Algorithms for Propositional Satisfiability Solving

Anbulagan
Anbulagan@nicta.com.au
http://users.rsise.anu.edu.au/~anbu/

Logic and Computation Program, NICTA
CSL, RSISE, Australian National University

Presented at DSL, SITACS, University of Wollongong, 16-17 January 2007
Roadmap

- SAT Resources
- Introduction to SAT
  - Real-world problems
  - Basic notation and definitions
  - Problems in CNF formula
  - Phase transition
- Algorithms for SAT solving
  - DP and DPLL procedure
  - Complete methods: Look-ahead based DPLL and CDCL-based DPLL
  - Stochastic methods: GSAT, Random Walk, Clause Weighting SLS
SAT Resources - 1

- Conferences
  - Main: SAT
  - Others: IJCAI, AAAI, ECAI, PRICAI, CP, IJCAR, CP-AI-OR
  - Applications: DAC, ICCAD, ICAPS

- Journals
  - Main: JSAT
  - Others: AIJ, JAIR, JAR, DAM, Constraint

- Competitions
  - Since 2002
  - Website: www.satcompetition.org
International SAT Competition

- Annual competition since 2002
- In 2005, there are 9 gold, 9 silver and 9 bronze medals
- More than 50 SAT solvers entered the contest
- 1,657 problems used: random, crafted and industrial
- 2 stages competition: (20 mins in 1st stage, 100 or 200 mins in 2nd stage)
- No competition in 2006 but there was a SAT-Race for solving only the industrial problems.

- Next competition starts soon.
  - Solver submission deadline is 31 January 2007.
SAT Resources - 2

- Benchmark Data
  - www.satlib.org

- More information about SAT
  - www.satlive.org

- SAT Solvers
  - Most of the solvers can be downloaded from the Internet
  - www.satcompetition.org
  - Authors website
Challenges

- Selman et. al.: IJCAI 1997
  - “Ten Challenges in Propositional Reasoning and Search”
- Kautz and Selman: CP 2003
  - “Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search”
Problems - Traffic Management
Many real world problems can be expressed as a list of constraints. Answer is assignment to variables that satisfy all the constraints.

Example:

- Scheduling people to work in shifts at a hospital.
  - Some people do not work at night.
  - No one can work more than H hours a week.
  - Some pairs of people cannot be on the same shift.
  - Is there assignment of people to shifts that satisfy all constraints?
Problems - Games

- **Sudoku**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Constraint:
There is no same number in the same row, column, or region.

- **N-Queens**

Q₁ Q₂
Q₃
Q₄

Constraint:
In chess, a queen can move horizontally, vertically, or diagonally.
Problems - Games

- Sudoku

```
1 2 3 4
3 4 1 2
2 1 4 3
4 3 2 1
```

Constraint:
There is no same number in the same row, column, or region.

- N-Queens

```
Q1
Q2
Q3
Q4
```

Constraint:
In chess, a queen can move horizontally, vertically, or diagonally.
A Simple Problem: Student-Courses

A student would like to decide on which subjects he should take for the next session. He has the following requirements:

- He would like to take Math or drop Biology.
- He would like to take Biology or Algorithms.
- He does not want to take Math and Algorithms together.

Which subjects this student can take?

\[ F = (X \lor \neg Y) \land (Y \lor Z) \land (\neg X \lor \neg Z) \]
There are 2 possible solutions:

- He could take Math and Biology together. (X=T, Y=T, Z=⊥)
- He could only take Algorithms. (X=⊥, Y=⊥, Z=T)
Practical Applications of SAT

- AI Planning and Scheduling
- Bioinformatics
- Bounded Model Checking
- Data Cleaning
- Diagnosis
- Electronic Design Automation and Verification
- FPGA routing
- Knowledge Discovery
- Security: cryptographic key search
- Software Verification
- Theorem Proving
## Propositional Logic

<table>
<thead>
<tr>
<th>English</th>
<th>Standard</th>
<th>Boolean</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>⊥</td>
<td>0</td>
<td>F</td>
</tr>
<tr>
<td>true</td>
<td>T</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>not $x$</td>
<td>$\neg x$</td>
<td>$x^\sim$</td>
<td>$-x, \sim x$</td>
</tr>
<tr>
<td>$x$ and $y$</td>
<td>$x \land y$</td>
<td>$xy$</td>
<td>$x &amp; y, x \cdot y$</td>
</tr>
<tr>
<td>$x$ or $y$</td>
<td>$x \lor y$</td>
<td>$x + y$</td>
<td>$x \mid y, x \lor y$</td>
</tr>
<tr>
<td>$x$ implies $y$</td>
<td>$x \Rightarrow y$</td>
<td>$x \leq y$</td>
<td>$x \rightarrow y, x \supset y$</td>
</tr>
<tr>
<td>$x$ iff $y$</td>
<td>$x \Leftrightarrow y$</td>
<td>$x = y$</td>
<td>$x \equiv y, x \sim y$</td>
</tr>
</tbody>
</table>
### Semantics of Boolean Operators

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x$</th>
<th>$x \land y$</th>
<th>$x \lor y$</th>
<th>$x \Rightarrow y$</th>
<th>$x \Leftrightarrow y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>⊥</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>T</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>T</td>
<td>T</td>
<td>⊥</td>
<td>T</td>
<td>T</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>T</td>
<td>⊥</td>
<td>⊥</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

**N.B.:**

- $x \lor y = \neg (\neg x \land \neg y)$
- $x \Rightarrow y = (\neg x \lor y)$
- $x \Leftrightarrow y = (x \Rightarrow y) \land (y \Rightarrow x)$
Basic Notation & Definitions

A SAT problem consists of

- Formula $F$: a conjunction of clauses using AND ($\land$) operator
- A set of Boolean variables $\{x_1, x_2, \ldots, x_n\}$
- Literal: $x_i$ is a positive literal of variable $x_i$, and $\neg x_i$ is its negation
- Clause: a disjunction of literals using OR ($\lor$) operator
- Unit Clause: a clause containing a single literal
- Binary Clause: a clause that contains two literals
- Empty Clause: a clause without any literal
- Pure Literal: a variable appearing only negatively or positively

\[
F = (x_1 \lor x_2 \lor x_3) \land (x_4 \lor \neg x_5) \land (\neg x_2 \lor x_4 \lor x_5) \land (\neg x_3)
\]
Solution for a given SAT problem

- **Satisfiable (SAT):**
  - There is at least an assignment of values {true, false} to the variables of the formula where all its clauses are satisfiable.
  - Only one model
  - Many models: find one quickly or find all

- **Unsatisfiable (UNSAT):**
  - There is no model found
  - MAX-SAT: satisfy maximum clauses

Solution of a SAT formula is when a solver can prove whether the formula is satisfiable (SAT) or unsatisfiable (UNSAT).
SAT Problems - Definition

Input: A formula $F$ in Conjunctive Normal Form (CNF)

Output: $F$ is satisfiable by a consistent assignment of truth value to variables or $F$ is unsatisfiable.

Example of a CNF formula:

$$F = (x_1 \lor x_2 \lor x_3) \land (x_4 \lor \neg x_5) \land (\neg x_2 \lor x_6 \lor x_7)$$

The first NP-Complete problem [Cook, 1971]
SAT Problems - Random Versus Structured

Random Problems: generated using a random problem generator.

The random SAT formula $F_k(n, m)$

- Given a set $V$ of $n$ Boolean variables $\{x_1, x_2, x_3, \ldots, x_n\}$, $m$ clauses of length $k$ are generated randomly. Each clause is produced by randomly choosing $k$ variables from $V$ and negating each with probability 0.5.
- Fixed clause lengths formula
- Mitchell et al., 1992

Structured Problems:

- Structures: symmetries, variable dependencies, clustering
- Generated from real-world problems
- Crafted problems

Random+Structured Problems:

- QWH = quasigroup with holes
- bQWH = balanced quasigroup with holes
- Problem generator is available
Simple Random Problem

- Random 3-SAT problem with 5 variables and 20 clauses
- generated using random SAT problem generator.

p cnf 5 20
-1 -2 -3 0
1 -3 -4 0
-1 -3 5 0
3 4 -5 0
-1 2 -5 0
-2 -3 -5 0
1 -2 -3 0
1 -3 4 0
1 -3 -5 0
-2 3 -5 0
2 -4 -5 0
-1 -2 -4 0
2 -3 4 0
-3 -4 -5 0
-2 4 5 0
-3 -4 5 0
-1 3 -5 0
2 -3 5 0
1 2 -3 0
-2 3 5 0
Structured Problem: par8-1-c.cnf

c parXX-Y-c denotes a parity problem on XX bits.
c Y is simply the instance number. c means that the instance has been simplified.
p cnf 64 254
-2 1 0
-3 -2 0
-3 -2 -1 0
3 2 -1 0
-3 2 1 0
3 -2 1 0
-4 2 0
-5 -4 0
-5 -4 -2 0
5 4 -2 0
-5 4 2 0
5 -4 2 0
.....
Structured Problem: Bounded Model Checking

The instance bmc-ibm-6.cnf, IBM, 1997

p cnf 51639 368352

-1 7 0
-1 6 0
-1 5 0
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0

i.e. \((\neg x_7)\) or \(x_7\)
and \((\neg x_5)\) or \(x_5\)
and … etc

10224 -10043 0
10224 -10044 0
10008 10009 10010 10011 10012 10013 10014 10015 10016 10017 10018 10019 10020 10021 10022
10023 10024 10025 10026 10027 10028 10029 10030 10031 10032 10033 10034 10035 10036
10037 10038 10039 10040 10041 10042 10043 10044 -10224 0 // a constraint with 64 literals at line 72054
10083 -10157 0
10083 -10227 0
10083 -10228 0
10157 10227 10228 -10083 0
Structured Problem: Bounded Model Checking

At the end of the file

\[
\begin{align*}
7 & -260 0 \\
1072 & 1070 0 \\
-15 & -14 -13 -12 -11 -10 0 \\
-15 & -14 -13 -12 -11 10 0 \\
-15 & -14 -13 -12 11 -10 0 \\
-15 & -14 -13 -12 11 10 0 \\
-7 & -6 -5 -4 -3 -2 0 \\
-7 & -6 -5 -4 -3 2 0 \\
-7 & -6 -5 -4 3 -2 0 \\
-7 & -6 -5 -4 3 2 0 \\
185 & 0
\end{align*}
\]

