Chaff:
Engineering an Efficient SAT Solver

Matthew W. Moskewicz,
Concor F. Madigan, Ying Zhao, Lintao Zhang,
Sharad Malik
Princeton University

Slides: Tamir Heyman
Some are from Malik’s presentation

Last modified by Will Klieber on Sep 7, 2011
Boolean Algebra Notation

- “+” denotes logical OR ("∨").
- “·” denotes logical AND ("∧").
- Overbar or postfix “’” denotes negation.

Example:
“(A ∨ (¬B ∧ C))” corresponds to “(A + (B’ · C))”.
Chaff Philosophy

- Make the core operations fast
  - profiling driven, most time-consuming parts:
    - Boolean Constraint Propagation (BCP) and Decision

- Emphasis on coding efficiency

- Emphasis on optimizing data cache behavior

- Search space pruning:
  - conflict resolution and learning
Chaff’s Main Procedures

- Efficient BCP
  - Two watched literals
  - Fast backtracking
- Efficient decision heuristic
  - Localizes search space
- Restarts
  - Increases robustness
Implication

- What “causes” an implication?
- When can it occur?
- All literals in a clause but one are assigned False.
Implication example

- The clause \((v_1 + v_2 + v_3)\) implies values only in the following cases.
  - In case \((F + F + v_3)\)
    - implies \(v_3 = T\)
  - In case \((F + v_2 + F)\)
    - implies \(v_2 = T\)
  - In case \((v_1 + F + F)\)
    - implies \(v_1 = T\)
Implication for N-literal clause

- Implication occurs after N-1 assignments to False to its literals.
- We can ignore the first N-2 assignments to this clause.
- The first N-2 assignments won’t have any effect on the BCP.
Watched Literals

- Each clause has two watched literals.
- Ignore any assignments to the other literals in the clause.
- BCP maintains the following invariant:
  - By the end of BCP, one of the watched literals is true or both are unassigned.
  - (Can watch a false literal only if other watch is true.)
- Guaranteed to find all implications found by normal unit prop.
BCP with watched Literals

- Identifying conflict clauses
- Identifying unit clauses
- Identifying associated implications
- Maintaining “BCP Invariant”
Example (1/13)

Input formula has the following clauses:

\[ v_2 + v_3 + v_1 + v_4 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1 + v_2' \]
\[ v_1' + v_4 \]

\((v_1')\) means \((-v_1)\)
Example (2/13)

Initially, we identify any two literals in each clause as the watched ones.

\[
\begin{align*}
    & v_2 + v_3 + v_1 + v_4 \\
    & v_1 + v_2 + v_3' \\
    & v_1' + v_2' \\
    & v_1' + v_4
\end{align*}
\]

(\(v_1'\)) means (\(\neg v_1\))
Example (3/13)

Stack: (v1=F)

v2 + v3 + v1 + v4
v1 + v2 + v3’
v1 + v2’
v1’ + v4

Assume we decide to set v1 the value F
Example (4/13)

Ignore clauses with a watched literal whose value is T.
(Such clauses are already satisfied.)
Example (5/13)

Stack: (v1=F)

\[ v_2 + v_3 + v_1 + v_4 \]

\[ v_1 + v_2' + v_3' \]

\[ v_1' + v_4 \]

- Ignore clauses where neither watched literal value changes
Example (6/13)

Stack:\((\mathbf{v1}=\mathbf{F})\)

\[
\begin{align*}
\text{v2} + \text{v3} + \text{v1} + \text{v4} \\
\text{v1} + \text{v2} + \text{v3}' \\
\text{v1} + \text{v2}' \\
\text{v1}' + \text{v4}
\end{align*}
\]

- Examine clauses with a watched literal whose value is F
Example (7/13)

\[ v_2 + v_3 + v_1 + v_4 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1' + v_2' \]
\[ v_1' + v_4 \]
Example (7/13)

\[\begin{align*}
v_2 + v_3 + v_1 + v_4 \\
v_1 + v_2 + v_3' \\
v_1 + v_2' \\
v_1' + v_4
\end{align*}\]

