Lecture 2 topics

- Conjunctive Normal Form and Tseytin’s transformation.
- Pure literal rule.
- Unit clauses (Boolean constraint propagation).
- Watched literals.

Conflict clauses

A conflict clause is a clause that is added to capture the causes of an inconsistency discovered during search.

Conflict clauses are very important. In a sense, they can “learn” from failed searches to improve future search.

They can also be thought of as “caching” previous search results.

Suppose, after a series of assignments, the solver discovers an inconsistency.

E.g., $x_1 = 1$, $x_2 = 0$, $x_3 = 1$

Then we know that: $\phi \rightarrow [x_1 \land \neg x_2 \land x_3 \rightarrow F]$

which is logically equivalent to: $\phi \rightarrow (\neg x_1 \lor x_2 \lor \neg x_3)$

**Terminology:** If $\phi \rightarrow \psi$, we call $\psi$ an *implicate* of $\phi$.

(Actually, I try not to call it that, because it is so easily confused with “implicant,” a more common but essentially opposite concept. But the GRASP paper uses the term a lot.)

*And the cool thing is that we can add this clause to $\phi$ without changing it:* $(\neg x_1 \lor x_2 \lor \neg x_3) \land \phi$ is logically equivalent to $\phi$. 
Implication graphs

To do useful clause learning, many SAT solvers maintain an implication graph, which captures the relevant variables that caused an implication in BCP.

Whenever we have to choose a variable assignment, the recursion depth is called the decision level of the variable.

The implication graph is a DAG. Vertices are of the form $x_i = v@d$, where $x_i$ is a variable, $v$ is a truth value, and $d$ is a decision level.

BCP adds to the graph when it does unit propagation.

Implication graph example

Conflicts are also included as nodes in the implication graph. Edges are called “implications.”

This is the example from the GRASP paper:

Failure-Driven Assertions

Things happen automatically that we might otherwise have had to invent and work hard to implement.

Failure-driven assertions: When a decision to set $x_i = v$ fails, the only other possibility is $x_i = \neg v$ – if you don’t undo previous decisions.

The conflict clause that was added automatically forces the solver to set $x_i = \neg v$.

If you add the conflict clause and then undo $x_i = v$, the new clause is unit clause and BCP immediately propagates $x_i = \neg v$, with no more implementation effort.

When this happens, $x_i$ is not a decision variable because it was set by BCP.
Conflict-directed backtracking

(I think this was called “backjumping” in older SAT algorithms.)
The naive algorithm backtracks one level after trying both
assignments for a variable.
Sometimes it is possible to jump back many levels, which can cut
off a massive portion of the search tree.

**Intuition:** If we get conflicts for both values of a variable at
decision level 20, and the previous relevant variable was at level 10,
changing variables at levels 11..19 is not going to make any
progress (none of those variables were relevant).
Conflict clauses exploit this automatically.

Conflict clauses exploit this automatically.

Better conflict clauses

Sometimes you can find better conflict clauses if there is one node
that separates the highest-level decision variable from the conflict.
Such a node is called a “unique implication point” (UIP).

That node allows us to use a smaller clause.

<table>
<thead>
<tr>
<th>Current Truth Assignment</th>
<th>Current Decision Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_0 = 0 \at 1, x_10 = 0 \at 3, x_{11} = 0 \at 3, x_{12} = 1 \at 2, x_{13} = 1 \at 2, \ldots}</td>
<td>{x_1 = 1 \at 6}</td>
</tr>
</tbody>
</table>

Node x_4 is a UIP, so we can add clause \((\neg x_1 \lor x_9 \lor x_4)\).

UIPs

Node x_4 is a UIP, so we can add clause \((\neg x_1 \lor x_9 \lor x_4)\).

<table>
<thead>
<tr>
<th>Current Truth Assignment</th>
<th>Current Decision Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x_0 = 0 \at 1, x_{10} = 0 \at 3, x_{11} = 0 \at 3, x_{12} = 1 \at 2, x_{13} = 1 \at 2, \ldots}</td>
<td>{x_1 = 1 \at 6}</td>
</tr>
</tbody>
</table>
Variable selection

There are many heuristics for choosing which variable to assign next (and whether to assign T or F first).

While the variable selection heuristic can have a profound impact on efficiency, it is not clear that there is robust “best” heuristic.

Example heuristics:

RAND: Randomly choose the next variable and assignment. This has turned out to be better than you might have expected in some tests.

DLIS (“Dynamic largest individual sum”): Use literal that appears most frequently in unsatisfied clauses.

VSIDS: Increment a counter for each literal when a clause is added with that variable. Choose literal with highest count. Periodically divide counts by a constant. This focuses effort on recently added conflict clauses. (Used in CHAFF and MiniSAT).

MOM (“Maximum Occurrences on clauses of Minimum Size”).
Let $f^*(\ell)$ be the number of occurrences of literal $\ell$ in the smallest unsatisfied clauses. Select literals that maximize:

$$[f^*(x) + f^*(\neg x)] * 2^k + f^*(x) * f^*(\neg x).$$

Intuition: Focuses on making small clauses smaller, and prefers literals that appear in many clauses, and variables where both polarities appear in many clauses.

MOM is an old heuristic, and part of a family that combine several metrics, such as clause size, number of occurrences of literals in various types of clauses, etc.

Other ideas

Exploit polynomial-time special cases:

- Horn clauses
- 2-sat
- Linear algebra in GF(2) (CNF with XOR instead of OR in clauses).

For now, it seems that only the last has really been helpful (cryptominisat), but maybe that will change.

Closing remark

Simple and fast beats complex and smart.

In SAT. For the moment . . .