CS 357 Lecture 4: Practical SAT Solving, cont.

David L. Dill
Department of Computer Science
Stanford University

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.

Conflict clauses

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: $\phi \rightarrow [x_1 \wedge \neg x_2 \wedge x_3 \rightarrow F]$

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

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

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

And the cool thing is that we can add this clause to ϕ without changing it: $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge \phi$ is logically equivalent to ϕ.

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."

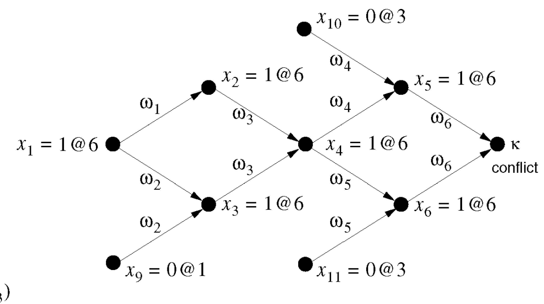
This is the example from the GRASP paper:

Current Truth Assignment: $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \ldots\}$

Current Decision Assignment: $\{x_1 = 1@6\}$

$$
\begin{aligned}
\omega_1 &= (\neg x_1 + x_2) \\
\omega_2 &= (\neg x_1 + x_3 + x_9) \\
\omega_3 &= (\neg x_2 + \neg x_3 + x_4) \\
\omega_4 &= (\neg x_4 + x_5 + x_{10}) \\
\omega_5 &= (\neg x_4 + x_6 + x_{11}) \\
\omega_6 &= (\neg x_5 + \neg x_6) \\
\omega_7 &= (x_1 + x_7 + \neg x_{12}) \\
\omega_8 &= (x_1 + x_8) \\
\omega_9 &= (\neg x_7 + \neg x_8 + \neg x_{13}) \\
&\ldots
\end{aligned}
$$

Clause Database



Implication Graph for Current Decision Assignment

Implication graph

One way to find a conflict clause is to trace back from the conflict until find the source nodes.

The $\vee$ of the negations of these nodes is a conflict clause.

In this case, the conflict clause $(\neg x_1 \vee x_9 \vee x_{10} \vee x_{11})$ can be added.

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).

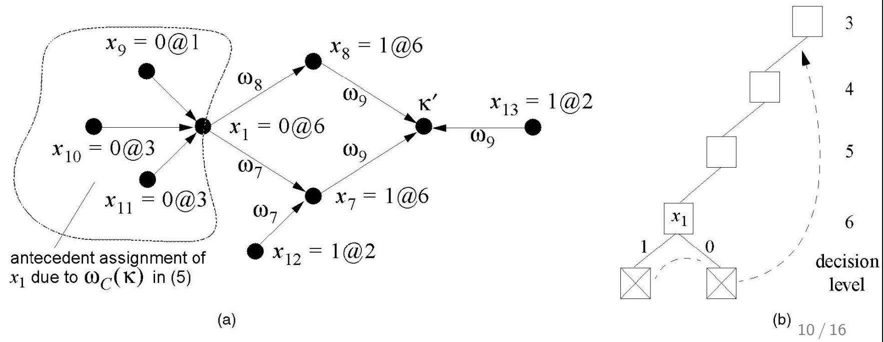
Conflict clauses exploit this automatically.

Conflict-directed backtracking

In the example, *after we add the first conflict clause*, the graph looks like this:

Even though we are at decision level 6, the highest-level decision variables in the conflict graph are at level 3.

Adding the resulting clause will cause all pending decisions at levels 4 and 5 to fail immediately (via BCP), so it immediately backtracks to level 3, skipping BCP.



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).

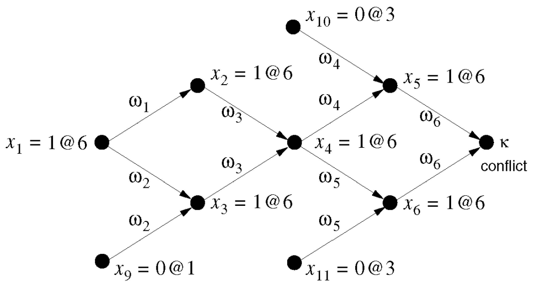
That node allows us to use a smaller clause.

Current Truth Assignment: $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \ldots\}$

Current Decision Assignment: $\{x_1 = 1@6\}$

$$
\begin{aligned}
\omega_1 &= (\neg x_1 + x_2)\\
\omega_2 &= (\neg x_1 + x_3 + x_9)\\
\omega_3 &= (\neg x_2 + \neg x_3 + x_4)\\
\omega_4 &= (\neg x_4 + x_5 + x_{10})\\
\omega_5 &= (\neg x_4 + x_6 + x_{11})\\
\omega_6 &= (\neg x_5 + \neg x_6)\\
\omega_7 &= (x_1 + x_7 + \neg x_{12})\\
\omega_8 &= (x_1 + x_8)\\
\omega_9 &= (\neg x_7 + \neg x_8 + \neg x_{13})\\
&\ldots
\end{aligned}
$$

Clause Database



Implication Graph for Current Decision Assignment

UIPs

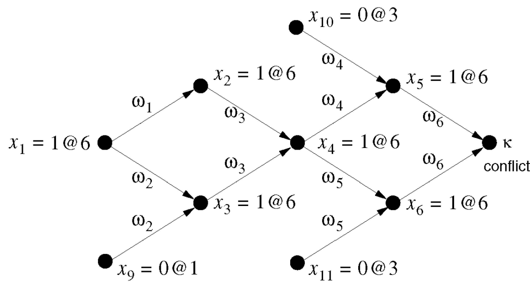
Node x_4 is a UIP, so we can add clause $(\neg x_1 \vee x_9 \vee x_4)$.

Current Truth Assignment: $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \ldots\}$

Current Decision Assignment: $\{x_1 = 1@6\}$

$$
\begin{aligned}
\omega_1 &= (\neg x_1 + x_2)\\
\omega_2 &= (\neg x_1 + x_3 + x_9)\\
\omega_3 &= (\neg x_2 + \neg x_3 + x_4)\\
\omega_4 &= (\neg x_4 + x_5 + x_{10})\\
\omega_5 &= (\neg x_4 + x_6 + x_{11})\\
\omega_6 &= (\neg x_5 + \neg x_6)\\
\omega_7 &= (x_1 + x_7 + \neg x_{12})\\
\omega_8 &= (x_1 + x_8)\\
\omega_9 &= (\neg x_7 + \neg x_8 + \neg x_{13})\\
&\ldots
\end{aligned}
$$

Clause Database



Implication Graph for Current Decision Assignment

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 hueristics:

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.

Variable selection, cont.

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 focusses 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 ℓ 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 . . .