**Note that:** \(2^{5169}\) is a very big number !!!

\[
2^{100} = 1,267,650,000,000,000,000,000,000,000,000
\]

Dew_Satz SAT solver (Anbulagan, 2005) solves this instance in 36 seconds.
Phase Transition in Random Problems

- Also called threshold phenomenon
- A phenomenon in which satisfiability problems suddenly change from easy (100% SAT) to hard (100% UNSAT) as abruptly as water freezing into ice.

\[ r_1 = 0; r_2 = 1; r_3 \approx 4.258; \]

- \( r = \) ratio number of clauses to number of variables (clause density).
- \( r_\alpha = \) critical value for formula with fixed clause lengths \( \alpha \).
- The critical value divides the space of SAT formulas into 3 regions, such as:
  - Under-constraint region: almost all the formulas are satisfiable and easy to solve.
  - Hard-constraint region: the region of 50% satisfiable formulas and hard to solve.
  - Over-constraint region: almost all the formulas are unsatisfiable and easy to solve.
Phase Transition: n=60:20:160
Phase Transition and Difficulty Level: n=200
How to Solve the Problems

- **Complete method: guarantee to obtain a solution**
  - Based on DPLL procedure [Davis et. al., 1962]
  - Enhanced by look-ahead: Satz, Dew_Satz, kcnfs, march_dl, …
  - Enhanced by CDCL: GRASP, RELSAT, Chaff, zChaff, MiniSat, Siege, Berkmin, Jerusat, Tinisat, …

- **Stochastic method: no guarantee to obtain a solution**
  - Stochastic Local Search:
    - Random Walk: WalkSAT, AdaptNovelty⁺, g2wsat, R+AdaptNovelty⁺, …
    - Clause Weighting: SAPS, PAWS, DDFW, R+DDFW⁺
  - Evolutionary algorithms
  - Neural networks
  - etc…

- **Hybrid approach**
Complete Method: Davis Putnam Procedure

- The original procedure (DP) used a resolution rule, leading to potentially exponential use of space. [Davis & Putnam, 1960]

- Davis, Logemann and Loveland replaced the resolution rule with a splitting rule. The new procedure is know as the DPLL or DPL procedure. [Davis et al., 1962]

- Despite its age, still one of the most popular and successful complete methods. Basic framework for many modern SAT solvers.

- Exponential time is still a problem.
DP Procedure

Procedure DP($F$)

for $l = 1$ to $\text{NumberOfVariableIn}(F)$

choose variable $x \in \text{VAR}(F)$

Resolvants = $\emptyset$

forall $(C_1, C_2)$ such that

$C_1 \in F$, $C_2 \in F$, $x \in C_1$, $\neg x \in C_2$

Resolvants = Resolvants $\cup$ resolve($C_1, C_2$)

// don’t generate tautological resolvants.

$C_x = \{ C : C \in F \text{ and } x \in C \}$

$F = F - C_x$

$F = F \cup \text{Resolvants}$

// $x$ is not in any clause in Resolvants. So now $x$ is not in $F$.

if $F = \emptyset$ return UNSATISFIABLE

else return SATISFIABLE
Resolution for SAT example

\[(x_1 \lor x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor \neg x_6) \land (\neg x_2 \lor x_5)\]
\[\Downarrow\]
\[(x_1 \lor x_3 \lor x_5) \land (\neg x_3 \lor \neg x_6 \lor x_5)\]
\[\Downarrow\]
\[(x_1 \lor x_5 \lor \neg x_6)\]

\[\Rightarrow \text{SAT}\]
Resolution for UNSAT example

\[(x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2)\]
\[
\Downarrow
\]
\[(x_2) \land (x_2 \lor \neg x_2) \land (\neg x_2 \lor x_2) \land (\neg x_2)\]
\[
\Downarrow
\]
\[\emptyset\]

⇒ UNSAT
Resolution for UNSAT example

\[(x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (\neg x_1 \lor \neg x_3)\]

\[\Downarrow\]

\[(x_1) \land (\neg x_1 \lor x_3) \land (\neg x_1 \lor \neg x_3)\]

\[\Downarrow\]

\[(x_3) \land (\neg x_3)\]

\[\Downarrow\]

\[\emptyset\]

\[\Rightarrow\] UNSAT
DPLL Procedure

Procedure DPLL($F$)

(Sat) if $F = \emptyset$ then “SAT”

(Empty) if $F$ contains the empty clause then “UNSAT”

(Unit Pr) if $F$ has unit clause $\{u\}$, then DPLL($F\{u/\text{true}\}$)

(Pure) if $F$ has pure literal $p$, then DPLL($F\{p/\text{true}\}$)

(Split) if DPLL($F\{l/\text{true}\}$) is satisfiable then “SAT”
else DPLL($F\{l/\text{false}\}$)

Unit Propagation (UP) is also called Boolean Constraint Propagation (BCP). It is used to detect unit clauses and conflict clauses.
DPLL: Basic Notation & Definitions

- **Branching variable**: a variable that can have 2 sub-branches
- **Free variable**: a variable without any value
- **Contradiction / dead-end / conflict**: an empty clause is found
- **Backtracking**: An algorithmic technique to find solutions by trying one of several choices. If the choice proves incorrect, computation *backtracks* or restarts at the point of choice and tries another choice.
Search Tree of DPLL Procedure

- Binary Search Tree
- Large search tree size $\iff$ Hard problem
- The hardness of a random SAT problem is independent from the SAT Solver used.
- Depth first search with backtracking
DPLL Tree of Student-Courses Problem

(X \lor \neg Y) \land (Y \lor Z) \land (\neg X \lor \neg Z)

Using DPLL Procedure
DPLL Performance: Original vs. Variants

- The worst case complexity of the algorithm in our experience is $O(2^{n/21.83-1.70})$, based on UNSAT problems.
- This is an improvement!… Notice that for example,

$$2^{100} = 1,267,650,000,000,000,000,000,000,000,000,000$$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
<th>#nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPLL (1962):</td>
<td>$O(2^{n/5})$</td>
<td>1,048,576</td>
</tr>
<tr>
<td>Satz (1997):</td>
<td>$O(2^{n/20.63+0.44})$</td>
<td>39</td>
</tr>
<tr>
<td>Satz215 (1999):</td>
<td>$O(2^{n/21.04-1.01})$</td>
<td>13</td>
</tr>
<tr>
<td>Kcnfs (2003):</td>
<td>$O(2^{n/21.10-1.35})$</td>
<td>10</td>
</tr>
</tbody>
</table>

- Look-ahead-enhanced DPLL based SAT solver can reliably solve problems with up to 700 variables.
Heuristics for DPLL Procedure

Objective: to reduce search tree size by choosing a best branching variable at each node of the search tree.

Central issue: How to select the next best branching variable?
Branching Heuristics in DPLL Procedure

- **Simple**
  - use simple heuristics for branching variable selection
  - Based on a literal or a variable occurrences counting

- **Sophisticated**
  - use sophisticated heuristics for branching variable selection
  - Need more resources and efforts.
Simple Branching Heuristics

- MOMS (Maximum Occurrences in Minimum Sized clauses) heuristics: pick the literal that occurs most often in the minimum size clauses.
  - Maximum binary occurrences
  - too simplistic
  - CSAT [Dubois et. al, 1993]

- Jeroslow-Wang’s heuristics [Jeroslow & Wang, 1990; Hooker & Vinay, 1995]: estimate the contribution of each literal $l$ to satisfying the clause set and pick the best

$$\text{score}(l) = \sum_{c \in F \land l \in c} 2^{-|c|}$$

for each clause $c$ the literal $l$ appears in $2^{-|c|}$ is added where $|c|$ is the number of literals in $c$. 
Sophisticated Branching Heuristics

- Look-ahead-based DPLL
- Unit Propagation Look-Ahead(UPLA) heuristics
  - Satz [Li & Anbulagan, 1997]
- Backbone Search heuristics
  - kcnfs [Dequen, 2003]
- Dynamic Variable Filtering(DVF) heuristics
  - ssc34 and ssc355 [Anbulagan, 2004]
  - LAS+NVO
- LAS+NVO+DEW heuristics
  - Dew_Satz [Anbulagan & Slaney, 2005]
Satz - Complete SAT Solver

- No physical modification on variables and clauses
- An efficient backtracking management
- Count the number of clauses at each node
- Using UPLA Heuristic for detecting contradictions earlier.
- Resolvents resolution as pre-processing (3-Resolution)
- Open for the integration of new ideas
UPLA Heuristics

- **UPLA** = Unit Propagation Look-Ahead
- **Goal:** to find the best branching variable by performing UPLA.
- UPLA was one of the main improvements to the DPLL procedure, after more than 3 decades.
- **Paper:** “Heuristics based on Unit Propagation for Satisfiability Problems” [Li and Anbulagan, IJCAI-1997].
Let $PROP$ be a binary predicate such that $PROP(x,i)$ is true iff $x$ is a variable that occurs both positively and negatively in binary clauses and occurs in at least $i$ binary clauses in $F$, and let $T$ be an integer, then $PROP_z(x)$ is defined to be the first of the three predicates $PROP(x,4)$, $PROP(x,3)$, $true$ (in this order) whose denotational semantics contains more than $T$ variables.

