	SApperloT	MXC	TNM	gNovelty2+	hybridGM3 / adaptg2wsat2009++
UNSAT	glucose	precosat	lysat	SATzilla2009_C	clasp	IUT_BMB_SAT	March hi	SATzilla2009_R	NA

Special prices

Parallel solver application	ManySAT
Parallel solver random	gNovelty2+
Best Minisat Hack	Minisat 09z

Past competitions

Carsten Sinz organized a new SAT Race in conjunction with the SAT 2008 Conference.

SAT 2007 competition

Outline

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 $\{0, 1\}$
  - 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: $\varphi(x_1, \ldots, x_3) = (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$
- Can encode any Boolean formula into CNF (more later)

Boolean Satisfiability (SAT)

- The Boolean satisfiability (SAT) problem:
  - Find an assignment to the variables $x_1, \ldots, x_n$ such that $\varphi(x_1, \ldots, x_n) = 1$, or prove that no such assignment exists
- SAT is an NP-complete decision problem [Cook'71]
  - SAT was the first problem to be shown NP-complete
  - There are no known polynomial time algorithms for SAT
  - 39-year old conjecture:
    Any algorithm that solves SAT is exponential in the number of variables, in the worst-case

Definitions

- Propositional variables can be assigned value 0 or 1
  - In some contexts variables may be unassigned
- A clause is satisfied if at least one of its literals is assigned value 1

  $(x_1 \lor \neg x_2 \lor \neg x_3)$
- A clause is unsatisfied if all of its literals are assigned value 0

  $(x_1 \lor \neg x_2 \lor \neg x_3)$
- A clause is unit if it contains one single unassigned literal and all other literals are assigned value 0

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

Outline

Algorithms for SAT

- Incomplete algorithms (i.e. can only prove (un)satisfiability):
  - Local search / hill-climbing
  - Genetic algorithms
  - Simulated annealing
  - ...
- Complete algorithms (i.e. can prove both satisfiability and unsatisfiability):
  - Proof system(s)
    - Natural deduction
    - Resolution
    - Stålmarck's method
    - Recursive learning
    - ...
  - Binary Decision Diagrams (BDDs)
  - Backtrack search / DPLL
    - Conflict-Driven Clause Learning (CDCL)
  - ...

Outline

Outline

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

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

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

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

- Start with (possibly random) assignment:
  $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

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

- Start with (possibly random) assignment:
  $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

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

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

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

- Start with (possibly random) assignment:
  $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

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

- Start with (possibly random) assignment:
  $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)
  - Done if all clauses satisfied

Organization of Local Search

- Local search is incomplete; *usually* it cannot prove unsatisfiability
  - Very effective in specific contexts
- Example:

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

- Start with (possibly random) assignment: $x_4 = 0, x_1 = x_2 = x_3 = 1$
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. x_4)
  - Done if all clauses satisfied
- Repeat (random) selection of assignment a number of times

Outline

Outline

Pure Literals

- A literal is pure if only occurs as a positive literal or as a negative literal in a CNF formula
  - Example:
    $\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
  - x_1 and x_3 and pure literals

- Pure literal rule:
  Clauses containing pure literals can be removed from the formula (i.e. just assign pure literals to the values that satisfy the clauses)
  - For the example above, the resulting formula becomes:
    $\varphi = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

- A reference technique until the mid 90s; nowadays seldom used

Unit Propagation

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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

- Unit clause rule:
  Given a unit clause, its only unassigned literal must be assigned value 1 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 0
- 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 unsatisfy clauses. Unsatisfied clauses create conflicts.

Outline

Resolution

- Resolution rule:
  - If a formula φ contains clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$, then one can infer $(\alpha \vee \beta)$

$$(x \vee \alpha) \wedge (\neg x \vee \beta) \vdash (\alpha \vee \beta)$$

- Resolution is a sound and complete rule

Resolution

- Resolution forms the basis of a complete algorithm for SAT
  - Iteratively apply the following steps: [Davis&Putnam'60]
    - Select variable x
    - Apply resolution rule between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$
    - Remove all clauses containing either x or $\neg x$
    - Apply the pure literal rule and unit propagation
  - Terminate when either the empty clause or the empty formula (equivalently, a formula containing only pure literals) is derived

