Automated Reasoning 2020

Lecture 6: CDCL - optimizations

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2020-09-05

Review of CDCL

- CNF input
- Decision, propagation, conflict, and backtracking
- Clause learning from conflict
- Clause minimization : first UIP strategy

Other heuristics

More heuristics that may improve the performance of SAT solvers

- 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 (extra topic)

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

Topic 6.1

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 6.1
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 6.1

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

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

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

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 6.2

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

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

Exercise: execute 2-watched literals

Exercise 6.3

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.

Detecting and assigning pure literals

ℓ may be assigned 1 immediately.

Definition 6.1

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 6.4

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 6.2

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 6.2

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 6.5

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.

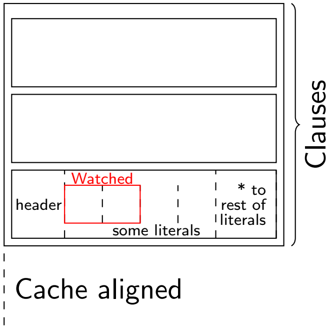
Example 6.3

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.

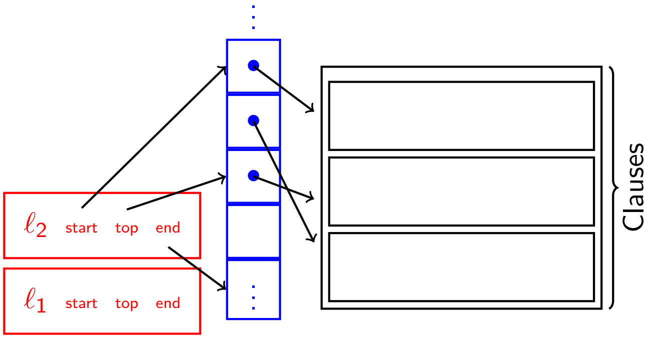
Binary and ternary clauses stored separately, since they become unit clauses too often

- $Occurs2$: 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 6.3

Runtime choices

Runtime choices

Let us study the following runtime choices available to the solvers

1. Variable ordering
2. Restarts
3. Learned clause deletion
4. Phase saving

Decision ordering

After each backtrack, we may choose a different order of assignment.

There are many proposed strategies for the decision order.

Desired property: allow different order after backtracking and less overhead

The following are two widely used strategies:

1. Select a literal with maximum occurrences in unassigned clauses
2. Variable state independent decaying sum

Very popular

Exercise 6.6

What is the policy in Z3?

Variable state independent decaying sum(VSIDS)

Each literal has a score. The highest-scored unassigned literal is the next decision, tie is broken randomly

- Initial score is the number of occurrences of the literals
- Score of a literal is incremented whenever a learned clause contains it
- In regular intervals, divide the scores by a constant (loop over all the variables)

decay

VSIDS is almost deterministic. Some solvers occasionally make random decisions to get out of potential local trap.

Exercise 6.7

Find the used decay rate, increment value, and the interval of update of scores in a solver.

Commentary: Variable state independent decaying sum gives greater weight to the occurrence in the later learned clauses. In some implementations, the weights of variables that appear in the conflict graph after the cut are also incremented.

VSIDS is effective

The principle of exploitation and exploration

- exploitation : decide literals that have participated in conflicts (local search)
- exploration : due to the decay, move on to search somewhere else (global search)

Exercise 6.8

Can we reduce effort of resizing scores of all the variables?

Commentary: Variants of VSIDS: [EènS örensson03/06]
Following the similar principal: Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Conflict-history Based Branching Heuristic for SAT Solvers. AAAI 2016

Restart

SAT solvers are likely to get trapped in a local search space.

Solution: restart CDCL with a different variable ordering

- Keep learned clauses across restarts
- Increase the interval of restarts such that tool becomes a complete solver
- To avoid trap of long restarts, we need a strategy to keep having short restarts

Exercise 6.9

Suggest a design of a parallel sat solver.

