SAT Solving

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

NICTA and Australian National University

Presented at COMP4360/6463 Overview of Logic and Computation course, ANU/NICTA, March-April 2009
Roadmap

• SAT Resources
• Introduction to SAT
  • Real-world problems
  • Basic notation and definitions
  • Problems in CNF formula
  • Phase transition phenomena
• Algorithms for SAT solving
  • DP and DPLL procedure
  • Complete methods: Look-ahead based DPLL and CL-based DPLL
  • Stochastic methods: GSAT, Random Walk, Clause Weighting SLS
  • SAT preprocessors
• 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
- 1657 problems used: random, crafted and industrial
- 2 stages competition: (20 mins in 1st stage, 100 or 200 mins in 2nd stage)
- Announcement at 2005 International Conference SAT Conference
- No competition in 2006 and 2008 but there is a SAT-Race for solving only the industrial problems.

- Next competition is in 2009
• 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**

```
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.
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

Constraint:
In chess, a queen can move horizontally, vertically, or diagonally.
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. ($X=T, Y=T, Z=\bot$)
- He could only take Algorithms. ($X=\bot, Y=\bot, Z=T$)
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>( \bot )</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>( \neg x, \sim x )</td>
</tr>
<tr>
<td>( x ) and ( y )</td>
<td>( x \land y )</td>
<td>( xy )</td>
<td>( x \land y, x \cdot y )</td>
</tr>
<tr>
<td>( x ) or ( y )</td>
<td>( x \lor y )</td>
<td>( x + y )</td>
<td>( x</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>$\bot$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$T$</td>
<td>$T$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$T$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$\bot$</td>
<td>$\bot$</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

- **Variable**: can take a value true or false
- **Literal**: if \( x \) is a variable, then \( x \) and \( \neg x \) are literals (respectively positive and negative)
- **Clause**: a disjunction \( l_1 \lor \ldots \lor l_n \) of literals \( l_1, \ldots, l_n \)
- **Formula** \( F \): a conjunction \( c_1 \lor \ldots \lor c_m \) of clauses \( c_1, \ldots, c_m \)
- **Unit Clause**: a clause consisting of only one literal
- **Binary Clause**: a clause consisting of two literals
- **Empty Clause**: a clause without any literal
- **Pure Literal**: a variable occurring only negatively or only 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 $\mathcal{F}$ in Conjunctive Normal Form (CNF)

Output: $\mathcal{F}$ is satisfiable by an assignment of truth values to variables or $\mathcal{F}$ is unsatisfiable.

Example of a CNF formula:

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

The first NP-Complete problem [Cook, 1971]

A central problem in mathematical logic, AI, and other fields of computer science and engineering.
SAT Problems - Random Versus Structured

Random problems:
- Used for testing SAT algorithms and NP-completeness
- Several models exist: constant probability, fixed clause length [Mitchell et al., 1992]
- Syntactic problem parameters determine difficulty
- Formulas have characteristics quite different from the kind of very big formulas coming from practical applications and that can be solved by current SAT algorithms.

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

Random+Structured problems:
- QWH = quasigroup with holes
- bQWH = balanced quasigroup with holes
A random 3-SAT problem with 64 variables and 254 clauses

- the fixed clause length model [Mitchell et al., 1992]:
  - Fix a set of $n$ variables $x_1, \ldots, x_n$
  - Generate $m$ clauses with 3 literals: randomly choose a variable and negate with probability 0.5
Structured Problem: par8-1.cnf

c par8-1-res
p cnf 350 382
  -2  3  146  0
  2  3  -146  0
  -2 -3  146  0
  -2 -3 -146  0
  -3  4  147  0
  3  4 -147  0
  3 -4  147  0
  -3 -4 -147  0
  -5  4  0
  5  -4  0
  -5  6  0
  5  -6  0
  ....
  ....
Structured Problem: Bounded Model Checking

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

c 6.6 MB data

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

........

