CS 433 Automated Reasoning 2025

Lecture 7: CDCL - optimizations

Instructor: Ashutosh Gupta

IITB India

Compile date: 2025-01-31

Review of CDCL

- CNF input
- Decision, propagation, conflict, and backtracking
- Clause learning from conflict

Other heuristics

More heuristics that may improve the performance of SAT solvers

- Clause minimization
- Lazy data structures
  - 2-watched literals
  - pure literals
- Optimal storage
  - Variables
  - Clauses
  - Occurrence maps

- Runtime choices
  1. Variable ordering
  2. Restarts
  3. Learned clause deletion
  4. Phase saving
- Pre/In-processing

Commentary: Clause learning is an algorithmic change. The above optimizations are clever data structures and implementations.

Topic 7.1

Conflict clause optimizations

Choices of conflict clause

Definition 7.1 (Functional definition of conflict clause)

We will say if a clause satisfies the following two conditions, it is a conflict clause.

1. *the clause does not change the set of satisfying assignments*
2. *the clause implies that the conflicting partial assignment will never be tried again*

There are other clever choices for conflict clauses.

Choices of conflict clauses

Some choices of clauses are found to be better than others

- Smaller conflict clauses prune more search space(Why?)

Example 7.1

Let us suppose there are three variables p, q, and r in the formula. How many solutions are rejected by the following clauses?

- $p \vee q \vee r$
- $p \vee q$
- p

- Decision variables may not be the variables that are the center of action for causing conflicts.

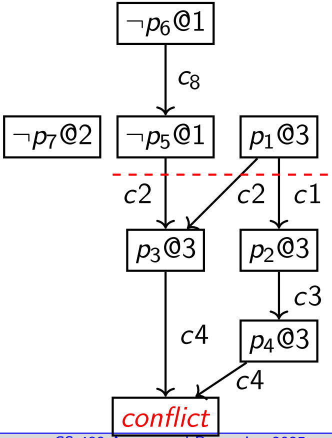
Commentary: The earlier presentation of the conflict clause may suggest that there is no choice in constructing the conflict clause. Now we will explore how to learn better conflict clauses that satisfy the above two objectives.

Cut clauses

Definition 7.2

Consider a cut of an implication graph that separates the decision nodes from the conflict node. Let $\ell_1, \dots, \ell_k$ be the literals that occur at the cut boundary. The cut clause for the cut is $\neg\ell_1 \vee \dots \vee \neg\ell_k$.

Example 7.2



Literals that are sources of cut edges

Cut clause : $\neg p_1 \vee p_5$

observation

Cut clauses may act as conflict clauses.

Exercise 7.1

Other choices for the cut clauses?

Cut clauses preserve models

Theorem 7.1

Cut clauses satisfy all the models of the input formula.

Proof.

Choose a cut. Consider the nodes at the boundary as the decision literals.

The graph from the boundary to the conflict is a valid implication graph. (Why?)

Therefore, the assignments of the input formula satisfy the cut clause. □

How to efficiently find the cuts that are small?

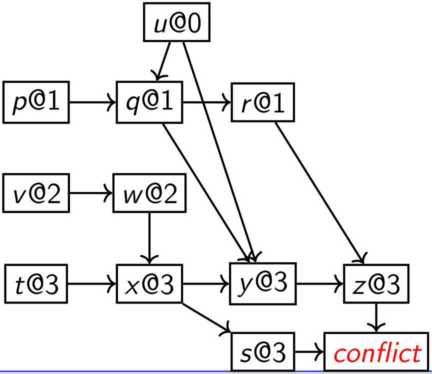
Unique implication point (UIP)

Definition 7.3

In an implication graph, node $\ell@d$ is a unique implication point at decision level d if $\ell@d$ occurs in each path from d^{th} decision literal to the conflict.

Example 7.3

Consider the following implication graph (Example source: SörenssonBiere-SAT09)



$$\begin{tikzcd}
 & u@0 \arrow[d] \arrow[dddr] & & \\
p@1 \arrow[r] & q@1 \arrow[r] \arrow[ddr] & r@1 \arrow[ddr] & \\
v@2 \arrow[r] & w@2 \arrow[d] & & \\
t@3 \arrow[r] & x@3 \arrow[r] \arrow[dr] & y@3 \arrow[r] & z@3 \arrow[d] \\
 & & s@3 \arrow[r] & \text{conflict}
\end{tikzcd}$$

Note: Edges need not be labeled with clauses.