Heavy tail restarts

Heavy tail behavior : often short restarts, but with significant chances of long restarts

Example 6.4

Let us consider a well know strategy called Luby restart [LubySinclairZuckerman93].

Let u be a unit number of conflicts. At ith run, we restart after $t_i u$, where

$$t_i = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \leq i < 2^k - 1. \end{cases}$$

Exercise 6.10

a. Plot t_i for first 70 points.

b. Show that v_n in the following reluctant doubling sequence is equal to t_n.[Knuth'12]

$(u1, v1) = (1, 1)$ *and* $(u_{n+1}, v_{n+1}) = (u_n \& - u_n) = v_n ? (u_n + 1, 1) : (u_n, 2v_n)$

Learned clause deletion

CDCL may learn a lot of clauses.

The solvers time to time delete some learned clauses.

The solver remains sound with deletions. However, the completeness may be compromised.

For completeness, reduce deletion of clauses over time.

Exercise 6.11

After learning how many clauses, we should start deleting?

(estimate via common sense; Imaging yourself in an interview!!!)

`https://arxiv.org/pdf/1402.1956.pdf` Gluecose `http://www.ijcai.org/Proceedings/09/Papers/074.pdf`

Deletion strategies

A solver may adopt a combination of the following choices.

Which clauses to delete?

- Delete long clauses with higher probability
- Never delete binary clauses
- Never delete active clauses, i.e., are participating in unit propagation

When to delete?

- At restart
- After crossing a threshold of number of the learned clauses; clauses involved in unit propagation can not be deleted

Deletion measure for clauses : Literal block distance

Definition 6.2

Literal block distance(LBD) = number of decisions in a learned clause

Larger LBD implies more likely to be deleted

LBD is a popular technique

Phase saving

- During CDCL run, the partial assignments satisfies a part of formula
- After restarts, we may want to use the same last partial assignment
- We save the last assigned phase of a variable. In future decisions, we use the same phase.
- Works well with rapid restarts

SAT solving: algorithm, science, or art

Algorithm:
We can not predict the impact of the optimizations based on theory. The current theoretical understanding is limited.

Science:
We need to run experiments to measure the performance.

Art:
Only SAT solving elders can tell you what strategy of solving is going to work on a new instance.

Latest trends in SAT solving

- Portfolio solvers
- Machine learned solver configuration
- Optimizations for applications, e.g., maxsat, unsatcore, etc.
- solving cryptography constraints

Exercise 6.12

Visit the latest SAT conference website. Read a paper and write a comment(400 chars max).

Topic 6.4

Problems

SAT solver for the following problems

- Please download the following SAT solver

  `https://github.com/arminbiere/cadical`

- Install the solver as instructed in the source code.

- Bench mark: Download main Track instances:

  `https://satcompetition.github.io/2020/downloads.html`

- The goal of the following problems would be to change the parameters regarding certain optimizations and draw the cactus plot for various choices of the parameters. (`-h` option will give you descriptions of the parameters.)

Play with exponential VSIDS (EVSIDS)

Exercise 6.13

Please modify the following parameters related to EVSIDS and draw cactus plot.

```
    score, scorefactor
```

Please follow the instructions at the start of problem section to do the above exercise.

Play with restarts

Exercise 6.14

Please modify the following parameters related to restarts and draw cactus plot.

```
restart, restartint, restartmargin, restartreusetrail,
reluctant, reluctantmax
```

Please follow the instructions at the start of problem section to do the above exercise.

Play with clause deletion

Exercise 6.15

Please modify the following parameters related to clause deletion and draw cactus plot.

```
    reduce, reduceint, reducetarget, reducetier1glue, reducetier2glue,
    emagluefast, emaglueslow
```

Please follow the instructions at the start of problem section to do the above exercise.