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

```
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
```

**Note that:** $2^{51639}$ 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
- $r = \text{the ratio of number of clauses and of variables (clause density)}.$
- As $r$ is increased, probability of being SAT goes abruptly from 1 to 0
- $r\alpha = \text{critical value for formula with fixed clause lengths } \alpha.$
- $r^2 = 1; \quad r^3 \approx 4.258;$

  The critical value divides the space of SAT problems into 3 regions:
  - Underconstrained: almost all the formulas are satisfiable and easy to solve.
  - Critically constrained: about 50 per cent of the formulas are satisfiable and hard to solve.
  - Overconstrained: almost all the formulas are unsatisfiable and easy to solve.
Phase Transition: n=60:20:160

The imagination driving Australia's ICT future
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 CL: 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**
The Resolution Rule

Resolution

\[ \frac{l \lor \phi \quad \bar{l} \lor \phi'}{\phi \lor \phi'} \]

One of \( l \) and \( \bar{l} \) is false.
Hence at least one of \( \phi \) and \( \phi' \) is true.
• 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 known 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.
Procedure DP($\mathcal{F}$)

for $i = 1$ to NumberofVariableIn($\mathcal{F}$) do

choose a variable $x$ occurring in $\mathcal{F}$

Resolvents := $\emptyset$

for all $(C_1,C_2)$ s.t. $C_1 \in \mathcal{F}$, $C_2 \in \mathcal{F}$, $x \in C_1$, $\neg x \in C_2$ do

  // don’t generate tautological resolvents.
  Resolvents = Resolvents $\cup$ resolve($C_1,C_2$)

  // $x$ is not in the current $\mathcal{F}$.

$\mathcal{F} := \{C \in \mathcal{F} | x$ does not occur in $C\} \cup$ Resolvents

if $\mathcal{F} := \emptyset$ then

  return UNSATISFIABLE

else

  return SATISFIABLE
DP Resolution for SAT example

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

\[\Downarrow\]

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

\[\Downarrow\]

\[(x_1 \lor x_5 \lor \neg x_6)\]

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

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

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

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

\[\downarrow\]

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

\[\downarrow\]

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

\[\downarrow\]

\[\emptyset\]

\[\Rightarrow\text{ UNSAT}\]
• **Branching variable**: a variable chosen for case analysis true/false

• **Free variable**: a variable with no value yet

• **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 choice-point in order to try another choice.
Unit Resolution

• Unrestricted application of the resolution rule is too expensive.

• Unit resolution restricts one of the clauses to be a unit clause consisting of only one literal.

• Performing all possible unit resolution steps on a clause set can be done in linear time.
Unit Propagation

Unit Resolution

\[
\frac{1}{\bar{1} \lor \phi} \quad \phi
\]

Unit Propagation algorithm $UP(\mathcal{F})$ for clause sets $\mathcal{F}$

1. If there is a unit clause $1 \in \mathcal{F}$, then replace every $\bar{1} \lor \phi \in \mathcal{F}$ by $\phi$ and remove all clauses containing $1$ from $\mathcal{F}$. As a special case the empty clause $\bot$ may be obtained.
2. If $\mathcal{F}$ still contains a unit clause, repeat step 1.
3. Return $\mathcal{F}$.

We sometimes write $\mathcal{F} \vdash_{UP} l$ if $l \in UP(\mathcal{F})$. 

The imagination driving Australia’s ICT future
Procedure DPLL($\mathcal{F}$)

(Sat) if $\mathcal{F} = \emptyset$ then return \texttt{true};

(Empty) if $\mathcal{F}$ contains the empty clause then return \texttt{false};

(UP) if $\mathcal{F}$ has unit clause $\{u\}$, then DPLL(UP($\mathcal{F} \cup \{u\}$));

(Pure) if $\mathcal{F}$ has pure literal $p$, then DPLL(UP($\mathcal{F} \cup \{p\}$));

(Split) choose a variable $x$;

if DPLL(UP($\mathcal{F} \cup \{x\}$)) = \texttt{true} then return \texttt{true}
else return DPLL(UP($\mathcal{F} \cup \{\neg x\}$));
Search Tree of DPLL Procedure

- Binary Search Tree
- Large search tree size $\Leftrightarrow$ Hard problem
- Depth first search with backtracking
DPLL Tree of Student-Courses Problem

(X ∨ ¬Y) & (Y ∨ Z) & (¬X ∨ ¬Z)

(Y ∨ Z) & (¬Z)

(Y) ∨ (Z)

Using DPLL Procedure

The imagination driving Australia’s ICT future
DPLL Performance: Original vs. Variants

- On hard random 3-SAT problems at \( r = 4.25 \)
- The worst case complexity of the algorithm in our experience is \( O(2^{n/21.83-1.70}) \), based on UNSAT problems.