Resolution – An Example

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

Resolution – An Example

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

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

Resolution – An Example

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

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

Resolution – An Example

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

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$

Resolution – An Example

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

$$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \quad \vdash$$

$$(x_3)$$

- Formula is SAT

Outline

Stålmarck's Method

- Recursive application of the branch-merge rule to each variable with the goal of identifying common assignments

  $$\varphi = (a \vee b)(\neg a \vee c)(\neg b \vee d)(\neg c \vee d)$$

  $$(a = 0) \rightarrow (b = 1) \rightarrow (d = 1)$$
  $$UP(a = 0) = \{a = 0, b = 1, d = 1\}$$

  $$(a = 1) \rightarrow (c = 1) \rightarrow (d = 1)$$
  $$UP(a = 1) = \{a = 1, c = 1, d = 1\}$$

  $$UP(a = 0) \cap UP(a = 1) = \{d = 1\}$$

  - Any assignment to variable a implies $d = 1$. Hence, $d = 1$ is a necessary assignment!
- Recursion can be of arbitrary depth

Outline

Recursive Learning

- Recursive evaluation of clause satisfiability requirements for identifying common assignments

  $\varphi = (a \vee b)(\neg a \vee c)(\neg b \vee d)(\neg c \vee d)$

  $(a = 1) \rightarrow (c = 1) \rightarrow (d = 1)$
  $UP(a = 1) = \{a = 1, c = 1, d = 1\}$

  $(b = 1) \rightarrow (d = 1)$
  $UP(b = 1) = \{b = 1, d = 1\}$

  $UP(a = 1) \cap UP(b = 1) = \{d = 1\}$

  - Every way of satisfying $(a \vee b)$ implies $d = 1$.
    Hence, $d = 1$ is a necessary assignment!
- Recursion can be of arbitrary depth

Outline

Historical Perspective I

- In 1960, M. Davis and H. Putnam proposed the DP algorithm:
  - Resolution used to eliminate 1 variable at each step
  - Applied the pure literal rule and unit propagation
- Original algorithm was inefficient

Historical Perspective II

- In 1962, M. Davis, G. Logemann and D. Loveland proposed an alternative algorithm:
  - Instead of eliminating variables, the algorithm would split on a given variable at each step
  - Also applied the pure literal rule and unit propagation
- The 1962 algorithm is actually an implementation of backtrack search
- Over the years the 1962 algorithm became known as the DPLL (sometimes DLL) algorithm

Basic Algorithm for SAT – DPLL

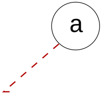
- Standard backtrack search
- At each step:
  - [DECIDE] Select decision assignment
  - [DEDUCE] Apply unit propagation and (optionally) the pure literal rule
  - [DIAGNOSIS] If conflict identified, then backtrack
    - If cannot backtrack further, return UNSAT
    - Otherwise, proceed with unit propagation
  - If formula satisfied, return SAT
  - Otherwise, proceed with another decision

An Example of DPLL

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

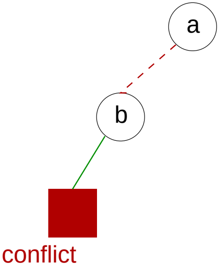
An Example of DPLL

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



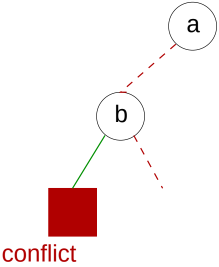
An Example of DPLL

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



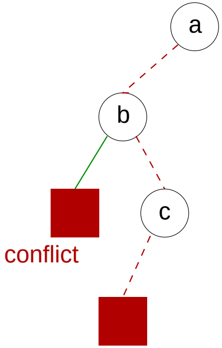
An Example of DPLL

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



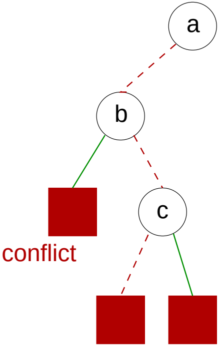
An Example of DPLL

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



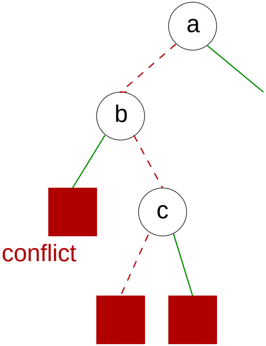
An Example of DPLL

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



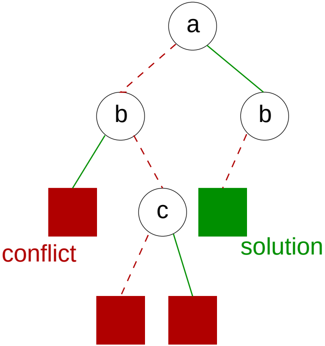
An Example of DPLL

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



An Example of DPLL

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



Outline

CDCL SAT Solvers

- Introduced in the 90's [Marques-Silva&Sakallah'96][Bayardo&Schrag'97]
- Inspired on DPLL
  - Must be able to prove both satisfiability and unsatisfiability
- New clauses are learnt from conflicts
- Structure of conflicts exploited (UIPs)
- Backtracking can be non-chronological
- Efficient data structures [Moskewicz&al'01]
  - Compact and reduced maintenance overhead
- Backtrack search is periodically restarted [Gomes&al'98]