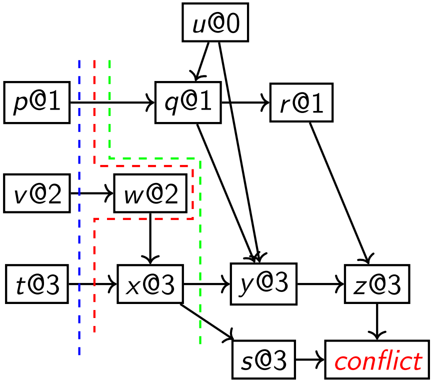
UIPs @ level 1 : $p@1, q@1$
UIPs @ level 2 : $v@2, w@2$
UIPs @ level 3 : $t@3, x@3$

First UIP strategy

Algorithm: Iteratively replace a decision literal by one of its UIP in the conflict clause.

Preferably choose UIP that is closest to the conflict, which may result in introduction of a single UIP that replaces multiple decision literals faster.

Example 7.4



Conflict clause using decision literals:

$\neg p \vee \neg v \vee \neg t$

We can replace v with w

$\neg p \vee \neg w \vee \neg t$

We can replace t with x

$\neg p \vee \neg x$

Exercise 7.2

Can we further move the cut?

Why first UIP?

- Likely smaller clauses
- Focuses on center of action
- Efficient to implement

Exercise 7.3

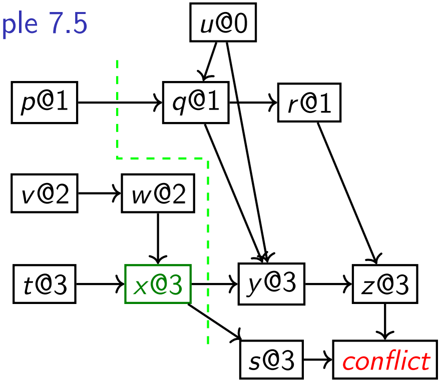
Is the first UIP strategy deterministic?

Back UIP clauses

Not using the decisions in the learned clause looses information.

Some solvers also keep the record of relations between the decisions and UIPs by learning additional clauses, called back clauses. [SabharwalSamulowitzSellmann-SAT12]

Example 7.5



First UIP conflict clause: $\neg p \lor \neg x$

Back clause: $\neg v \lor \neg t \lor x$

One may learn both the clauses.

Topic 7.2

Lazy data structures

Detecting unit clauses

Näive procedure:

1. For each unassigned clause count unassigned literals
2. If there is exactly one unassigned literal, apply unit clause propagation

Observation:
To decide if a clause is ready for unit propagation,
we need to count only 0, 1, and many, i.e., look at only two literals that are not false.

Let us use the insight to optimize the unit clause propagation.

2-watched literals for detecting unit clauses

For each clause we select two literals and we call them watched literals.

In a clause,

- if watched literals are non-false, the clause is not a unit clause
- if any of the two becomes false, we search for another two non-false literals
- if we can not find another two, the clause is a unit clause

Exercise 7.4

Why this scheme may reduce the effort in searching for the unit clauses?

Commentary: If we maintain this additional information, we need to ensure that the potential benefit of of the information outweighs the overhead of maintaining the additional information.

Example: understanding 2-watched literals

Example 7.6

Consider clause $p_1 \lor p_2 \lor \neg p_3 \lor \neg p_4$ in a formula. Let us initially watch p_1 and p_2 in the clause.

$* \triangleq$ *watched literals,* ☺ $\triangleq$ *no work needed!*

Initially: $m = \{\}$	$p_1^* \lor p_2^* \lor \neg p_3 \lor \neg p_4$	
$\vdots$	$\vdots$	
Assign $p_1 = 0$: $m = \{\ldots, p_1 \mapsto 0\}$	$p_1 \lor p_2^* \lor \neg p_3^* \lor \neg p_4$	*(work)*
Assign $p_2 = 1$: $m = \{\ldots, p_1 \mapsto 0, p_2 \mapsto 1\}$	$p_1 \lor p_2^* \lor \neg p_3^* \lor \neg p_4$	☺
Backtrack to p_1: $m = \{\ldots\}$	$p_1 \lor p_2^* \lor \neg p_3^* \lor \neg p_4$	☺
Assign $p_4 = 1$: $m = \{\ldots, p_4 \mapsto 1\}$	$p_1 \lor p_2^* \lor \neg p_3^* \lor \neg p_4$	☺

The benefit: often no work to be done!