$T$ is fixed to 10 in Satz.
Algorithm 1 LA-BranchingRule($\mathcal{F}$)

1: for each variable $x_i \in \mathcal{V}$ do
2:   Let $\mathcal{F}_i'$ and $\mathcal{F}_i''$ be two copies of $\mathcal{F}$;
3:   $\mathcal{F}_i' := \text{UP}(\mathcal{F}_i' \cup \{x_i\})$;
4:   $\mathcal{F}_i'' := \text{UP}(\mathcal{F}_i'' \cup \{\bar{x}_i\})$;
5:   if empty clause $\in \mathcal{F}_i'$ and empty clause $\in \mathcal{F}_i''$ then
6:     return "unsatisfiable";
7:   else if empty clause $\in \mathcal{F}_i'$ then
8:     $\mathcal{F} := \mathcal{F}_i''$;
9:   else if empty clause $\in \mathcal{F}_i''$ then
10:  $\mathcal{F} := \mathcal{F}_i'$;
11:  else
12:     $w(x_i) := \text{diff}(\mathcal{F}_i', \mathcal{F})$;
13:     $w(\bar{x}_i) := \text{diff}(\mathcal{F}_i'', \mathcal{F})$;
14:     $\mathcal{W}(x_i) := w(x_i) \ast w(\bar{x}_i) + w(x_i) + w(\bar{x}_i)$;
15:  end if
16: end for
17: return $x_i$ with highest $\mathcal{W}(x_i)$ to branch on;
LAS+NVO Heuristics

Algorithm 3 NVO-LAS-BranchingRule($\mathcal{F}$)
1: Push each variable $x_i \in \mathcal{V}$ to NVO_STACK;
2: repeat
3: \hspace{1em} $\mathcal{F}_{\text{init}} := \mathcal{F}$;
4: \hspace{1em} for each variable $x_i \in \text{NVO\_STACK}$ do
5: \hspace{2em} Let $\mathcal{F}_i'$ and $\mathcal{F}_i''$ be two copies of $\mathcal{F}$;
6: \hspace{2em} $\mathcal{F}_i' := \text{UP}(\mathcal{F}_i' \cup \{x_i\})$;
7: \hspace{2em} $\mathcal{F}_i'' := \text{UP}(\mathcal{F}_i'' \cup \{\overline{x_i}\})$;
8: \hspace{2em} if empty clause $\in \mathcal{F}_i'$ and empty clause $\in \mathcal{F}_i''$ then
9: \hspace{3em} return "unsatisfiable";
10: \hspace{2em} else if empty clause $\in \mathcal{F}_i'$ then
11: \hspace{3em} $\mathcal{F} := \mathcal{F}_i''$;
12: \hspace{3em} NVO($x_i$);
13: \hspace{2em} else if empty clause $\in \mathcal{F}_i''$ then
14: \hspace{3em} $\mathcal{F} := \mathcal{F}_i'$;
15: \hspace{3em} NVO($x_i$);
16: \hspace{2em} else
17: \hspace{3em} $w(x_i) := \text{diff}(\mathcal{F}_i', \mathcal{F})$;
18: \hspace{3em} $w(\overline{x}_i) := \text{diff}(\mathcal{F}_i'', \mathcal{F})$;
19: \hspace{3em} $\mathcal{W}(x_i) := w(x_i) \ast w(\overline{x}_i) + w(x_i) + w(\overline{x}_i)$;
20: \hspace{2em} end if
21: \hspace{1em} end for
22: until $\mathcal{F} = \mathcal{F}_{\text{init}}$
23: NVO($x_i$);
24: return $x_i$ with highest $\mathcal{W}(x_i)$ to branch on;

- **LAS** = look-ahead Saturation
- **NVO** = Neighbourhood Variables Ordering
- **The idea of integrating LAS:** do UPLA process until the sub-formulae at each node becomes non-reducible. Then execute MOMS heuristic to choose a best branching variable.
- **Result:** Success in finding a best branching variable at each node and reduce significantly the number of branching nodes.
- **The idea of integrating NVO:** attempt to limit the number of free variables examined by exploring next only the neighbourhood variables of the current assigned variable.
## Empirical Results on Security problem (cnf-r3*)

<table>
<thead>
<tr>
<th>Probs.</th>
<th># Vars</th>
<th>#Cls</th>
<th>satz215</th>
<th>ssc34</th>
<th>ssc355</th>
<th>satz215</th>
<th>ssc34</th>
<th>ssc355</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1-k1.1</td>
<td>21536</td>
<td>8966</td>
<td>2008485</td>
<td>1265</td>
<td>3551</td>
<td>2966</td>
<td>71</td>
<td>124</td>
</tr>
<tr>
<td>b1-k1.2</td>
<td>152608</td>
<td>8891</td>
<td>N/A</td>
<td>3002</td>
<td>1500</td>
<td>&gt;3600</td>
<td>174</td>
<td>53</td>
</tr>
<tr>
<td>b2-k1.1</td>
<td>152608</td>
<td>17857</td>
<td>128061</td>
<td>0</td>
<td>0</td>
<td>792</td>
<td>1.05</td>
<td>0.88</td>
</tr>
<tr>
<td>b2-k1.2</td>
<td>414752</td>
<td>17960</td>
<td>181576</td>
<td>0</td>
<td>0</td>
<td>1254</td>
<td>1.19</td>
<td>1.09</td>
</tr>
<tr>
<td>b3-k1.1</td>
<td>283680</td>
<td>26778</td>
<td>31647</td>
<td>0</td>
<td>0</td>
<td>448</td>
<td>1.89</td>
<td>1.51</td>
</tr>
<tr>
<td>b3-k1.2</td>
<td>676896</td>
<td>27503</td>
<td>38279</td>
<td>0</td>
<td>0</td>
<td>600</td>
<td>2.25</td>
<td>1.64</td>
</tr>
<tr>
<td>b4-k1.1</td>
<td>414752</td>
<td>35817</td>
<td>11790</td>
<td>0</td>
<td>0</td>
<td>348</td>
<td>3.00</td>
<td>2.41</td>
</tr>
<tr>
<td>b4-k1.2</td>
<td>939040</td>
<td>35963</td>
<td>20954</td>
<td>0</td>
<td>0</td>
<td>624</td>
<td>3.37</td>
<td>2.71</td>
</tr>
</tbody>
</table>

ssc34 uses LAS, while ssc355 uses LAS+NVO
Empirical Results on Random 3-SAT Problem

Mean search tree size of each DPLL procedure as a function of nb. of variables for hard random 3-SAT problems at ratio 4.25 (1000 problems are solved at each point)
Empirical Results on Random 3-SAT Problem

Mean search tree size of each DPLL procedure as a function of number of variables for hard random unsatisfiable 3-SAT problems at ratio 4.25
Empirical Results on Random 3-SAT Problem

• On hard random 3-SAT problems with 350 variables (300 problems are solved), mean search tree size of:
  • Satz215: 36156 branching nodes
  • kcnfs: 24669 branching nodes
  • ssc34: 15507 branching nodes
  • ssc355: 13675 branching nodes

• Search tree size of ssc355 is 164% and 80% smaller, respectively than those for Satz215 and kcnfs.
More Reasoning & Less Searching

- Problem: v350c1488g255 (unsatisfiable)

<table>
<thead>
<tr>
<th></th>
<th># Branch. Nodes</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssc355</td>
<td>65,784</td>
<td>189</td>
</tr>
<tr>
<td>kcnfs</td>
<td>93,655</td>
<td>40</td>
</tr>
<tr>
<td>Satz215</td>
<td>123,735</td>
<td>61</td>
</tr>
<tr>
<td>OKsolver</td>
<td>275,159</td>
<td>438</td>
</tr>
<tr>
<td>MiniSat</td>
<td>25,456,254</td>
<td>1660</td>
</tr>
<tr>
<td>Siege</td>
<td>n/a</td>
<td>&gt; 9000</td>
</tr>
<tr>
<td>zChaff</td>
<td>n/a</td>
<td>&gt; 9000</td>
</tr>
<tr>
<td>Tinisat</td>
<td>n/a</td>
<td>&gt; 9000</td>
</tr>
</tbody>
</table>

- Number of Branching Nodes versus Runtime
- More reasoning at each node increases the runtime cost.
• **Backbone variable** is a variable which is assigned the same value for all solutions to the SAT/CSP problems.

• Such variables are also called **frozen variables**.

• Detection of backbone variables during LA process.

• The cnfs and kcnfs solvers implemented a pseudo-backbone variables detection heuristic.
Backbone Variable: an example

\[(x_1 \lor \neg x_2) \land (x_1 \lor \neg x_3) \land (\neg x_1 \lor x_7) \land (\neg x_1 \lor x_8) \land (x_4 \lor \neg x_7 \lor \neg x_8) \land (\neg x_4 \lor x_5 \lor x_6) \land (x_4 \lor x_2 \lor x_3)\]

Find the backbone variable(s)!
Backbone Variable: an example

\[(x_1 \lor \neg x_2) \land \neg x_1 \lor x_7 \land \neg x_1 \lor x_8 \land (x_4 \lor \neg x_7 \lor \neg x_8) \land \neg x_4 \lor x_5 \lor x_6 \land (x_4 \lor x_2 \lor x_3)\]

Find the backbone variable(s)!

The answer is \(x_4\)
Comparison Results

Graph showing the comparison results for different solvers:
- OK solver
- satz215
- kcnfs
- ssc355

The x-axis represents the number of variables, while the y-axis shows the mean backtracks. The graphs illustrate the performance of each solver as the number of variables increases.
LAS+NVO+DEW Heuristics

Algorithm 2 DewSatz-BranchingRule(\(\mathcal{F}\))

1: Push each variable \(x_i \in \mathcal{V}\) to NVO_STACK at the root node;
2: repeat
3: \(\mathcal{B} := \emptyset;\)
4: \(\mathcal{F}_{\text{init}} := \mathcal{F};\)
5: for each variable \(x_i \in \text{NVO\_STACK}\) do
6: \(\) Let \(\mathcal{F}'_i\) and \(\mathcal{F}''_i\) be two copies of \(\mathcal{F};\)
7: \(\) if \(w(x_i) = 0\) then
8: \(\) \(\mathcal{F}'_i := \text{UP}(\mathcal{F}'_i \cup \{x_i\});\)
9: \(\) if \(w(\bar{x}_i) = 0\) then
10: \(\) \(\mathcal{F}''_i := \text{UP}(\mathcal{F}''_i \cup \{\bar{x}_i\});\)
11: \(\) if empty clause \(\in \mathcal{F}'_i\) and empty clause \(\in \mathcal{F}''_i\) then
12: \(\) return Unsatisfiable;
13: \(\) else if empty clause \(\in \mathcal{F}'_i\) then
14: \(\) \(\mathcal{F} := \mathcal{F}''_i;\)
15: \(\) \(\text{NVO}(x_i);\)
16: \(\) else if empty clause \(\in \mathcal{F}''_i\) then
17: \(\) \(\mathcal{F} := \mathcal{F}'_i;\)
18: \(\) \(\text{NVO}(\bar{x}_i);\)
19: \(\) else
20: \(\) \(\mathcal{B} := \mathcal{B} \cup \{x_i\};\)
21: \(\) \(w(x_i) := \text{diff}(\mathcal{F}'_i, \mathcal{F});\)
22: \(\) \(w(\bar{x}_i) := \text{diff}(\mathcal{F}''_i, \mathcal{F});\)
23: \(\) \(\text{Compute\_DEW}(x_i);\)
24: until \(\mathcal{F} = \mathcal{F}_{\text{init}}\)
25: for each variable \(x_i \in \mathcal{B}\) do
26: \(\) \(\mathcal{W}(x_i) := w(x_i) \times w(\bar{x}_i) + w(x_i) + w(\bar{x}_i);\)
27: \(\) \(\text{NVO}(x_i);\)
28: return \(x_i\) with highest \(\mathcal{W}(x_i)\) to branch on;

The basic idea of integrating DEW (dynamic equivalency weighting):

Whenever the binary equivalency clause \((x_i \iff x_j)\), which is equivalent to 2 CNF clauses \((\neg x_i \lor x_j)\) and \((x_i \lor \neg x_j)\), occurs in the formula at a node, Satz needs to perform look-ahead on \(x_i, \neg x_i, x_j,\) and \(\neg x_j\).