Play with phase saving

Exercise 6.16

Please modify the following parameters related to phase saving and draw cactus plot.

```
forcephase, phase, rephase, rephaseint,
stabilize, stabilizefactor, stabilizeint, stabilizemaxint,
stabilizeonly, stabilizephase,
```

Please follow the instructions at the start of problem section to do the above exercise.

Play with chronological backtracking (Clause learning)

Exercise 6.17

Please modify the following parameters related to chronological backtracking and draw the cactus plot.

```
chrono, chronoalways, chronolevelim, chronoreusetrail,
```

Please follow the instructions at the start of problem section to do the above exercise.

Topic 6.5

Extra slides : pre(in)-processing

Pre(in)-processing

Simplify input before CDCL

- Eliminate tautologies/Unit clauses/Pure literal elimination
- Subsumption/Self-subsuming resolution
- Blocked clause elimination
- Literal equivalence
- Bounded variable elimination/addition
- Failed literal probing/Vivification
- Stamping
- ....

http://theory.stanford.edu/~barrett/summerschool/soos.pdf

Source of Lingeling (http://fmv.jku.at/lingeling/)

Obvious eliminations

- Eliminate tautologies
  - Remove clauses like $p_1 \lor \neg p_1 \lor ....$
- Assign unit clauses
  - Unit propagation at 0th decision level.
- Pure literal elimination
  - Remove all the clauses that contain the literal

Exercise 6.18

a. What is the cost of eliminating tautologies?

b. What is the cost of pure literal elimination?

Commentary: Sorted clause make tautology detection efficient. Pre-computing of occurrence while parsing helps identifying pure literals.

Subsumption

Remove clause C' if $C \subset C'$ is present.

- Use backward subsumption: for a C search for weaker clauses
- Only search using short C
- Iterate over the occurrence list of the literal in C that has the smallest occur size.
- Containment check is sped up using bloom filter.

Example 6.5

$p \vee q \vee r$ is a redundant clause if $p \vee q$ is present.

Subsumption algorithm

The fingerprint used in Lingeling for Bloom filter.

$$fingerPrint(C) = |_{\ell \in C}(1 << (atom(\ell)\&31))$$

$atom(\ell)$ returns the atom in literal ℓ.

Algorithm 6.1: Subsumption(F)

```
for $C \in F$ such that $|C| < shortLimit$ do
    $sigC := fingerPrint(C)$;
    $\ell :=$ literal in $C$ with smallest $|OccurList(\ell)|$;
    for $C' \in OccurList(\ell)$ such that $C' \neq C$ do
        if $sigC$ ?? $fingerPrint(C')$ then
            if $C \subset C'$ then
                $F := (F - \{C'\})$;
```

Exercise 6.19

Complete the missing operator '??'.

Self-subsumption (Strengthening)

Replace clause $C' \vee \ell$ by C' if for some $C \subset C'$, $C \vee \neg\ell$ is present.

Example 6.6

$p \vee q \vee r \vee \neg s$ should be replaced by $p \vee q \vee r$ if $r \vee s$ is present.

Algorithm 6.2: SelfSubsumption(F)

```
for $C \in F$ such that $|C| < shortLimit$ do
    $sigC := fingerPrint(C)$;
    $\ell :=$ literal in $C$ with smallest $|OccurList(\ell) \cup OccurList(\neg\ell)|$;
    for $C' \in OccurList(\ell) \cup OccurList(\neg\ell)$ such that $C' \neq C$ do
        if $sigC$ ?? $fingerPrint(C')$ then
            if $D' \vee \neg\ell' = C'$ and $D \vee \ell' = C$ and $D \subset D'$ then
                $F := (F - \{C'\}) \cup D'$;
```

Commentary: Same answer for ?? as in the previous slide.