- Can solve instances with hundreds of thousand variables and tens of million clauses

CDCL SAT Solvers

- Introduced in the 90's [Marques-Silva&Sakallah'96][Bayardo&Schrag'97]
- Inspired on DPLL
  - Must be able to prove both satisfiability and unsatisfiability
- New clauses are learnt from conflicts
- Structure of conflicts exploited (UIPs)
- Backtracking can be non-chronological
- Efficient data structures
  - Compact and reduced maintenance overhead
- Backtrack search is periodically restarted

- Can solve instances with hundreds of thousand variables and tens of million clauses

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 $c = 0$ and $f = 0$

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 $c = 0$ and $f = 0$
  - Assign $a = 0$ 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 $c = 0$ and $f = 0$
  - Assign $a = 0$ 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 $c = 0$ and $f = 0$
  - Assign $a = 0$ 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 $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$

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 $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$

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 $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$
  - Learn new clause $(a \vee c \vee f)$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

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

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

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

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \lor c \lor f)$ implies $a = 1$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

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

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

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

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

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

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$

Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

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

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (c = 1) \vee (f = 1)$

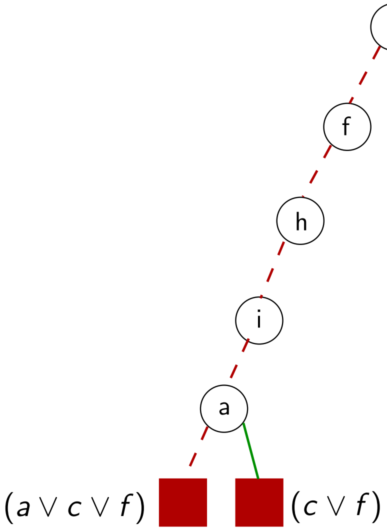
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

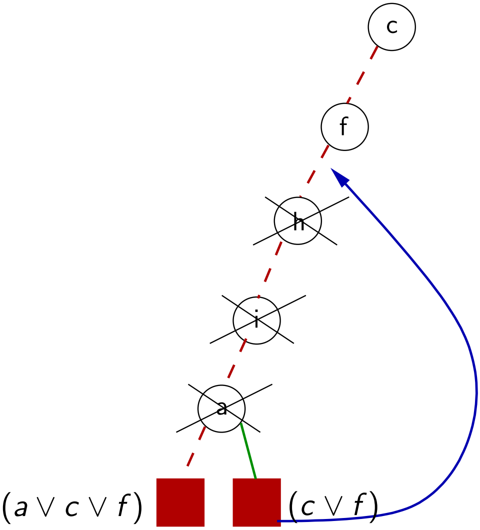
$$\begin{aligned}\varphi \;=\; & (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \wedge \\ & (a \vee c \vee f) \wedge (\neg a \vee g) \wedge (\neg g \vee b) \wedge (\neg h \vee j) \wedge (\neg i \vee k)\end{aligned}$$

  - Assume decisions $c = 0$, $f = 0$, $h = 0$ and $i = 0$
  - Assignment $a = 0$ caused conflict $\Rightarrow$ learnt clause $(a \vee c \vee f)$ implies $a = 1$
  - A conflict is again reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (c = 1) \vee (f = 1)$
  - Learn new clause $(c \vee f)$

Non-Chronological Backtracking



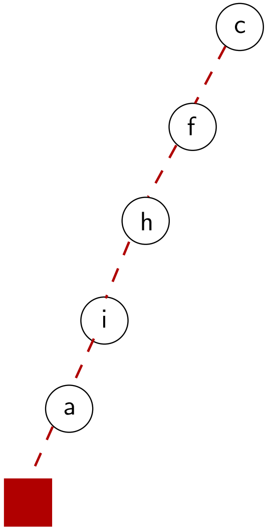
Non-Chronological Backtracking



- Learnt clause: $(c \vee f)$
- Need to backtrack, given new clause
- Backtrack to most recent decision: $f = 0$