As result, variables \(x_i\) and \(x_j\) will be associated the same weight.

Clearly, the look-aheads on \(x_i\) and \(\neg x_j\) are redundant, so we avoid them by assigning the implied literal \(x_j\) (\(\neg x_i\)’s) the weight of its parent literal \(x_i\) (\(\neg x_i\)’s), and then by avoiding look-ahead on literals with weight zero.

By doing so, we save two look-aheads.
EqSatz

- Based on Satz
- Enhanced with equivalency reasoning during search process.
  - Substitute the equivalent literals during the search in order to reduce the number of active variables in the current formula.
  - Example: given the clause \( x_i \Leftrightarrow x_j \), we can substitute \( x_j \) by \( x_i \).
Equivalency: Reasoning vs. Weighting

On 32-bit Parity Learning problem
A challenging problem [Selman et al., 1997]

<table>
<thead>
<tr>
<th>Instance (#Vars/#Cls)</th>
<th>Satz</th>
<th>Dew_Satz</th>
<th>EqSatz</th>
<th>Lsat</th>
<th>March_eq</th>
<th>zChaff</th>
</tr>
</thead>
<tbody>
<tr>
<td>par32-1 (3176/10227)</td>
<td>&gt;24h</td>
<td>12,918</td>
<td>242</td>
<td>126</td>
<td>0.22</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-2 (3176/10253)</td>
<td>&gt;24h</td>
<td>5,804</td>
<td>69</td>
<td>60</td>
<td>0.27</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-3 (3176/10297)</td>
<td>&gt;24h</td>
<td>7,198</td>
<td>2,863</td>
<td>183</td>
<td>2.89</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-4 (3176/10313)</td>
<td>&gt;24h</td>
<td>11,005</td>
<td>209</td>
<td>86</td>
<td>1.64</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-5 (3176/10325)</td>
<td>&gt;24h</td>
<td>17,564</td>
<td>2,639</td>
<td>418</td>
<td>8.07</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-1-c (1315/5254)</td>
<td>&gt;24h</td>
<td>10,990</td>
<td>335</td>
<td>270</td>
<td>2.63</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-2-c (1303/5206)</td>
<td>&gt;24h</td>
<td>411</td>
<td>13</td>
<td>16</td>
<td>2.19</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-3-c (1325/5294)</td>
<td>&gt;24h</td>
<td>4,474</td>
<td>1,220</td>
<td>374</td>
<td>6.65</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-4-c (1333/5326)</td>
<td>&gt;24h</td>
<td>7,090</td>
<td>202</td>
<td>115</td>
<td>0.45</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>par32-5-c (1339/5350)</td>
<td>&gt;24h</td>
<td>11,899</td>
<td>2,896</td>
<td>97</td>
<td>6.44</td>
<td>&gt;24h</td>
</tr>
</tbody>
</table>

The first solver which solved all the instances.
Equivalency: Reasoning vs. Weighting

On BMC and circuit-related problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Dew_Satz</th>
<th>EqSatz</th>
<th>March_eq</th>
<th>zChaff</th>
</tr>
</thead>
<tbody>
<tr>
<td>barrel6</td>
<td>4.13</td>
<td>0.17</td>
<td>0.13</td>
<td>2.95</td>
</tr>
<tr>
<td>barrel7</td>
<td>8.62</td>
<td>0.23</td>
<td>0.25</td>
<td>11</td>
</tr>
<tr>
<td>barrel8</td>
<td>72</td>
<td>0.36</td>
<td>0.38</td>
<td>44</td>
</tr>
<tr>
<td>barrel9</td>
<td>158</td>
<td>0.80</td>
<td>0.87</td>
<td>66</td>
</tr>
<tr>
<td>longmult10</td>
<td>64</td>
<td>385</td>
<td>213</td>
<td>872</td>
</tr>
<tr>
<td>longmult11</td>
<td>79</td>
<td>480</td>
<td>232</td>
<td>1,625</td>
</tr>
<tr>
<td>longmult12</td>
<td>97</td>
<td>542</td>
<td>167</td>
<td>1,643</td>
</tr>
<tr>
<td>longmult13</td>
<td>127</td>
<td>617</td>
<td>53</td>
<td>2,225</td>
</tr>
<tr>
<td>longmult14</td>
<td>154</td>
<td>706</td>
<td>30</td>
<td>1,456</td>
</tr>
<tr>
<td>longmult15</td>
<td>256</td>
<td>743</td>
<td>23</td>
<td>392</td>
</tr>
<tr>
<td>philips-org</td>
<td>697</td>
<td>1974</td>
<td>&gt;5,000</td>
<td>&gt;5,000</td>
</tr>
<tr>
<td>philips</td>
<td>295</td>
<td>2401</td>
<td>726</td>
<td>&gt;5,000</td>
</tr>
</tbody>
</table>
Results on Hard Random k-SAT Problems

- **Benchmark:** from 2005 International SAT Competition.
- **Experiment:** Each solver was timed out after 200 minutes in the second stage of 2005 International SAT Competition.
- Nb. of hard random k-SAT problems solved by a given DPLL solver.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Problem</th>
<th>SAT (285)</th>
<th>UNSAT (105)</th>
<th>ALL (390)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kcnfs</td>
<td></td>
<td>92</td>
<td>75</td>
<td>167</td>
</tr>
<tr>
<td>Dew_Satz</td>
<td></td>
<td>68</td>
<td>50</td>
<td>118</td>
</tr>
<tr>
<td>March_dl</td>
<td></td>
<td>56</td>
<td>43</td>
<td>99</td>
</tr>
</tbody>
</table>

- **Dew_Satz** won 2 bronze medals for UNSAT and ALL categories.
• Parallelization for reduce the problem solving time
• Using dynamic load balancing mechanism based on the work stealing techniques.
• Significant improvement.
• Paper by [Jurkowiak, Li and Utard, 2005]
MaxSatz

- MAX-SAT is an optimisation variant of SAT
- For over-constraint problems
- MAX-SAT is a special case of weighted MAX-SAT where all clauses have weight one.
- The goal: to find a variable assignment that satisfies a maximal number of clauses of a given CNF formula.
- Paper by [Li, Manya and Planes, 2006]
CDCL-based DPLL
Backjumping

- Idea: when a branch fails,
  - Reveal the sub-assignment causing the contradiction (conflict set)
  - Backtrack to the most recent branching point in the conflict set

- A conflict set is constructed from the conflict clause by tracking backwards the unit-implications causing it and by keeping the branching literals.

- When a branching point fails, a conflict set is obtained by resolving the two conflict sets of the two branches.

- May avoid a lot of redundant search.
Backjumping: an example

\[ (\neg x_1 \lor x_2) \land \\
(\neg x_1 \lor x_3 \lor x_9) \land \\
(\neg x_2 \lor \neg x_3 \lor x_4) \land \\
(\neg x_4 \lor x_5 \lor x_{10}) \land \\
(\neg x_4 \lor x_6 \lor x_{11}) \land \\
(\neg x_5 \lor \neg x_6) \land \\
(x_1 \lor x_7 \lor \neg x_{12}) \land \\
(x_1 \lor x_8) \land \\
(\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \land \\
\ldots \ldots \]
Backjumping: an example

\((\neg x_1 \lor x_2) \land\)
\((\neg x_1 \lor x_3 \lor x_9) \land\)
\((\neg x_2 \lor \neg x_3 \lor x_4) \land\)
\((\neg x_4 \lor x_5 \lor x_{10}) \land\)
\((\neg x_4 \lor x_6 \lor x_{11}) \land\)
\((\neg x_5 \lor \neg x_6) \land\)
\((x_1 \lor x_7 \lor \neg x_{12}) \land\)
\((x_1 \lor x_8) \land\)
\((\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \land\)

.....

\{\ldots, \neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}, \ldots\} \ (initial\ assignment)
Backjumping: an example

\[(\neg x_1 \lor x_2) \land \neg x_1 \lor x_3 \lor x_9 \land \neg x_2 \lor \neg x_3 \lor x_4 \land \neg x_4 \lor x_5 \lor x_{10} \land \neg x_4 \lor x_6 \lor x_{11} \land \neg x_5 \lor \neg x_6 \land x_1 \lor x_7 \lor \neg x_{12} \land \neg x_7 \lor \neg x_8 \lor \neg x_{13} \land \text{removed} \]

\[\{\ldots, \neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}, \ldots, x_1\} \text{ (branch on } x_1)\]