Commentary: We see that the overhead of maintaining the information is limited.

Data structure for 2-watched literals

- Create a map from literals to a list of clauses where the literals are watched

- If a literal becomes false, one of the following happens in a clause where it is watched
  1. the clause has become a unit clause
  2. conflict has occurred
  3. the clause is moved to the other literals in the clause watch list

Only in the last case, the data structure changes.

- No other operation in the assignment triggers an action on watched data structure

Exercise 7.5

a. What if we have only one watched literal. Will it work?

b. Is this idea extendable for counting 0, 1, 2, and many?

Detecting and assigning pure literals

ℓ may be assigned 1 immediately.

Definition 7.4

A literal ℓ is called pure in F if $\bar{\ell}$ does not occur in F.

In CDCL run, more literals may become pure(Why?), which may be assigned 1 like unit propagation.

However, this optimization is at odds with 2-watched literal optimization.

- 2-watched literal visits the clauses that have literals that are just assigned false and watched
- Adding traversal for pure literals will defeat the benefit of 2-watched literal.

Exercise 7.6

Can we use 2-watched literal like data structure to improve pure literal search?

Often not implemented

Commentary: We saw two optimizations that are at odds with each other. Often newly proposed optimizations find it hard to work with existing ones in the tools. Anecdotal fact: Some Quantified Boolean Formula(QBF) solvers do implement pure literal removal. Similar to 2-watched literal idea, they watch clauses to check if some literal is still active.

Topic 7.3

Optimal storage

Storing variables

Variables are contiguous numbers

- A word of the machine is used to store a variable name
- Positive integer is the positive literal
- Negative integer is the negated literal
- Variable numbers are used as index to the data structures

Example 7.7

If a formula has five variables, we say we have variables $1, 2, 3, 4, 5$*.*

We say $-1, -2, -3, -4,$ *and* -5 *are the negative literals.*

Exercise 7.7

What is the maximum number of variables allowed in the design?

Current assignment

We need to store 2 bits to store variable state (true, false, and unassigned) in a bitvector.

Along with recording value on each variable, we record current assignment is a list of literals, which allows efficient push and pop.

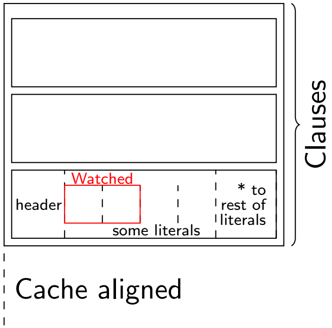
Example 7.8

An assignment

$$[-4, \quad -2, \quad 3, \quad -50]$$

Clause store

- Lists of clauses stored as array of arrays
- Clause header and a few literals are stored together and then linked list
- Rearrange clauses if changes in watched literal
- In some implementations, clauses are aligned with cache line
- Pre allocate clauses in bulk to avoid system overhead when conflict clauses are added



Commentary: The above design is not universal. It is highly dependent on the other choices in the algorithm.
Please look at: `http://www.easychair.org/publications/download/Towards_Improving_the_Resource_Usage_of_SAT-solvers`
A typical SAT solver is full of "little" optimizations. Each is simple, but it takes several years to learn all of them. If it interests you, read sources of some solvers.

Occurrence map

In CDCL, we run unit propagation repeatedly. As a literal ℓ becomes true, we need to

- disable clauses containing ℓ
- check for unit propagation in clauses containing $\bar{\ell}$

Undo during backtracking.

For fast access, we keep occurrence map : literals $\rightarrow$ clauses.