- This is a big improvement!

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

<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>
<tr>
<td>Opt_Satz (2003):</td>
<td>( O(2^{n/21.83-1.70}) )</td>
<td>7</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.
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 $c$. 
Look-ahead based DPLL
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]
Unit Propagation Look-ahead (UPLA)

- Heuristics like MOMS choose branching variables based on properties of the occurrences of literals.
- What if one could look ahead what the consequences of choosing a certain branch variable are?
- [Freeman, 1995; Crawford & Auton, 1996; Li & Anbulagan, 1997a]

Unit Propagation Based Look-Ahead (UPLA)

1. Set a literal \( l \) true and perform unit propagation:
   \[ \mathcal{F}' = UP(\mathcal{F} \cup \{l\}). \]
2. (If the empty clause is obtained, see the next slide.)
3. Compute a heuristic value for \( \mathcal{F}' \).

Choose a literal with the highest value.
Unit Propagation Look-ahead (UPLA)
 Failed Literals

UPLA for some literals may lead to the empty clause.

**Lemma**

\[ \text{If } \mathcal{F} \cup \{l\} \vdash_{UP} \bot, \text{ then } \mathcal{F} \models \bar{l}. \]

Here \( l \) is a failed literal.
Failed literals may be set false: \( \mathcal{F} := UP(\mathcal{F} \cup \{\bar{l}\}). \)
After setting a literal $l$ true and performing UP, calculate the weight of $l$: $w(l) = \text{diff}(\mathcal{F}, \text{UP}(\mathcal{F} \cup \{l\})) = \text{the number of clauses of minimal size in UP}(\mathcal{F} \cup \{l\})$ but not in $\mathcal{F}$.

A literal has a high weight if setting it true produces many clauses of minimal size (typically: clauses with 2 literals).

For branching choose a variable of maximal weight $w(x) \cdot w(\neg x) + w(x) + w(\neg x)$. 
Unit Propagation Look-ahead (UPLA)
Restricted UPLA

- Heuristics based on UPLA are often much more informative than simpler ones like MOMS.
- But doing UPLA for every literal is very expensive.
- Li and Anbulagan [1997a] propose the use of predicates PROP for selecting a small subset of the literals for UPLA.
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.