(unit \(x_2, x_3\))
Backjumping: an example

\[
\begin{align*}
&(-x_1 \lor x_2) \land \text{ removed} \\
&(-x_1 \lor x_3 \lor x_9) \land \text{ removed} \\
&(-x_2 \lor -x_3 \lor x_4) \land \\
&(-x_4 \lor x_5 \lor x_{10}) \land \\
&(-x_4 \lor x_6 \lor x_{11}) \land \\
&(-x_5 \lor -x_6) \land \\
&(x_1 \lor x_7 \lor -x_{12}) \land \text{ removed} \\
&(x_1 \lor x_8) \land \text{ removed} \\
&(-x_7 \lor -x_8 \lor -x_{13}) \land \\
\ldots \ldots \\
&\{\ldots, -x_9, -x_{10}, -x_{11}, x_{12}, x_{13}, \ldots, x_1, x_2, x_3\} \\
&(\text{unit } x_4)
\end{align*}
\]
Backjumping: an example

\[(\neg x_1 \lor x_2) \land \text{removed}\]
\[(\neg x_1 \lor x_3 \lor x_9) \land \text{removed}\]
\[(\neg x_2 \lor \neg x_3 \lor x_4) \land \text{removed}\]
\[(\neg x_4 \lor x_5 \lor x_{10}) \land \text{removed}\]
\[(\neg x_4 \lor x_6 \lor x_{11}) \land \text{removed}\]
\[(\neg x_5 \lor \neg x_6) \land \text{removed}\]
\[(x_1 \lor x_7 \lor \neg x_{12}) \land \text{removed}\]
\[(x_1 \lor x_8) \land \text{removed}\]
\[(\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \land \text{removed}\]

\[
\{\ldots, \neg x_9, \neg x_{10}, \neg x_{11}, x_1, x_{12}, x_{13}, \ldots, x_1, x_2, x_3, x_4\}
\]
\[(\text{unit } x_5, x_6)\]
Backjumping: an example

\[ (¬x_1 v x_2) \land \]
\[ (¬x_1 v x_3 v x_9) \land \]
\[ (¬x_2 v ¬x_3 v x_4) \land \]
\[ (¬x_4 v x_5 v x_10) \land \]
\[ (¬x_4 v x_6 v x_11) \land \]
\[ (¬x_5 v ¬x_6) \land \]
\[ (x_1 v x_7 v ¬x_12) \land \]
\[ (x_1 v x_8) \land \]
\[ (¬x_7 v ¬x_8 v ¬x_13) \land \]

\[ \ldots \]
\[ \{ \ldots, ¬x_9, ¬x_{10}, ¬x_{11}, x_{12}, x_{13}, \ldots, x_1, x_2, x_3, x_4, x_5, x_6 \} \]

Conflict set: \{¬x_9, ¬x_{10}, ¬x_{11}, x_1\} ⇒ backtrack to x_1
Backjumping: an example

\[
\begin{align*}
(-x_1 \lor x_2) \land \quad & \text{removed} \\
(-x_1 \lor x_3 \lor x_9) \land \quad & \text{removed} \\
(-x_2 \lor -x_3 \lor x_4) \land \\
(-x_4 \lor x_5 \lor x_{10}) \land \\
(-x_4 \lor x_6 \lor x_{11}) \land \\
(-x_5 \lor -x_8) \land \\
(x_1 \lor x_7 \lor -x_{12}) \land \\
(x_1 \lor x_8) \land \\
(-x_7 \lor -x_8 \lor -x_{13}) \land \\
\ldots \\
\{\ldots, -x_9, -x_{10}, -x_{11}, x_{12}, x_{13}, \ldots, -x_1\} \quad & \text{(branch on } -x_1) \\
\text{(unit } x_7, x_8) 
\end{align*}
\]
Backjumping: an example

\[
\begin{align*}
&(-x_1 \lor x_2) \land \quad \text{removed} \\
&(-x_1 \lor x_3 \lor x_9) \land \quad \text{removed} \\
&(-x_2 \lor \neg x_3 \lor x_4) \land \\
&(-x_4 \lor x_5 \lor x_{10}) \land \\
&(-x_4 \lor x_6 \lor x_{11}) \land \\
&(-x_5 \lor \neg x_6) \land \\
&(x_1 \lor x_7 \lor \neg x_{12}) \land \quad \text{removed} \\
&(x_1 \lor x_8) \land \quad \text{removed} \\
&(-x_7 \lor \neg x_8 \lor \neg x_{13}) \land \quad \text{conflict} \\
\end{align*}
\]

\[
\ldots
\]

\[
\{ \ldots, \neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}, \ldots, \neg x_1, x_7, x_8 \}
\]

Conflict set: \{x_{12}, x_{13}, \neg x_1\}
Backjumping: an example

\[ (\neg x_1 \lor x_2) \land \quad \text{removed} \]
\[ (\neg x_1 \lor x_3 \lor x_9) \land \quad \text{removed} \]
\[ (\neg x_2 \lor \neg x_3 \lor x_4) \land \]
\[ (\neg x_4 \lor x_5 \lor x_{10}) \land \]
\[ (\neg x_4 \lor x_6 \lor x_{11}) \land \]
\[ (\neg x_5 \lor \neg x_6) \land \]
\[ (x_1 \lor x_7 \lor \neg x_{12}) \land \quad \text{removed} \]
\[ (x_1 \lor x_8) \land \quad \text{removed} \]
\[ (\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \land \quad \text{conflict} \]

Conflict set: \( \{x_{12}, x_{13}, \neg x_1\} \lor \{\neg x_9, \neg x_{10}, \neg x_{11}, x_1\} \)
\[ \Rightarrow \{\neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}\} \Rightarrow \text{backtrack to } x_{13} \]
Look-Ahead vs. Look-Back

(1) Heuristique UP
(2) Backtracking simple
(3) Backjumping
Learning

- Idea: when a conflict set $C$ is revealed, then $\neg C$ can be added to the clause set
  - DPLL will never again generate an assignment containing $C$.

- May avoid a lot of redundant search.

- Problem: may cause a blow up in space
  - Techniques to control learning and to drop learned clauses when necessary.

- Learning is very effective in pruning the search space for structured problems.
Learning: an example

\[
(\neg x_1 \lor x_2) \land \quad \text{removed}
\]
\[
(\neg x_1 \lor x_3 \lor x_9) \land \quad \text{removed}
\]
\[
(\neg x_2 \lor \neg x_3 \lor x_4) \land \quad \text{removed}
\]
\[
(\neg x_4 \lor \neg x_5 \lor x_{10}) \land \quad \text{removed}
\]
\[
(\neg x_4 \lor x_6 \lor x_{11}) \land \quad \text{removed}
\]
\[
(\neg x_5 \lor \neg x_6) \land \quad \text{conflict}
\]
\[
(x_1 \lor x_7 \lor \neg x_{12}) \land \quad \text{removed}
\]
\[
(x_1 \lor x_8) \land \quad \text{removed}
\]
\[
(\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \land
\]
\[
\ldots
\]
\[
(x_9 \lor x_{10} \lor x_{11} \lor \neg x_1) \quad \text{learned clause}
\]

Conflict set: \{\neg x_9, \neg x_{10}, \neg x_{11}, x_1\}
Learn: \(x_9 \lor x_{10} \lor x_{11} \lor \neg x_1\)
Pseudocode of MiniSat

Algorithm 1 MINISAT
1: loop
2: propagate()
3: if not conflict then
4: if all variables assigned then
5: return SATISFIABLE
6: else
7: decide()
8: else
9: analyze()
10: if top-level conflict found then
11: return UNSATISFIABLE
12: else
13: backtrack()
Decision Heuristic in CDCL-based Solvers

• **VSIDS = Variable State Independent Decaying Sum**
  - It keeps a score for each phase of a variable. Initially, the scores are the number of occurrences of a literal in the initial formula. VSIDS increases the score of a variable by a constant whenever an added clause contains the variable. Moreover, as the search progresses, periodically all scores are divided by a constant number. VSIDS will choose a free variable with the highest score to branch.
  - Used in zChaff
  - An improved version is used in MiniSat where variable activities are decayed 5% after each conflict.

• **VMTF = Variable Move To Front**
  - The initial order of the list is sorted by the occurrence of the variables in the formula. Every time a new clause is learnt and then added to the database, a constant number of the variables from the clause are moved to the front of the list. The list is resorted according to the occurrence of variables in clauses database every time that the restart occurs.
  - Used in Siege
Restart Policy in CDCL-based Solvers

- Abandon the current search space and restart a new one after exceeding certain conditions, such as number of backtracks.
- Increase the backtrack cutoff value by a constant amount to allow solving unsatisfiable formula.
- The clauses learned prior to the restart are considered in the new search. They will help to prune the search space.
- The effect of restarts on the efficiency of clause learning [Huang, 2007]
Watched Literals Mechanism

- For efficient unit propagation and backtrack processes
- Using 2-literal watching in each clause
- The Quest for Efficient Boolean Satisfiability Solver [Zhang and Malik, 2002]
The imagination driving Australia’s ICT future.

Results of SAT-Race 2006

- minisat2.0
- Eureka
- Rsat
- CadenceMiniSat
- Actin
- Barcelogic
- PicoSAT
- QPicoSAT
- TINISAT
- SAT4J
- QCompSAT
- zChaff
- CompSAT
- MXC
- Mucsat
- HyperSAT
Stochastic Methods: motivation

- DPLL can reliably solve hard random problems with up to 700 variables.
- … but problems commonly arising in practice often need 100,000s - millions of variables.
- We might need “anytime answers” which can provide a “best guess” at any point we stop the algorithm.

SLS Algorithms:
- GSAT
- Random Walk: WalkSAT, AdaptNovelty⁺, g2wsat
- Clause Weighting: SAPS, PAWS, DDFW, DDFW⁺

- Other approach: SP (Survey Propagation)
Local Optima and Global Optimum in SLS

- **Local optima**
- **Global optimum**

The diagram illustrates the concept of finding the global optimum in a solution space, which is the objective function value.
Flipping Coins: The “Greedy” Algorithm

- This algorithm is due to Koutsopias and Papadimitriou
- Main idea: flip variables till you can no longer increase the number of satisfied clauses.

Procedure greedy($F$)

$T = \text{random}(F)$  // random assignment

repeat until no improvement possible

$T = T$ with variable flipped that increases the number of satisfied clauses

end
The GSAT Procedure

- This algorithm is due to Selman, Levesque and Mitchell
- Adds restarts to the simple “greedy” algorithm, and also allows sideways flips.

Procedure GSAT($F$, MAX_TRIES, MAX_FLIPS)

```
for i=1 to MAX_TRIES  // these are the restarts
    $T = \text{random}(F)$  // random assignment
    for j=1 to MAX_FLIPS  // to ensure termination
        if $T$ satisfies $F$ then return $T$
        Flip any variable in $T$ that results in greatest increase in number of satisfied clauses
        // it does not matter if the number does not increase.
        // This are the sideways flips
    end
end
return “No satisfying assignment found”
```

End GSAT
The WALKSAT Procedure

- The procedure is due to Selman, Kautz and Cohen

Procedure WalkSAT($F$, MAX_TRIES, MAX_FLIPS, VSH)

for i=1 to MAX_TRIES  // these are the restarts
    $T = \text{random}(F)$  // random assignment
for j=1 to MAX_FLIPS  // to ensure termination
    if $T$ satisfies $F$ then return $T$.
    choose unsatisfied clause $C \in F$ at random.
    choose a variable $x \in C$ according to VSH.
    $T = T$ with variable $x$ flipped.
end
end
return “No satisfying assignment found”
End WalkSAT
• WalkSAT variants depend on the setting of their noise parameter.