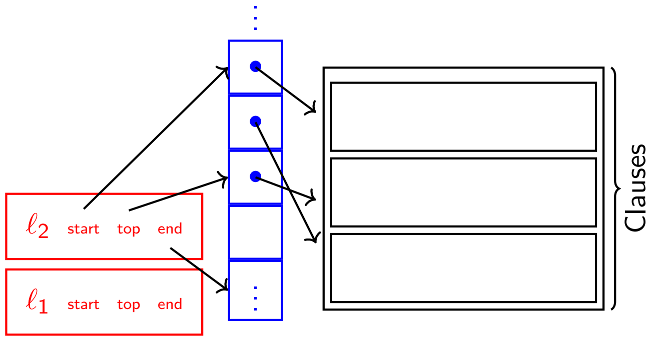
Binary and ternary clauses stored separately, since they become unit clauses too often

- $Occurs2$: literals $\rightarrow$ Binary clauses
- $Occurs3$: literals $\rightarrow$ Ternary clauses
- $OccursL$: literals $\rightarrow$ Large clauses

Efficient data structure for literals → Large clauses

SAT solvers are memory intensive. Minimize: dereferencing, resizing, and cache misses.

- For each literal, we maintain a stack using three pointers start, top, and end
- All stacks are laid out on a single array of pointers to clauses
- End pointer informs if neighbouring stacks are about to clash and it is time to resize



Commentary: The priority changes if our solver is multi-threaded or allows clause deletion. More on efficient maps: Armin Biere, PicoSAT Essentials, 2008.

Topic 7.4

Problems

UIP

Exercise 7.8

Consider the following implication graph generated in a CDCL solver.

$$\begin{tikzcd}
\boxed{\neg p_1 @ 1} \arrow[r] \arrow[dr] & \boxed{p_4 @ 1} \arrow[ddrrr] & & & \\
\boxed{p_2 @ 2} \arrow[r] \arrow[dr] & \boxed{p_5 @ 2} \arrow[dr] \arrow[ddr] & & & \\
\boxed{p_3 @ 3} \arrow[r] \arrow[dr] & \boxed{\neg p_6 @ 3} \arrow[d] & \boxed{p_8 @ 3} \arrow[r] & \boxed{\neg p_{10} @ 3} \arrow[r] \arrow[dr] & \boxed{p_{11} @ 3} \arrow[d] \\
 & \boxed{p_7 @ 3} \arrow[ur] \arrow[r] & \boxed{p_9 @ 3} \arrow[ur] & & \boxed{\textit{conflict}}
\end{tikzcd}$$

a. Identify the UIPs in the each level.

b. Give the first UIP clause and corresponding back clauses.

Smallest conflict clause

Exercise 7.9

Prove/disprove: For a given implication graph, UIP strategy will always produce smallest conflict clause.

Exercise : back clauses

Exercise 7.10

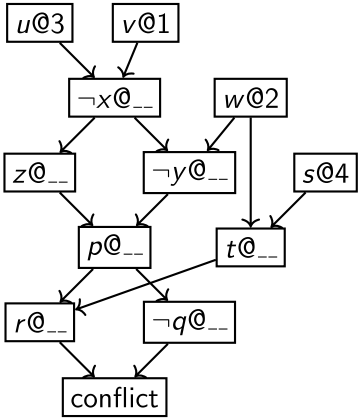
Let us suppose we learned conflict clause $p_1 \vee \neg p_2 \vee p_3$ *using first UIP strategy cut while analyzing an implication graph. We also learned back clauses* $p_4 \vee \neg p_1$ *and* $p_2 \vee \neg p_5$ *from the cut. Which of the following are among the decision literals in the implication graph?*

- $\neg p_3$
- p_5
- p_3
- $\neg p_1$
- $\neg p_2$

Conflict clauses

Exercise 7.11

Consider the following implication graph generated in a CDCL solver.



a. *Assign decision level to every node (write within the node)*
b. *What is the conflict clause due to the implication graph?*
b. *Write unique implication points(UIPs) for each level*
c. *Give the first UIP conflict clause.*

Exercise: execute 2-watched literals

Exercise 7.12

Let the following be a sequence of partial models occurring in a run of CDCL

1. p_1
2. p_1, p_2
3. $p_1, p_2, \neg p_3, p_5$
4. p_1
5. $p_1, \neg p_3$
6. $p_1, \neg p_3, \neg p_5$
7. $p_1, \neg p_3, \neg p_5, p_4$
8. p_1
9. $p_1, \neg p_4$
10. $p_1, \neg p_4, \neg p_2$

Now consider clause $\neg p_1 \vee p_3 \vee p_4 \vee p_5$ with initial watched literals $\neg p_1$ and p_3. Give the watched literals in the clause after each of the above partial models.

End of Lecture 7