Stack: \((v_1=F)\)

\[\begin{align*}
\text{In the second clause, replace the watched literal } v_1 \text{ with } v_3'
\end{align*}\]
Example (8/13)

\[
\begin{align*}
&v_2 + v_3 + v_1 + v_4 \\
v_1 + v_2 + v_3'
\end{align*}
\]

Stack: \((v_1=\text{F})\)

- The third clause is a unit and implies \(v_2=\text{F}\)
- We record the new implication, and add it to a queue of assignments to process.

\[
\begin{align*}
&v_2 + v_3 + v_1 + v_4 \\
v_1 + v_2 + v_3'
\end{align*}
\]

Stack: \((v_1=\text{F})\) Pending: \((v_2=\text{F})\)
Example (9/13)

\[
\begin{align*}
\text{Stack:}(v1=F, \quad v2=F) \\
\end{align*}
\]

Next, we process v2.
We only examine the first 2 clauses
Example (10/13)

\[
\begin{align*}
v2 + \overline{v3} + v1 + v4 &\quad\rightarrow\quad v2 + \overline{v3} + v1 + v4 \\
v1 + v2 + v3' &\quad\rightarrow\quad v1 + v2 + v3' \\
v1' + v2' &\quad\rightarrow\quad v1' + v4 \\
v1' + v4 &\quad\rightarrow\quad v1 = F, v2 = F \\
\end{align*}
\]

Stack: (v1=F, v2=F) Pending: (v3=F)

- In the first clause, we replace v2 with v4
- The second clause is a unit and implies v3=F
- We record the new implication, and add it to the queue
Example (11/13)

\[ v_2 + v_3 + v_1 + v_4 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1 + v_2' \]
\[ v_1' + v_4 \]

Stack: \((v_1=F, v_2=F, v_3=F)\)

Next, we process \(v_3'\). We only examine the first clause.
Example (12/13)

\[ v_2 + v_3 + v_1 + v_4 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1 + v_2' \]
\[ v_1' + v_4 \]

Stack: \((v_1=F, v_2=F, v_3=F)\)

\[ v_2 + v_3 + v_1 + v_4 \]
\[ v_1 + v_2 + v_3' \]
\[ v_1 + v_2' \]
\[ v_1' + v_4 \]

Stack: \((v_1=F, v_2=F, v_3=F)\)
Pending: \((v_4=T)\)

- The first clause is a unit and implies \(v_4=T\).
- We record the new implication, and add it to the queue.
Example (13/13)

\[
\begin{align*}
\text{Stack:} & \quad (v1=F, v2=F, v3=F, v4=T) \\
& v2 + v3 + v1 + v4 \\
& v1 + v2 + v3' \\
& v1 + v2' \\
& v1' + v4
\end{align*}
\]

- There are no pending assignments, and no conflict
- Therefore, BCP terminates and so does the SAT solver
Identify conflicts

Stack: (v1=F, v2=F, v3=F)

- What if the first clause does not have v4?
- When processing v3', we examine the first clause.
- This time, there is no alternative literal to watch.
- BCP returns a conflict
Backtrack

\[
\begin{align*}
&v_2 + v_3 + v_1 \\
&v_1 + v_2 + v_3' \\
&v_1 + v_2' \\
&v_1' + v_4
\end{align*}
\]

Stack:()

- We do not need to move any watched literal
BCP Summary

- During forward progress (decisions, implications)
  - Examine clauses where watched literal is set to F
  - Ignore clauses with assignments of literals to T
  - Ignore clauses with assignments to non-watched literals
Backtrack Summary

- Unwind Assignment Stack
- No action is applied to the watched literals

Overall
- Minimize clause access
Chaff Decision Heuristic VSIDS

- Variable State Independent Decaying Sum
  - Rank variables based on literal count in the initial clause database.
  - Only increment counts as new clauses are added.
  - Periodically, divide all counts by a constant.
VSIDS Example (1/2)

Initial data base

\[
\begin{align*}
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10'}
\end{align*}
\]

Scores:

4: \(x_8\)  
3: \(x_1, x_7\)  
2: \(x_3\)  
1: \(x_2, x_4, x_9, x_{10}, x_{11}, x_{12}\)

New clause added

\[
\begin{align*}
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10'} \\
x_7 + x_{10} + x_{12'}
\end{align*}
\]

Scores:

4: \(x_8, x_7\)  
3: \(x_1\)  
2: \(x_3, x_{10}, x_{12}\)  
1: \(x_2, x_4, x_9, x_{11}\)

watch what happens to \(x_8, x_7\) and \(x_1\)
VSIDS Example (2/2)

Counters divided by 2

\[
\begin{align*}
x1 + x4 \\
x1 + x3' + x8' \\
x1 + x8 + x12 \\
x2 + x11 \\
x7' + x3' + x9 \\
x7' + x8 + x9' \\
x7 + x8 + x10' \\
x7 + x10 + x12' \\
\end{align*}
\]

Scores:

2: x8,x7
1: x3,x10,x12,x1
0: x2,x4,x9,x11

New clause added

\[
\begin{align*}
x1 + x4 \\
x1 + x3' + x8' \\
x1 + x8 + x12 \\
x2 + x11 \\
x7' + x3' + x9 \\
x7' + x8 + x9' \\
x7 + x8 + x10' \\
x7 + x10 + x12' \\
x12' + x10 \\
\end{align*}
\]

Scores:

2: x8,x7,x12,x10
1: x3,x1
0: x2,x4,x9,x11

watch what happens to x8, x10
VSIDS - Summary

- **Quasi-static:**
  - Static because it is independent of variable values
  - Not static because it gradually changes as new clauses are added
    - Decay causes bias toward *recent* conflicts.
  - Use heap to find an unassigned variable with the highest ranking
Interplay of BCP and the Decision Heuristic

- This is only an intuitive description ...
  - Reality depends heavily on specific instances

- Take some variable ranking
  - Assume several decisions are made
  - Say \( v_2 = T, v_7 = F, v_9 = T, v_1 = T \) (and any implications thereof)
Interplay of BCP and the Decision Heuristic (cont’)

- Then a conflict is encountered and forces $v_2 = F$
- The next decisions may still be $v_7 = F$, $v_9 = T$, $v_1 = T$
  - VSIDS variable ranks change slowly...
- But the BCP engine has recently processed these assignments ...
  - so these variables are unlikely to still be watched.
Interplay of BCP and the Decision Heuristic (cont’)

- In a more general sense
- The more “active” a variable is, the more likely it is to *not* be watched.
- Because BCP is likely to replace it
Interplay of Learning and the Decision Heuristic

- Again, this is an intuitive description ...
- Learned clauses capture relationships between variables
- Decision heuristic influences which variables appear in learned clauses
  - Decisions $\rightarrow$ implications $\rightarrow$ conflicts $\rightarrow$ learned clause
Interplay of Learning and the Decision Heuristic (cont’)

- Important for decisions to keep the search strongly localized
  - Especially when there are 100k variables!

- In VSIDS, learned clauses bias decision strategy
  - Focusing in a smaller set of variables
Restart

- Abandon the current search tree and reconstruct a new one
- Helps reduce runtime variance between instances- adds to robustness of the solver
- The clauses learned prior to the restart are *still there* after the restart and can help pruning the search space
### Timeline

<table>
<thead>
<tr>
<th>Year</th>
<th>Algorithm</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>DP</td>
<td>≈ 10 var</td>
</tr>
<tr>
<td>1962</td>
<td>DLL</td>
<td>≈ 10 var</td>
</tr>
<tr>
<td>1986</td>
<td>BDD</td>
<td>≈ 100 var</td>
</tr>
<tr>
<td>1988</td>
<td>SOCRATES</td>
<td>≈ 3k var</td>
</tr>
<tr>
<td>1992</td>
<td>GSAT</td>
<td>≈ 300 var</td>
</tr>
<tr>
<td>1994</td>
<td>Hannibal</td>
<td>≈ 3k var</td>
</tr>
<tr>
<td>1996</td>
<td>GRASP</td>
<td>≈ 1k var</td>
</tr>
<tr>
<td>1996</td>
<td>SATO</td>
<td>≈ 1k var</td>
</tr>
<tr>
<td>1996</td>
<td>Stålmarck</td>
<td>≈ 1000 var</td>
</tr>
<tr>
<td>2001</td>
<td>Chaff</td>
<td>≈ 10k var</td>
</tr>
</tbody>
</table>