• **Noise parameter**: to control the degree of greediness in the variable selection process. It takes value between zero and one.

• AdaptNovelty$^+$ is for adaptively tuning the noise level based on the detection of stagnation.
Dynamic Local Search: The basic idea

• Use clause weighting mechanism
  • Increase weights on unsatisfied clauses in local minima in such a way that further improvement steps become possible
  • Adjust weights periodically when no further improvement steps are available in the local neighborhood
Dynamic Local Search: A brief history

- Breakout Method [Morris, 1993]
- Weighted GSAT [Selman and Kautz, 1993]
- Learning short-term clause weights for GSAT [Frank, 1997]
- Discrete Lagrangian Method (DLM) [Wah and Shang, 1997]
- Smoothed Descent and Flood [Schuurmans and Southey, 2000]
- Scaling and Probabilistic Smoothing (SAPS) [Hutter, Tompkins, and Hoos, 2002]
- Pure Additive Weighting Scheme (PAWS) [Thornton et al., 2004]
- Divide and Distribute Fixed Weight (DDFW) [Ishtaiwi et al., 2005]
- Adaptive DDFW (DDFW⁺) [Ishtaiwi et al., 2006]
DDFW$^+$

- Adaptive DDFW
- No parameter tuning
- Dynamically alters the total amount of weight that DDFW distributes according to the degree of stagnation in the search.
- The weight initialization value is set at 2 and could be altered during the search between 2 and 3.
- R+DDFW$^+$ is the current best SLS solver for solving random and structured problems
Search for improving flips
(Repeat until MaxFlips or Sol. Found)

For all clauses
\( W_{\text{init}} = 2 \)

Distribute Weights

If Found

Yes

flip

No

If Flips > Literals

Yes

Reset Weights
\( W_{\text{SAT}} = 2 \)
\( W_{\text{UNSAT}} = 3 \)

No

No

Taken from Abdul Sattar's presentation slide at CP’06
R+DDFW⁺

3-Resolution

For all clauses
\( W_{\text{init}} = 2 \)

Search for improving flips
(Repeat until MaxFlips or Sol. Found)

If Found

Distribute Weights

Yes

If Flips > Literals

No

Reset Weights
\( W_{\text{SAT}} = 2 \)
\( W_{\text{UNSAT}} = 3 \)

No

Yes

Taken from Abdul Sattar’s presentation slide at CP’06
Comparison Results of SLS on Random Problems

![Graph showing percentage solved over time for different algorithms on random 3SAT problems](image)

- R+DDFW
- R+DDFW+
- R+AdaptNovelty+
- R+G^2WSAT

a. Random 3SAT problems (50 Instances)
Comparison Results of SLS on Ferry Planning Problems

b. Industrial Ferry problems (16 Instances)
Old Resolution meets Modern SLS

- Adding restricted resolution as a preprocessor
- See [Anbulagan et al., 2005]

```
Algorithm 1 ComputeResolvents()
1: for each clause \( c_1 \) of length \( \leq 3 \) in \( \mathcal{F} \) do
2:     for each literal \( l \) of \( c_1 \) do
3:         for each clause \( c_2 \) of length \( \leq 3 \) in \( \mathcal{F} \) s.t. \( \bar{l} \in c_2 \) do
4:             Compute resolvent \( r = (c_1 \setminus \{l\}) \cup (c_2 \setminus \{\bar{l}\}) \);
5:             if \( r \) is empty then
6:                 return "unsatisfiable";
7:             else
8:                 if \( r \) is of length \( \leq 3 \) then
9:                     \( \mathcal{F} := \mathcal{F} \cup \{r\} \);
10:                end if
11:            end if
12:         end for
13:     end for
14: end for
```

\[ (x_1 \lor x_2 \lor x_3) \land
(\neg x_1 \lor x_2 \lor x_4) =
(x_2 \lor x_3 \lor \neg x_4) \]
From 2005 International SAT Competition

- R+AdaptNovelty+ won the Gold Medal.
- Joint work with IIIS-Griffith University
- Solves 209 of 285 random SAT problems.
- The 2\textsuperscript{nd} and 3\textsuperscript{rd} place are g2wsat (178) and VW (170).
- The 2004 winner, AdaptNovelty+ could only solve 119 problems
### Boosting SLS using Resolution

- **Old resolution meets modern SLS**
  - [Anbulagan et al., AAAI-2005]
  - Limited by using only the 3-Resolution preprocessor.
Resolution-based Preprocessors

- 3-Resolution [Li and Anbulagan, CP-1997]: computes resolvents for all pairs of clauses of length \( \leq 3 \)
- 2-SIMPLIFY [Brafman, IJCAI-2001]: constructs an implication graph from all binary clauses of a problem instance and uses a restricted variant of hyper-resolution.
- HyPre [Bacchus and Winter, SAT-2003]: reasons with binary clauses and do full hyper-resolution.
- NiVER [Subbarayan and Pradhan, SAT-2004]: Non increasing Variable Elimination Resolution.
- SatELite [Één and Biere, SAT-2005]: improved NiVER with a variable elimination by substitution rule.
Problems

- Hard random 3-SAT (3sat), 10 instances, SAT2005
- Quasigroup existence (qg), 10 instances, SATLIB
- 10 Real-world domains
  - All interval series (ais), 6 instances, SATLIB
  - BMC-IBM (bmc), 3 instances, SATLIB
  - BW planning (bw), 4 instances, SATLIB
  - Job-shop scheduling e*ddr* (edd), 6 instances, SATLIB
  - Ferry planning (fer), 5 instances, SAT2005
  - Logistics planning (log), 4 instances, SATLIB
  - Parity learning par16* (par), 5 instances, SATLIB
  - “single stuck-at” (ssa), 4 instances, SATLIB
  - Cryptographic problem (vmcp), 5 instances, SAT2005
  - Models generated from Alloy (vpn), 2 instances, SAT2005
- Problem instance size
  - The smallest (ais6) contains 61 variables and 581 clauses
  - The largest (vpn-1962) contains 267,766 variables and 1,002,957 clauses
The Impact of Preprocessor Variables Reduction
The Impact of Preprocessor

Clauses Reduction

![Graph showing the impact of preprocessor clauses reduction with various symbols representing different tools: 3-Resolution, 2-SIMPLIFY, HyPre, NiVER, and SatELite. The x-axis represents different problems, and the y-axis represents the reduction ratio.](image_url)
The Impact of Preprocessor

Literals Reduction

The graph represents the impact of preprocessor literals reduction across various problems, with different markers indicating different preprocessor tools. The x-axis represents the problem names (e.g., 3sat, ais, bmc, bw,edd, fer, log, par, qg, ssa, vmpc, vpn), and the y-axis represents the reduction ratio. The tools shown include 3-Resolution, 2-SIMPLIFY, HyPre, NiVER, and SatELite.
The Impact of Preprocessor

Preprocessing Time

![Graph showing the impact of preprocessing time on various problems](image)

- 3-Resolution
- 2-SIMPLIFY
- HyPre
- NiVER
- SatELite
Random-Walk:
- AdaptNovelty$^+$ [Hoos, AAAI-2002]: enhancing Novelty$^+$ with adaptive noise mechanism.
- g2wsat [Li and Huang, SAT-2005]: deterministically picks the best promising decreasing variable to flip.

Clause Weighting:
- PAWS$_{10}$: PAWS [Thornton et al., AAAI-2004] with smooth parameter fixed to 10
- RSAPS: reactive version of SAPS [Hutter et al., CP-2002]
Empirical Study

- 12 classes of problems: random, quasigroup, real-world
  - 64 problem instances
- 5 resolution-based preprocessors
- 4 SLS solvers: random walk vs. clause weighting
- The total of 153,600 runs
  - 100 runs for each instance
  - 128,000 runs on preprocessed instances
  - 25,600 runs on original instances
- Time limit for each run is 1200 seconds for random, ferry, and cryptographic problems and 600 seconds for the other ones.
- On Linux Pentium IV computer with 3.00GHz CPU and 1GB RAM
### Empirical Results