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-based Branching Variable Selection

Algorithm 1 LA-BranchingRule(\(F\))

1: for each variable \(x_i \in V\) do
2:   \(F'_i\) and \(F''_i\) be two copies of \(F\);
3:   \(F'_i := \text{UP}(F'_i \cup \{x_i\})\);
4:   \(F''_i := \text{UP}(F''_i \cup \{\overline{x_i}\})\);
5: if empty clause \(\in F'_i\) and empty clause \(\in F''_i\) then
6:     return "unsatisfiable";
7: else if empty clause \(\in F'_i\) then
8:     \(F := F''_i\);
9: else if empty clause \(\in F''_i\) then
10: \(F := F'_i\);
11: else
12: \(w(x_i) := \text{diff}(F'_i, F)\);
13: \(w(\overline{x_i}) := \text{diff}(F''_i, F)\);
14: \(W(x_i) := w(x_i) \ast w(\overline{x_i}) + w(x_i) + w(\overline{x_i})\);
15: end if
16: end for
17: return \(x_i\) with highest \(W(x_i)\) 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: \quad $\mathcal{F}_{init} := \mathcal{F}$;
4: \quad for each variable $x_i \in \text{NVO STACK}$ do
5: \quad \quad Let $\mathcal{F}_i'$ and $\mathcal{F}_i''$ be two copies of $\mathcal{F}$;
6: \quad \quad $\mathcal{F}_i' := \text{UP}(\mathcal{F}_i' \cup \{x_i\})$;
7: \quad \quad $\mathcal{F}_i'' := \text{UP}(\mathcal{F}_i'' \cup \{\bar{x}_i\})$;
8: \quad \quad if empty clause $\in \mathcal{F}_i'$ and empty clause $\in \mathcal{F}_i''$ then
9: \quad \quad \quad return "unsatisfiable";
10: \quad \quad else if empty clause $\in \mathcal{F}_i'$ then
11: \quad \quad \quad $\mathcal{F} := \mathcal{F}_i''$;
12: \quad \quad \quad NVO($x_i$);
13: \quad \quad else if empty clause $\in \mathcal{F}_i''$ then
14: \quad \quad \quad $\mathcal{F} := \mathcal{F}_i'$;
15: \quad \quad \quad NVO($x_i$);
16: \quad \quad else
17: \quad \quad \quad $w(x_i) := \text{diff}(\mathcal{F}_i', \mathcal{F})$;
18: \quad \quad \quad $w(\bar{x}_i) := \text{diff}(\mathcal{F}_i'', \mathcal{F})$;
19: \quad \quad \quad $\mathcal{W}(x_i) := w(x_i) \cdot w(\bar{x}_i) + w(x_i) + w(\bar{x}_i)$;
20: \quad \quad \quad end if
21: \quad end for
22: until $\mathcal{F} = \mathcal{F}_{init}$
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>Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>satz215</td>
</tr>
<tr>
<td>b1-k1.1</td>
<td>21536</td>
<td>8966</td>
<td>2008485</td>
<td>1265</td>
<td>3551</td>
<td>2966</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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

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

Find the backbone variable(s)!

The answer is \(x_4\)
Comparison Results

The imagination driving Australia’s ICT future
The basic idea of integrating DEW (dynamic equivalency weighting):

Whenever the binary equivalency clause \((x_i \leftrightarrow x_j)\), 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 \(x_j\) (\(\neg x_j\)'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.
• 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 \(x_i\).
On 32-bit Parity Learning problem

A challenging problem [Selman et al., 1997]

EqSatz is the first solver which solved all the instances

Lsat and March_eq perform equivalency reasoning at pre-search phase.

<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;36h</td>
<td>12,918</td>
<td>242</td>
<td>126</td>
<td>0.22</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-2 (3176/10253)</td>
<td>&gt;36h</td>
<td>5,804</td>
<td>69</td>
<td>60</td>
<td>0.27</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-3 (3176/10297)</td>
<td>&gt;36h</td>
<td>7,198</td>
<td>2,863</td>
<td>183</td>
<td>2.89</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-4 (3176/10313)</td>
<td>&gt;36h</td>
<td>11,005</td>
<td>209</td>
<td>86</td>
<td>1.64</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-5 (3176/10325)</td>
<td>&gt;36h</td>
<td>17,564</td>
<td>2,639</td>
<td>418</td>
<td>8.07</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-1-c (1315/5254)</td>
<td>&gt;36h</td>
<td>10,990</td>
<td>335</td>
<td>270</td>
<td>2.63</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-2-c (1303/5206)</td>
<td>&gt;36h</td>
<td>411</td>
<td>13</td>
<td>16</td>
<td>2.19</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-3-c (1325/5294)</td>
<td>&gt;36h</td>
<td>4,474</td>
<td>1,220</td>
<td>374</td>
<td>6.65</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-4-c (1333/5326)</td>
<td>&gt;36h</td>
<td>7,090</td>
<td>202</td>
<td>115</td>
<td>0.45</td>
<td>&gt;36h</td>
</tr>
<tr>
<td>par32-5-c (1339/5350)</td>
<td>&gt;36h</td>
<td>11,899</td>
<td>2,896</td>
<td>97</td>
<td>6.44</td>
<td>&gt;36h</td>
</tr>
</tbody>
</table>
## Equivalency: Reasoning vs. Weighting

### Empirical Results (2/2)

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

- Dew_Satz won 2 bronze medals for UNSAT and ALL categories.
