Visualizing CDCL VSIDS and phase values

Jan Elffers

KTH Royal Institute of Technology

July 8, 2019

Joint work with Jakob Nordström
CDCL solvers crucially use heuristics for, e.g.:

- Variable decisions.
- Clause database management.
- Restarts.
CDCL solvers crucially use heuristics for, e.g.:

- Variable decisions.
- Clause database management.
- Restarts.

Heuristics often work very well.

Limited understanding of why.

This presentation: tool for visualizing CDCL heuristics (so far on crafted benchmarks)
Classic decision heuristic: variable with highest VSIDS score [MMZ\textsuperscript{+}01]

If variable \( v \) was active in conflicts \( t_1, \ldots, t_k \) and \( T \) conflicts have passed,

\[
activity(v) = \sum_{1 \leq i \leq k} \alpha^{T-t_i}
\]

\( \alpha \) decay factor \( 1/2 \leq \alpha < 1. \) (default 0.95)

*Phase saving* [PD07]: Set variable to the polarity (phase) it was last propagated to.
Visualizing VSIDS scores

- Project arose out of understanding how CDCL solves tricky combinatorial benchmarks.
- By visualizing aspects of proof search (VSIDS, phase) we try to understand CDCL.
- Crafted benchmarks often defined in terms of graphs or matrices.
Visualizing VSIDS scores

- Brighter color = higher score
- Runs during CDCL execution.
- Refreshes every 100 conflicts.*
- At most 25 “frames”/second.*

*numbers can be changed
How to implement your own visualization

Visualizer written in Qt framework.
  ▶ Before the solver starts running, a method is called to draw shapes on the grid.
    Each shape = a variable.
    In this method you can draw your own visualization.
  ▶ With each new frame, a method draw_vsids(vector<double> scores) is called.
    Gives shapes colors corresponding to VSIDS scores.
    Can be overridden to define custom color scheme.
Smoothing of VSIDS animation

CDCL VSIDS scores can change quickly: becomes a blur.

- Use exponential moving average on VSIDS scores:

\[ \text{expAverage}(v)_t = \sum_{\Delta t \geq 0} \alpha^{\Delta t} \text{activity}(v)_{t-\Delta t} \]

set \( \alpha \) to say 0.99.

\( \text{activity}(v)_t \) are VSIDS scores (= exponential averages) themselves

- Visualizer reads VSIDS scores from solver.

- Visualize frames every 100 conflicts.

However, want average over all conflicts

Cannot read full VSIDS scores after every conflict.
Our implementation: let solver maintain 2 VSIDS scores: standard and more slowly decaying.

Mathematical guarantee:

**Theorem**

Exponential average with parameter $d_{\text{slow}}$ over decay $d$ VSIDS is a linear combination of decay $d$ VSIDS and decay $d_{\text{slow}}$ VSIDS.

$\Rightarrow$ slower decay factor approximates exponential moving average.
Phase VSIDS

Similar problem with phases. Can apply same VSIDS idea. For $0 < \alpha < 1$, one can define

$$\text{vsidsPhase}(v)_t = (1 - \alpha) \text{phase}(v)_t + \alpha \cdot \text{vsidsPhase}(v)_{t-1}$$

where false $= -1$, true $= 1$.
Only for visualization purposes currently.
Integration in CDCL solver

Communication via text streams on stdin / stdout

- Add a command line parameter “interactive” to the solver.
- Add a method `interactive()` to the solver.

Idea:

1. read #conflicts to run
2. run solver for this many conflicts
3. output `frame`:
   - The secondary VSIDS scores, normalized to [0, 1].
   - Phases mapped to interval [−1.0, 1.0] (−1 = false, 1 = true).
1. VSIDS:
   (a) Ordering principle
   (b) Flow formula
2. Phase:
   (a) Pebbling formula with XOR
   (b) Pigeonhole principle
Demo 1a: ordering principle

A formula with variables $x_{i,j}$ for each $1 \leq i \neq j \leq n$ corresponding to edges of a complete graph.
Super easy in theory.
Hard for CDCL if VSIDS decay factor too high (too close to 1).

**Clause deletion** causes VSIDS resets, not restarts.
Formula defined on an undirected graph.  
Super easy in theory. 
Variables \( x_{u,v} \) for all directed edges \((u, v)\).  
Constraints (\( N(u) \) are neighbours of vertex \( u \)): 

\[
\forall u : \sum_{v \in N(u)} x_{u,v} - x_{v,u} = 1
\]

Phenomenon observed: VSIDS “freezes” on some instances. 
Sample graph is 6-regular random graph on 500 nodes.
Demo 2a: pebbling formula with XOR

A formula defined on a single-sink DAG with indegree 2.
Super easy in theory.

Two variables per vertex: $x_{u,1}$ and $x_{u,2}$. Constraints:

- $x_{u,1} \oplus x_{u,2} = 1$ for $u$ a source.
- $x_{u,1} \oplus x_{u,2} = 0$ for $u$ the sink.
- If $u$ has predecessors $v, w$,

$$((x_{v,1} \oplus x_{v,2} = 1) \land (x_{w,1} \oplus x_{w,2} = 1)) \rightarrow (x_{u,1} \oplus x_{u,2} = 1)$$

Visualize $\text{phase}(x_u) := \text{phase}(x_{u,1}) \cdot \text{phase}(x_{u,2})$ for all vertices $u$.
(remember $\text{phase} \in \{-1, 1\}$ without phase VSIDS)
Demo 2b: pigeonhole principle

Formula claims that $n$ pigeons do not fit into $n - 1$ holes. Variables $x_{i,j}$ for $1 \leq i \leq n$, $1 \leq j \leq n - 1$. We run CDCL without restarts, and with and without phase VSIDS.
Conclusion

What is already implemented:
- Various visualizations for studying combinatorial formulas.
- Interactive version of Minisat for use with the visualizer.

What a user could add:
- Additional visualizations.
- Interactive versions of other CDCL SAT solvers.

The software is available on Github:
github.com/elffersj/cdcl-visualizer
Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: Engineering an efficient SAT solver.

Knot Pipatsrisawat and Adnan Darwiche.
A lightweight component caching scheme for satisfiability solvers.