Lecture Outline

- CDCL round-up
  - Variable selection heuristics for CDCL
  - What are good learned clauses?
  - Forgetting learned clauses
- Internal structure of SAT instances
  - Visualization
  - Decomposition / partitioning
CDCL Algorithm

**input**: Formula $F$ in CNF

**output**: SAT / UNSAT

1. $dl \leftarrow 0$  // initialize decision level
2. $V \leftarrow \emptyset$  // initialize trail (variable assignment)
3. if unit_propagation($F$, $V$) == CONFLICT then
   4. return UNSAT
4. while not all variables assigned do
5.    $(x, b) \leftarrow$ pick branching literal
6.    $dl \leftarrow dl + 1$
7.    $V \leftarrow V \cup \{(x, b)\}$
8.    if unit_propagation($F$, $V$) == CONFLICT then
9.       $(c, bl) \leftarrow$ analyze conflict
10.      if $bl < 0$ then
11.         return UNSAT
12.      else
13.          add_clause(c)
14.          backtrack to $bl$
15.      $dl \leftarrow bl$
16. return SAT
Variable Selection in CDCL

- Previous heuristics (MOMS, Bohm’s, etc.): global, “static”
- E.g. MOMS: $S(x) = (f^*(x) + f^*(\overline{x})) \times 2^k + (f^*(x) \times f^*(\overline{x}))$
- static: $S(x)$ often computed only at root node of search
- global: based on whole CNF
Variable Selection in CDCL

- Previous heuristics (MOMS, Bohm’s, etc.): **global, “static”**
  - E.g. MOMS: \( S(x) = (f^*(x) + f^*(\overline{x})) \times 2^k + (f^*(x) \times f^*(\overline{x})) \)
  - **static**: \( S(x) \) often computed only at root node of search
  - **global**: based on whole CNF

- **Idea for CDCL**: Make heuristics more “focused”
  - try to find small unsatisfiable subsets
  - prefer variables that occurred in a recent conflict
VSIDS Heuristic

- **VSIDS**: Variable State Independent Decaying Sum
  - **General approach**: Compute score for each variable, select variable with highest score
  - Initial variable score is number of literal occurrences
  - New conflict clause $c$: Score is incremented for all variables in $c$
  - Periodically, divide all scores by a constant
VSIDS Heuristic

- **VSIDS**: Variable State Independent Decaying Sum
  - **General approach**: Compute score for each variable, select variable with highest score
  - Initial variable score is number of literal occurrences
  - New conflict clause $c$: Score is incremented for all variables in $c$
  - Periodically, divide all scores by a constant

- First presented in SAT solver Chaff, 2001 [1]
VSIDS Heuristic

- **VSIDS**: Variable State Independent Decaying Sum
  - **General approach**: Compute score for each variable, select variable with highest score
  - Initial variable score is number of literal occurrences
  - New conflict clause $c$: Score is incremented for all variables in $c$
  - Periodically, divide all scores by a constant

- First presented in SAT solver Chaff, 2001 [1]
- VSIDS (or a variant of it) implemented in most current CDCL solvers
VSIDS Example

Initial $F$:

\{ $x_1$, $x_4$ \}
\{ $x_1$, $\overline{x_3}$, $x_8$ \}
\{ $x_1$, $x_8$, $x_{12}$ \}
\{ $x_2$, $x_{11}$ \}
\{ $\overline{x_7}$, $\overline{x_3}$, $x_9$ \}
\{ $\overline{x_7}$, $x_8$, $\overline{x_9}$ \}
\{ $x_7$, $x_8$, $\overline{x_{10}}$ \}

Scores:

4 : $x_8$
3 : $x_1$, $x_7$
2 : $x_3$
1 : $x_2$, $x_4$, $x_9$, $x_{10}$, $x_{11}$, $x_{12}$
VSIDS Example

Initial $F$:
\[
\begin{align*}
\{x_1, x_4\} \\
\{x_1, \overline{x_3}, x_8\} \\
\{x_1, x_8, x_{12}\} \\
\{x_2, x_{11}\} \\
\{\overline{x_7}, \overline{x_3}, x_9\} \\
\{\overline{x_7}, x_8, x_9\} \\
\{x_7, x_8, \overline{x_{10}}\}
\end{align*}
\]

$F$ with new learned clause added:
\[
\begin{align*}
\{x_1, x_4\} \\
\{x_1, \overline{x_3}, x_8\} \\
\{x_1, x_8, x_{12}\} \\
\{x_2, x_{11}\} \\
\{\overline{x_7}, \overline{x_3}, x_9\} \\
\{\overline{x_7}, x_8, x_9\} \\
\{x_7, x_8, \overline{x_{10}}\} \\
\{x_7, x_{10}, \overline{x_{12}}\} \quad \text{(new learned clause)}
\end{align*}
\]

Scores:
\[
\begin{align*}
4 : x_8 \\
3 : x_1, x_7 \\
2 : x_3 \\
1 : x_2, x_4, x_9, x_{10}, x_{11}, x_{12}
\end{align*}
\]

