Automated Reasoning 2018

Lecture 5: CDCL - optimizations

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2018-07-31

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 structure
  - ▶ 2-watched literals
  - ▶ pure literals
- ▶ Other optimizations
  - ▶ variable ordering
  - ▶ restarts
  - ▶ learned clause deletion
  - ▶ cache aware implementation
- ▶ pre-processing

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

Topic 5.1

Lazy data structures

Data stracture to keep the formulas

▶ Variables are contiguous numbers
  ▶ Variable number are used as index to the data structures
  ▶ Positive integer is the positive literal
  ▶ Negative integer is the negated literal

▶ Current assignment a list of literals

▶ Occurrence map *OccurList* : literals → clauses
  ▶ Stored as array or arrays
  ▶ Binary and ternary clauses stored exclusively, since they become unit clauses too often

Exercise 5.1
What is the maximum number of variables allowed in the design?

Detetecting Unit clauses

Naive procedure:

- ▶ For each unassigned clause count unassigned literals
- ▶ 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.,
    we need to look at only two literals that are not false.

Let us look at a data structure the optimizes the unit clause propagation using the insight.

2-watched literals for detecting unit clauses

For each clause we choose 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 look for another two non-false literals
- ▶ If we can not find another two, the clause is a unit clause

Exercise 5.2
Why this scheme may reduce the effort in searching for the unit clauses?

Example: 2-watched literals

Example 5.1

Consider clause $p_1 \vee p_2 \vee \neg p_3 \vee \neg p_4$ in a formula among other variables and clauses. Let us suppose initially we watch p_1 and p_2 in the clause.

$^* \triangleq$ *watched literals.*
$\smiley \triangleq$ *no work to be done!*

Initially: $m = \{\}$ $p_1^* \vee p_2^* \vee \neg p_3 \vee \neg p_4$

$\vdots$ $\vdots$

Assign $p_1 = 0$: $m = \{\ldots, p_1 \mapsto 0\}$ $p_1 \vee p_2^* \vee \neg p_3^* \vee \neg p_4$ *(work)*

Assign $p_2 = 1$: $m = \{\ldots, p_1 \mapsto 0, p_2 \mapsto 1\}$ $p_1 \vee p_2^* \vee \neg p_3^* \vee \neg p_4$ $\smiley$

Backtrack to p_1: $m = \{\ldots\}$ $p_1 \vee p_2^* \vee \neg p_3^* \vee \neg p_4$ $\smiley$

Assign $p_4 = 1$: $m = \{\ldots, p_4 \mapsto 1\}$ $p_1 \vee p_2^* \vee \neg p_3^* \vee \neg p_4$ $\smiley$

> The benefit: often no work to be done!

Data structure for 2-watched literals

- A map from literals to a list of clauses where the literal is watched

- If a literal becomes false, one of the following happens in a clause where it is watched
    - the clause has become a unit clause
    - conflict has occurred
    - the clause is moved to the other literals in the clause for watching

    > Only in the last case, the data structure changes.

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

Exercise 5.3
Is this idea extendable for counting 0, 1, 2, and many?

Detecting pure literals

Definition 5.1
A literal ℓ is called pure *in F if $\bar{\ell}$ does not occur in F.*

Benefit: ℓ may be assigned 1 immediately.

As DPLL proceeds, more and more literals may become pure literals.(why?)
We may assign them similar to unit clause propagation.

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

▶ In each step, 2-watched literal optimization only visits those clauses that have literals that are just assigned and watched

▶ Adding such data structure will defeat the benefit of 2-watched literal.

> 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 ensure to check if some literal is still active.

Topic 5.2

Other design choices
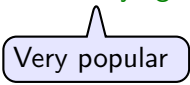
Decision ordering

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 5.4
What is the policy in Z3? Choose a solver and find the policy in the solver?

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

decay

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

Exercise 5.5
Characterize the scheme? Why may this scheme be effective?

Commentary: Variable state independent decaying sum gives greater weight to the occurrence in the later learned clauses.

Restart

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

Solution: restart CDCL with a different variable ordering
- ► Keep learned clauses across restarts
- ► Slowly increase the interval of restarts such that tool becomes a complete solver (various strategies in the literature.)

Exercise 5.6
Suggest a design of a parallel sat solver.

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 5.7
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 strategy

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

Cache aware CDCL

SAT solvers are memory intensive.

The implementation should try to make localized accesses.

- ▶ Clause headers and upto four literals are stored together
- ▶ Pre allocate clauses in bulk to avoid system overhead
- ▶ Clauses should be aligned with cache line
- ▶ Use only 2 bits to store state (true, false, and unassigned)

http://www.easychair.org/publications/download/Towards_Improving_the_Resource_Usage_of_SAT-solvers

SAT solving: algorithm, science, or art

Algorithm:
We can not predict the impact of the optimizations based on the theory. At least 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 of SAT.

Topic 5.3

Pre-processing

Pre-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
- ▶ Stamping
- ▶ ....

https://cs.nyu.edu/~barrett/summerschool/soos.pdf

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

Obvious eliminations

▶ Eliminate tautologies
  ▶ Remove clauses like $p_1 \vee \neg p_1 \vee ....$

▶ Assign unit clauses
  ▶ Unit propagation at 0th decision level.

▶ Pure literal elimination
  ▶ Remove all the clauses that contain the literal

Exercise 5.8
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 5.2

$p \vee q \vee r$ 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 5.1: 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 5.9

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 5.3

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

Algorithm 5.2: 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.

Definition 5.2

called blocking literal

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 5.4

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 5.10

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

Soundness of blocking clause elimination

Theorem 5.1
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. $\{\ell, \neg\ell\} \cap C' = \emptyset$: trivial.(why?) □

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.

Latest trends in SAT solving

- ▶ Portfolio solvers
- ▶ Optimizations for applications, e.g., maxsat, unsatcore, etc.
- ▶ solving cryptography constraints

Exercise 5.11

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

End of Lecture 5