<table>
<thead>
<tr>
<th>Instances</th>
<th>Prep.</th>
<th>#Vars/#Cls/#Lits</th>
<th>Ptime</th>
<th>g2wsat</th>
<th>AdaptNovelty+</th>
<th>PAWS&lt;sub&gt;10&lt;/sub&gt;</th>
<th>RSAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%</td>
<td>%</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stime</td>
<td>Stime</td>
<td>Stime</td>
<td></td>
</tr>
<tr>
<td>ais16</td>
<td>origin</td>
<td>481/10621/25261</td>
<td>n/a</td>
<td>1</td>
<td>596.01</td>
<td>2</td>
<td>591.38</td>
</tr>
<tr>
<td></td>
<td>2-SIM</td>
<td>481/10621/25261</td>
<td>0.06</td>
<td>10</td>
<td>558.64</td>
<td>3</td>
<td>593.65</td>
</tr>
<tr>
<td></td>
<td>HyPre</td>
<td>481/10621/25261</td>
<td>0.04</td>
<td>5</td>
<td>583.78</td>
<td>1</td>
<td>596.47</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>465/10365/34140</td>
<td>0.17</td>
<td>96</td>
<td>147.93</td>
<td>100</td>
<td>50.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw_large_d</td>
<td>origin</td>
<td>6325/131973/294118</td>
<td>n/a</td>
<td>0</td>
<td>600.00</td>
<td>99</td>
<td>165.36</td>
</tr>
<tr>
<td></td>
<td>3-Res</td>
<td>6325/141421/322458</td>
<td>0.85</td>
<td>0</td>
<td>600.00</td>
<td>88</td>
<td>244.74</td>
</tr>
<tr>
<td></td>
<td>2-SIM</td>
<td>4644/82453/183302</td>
<td>1.57</td>
<td>0</td>
<td>600.00</td>
<td>37</td>
<td>481.55</td>
</tr>
<tr>
<td></td>
<td>HyPre</td>
<td>3572/86285/188327</td>
<td>15.08</td>
<td>51</td>
<td>410.81</td>
<td>75</td>
<td>338.64</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>6307/131653/303116</td>
<td>1.22</td>
<td>0</td>
<td>600.00</td>
<td>99</td>
<td>123.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ferry8-ks99a-4004</td>
<td>origin</td>
<td>1259/15259/31167</td>
<td>n/a</td>
<td>58</td>
<td>508.44</td>
<td>2</td>
<td>1197</td>
</tr>
<tr>
<td></td>
<td>3-Res</td>
<td>1241/15206/31071</td>
<td>0.11</td>
<td>64</td>
<td>435.55</td>
<td>12</td>
<td>1150</td>
</tr>
<tr>
<td></td>
<td>2-SIM</td>
<td>1233/14562/29783</td>
<td>0.00</td>
<td>56</td>
<td>531.99</td>
<td>26</td>
<td>1046</td>
</tr>
<tr>
<td></td>
<td>HyPre</td>
<td>1209/20960/42471</td>
<td>0.13</td>
<td>30</td>
<td>947.43</td>
<td>8</td>
<td>1136</td>
</tr>
<tr>
<td></td>
<td>NiVER</td>
<td>976/14952/30817</td>
<td>0.00</td>
<td>58</td>
<td>510.05</td>
<td>40</td>
<td>937.56</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>813/14720/34687</td>
<td>0.19</td>
<td>84</td>
<td>210.65</td>
<td>100</td>
<td>209.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>par16-1</td>
<td>origin</td>
<td>1015/3310/8788</td>
<td>n/a</td>
<td>12</td>
<td>534.30</td>
<td>35</td>
<td>479.82</td>
</tr>
<tr>
<td></td>
<td>3-Res</td>
<td>607/1815/4713</td>
<td>0.04</td>
<td>9</td>
<td>548.57</td>
<td>99</td>
<td>145.44</td>
</tr>
<tr>
<td></td>
<td>2-SIM</td>
<td>317/1266/3652</td>
<td>0.03</td>
<td>74</td>
<td>312.23</td>
<td>100</td>
<td>54.26</td>
</tr>
<tr>
<td></td>
<td>HyPre</td>
<td>317/1324/3790</td>
<td>0.01</td>
<td>63</td>
<td>331.39</td>
<td>100</td>
<td>32.47</td>
</tr>
<tr>
<td></td>
<td>NiVER</td>
<td>632/2512/7058</td>
<td>0.02</td>
<td>81</td>
<td>200.98</td>
<td>97</td>
<td>153.91</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>201/1173/4121</td>
<td>0.05</td>
<td>100</td>
<td>22.99</td>
<td>100</td>
<td>17.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qg7-13</td>
<td>origin</td>
<td>2197/97072/256426</td>
<td>n/a</td>
<td>21</td>
<td>518.52</td>
<td>0</td>
<td>600.00</td>
</tr>
<tr>
<td></td>
<td>3-Res</td>
<td>141/45362/115164</td>
<td>2.06</td>
<td>27</td>
<td>511.52</td>
<td>49</td>
<td>381.89</td>
</tr>
<tr>
<td></td>
<td>2-SIM</td>
<td>1333/41647/106430</td>
<td>0.75</td>
<td>75</td>
<td>326.24</td>
<td>84</td>
<td>135.23</td>
</tr>
<tr>
<td></td>
<td>HyPre</td>
<td>1207/41110/105189</td>
<td>0.95</td>
<td>100</td>
<td>24.57</td>
<td>99</td>
<td>27.32</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>1412/45967/116374</td>
<td>0.35</td>
<td>17</td>
<td>541.08</td>
<td>55</td>
<td>339.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3sat-1648</td>
<td>origin</td>
<td>10000/42000/126000</td>
<td>n/a</td>
<td>68</td>
<td>744.99</td>
<td>16</td>
<td>1145</td>
</tr>
<tr>
<td></td>
<td>3-Res</td>
<td>10000/42081/126243</td>
<td>0.40</td>
<td>66</td>
<td>747.21</td>
<td>44</td>
<td>1022</td>
</tr>
<tr>
<td></td>
<td>NiVER</td>
<td>9982/41981/125993</td>
<td>0.43</td>
<td>78</td>
<td>600.19</td>
<td>30</td>
<td>1082</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>9775/41747/126601</td>
<td>0.66</td>
<td>70</td>
<td>672.08</td>
<td>22</td>
<td>1133</td>
</tr>
</tbody>
</table>
RTDs on Structured Problems
Multiple Preprocessing and Preprocessor Ordering

<table>
<thead>
<tr>
<th>Instances</th>
<th>Preprocessor</th>
<th>#Vars/#Cls/#Lits</th>
<th>Ptime</th>
<th>Succ. Rate</th>
<th>CPU Time</th>
<th>Flips</th>
</tr>
</thead>
<tbody>
<tr>
<td>ferry7-ks99i-</td>
<td>origin</td>
<td>1946/22336/45706</td>
<td>n/a</td>
<td>100</td>
<td>192.92</td>
<td>55,877,724</td>
</tr>
<tr>
<td>4001</td>
<td>SatELite</td>
<td>1286/21601/50644</td>
<td>0.27</td>
<td>100</td>
<td>4.39</td>
<td>897,165</td>
</tr>
<tr>
<td>HyPre</td>
<td>1881/32855/66732</td>
<td>0.19</td>
<td>100</td>
<td>2.34</td>
<td>3.26</td>
<td>494,122</td>
</tr>
<tr>
<td>HyPre &amp; Sat</td>
<td>1289/29078/76551</td>
<td>0.72</td>
<td>100</td>
<td>2.17</td>
<td>3.05</td>
<td>359,981</td>
</tr>
<tr>
<td>Sat &amp; HyPre</td>
<td>1272/61574/130202</td>
<td>0.59</td>
<td>100</td>
<td>0.83</td>
<td>1.17</td>
<td>83,224</td>
</tr>
<tr>
<td>ferry8-ks99i-</td>
<td>origin</td>
<td>2547/32525/66425</td>
<td>n/a</td>
<td>42</td>
<td>1,200.00</td>
<td>302,651,507</td>
</tr>
<tr>
<td>4005</td>
<td>SatELite</td>
<td>1696/31589/74007</td>
<td>0.41</td>
<td>100</td>
<td>44.96</td>
<td>7,563,160</td>
</tr>
<tr>
<td>HyPre</td>
<td>2473/48120/97601</td>
<td>0.29</td>
<td>100</td>
<td>9.50</td>
<td>19.61</td>
<td>1,629,417</td>
</tr>
<tr>
<td>HyPre &amp; Sat</td>
<td>1700/43296/116045</td>
<td>1.05</td>
<td>100</td>
<td>5.19</td>
<td>10.86</td>
<td>1,077,364</td>
</tr>
<tr>
<td>Sat &amp; HyPre</td>
<td>1680/92321/194966</td>
<td>0.90</td>
<td>100</td>
<td>2.23</td>
<td>3.62</td>
<td>252,778</td>
</tr>
<tr>
<td>par16-4</td>
<td>origin</td>
<td>1015/3324/8844</td>
<td>n/a</td>
<td>4</td>
<td>600.00</td>
<td>273,700,514</td>
</tr>
<tr>
<td>HyPre</td>
<td>324/1352/3874</td>
<td>0.01</td>
<td>100</td>
<td>10.14</td>
<td>13.42</td>
<td>5,230,084</td>
</tr>
<tr>
<td>SatELite</td>
<td>210/1201/4189</td>
<td>0.05</td>
<td>100</td>
<td>5.25</td>
<td>7.33</td>
<td>2,230,524</td>
</tr>
<tr>
<td>Sat &amp; HyPre</td>
<td>210/1210/4207</td>
<td>0.05</td>
<td>100</td>
<td>4.73</td>
<td>6.29</td>
<td>1,987,638</td>
</tr>
<tr>
<td>HyPre &amp; Sat</td>
<td>198/1232/4352</td>
<td>0.04</td>
<td>100</td>
<td>1.86</td>
<td>2.80</td>
<td>1,333,372</td>
</tr>
</tbody>
</table>

Table 3. RSAPS performance on ferry planning and par16-4 instances.
Boosting DPLL using Resolution-based Preprocessing
### Empirical Results on Parity and Planning

<table>
<thead>
<tr>
<th>Instance</th>
<th>Prep.</th>
<th>#Var/#Cls/#Lits</th>
<th>Ptime</th>
<th>Dew_Satz</th>
<th>MiniSat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stime</td>
<td>#BackT</td>
</tr>
<tr>
<td>par32-4</td>
<td>Orig</td>
<td>3176/10313/27645</td>
<td>n/a</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>3Res</td>
<td>2385/7433/19762</td>
<td>0.08</td>
<td>10,425</td>
<td>10,036,154</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>849/5160/18581</td>
<td>0.21</td>
<td>12,820</td>
<td>18,230,746</td>
</tr>
<tr>
<td></td>
<td>Hyp+3Res</td>
<td>1331/6055/16999</td>
<td>0.36</td>
<td>9,001</td>
<td>17,712,997</td>
</tr>
<tr>
<td></td>
<td>3Res+Hyp</td>
<td>1331/5567/16026</td>
<td>0.11</td>
<td>5,741</td>
<td>10,036,146</td>
</tr>
<tr>
<td></td>
<td>Niv+3Res</td>
<td>1333/5810/16503</td>
<td>0.34</td>
<td>6,099</td>
<td>10,036,154</td>
</tr>
<tr>
<td></td>
<td>3Res+Niv</td>
<td>1290/5297/15481</td>
<td>0.10</td>
<td>14,003</td>
<td>25,092,756</td>
</tr>
<tr>
<td></td>
<td>3Res+Sat</td>
<td>850/5286/18958</td>
<td>0.35</td>
<td>3,552</td>
<td>7,744,986</td>
</tr>
<tr>
<td></td>
<td>Sat+3Res</td>
<td>849/5333/19052</td>
<td>0.38</td>
<td>3,563</td>
<td>7,744,986</td>
</tr>
<tr>
<td></td>
<td>Sat+2Sim</td>
<td>848/5154/18565</td>
<td>0.26</td>
<td>12,862</td>
<td>18,230,746</td>
</tr>
</tbody>
</table>

| ferry10_ks99a | Orig  | 1977/29041/59135 | n/a   | >15000   | n/a     | 0.03  | 710        |
|               | 3Res  | 1955/28976/59017 | 0.13  | >15000   | n/a     | 0.03  | 827        |
|               | Hyp   | 1915/40743/82551 | 0.29  | >15000   | n/a     | 0.04  | 563        |
|               | Niv   | 1544/28578/58619 | 0.02  | >15000   | n/a     | 0.01  | 0          |
|               | Sat   | 1299/28246/66432 | 0.44  | >15000   | n/a     | 0.03  | 909        |
|               | 2Sim  | 1945/27992/57049 | 0.05  | >15000   | n/a     | 0.05  | 1,565      |
|               | Sat+2Sim | 1299/69894/149728 | 0.69  | 0.28     | 1       | 0.04  | 419        |
|               | 3Res+2Sim+Niv | 1793/21099/43369 | 0.43  | 0.08     | 0       | 0.06  | 1,278      |
|               | Niv+Hyp+2Sim+3Res | 1532/24524/50463 | 0.54  | 5.19     | 3,949   | 0.02  | 454        |
## Empirical Results on BMC