Clause learning (CL) 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

\[
\begin{align*}
&\neg x_1 \lor x_2) \land \\
&\neg x_1 \lor x_3 \lor x_9) \land \\
&\neg x_2 \lor x_3 \lor x_4) \land \\
&\neg x_4 \lor x_5 \lor x_10) \land \\
&\neg x_4 \lor x_6 \lor x_11) \land \\
&\neg x_5 \lor x_6) \land \\
&(x_1 \lor x_7 \lor x_12) \land \\
&(x_1 \lor x_8) \land \\
&\neg x_7 \lor x_8 \lor x_13 \land
\end{align*}
\]

......
Backjumping: an example

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

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

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

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

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

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

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

Conflict set: \{-x_9, -x_{10}, -x_{11}, x_1\} \Rightarrow \text{backtrack to } x_1
Backjumping: an example

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

\(
\{\ldots, \neg x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}, \ldots, \neg x_1\} \text{ (branch on } \neg x_1) \\
\text{(unit } x_7, x_8\text{) }
\)
Backjumping: an example

\[ \neg x_1 \lor x_2 \land \neg x_1 \lor x_3 \lor x_9 \land \neg x_2 \lor \neg x_3 \lor x_4 \land \neg x_4 \lor x_5 \lor x_{10} \land \neg x_4 \lor x_6 \lor x_{11} \land \neg x_5 \lor \neg x_6 \land x_1 \lor x_7 \lor \neg x_{12} \land x_1 \lor x_8 \land \neg x_7 \lor \neg x_8 \lor \neg x_{13} \land \neg x_1 \lor x_12 \lor x_{13} \lor \neg x_1 \lor x_7 \lor x_8 \]

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

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

Conflict set: \{x_{12}, x_{13}, \neg x_1\} \lor \{-x_9, \neg x_{10}, \neg x_{11}, x_1\}

\[\Rightarrow \{-x_9, \neg x_{10}, \neg x_{11}, x_{12}, x_{13}\} \Rightarrow \text{backtrack to } x_{13}\]
Look-Ahead vs. Look-Back

C. M. Li & Anbulagan, CP’97

(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

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

\[ \ldots \]
\[ (x_9 \lor x_{10} \lor x_{11} \lor \neg x_1) \land \text{ learned clause} \]

Conflict set: \{\neg x_9, \neg x_{10}, \neg x_{11}, x_1\}
Learn: \( x_9 \lor x_{10} \lor x_{11} \lor \neg x_1 \)
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]
Results of SAT-Race 2006
SLS Algorithms
Local Optima and Global Optimum in SLS

- Objective function value vs. solution space
- Local optima: multiple peaks and valleys
- Global optimum: the lowest point in the solution space
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($F$)

$T = \text{random (} F \text{)}$  // 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
    $T$ = 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
AdaptNovelty$^+$

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

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

If Found

Distribute Weights

Yes

No

If Flips > Literals

No

Yes

Reset Weights

\( W_{\text{SAT}} = 2 \)

\( W_{\text{UNSAT}} = 3 \)

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)

Distribute Weights

If Found

Yes

flip

No

If Flips > Literals

Yes

No

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

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]

- 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 $F$ do
2:   for each literal $l$ of $c_1$ do
3:     for each clause $c_2$ of length $\leq 3$ in $F$ s.t. $\overline{l} \in c_2$ do
4:       Compute resolvent $r = (c_1 - \{l\}) \cup (c_2 - \{\overline{l}\})$;
5:       if $r$ is empty then
6:         return "unsatisfiable";
7:       else
8:       if $r$ is of length $\leq 3$ then
9:         $F := F \cup \{r\}$;
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$^{nd}$ and 3$^{rd}$ place are g2wsat (178) and VW (170).