Blocked clause elimination

Now, we will look at a more general condition than pure literal to remove clauses.

called blocking literal

Definition 6.3

A clause $C \in F$ is a blocked clause in F, if there is a literal $\ell \in C$ such that for each $C' \in F$ with $\neg\ell \in C'$, there is a literal ℓ' such that $\ell' \in C$ and $\neg\ell' \in C' \setminus \{\neg\ell\}$.

claim:

We can safely disable blocked clauses, without affecting satisfiability.

Example: blocked clause elimination

Example 6.7

In the following clauses, p_1 is a blocking literal in the blocking clause C_1.

$C_1 = (p_1 \lor p_2 \lor \neg p_3) \land$
$C_2 = (\neg p_3 \lor \neg p_2) \land$
$C_3 = (\neg p_1 \lor \neg p_2) \land$
$C_4 = (p_1 \lor \neg p_5) \land$
$C_5 = (\neg p_1 \lor p_3 \lor p_4)$

Only, C_3 and C_5 contain $\neg p_1$.

$p_3 \in C_5$ is helping p_1 to become blocked literal in C_1, since negation of p_3 is present in C_1.

Exercise 6.20

Which literal in C_3 helping p_1 to become blocked literal in C_1?

Soundness of blocking clause elimination

Theorem 6.1

If C is a blocking clause in F, then F and $F \setminus C$ are equisatisfiable.

Proof.

Wlog, let $C = \ell_1 \vee \cdots \vee \ell_k$ and ℓ_1 be the blocking literal.
Let us suppose $m \models F \setminus C$ and $m \not\models C$, otherwise proof is trivial.
Therefore, $m(\ell_i) = 0$.

claim: $m[\ell_1 \mapsto 1] \models F$
Choose $C' \in F$. Now three cases.

1. $\neg\ell_1 \in C'$: there is ℓ_i for $i > 1$(why?) such that $\ell_i \in C$ and $\neg\ell_i \in C'$.
   Since $m(\ell_i) = 0$, $m[\ell_1 \mapsto 1] \models C'$.(why?)
2. $\ell_1 \in C'$: Since $m[\ell_1 \mapsto 1] \models C'$, $m[\ell_1 \mapsto 1] \models C'$.
3. $\{\ell_1, \neg\ell_1\} \cap C' = \emptyset$: trivial.(why?)

□

Commentary: A näive implementation will be inefficient. To find an efficient implementation, read `http://fmv.jku.at/papers/JarvisaloBiereHeule-TACAS10.pdf`

Vivification

Let us suppose $C = (\ell_1 \vee ... \vee \ell_n) \in F$ and

$$\textsc{UnitPropagation}(\emptyset, F \wedge \neg\ell_1 \wedge \dots \wedge \neg\ell_{i-1} \wedge \neg\ell_{i+1} \wedge \dots \wedge \neg\ell_n)$$

results in conflict.

We replace C in F by $C \setminus \{\ell_{i-1}\}$.

Implemented in many state of the art solvers.

Topic 6.6

More problems

Compaction

The pre-processing changes the set of variables and clauses.

Before running CDCL,

- the solvers rename all the variables with contiguous numbers and
- clause lists are also compacted.

This increases cache locality, and fewer cache misses.

Play with vivifications

Exercise 6.21
Please modify the following parameters related to vivification and draw cactus plot.

```
vivify, vivifymaxeff, vivifymineff, vivifyonce, vivifyredeff, vivifyreleff
```

Please follow the instructions at the start of problem section of the previous lecture to do the above exercise.

Play with failed literal probing

Exercise 6.22

Please modify the following parameters related to failed literal probing and draw the cactus plot.

```
probe, probehbr, probeint, probemaxeff, probemineff, probereleff, proberounds
```

Please follow the instructions at the start of problem section of the previous lecture to do the above exercise.

Play with blocked clause elimination

Exercise 6.23

Please modify the following parameters related to blocked clause elimination and draw the cactus plot.

```
block, blockmaxclslim, blockminclslim, blockocclim,
```

Please follow the instructions at the start of problem section of the previous lecture to do the above exercise.

End of Lecture 6