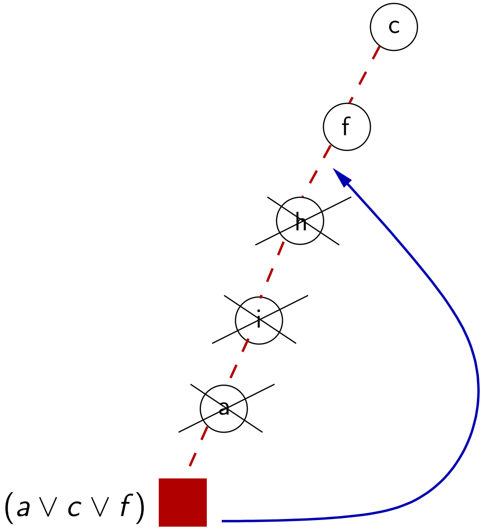
- Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers

Most Recent Backtracking Scheme

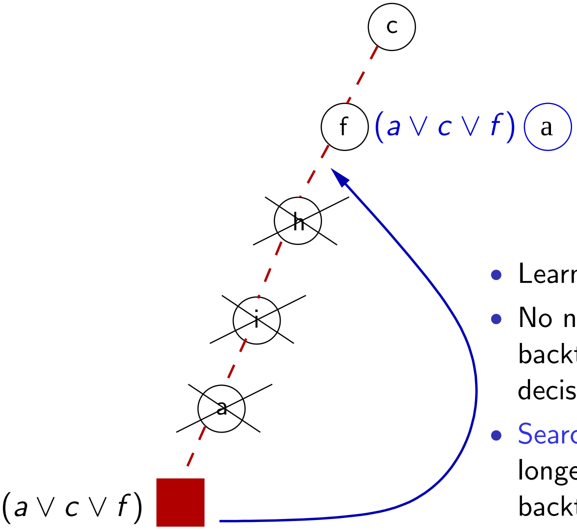


$(a \vee c \vee f)$

Most Recent Backtracking Scheme

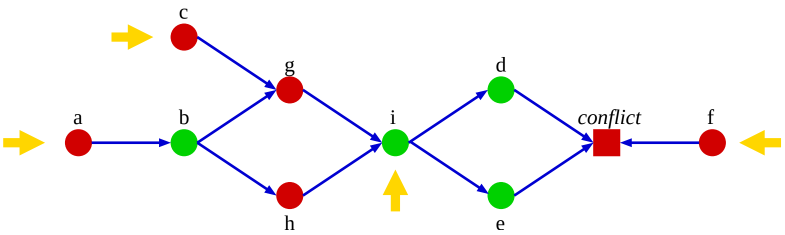


Most Recent Backtracking Scheme



- Learnt clause: $(a \lor c \lor f)$
- No need to assign $a = 1$ - backtrack to most recent decision: $f = 0$
- Search algorithm is no longer a traditional backtracking scheme

Unique Implication Points (UIPs)



- Exploit structure from the implication graph
  - To have a more aggressive backtracking policy
- Identify additional clauses to be learnt [Marques-Silva&Sakallah'96]
  - Create clauses $(a \vee c \vee f)$ and $(\neg i \vee f)$
  - Imply not only $a = 1$ but also $i = 0$
- 1st UIP scheme is the most efficient [Zhang&al'01]
  - Create only one clause $(\neg i \vee f)$
  - Avoid creating similar clauses involving the same literals

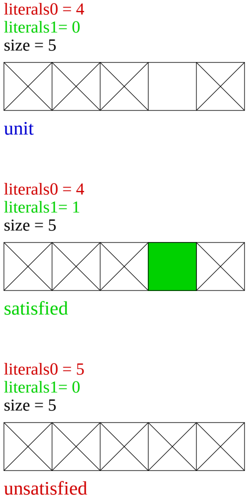
Clause deletion policies

- Keep only the small clauses [Marques-Silva&Sakallah'96]
  - For each conflict record one clause
  - Keep clauses of size $\leq K$
  - Large clauses get deleted when become unresolved
- Keep only the relevant clauses [Bayardo&Schrag'97]
  - Delete unresolved clauses with $\leq M$ free literals
- Keep only the clauses that are used [Goldberg&Novikov'02]
  - Keep track of clauses activity