- The 2004 winner, AdaptNovelty$^+$ could only solve 119 problems
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.
  - Self-subsuming resolution: $C_1 = (x, a, b)$ and $C_2 = (-x, a)$, resolving on $x$ will produce $R = (a, b)$
NiVER CNF Preprocessor

Algorithm 1 NiVER CNF Preprocessor
1: NiVER(F)
2: repeat
3:   entry = FALSE
4:   for all $V \in \text{Var}(F)$ do
5:     $P_C = \{ C \mid C \in F, \overline{V} \in C \}$
6:     $N_C = \{ C \mid C \in F, \overline{V} \in C \}$
7:     $R = \{ \}$
8:     for all $P \in P_C$ do
9:       for all $N \in N_C$ do
10:          $R = R \cup \text{Resolve}(P,N)$
11:     Old_Num_Lits = Number of Literals in $(P_C \cup N_C)$
12:     New_Num_Lits = Number of Literals in $R$
13:     if (Old_Num_Lits $\geq$ New_Num_Lits) then
14:        $F = F - (P_C \cup N_C)$, $F = F + R$, entry = TRUE
15:     end if
16:   end for
17: end for
18: until $\neg entry$
19: return $F$

Clauses with literal \( l_{44} \)
\[
(l_{44} + l_{6315} + \overline{l}_{15605}) \\
(l_{44} + l_{6192} + \overline{l}_{6315}) \\
(l_{44} + \overline{l}_{3951} + \overline{l}_{11794})
\]

Clauses with literal \( \overline{l}_{44} \)
\[
(\overline{l}_{44} + l_{6315} + l_{15605}) \\
(\overline{l}_{44} + l_{6192} + \overline{l}_{6315}) \\
(\overline{l}_{44} + \overline{l}_{3951} + l_{11794})
\]

Old_Num_Lits = 18

Number of Clauses deleted = 6

Added Resolvents
\[
(l_{6315} + \overline{l}_{15605} + \overline{l}_{3951} + l_{11794}) \\
(l_{6192} + \overline{l}_{6315} + \overline{l}_{3951} + l_{11794}) \\
(\overline{l}_{3951} + l_{11794} + l_{6315} + l_{15605}) \\
(\overline{l}_{3951} + \overline{l}_{11794} + l_{6192} + \overline{l}_{6315})
\]

Discarded Resolvents (Tautologies)
\[
(l_{6315} + \overline{l}_{15605} + l_{6315} + l_{15605}) \\
(l_{6315} + \overline{l}_{15605} + \overline{l}_{6192} + \overline{l}_{6315}) \\
(l_{6192} + \overline{l}_{6315} + l_{6315} + l_{15605}) \\
(l_{6192} + \overline{l}_{6315} + \overline{l}_{6192} + \overline{l}_{6315}) \\
(\overline{l}_{3951} + l_{11794} + \overline{l}_{3951} + l_{11794})
\]

New_Num_Lits = 16

Number of Clauses added = 4

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 (vmpc), 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 variables reduction is illustrated in the graph. The x-axis represents different problems (3sat, ais, bmc, bw, edd, fer, log, par, qg, ssa, vmpc, vpn), while the y-axis shows the reduction ratio. The graph compares various preprocessor methods, including 3-Resolution, 2-SIMPLIFY, HyPre, NiVER, and SatELite.