<table>
<thead>
<tr>
<th>Instance</th>
<th>Prep.</th>
<th>#Var/#Cls/#Lits</th>
<th>Ptime</th>
<th>Dew_Satz</th>
<th>MiniSat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stime</td>
<td>#BackT</td>
<td>Stime</td>
</tr>
<tr>
<td>BMC-IBM-12</td>
<td>Orig</td>
<td>39598/194778/515536</td>
<td>n/a</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Hyp</td>
<td>12205/87082/228241</td>
<td>92</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Niv</td>
<td>27813/168440/476976</td>
<td>0.69</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>3Res</td>
<td>32606/160555/419341</td>
<td>2.77</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>15176/109121/364968</td>
<td>4.50</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Niv+3Res</td>
<td>12001/100114/253071</td>
<td>86</td>
<td>106</td>
<td>6</td>
</tr>
<tr>
<td>BMC-alpha-25449</td>
<td>Orig</td>
<td>663443/3065529/7845396</td>
<td>n/a</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>12408/76025/247622</td>
<td>129</td>
<td>6.94</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Sat+Hyp</td>
<td>9091/61789/203593</td>
<td>566</td>
<td>7.82</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Sat+Niv</td>
<td>12356/75709/246367</td>
<td>130</td>
<td>4.48</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Sat+3Res</td>
<td>12404/77805/249192</td>
<td>130</td>
<td>8.84</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Sat+2Sim</td>
<td>10457/71128/229499</td>
<td>131</td>
<td>6.37</td>
<td>10</td>
</tr>
<tr>
<td>BMC-alpha-4408</td>
<td>Orig</td>
<td>1080015/3054591/7395935</td>
<td>n/a</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>23657/112343/364874</td>
<td>47</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat+Hyp</td>
<td>13235/88976/263053</td>
<td>56</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat+Niv</td>
<td>22983/108603/351369</td>
<td>49</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat+3Res</td>
<td>23657/117795/380389</td>
<td>48</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat+2Sim</td>
<td>17470/129245/375444</td>
<td>52</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Sat+2Sim+3Res</td>
<td>16837/98726/305057</td>
<td>53</td>
<td>&gt;15000</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Empirical Results on FPGA Routing

<table>
<thead>
<tr>
<th>Instance</th>
<th>Dew_Satz</th>
<th>MiniSat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Solved</td>
<td>Stime</td>
</tr>
<tr>
<td>bart (21 SAT)</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td>homer (15 UNSAT)</td>
<td>15</td>
<td>2,662</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>Prep.</th>
<th>#Var/#Cls/#Lits</th>
<th>Ptime</th>
<th>Dew_Satz</th>
<th>MiniSat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stime</td>
<td>#BackT</td>
<td>Stime</td>
</tr>
<tr>
<td>bart-28</td>
<td>Orig</td>
<td>428/2907/7929</td>
<td>n/a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>413/2892/11469</td>
<td>0.06</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sha</td>
<td>1825/8407/27003</td>
<td>0.37</td>
<td>0.06</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Sha+3Res</td>
<td>1764/7702/24400</td>
<td>0.46</td>
<td>0.04</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Sha+Hyp</td>
<td>1764/8349/26138</td>
<td>0.41</td>
<td>0.05</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Sha+Niv</td>
<td>1781/8358/26759</td>
<td>0.38</td>
<td>0.05</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Sha+Sat</td>
<td>1728/8254/30422</td>
<td>0.53</td>
<td>0.10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sha+2Sim</td>
<td>1750/7892/24682</td>
<td>0.39</td>
<td>0.05</td>
<td>17</td>
</tr>
<tr>
<td>homer-20</td>
<td>Orig</td>
<td>440/4220/8800</td>
<td>n/a</td>
<td>941</td>
<td>19,958,400</td>
</tr>
<tr>
<td></td>
<td>Sat</td>
<td>400/4180/15200</td>
<td>0.08</td>
<td>1,443</td>
<td>6,982,425</td>
</tr>
<tr>
<td></td>
<td>Sha</td>
<td>1999/10340/29988</td>
<td>0.28</td>
<td>369</td>
<td>350,610</td>
</tr>
<tr>
<td></td>
<td>Sha+3Res</td>
<td>1907/8793/25027</td>
<td>0.37</td>
<td>362</td>
<td>405,059</td>
</tr>
<tr>
<td></td>
<td>Sha+Hyp</td>
<td>1905/10527/29129</td>
<td>0.34</td>
<td>1,306</td>
<td>1,451,567</td>
</tr>
<tr>
<td></td>
<td>Sha+Niv</td>
<td>1941/10276/29671</td>
<td>0.29</td>
<td>379</td>
<td>349,842</td>
</tr>
<tr>
<td></td>
<td>Sha+Sat</td>
<td>1723/9420/30986</td>
<td>0.54</td>
<td>822</td>
<td>300,605</td>
</tr>
<tr>
<td></td>
<td>Sha+2Sim</td>
<td>1879/9419/26188</td>
<td>0.31</td>
<td>114</td>
<td>120,297</td>
</tr>
</tbody>
</table>
## Using Dew_Satz

### Multiple Preprocessing and Preprocessor Ordering

<table>
<thead>
<tr>
<th>Instance</th>
<th>Prep.</th>
<th>#Var/#Cls/#Lits</th>
<th>Ptime</th>
<th>Stime</th>
<th>#BackT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BMC-IBM-12</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hyp+3Res+Niv</td>
<td>10805/83643/204679</td>
<td>96.11</td>
<td>&gt;15,000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Niv+Hyp+3Res</td>
<td>12001/100114/253071</td>
<td>85.81</td>
<td>106</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>3Res+Hyp+Niv</td>
<td>10038/82632/221890</td>
<td>89.56</td>
<td>&gt;15,000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>3Res+Niv+Hyp</td>
<td>11107/99673/269405</td>
<td>58.38</td>
<td>&gt;15,000</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>ferry10_ks99a</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2Sim+Niv+Hyp+3Res</td>
<td>1518/32206/65806</td>
<td>0.43</td>
<td>&gt;15,000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Niv+3Res+2Sim+Hyp</td>
<td>1532/25229/51873</td>
<td>0.49</td>
<td>11,345</td>
<td>17,778,483</td>
</tr>
<tr>
<td></td>
<td>3Res+2Sim+Niv+Hyp</td>
<td>1793/20597/42365</td>
<td>0.56</td>
<td>907</td>
<td>1,172,964</td>
</tr>
<tr>
<td></td>
<td>Niv+Hyp+2Sim+3Res</td>
<td>1532/24524/50463</td>
<td>0.54</td>
<td>5</td>
<td>3,949</td>
</tr>
<tr>
<td><strong>ferry10_ks99a</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2Sim+Niv</td>
<td>1518/27554/56565</td>
<td>0.08</td>
<td>&gt;15,000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>2Sim+Niv+2Sim</td>
<td>1518/18988/39433</td>
<td>0.27</td>
<td>3,197</td>
<td>6,066,241</td>
</tr>
<tr>
<td></td>
<td>2Sim+Niv+  2Sim+Niv</td>
<td>1486/18956/39429</td>
<td>0.29</td>
<td>129</td>
<td>290,871</td>
</tr>
<tr>
<td></td>
<td>2Sim+Niv+  2Sim+Niv+2Sim</td>
<td>1486/23258/48033</td>
<td>0.48</td>
<td>7,355</td>
<td>8,216,100</td>
</tr>
</tbody>
</table>
Preprocessing + DPLL
Conclusion: on SAT Algorithms

- Complete method
  - LA-based DPLL
  - CDCL-based DPLL

- Incomplete method (SLS)
  - Random Walk
  - DLS (clause weighting SLS)
  - Resolution+SLS

- Resolution + SLS
Conclusion: on Complete SAT Algorithms

3 classes of SAT solvers in terms of their capability for solving problems:

- **High performance on random problems**
  - Kcnfs
  - Based on look-ahead

- **High performance on most of structured problems**
  - zChaff, Jerusat, Berkmin, Siege, MiniSat, Tinisat, etc…
  - Based on CDCL

- **Good performance on random & high performance on some classes of structured problems**
  - Satz, Dew_Satz, march_dl, etc…
  - Based on look-ahead
### Conclusion: SAT Algorithms Performance Comparison

<table>
<thead>
<tr>
<th>Algo</th>
<th>Complete SAT solvers</th>
<th>Incomplete SAT solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob</td>
<td></td>
<td></td>
</tr>
<tr>
<td>random</td>
<td>DPLL</td>
<td>Random Walk</td>
</tr>
<tr>
<td></td>
<td>Can reach 700 vars</td>
<td>10000 vars with 68% sr (g2wsat)</td>
</tr>
<tr>
<td></td>
<td>10 - 40 times slower than DPLL</td>
<td>10000 vars with 0% sr (RSAPS)</td>
</tr>
<tr>
<td>structured real-world</td>
<td>better than CDCL on 30%</td>
<td>better than CW on 30%</td>
</tr>
<tr>
<td></td>
<td>better than DPLL on 70%</td>
<td>better than RW on 70%</td>
</tr>
</tbody>
</table>

R+DDFW$^+$ is the current best SLS solver for both random and realistic problems
Questions........