Data Structures

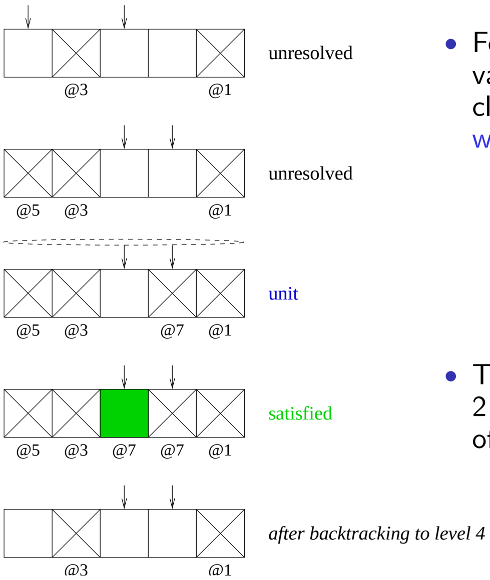
- Key point: only unit and unsatisfied clauses *must* be detected during search
  - Formula is unsatisfied when at least one clause is unsatisfied
  - Formula is satisfied when all the variables are assigned and there are no unsatisfied clauses
- In practice: unit and unsatisfied clauses may be identified using only two references
- Standard data structures (adjacency lists):
  - Each variable x keeps a reference to all clauses containing a literal in x
- Lazy data structures (watched literals):
  - For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched

Standard Data Structures (adjacency lists)



- Each variable x keeps a reference to all clauses containing a literal in x
  - If variable x is assigned, then all clauses containing a literal in x are evaluated
  - If search backtracks, then all clauses of all newly unassigned variables are updated
- Total number of references is L, where L is the number of literals

Lazy Data Structures (watched literals)

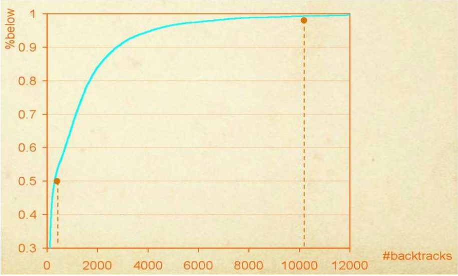


- For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched
  - If variable x is assigned, only the clauses where literals in x are watched need to be evaluated
  - If search backtracks, then nothing needs to be done
- Total number of references is $2 \times C$, where C is the number of clauses
  - In general $L \gg 2 \times C$, in particular if clauses are learnt

Search Heuristics

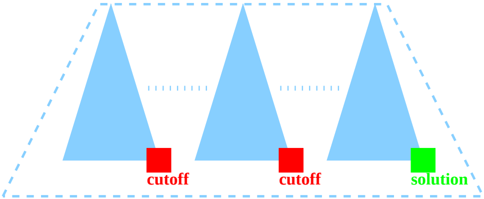
- Standard data structures: heavy heuristics
  - DLIS: Dynamic Large Individual Sum [Marques-Silva'99]
    - Selects the literal that appears most frequently in unresolved clauses
- Lazy data structures: light heuristics
  - VSIDS: Variable State Independent Decaying Sum [Moskewicz&al'01]
    - Each literal has a counter, initialized to zero
    - When a new clause is recorded, the counter associated with each literal in the clause is incremented
    - The unassigned literal with the highest counter is chosen at each decision
  - Other variations
    - Counters updated also for literals in the clauses involved in conflicts [Goldberg&Novikov'02]

Restarts I

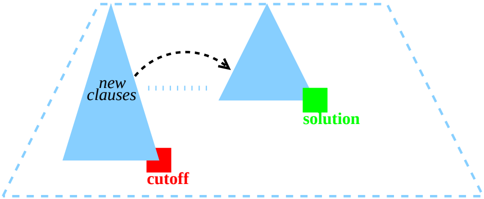


- Plot for processor verification instance with branching randomization and 10000 runs
  - More than 50% of the runs require less than 1000 backtracks
  - A small percentage requires more than 10000 backtracks
- Run times of backtrack search SAT solvers characterized by heavy-tail distributions

Restarts II



- Repeatedly restart the search each time a cutoff is reached
  - Randomization allows to explore different paths in search tree
- Resulting algorithm is incomplete
  - Increase the cutoff value
  - Keep clauses from previous runs



Conclusions

- The ingredients for having an efficient SAT solver
  - Mistakes are not a problem
    - Learn from your conflicts
    - ... and perform non-chronological backtracking
    - Restart the search
  - Be lazy!
    - Lazy data structures
    - Low-cost heuristics

Thank you!