The Impact of Preprocessor

Clauses Reduction

The imagination driving Australia’s ICT future
The Impact of Preprocessor

Literals Reduction

The imagination driving Australia’s ICT future
The Impact of Preprocessor

Preprocessing Time

![Graph showing the impact of preprocessing time for different problems and tools.](image)
SLS Solvers

- 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
<table>
<thead>
<tr>
<th>Instances</th>
<th>Prep.</th>
<th>#Vars/#Cls/#Lits</th>
<th>Ptime</th>
<th>g2wsat</th>
<th>AdaptNovelty⁺</th>
<th>PAWS₁₀</th>
<th>RSAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>%</td>
<td>Stime</td>
<td>%</td>
</tr>
<tr>
<td>as16</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>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>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>119.7</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>115.0</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>104.6</td>
</tr>
<tr>
<td></td>
<td>HyPre</td>
<td>1209/20906/42471</td>
<td>0.13</td>
<td>30</td>
<td>947.43</td>
<td>8</td>
<td>113.6</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>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>548.27</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>632/2521/7058</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>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>1412/5362/115164</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>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>114.5</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
<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></td>
<td>HyPre</td>
<td>1881/32855/66732</td>
<td>0.19</td>
<td>100</td>
<td>2.34</td>
<td>494,122</td>
</tr>
<tr>
<td></td>
<td>HyPre &amp; Sat</td>
<td>1289/29078/76551</td>
<td>0.72</td>
<td>100</td>
<td>2.17</td>
<td>359,981</td>
</tr>
<tr>
<td></td>
<td>Sat &amp; HyPre</td>
<td>1272/61574/130202</td>
<td>0.59</td>
<td>100</td>
<td>0.83</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></td>
<td>HyPre</td>
<td>2473/48120/97601</td>
<td>0.29</td>
<td>100</td>
<td>9.50</td>
<td>1,629,417</td>
</tr>
<tr>
<td></td>
<td>HyPre &amp; Sat</td>
<td>1700/43966/116045</td>
<td>1.05</td>
<td>100</td>
<td>5.19</td>
<td>1,077,364</td>
</tr>
<tr>
<td></td>
<td>Sat &amp; HyPre</td>
<td>1680/92321/194966</td>
<td>0.90</td>
<td>100</td>
<td>2.23</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></td>
<td>HyPre</td>
<td>324/1352/3874</td>
<td>0.01</td>
<td>100</td>
<td>10.14</td>
<td>5,230,084</td>
</tr>
<tr>
<td></td>
<td>SatELite</td>
<td>210/1201/4189</td>
<td>0.05</td>
<td>100</td>
<td>5.25</td>
<td>2,230,524</td>
</tr>
<tr>
<td></td>
<td>Sat &amp; HyPre</td>
<td>210/1210/4207</td>
<td>0.05</td>
<td>100</td>
<td>4.73</td>
<td>1,987,638</td>
</tr>
<tr>
<td></td>
<td>HyPre &amp; Sat</td>
<td>198/1232/4352</td>
<td>0.04</td>
<td>100</td>
<td>1.86</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>Stime</td>
<td>#BackT</td>
<td>Stime</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></td>
<td>Stime</td>
<td>#BackT</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+Hyp+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>

The imagination driving Australia’s ICT future
### 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/#Cl's/#Lits</th>
<th>Ptime</th>
<th>Dew_Satz</th>
<th>MiniSat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ptime</td>
<td>Stime</td>
<td>#BackT</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>
</tbody>
</table>

| homer-20 | Orig  | 440/4220/8800    | n/a   | 941      | 19,958,400 | >15,000 | n/a       |
|          | Sat   | 400/4180/15200   | 0.08  | 1,443    | 6,982,425  | 11,448  | 57,302,582 |
|          | Sha   | 1999/10342/29988 | 0.28  | 369      | 350,610    | 1.83    | 22,950    |
|          | Sha+3Res | 1907/8793/25027 | 0.37  | 362      | 405,059    | 1.41    | 18,273    |
|          | Sha+Hyp | 1905/10527/29129 | 0.34  | 1,306    | 1,451,567  | 1.10    | 13,927    |
|          | Sha+Niv | 1941/10276/29671 | 0.29  | 379      | 349,842    | 0.91    | 13,543    |
|          | Sha+Sat | 1723/9420/30986 | 0.54  | 822      | 300,605    | 1.00    | 13,831    |
|          | Sha+2Sim | 1879/9419/26188 | 0.31  | 114      | 120,297    | 0.40    | 6,612     |

The imagination driving Australia’s ICT future
## Using Dew_Satz

<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>BMC-IBM-12</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>ferry10_ks99a</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>ferry10_ks99a</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

The imagination driving Australia’s ICT future
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
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
Questions........