Scores:
\[
\begin{align*}
4 : x_8, x_7 \\
3 : x_1 \\
2 : x_3, x_{10}, x_{12} \\
1 : x_2, x_4, x_9, x_{11}
\end{align*}
\]
Implementation of VSIDS

- **Possible**: Keep list of variables sorted by score
Implementation of VSIDS

- **Possible:** Keep list of variables sorted by score
- **Many implementations:** Use priority queues
  - **Operations:**
    - `insert_with_priority`, `pull_highest_priority_element`
Implementation of VSIDS

- **Possible:** Keep list of variables sorted by score
- **Many implementations:** Use priority queues
  - Operations:
    - `insert_with_priority`, `pull_highest_priority_element`
- **Often implemented as binary heaps**
  - Insert: $\mathcal{O}(\log n)$
  - Delete: $\mathcal{O}(\log n)$
  - Peek: $\mathcal{O}(1)$
Question: Why periodically divide scores?
Answer: Give priority to recently learned clauses
Chaff: half scores every 256 conflicts (“decay”); sort priority queue after each decay only
Variants of VSIDS:
- Berkmin’s strategy (Berkmin, 2002)
- VMTF: variable move to front (Siege, 2004)
- CMTF: clause move to front (HaifaSAT, 2008)
Comparison of Heuristics

SAT Competition 2013 Application Track Benchmarks Solved by Lingeling

Introduction Repetition CDCL Round-Up Structure

Carsten Sinz, Tomáš Balyo – SAT Solving

June 6, 2016

9/20
Learned Clause Removal

- **Problem:** Too many learned clauses!
  - ...and not all of them are helpful (e.g. subsumed clauses)
  - BCP gets slower, memory consumption
- **Solution:** Forget clauses after some time
  - also called **Clause Database Reduction**
  - size heuristics: discard long clauses
  - least recently used (LRU) heuristics: discard clauses not involved in recent conflict clause generation
  - blue: “Glucose level”: number of decision levels in learned clauses (called LBD in original paper [2])
Structure in SAT Instances
Notions of Structure

- Often graph-based: incidence graph (aka factor graph), primal graph (aka variable interaction graph)
- \( F = (C_1, C_2, C_3) = (\{x, y, \overline{z}\}, \{\overline{x}, y\}, \{u, y, z\}) \)

![Diagrams of hypergraph, factor graph, variable interaction graph, and resolution graph]
Primal Graph and DPLL

During search delivers considerably more information. We therefore simulated such depth-bounded runs of a typical DPLL algorithm in our experiments. The depth-bounded search tree obtained that way for the `longmult8` instance is shown in Figure 6.

![Search tree of a depth-bounded run of the DPLL algorithm on the longmult8 benchmark problem.](image)

Figure 7. Variable interaction graph of the instance `longmult8` (top-level, after unit propagation).

For comparison reasons, we initiated experiments with further SAT instances outside the realm of hardware verification. We used an instance from automotive product configuration, one instance of the well-known pigeon hole problems (`hole10`) and a random 3-SAT formula with 100 variables and 425 clauses. The last two instances are known to be hard for resolution-based SAT-solvers, whereas the configuration instance is known to be very easy. The variable interaction graphs shown on the left of Figure 12 correspond to a state during the run of a DPLL algorithm where a few literals already have been fixed. After setting of three further variables and subsequent unit propagation the interaction graphs shown on the right resulted.

Besides generating variable interaction graphs, we also built resolution graphs. Figure 13 shows two resolution graphs for restriction `A` of instances `longmult8` and for instance `bmc-ibm-2`. Clustering and symmetry of the resolution graphs is similar to the respective variable interaction graphs. This suggests that properties of an instance like symmetries carry over from one representation to the other (for the component structure of a SAT instance this is obvious).

---

`longmult8`: output bit 8 of 16-bit hardware multiplier
longmult8: output bit 8 of 16-bit hardware multiplier
longmulti8: output bit 8 of 16-bit hardware multiplier
Primal Graph and DPLL

longmult8: output bit 8 of 16-bit hardware multiplier
**Primal Graph and DPLL**

```
\begin{align*}
\text{longmult8: output bit 8 of 16-bit hardware multiplier}
\end{align*}
```
Change of Structure in CDCL

left: original problem, right: after three decisions and UP configuration – pigeon hole – random
Definition: Diameter

Let $V$ be the set of variables of an ordered formula $\phi_\pi$. For $x \in V$, let $f(x), l(x)$ be the index of the clause that $x$ appears for the first and last time, respectively. The ordered diameter is

$$D(\phi_\pi) = \max_{x \in V} (l(x) - f(x)),$$

the unordered diameter is

$$\Delta(\phi) = \min_{\pi} D(\phi_\pi).$$
Diameter Example

\[ \phi_{pi} = (\{x_1, x_2, x_3\}, \{x_2, x_4, x_5\}, \{x_6, x_7, x_4\}, \{x_9, x_1, x_10\}) \]

\[ D(\phi_{\pi}) = 3 \]

\[ \Delta(\phi_{\pi}) = 2 \]


URL http://dl.acm.org/citation.cfm?id=1661445.1661509