Conflict Driven Clause Learning (CDCL)

The CDCL calculus tests satisfiability of a finite set N of propositional clauses.

I assume that $\bot \not\in N$ and that the clauses in N do not contain duplicate literal occurrences. Furthermore, duplicate literal occurrences are always silently removed during rule applications of the calculus. (Exhaustive Condensation.)

The CDCL calculus explicitely builds a candidate model for a clause set. If such a sequence of literals $L_1, \ldots, L_n$ satisfies the clause set N, it is done. If not, there is a false clause $C \in N$ with respect to $L_1, \ldots, L_n$.

Now instead of just backtracking through the literals $L_1, \ldots, L_n$, CDCL generates in addition a new clause, called *learned clause* via resolution, that actually guarantees that the subsequence of $L_1, \ldots, L_n$ that caused C to be false will not be generated anymore.

This causes CDCL to be exponentially more powerful in proof length than its predecessor DPLL or Tableau.

CDCL State

A CDCL problem state is a five-tuple $(M; N; U; k; D)$ where

M a sequence of annotated literals, called a *trail*,

N and U are sets of clauses,

$k \in \mathbb{N}$, and

D is a non-empty clause or $\top$ or $\bot$, called the *mode* of the state.

The set N is initialized by the problem clauses where the set U contains all newly learned clauses that are consequences of clauses from N derived by resolution.

Modes of CDCL States

$(\epsilon; N; \emptyset; 0; \top)$	is the start state for some clause set N
$(M; N; U; k; \top)$	is a final state, if $M \models N$ and all literals from N are defined in M
$(M; N; U; k; \bot)$	is a final state, where N has no model
$(M; N; U; k; \top)$	is an intermediate model search state if $M \not\models N$
$(M; N; U; k; D)$	is a backtracking state if $D \not\in \{\top, \bot\}$

The Role of Levels

Literals in $L \in M$ are either annotated with a number, a level, i.e., they have the form L^k meaning that L is the k^{th} guessed decision literal, or they are annotated with a clause that forced the literal to become true.

A literal L is of *level* k with respect to a problem state $(M; N; U; j; C)$ if L or $\mathrm{comp}(L)$ occurs in M and L itself or the first decision literal left from L ($\mathrm{comp}(L)$) in M is annotated with k. If there is no such decision literal then $k = 0$.

A clause D is of *level* k with respect to a problem state $(M; N; U; j; C)$ if k is the maximal level of a literal in D.

CDCL Rules

Propagate $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (ML^{C \vee L}; N; U; k; \top)$
provided $C \vee L \in (N \cup U)$, $M \models \neg C$, and L is undefined in M

Decide $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (ML^{k+1}; N; U; k+1; \top)$
provided L is undefined in M

Conflict $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (M; N; U; k; D)$
provided $D \in (N \cup U)$ and $M \models \neg D$

Skip $(ML^{C\lor L}; N; U; k; D) \Rightarrow_{\text{CDCL}} (M; N; U; k; D)$
provided $D \notin \{\top, \bot\}$ and $\text{comp}(L)$ does not occur in D

Resolve $(ML^{C\lor L}; N; U; k; D \lor \text{comp}(L)) \Rightarrow_{\text{CDCL}}$
$(M; N; U; k; D \lor C)$
provided D is of level k

Backtrack $(M_1 K^{i+1} M_2; N; U; k; D \lor L) \Rightarrow_{\text{CDCL}}$
$(M_1 L^{D\lor L}; N; U \cup \{D \lor L\}; i; \top)$
provided L is of level k and D is of level i.

Restart $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (\epsilon; N; U; 0; \top)$
provided $M \not\models N$

Forget $(M; N; U \uplus \{C\}; k; \top) \Rightarrow_{\text{CDCL}} (M; N; U; k; \top)$
provided $M \not\models N$

2.9.5 Definition (Reasonable CDCL Strategy)

A CDCL strategy is *reasonable* if the rules Conflict and Propagate are always preferred over all other rules.

2.9.6 Proposition (CDCL Basic Properties)

Consider CDCL run deriving $(M; N; U; k; C)$ by any strategy but without Restart and Forget. Then the following properties hold:

1. M is consistent.
2. All learned clauses are entailed by N.
3. If $C \notin \{\top, \bot\}$ then $M \models \neg C$.
4. If $C = \top$ and M contains only propagated literals then for each valuation $\mathcal{A}$ with $\mathcal{A} \models N$ it holds that $\mathcal{A} \models M$.
5. If $C = \top$, M contains only propagated literals and $M \models \neg D$ for some $D \in (N \cup U)$ then N is unsatisfiable.
6. If $C = \bot$ then CDCL terminates and N is unsatisfiable.
7. k is the maximal level of a literal in M.
8. Each infinite derivation contains an infinite number of Backtrack applications.

2.9.7 Lemma (CDCL Redundancy)

Consider a CDCL derivation by a reasonable strategy. Then CDCL never learns a clause contained in $N \cup U$.

2.9.10 Lemma (CDCL Soundness)

In a reasonable CDCL derivation, CDCL can only terminate in two different types of final states: $(M; N; U; k; \top)$ where $M \models N$ and $(M; N; U; k; \bot)$ where N is unsatisfiable.

2.9.11 Proposition (CDCL Soundness)

The rules of the CDCL algorithm are sound: (i) if CDCL terminates from $(\epsilon; N; \emptyset; 0; \top)$ in the state $(M; N; U; k; \top)$, then N is satisfiable, (ii) if CDCL terminates from $(\epsilon; N; \emptyset; 0; \top)$ in the state $(M; N; U; k; \bot)$, then N is unsatisfiable.

2.9.12 Proposition (CDCL Strong Completeness)

The CDCL rule set is complete: for any valuation M with $M \models N$ there is a reasonable sequence of rule applications generating $(M'; N; U; k; \top)$ as a final state, where M and M' only differ in the order of literals.

2.9.13 Proposition (CDCL Termination)

Assume the algorithm CDCL with all rules except Restart and Forget is applied using a reasonable strategy. Then it terminates in a state $(M; N; U; k; D)$ with $D \in \{\top, \bot\}$.

The Overall Picture

Application System + Problem
System Algorithm + Implementation
Algorithm Calculus + Strategy
Calculus Logic + States + Rules
Logic Syntax + Semantics

Algorithm: 5 CDCL(S)

Input : An initial state $(\epsilon; N; \emptyset; 0; \top)$.
Output : A final state $S = (M; N; U; k; \top)$ or $S = (M; N; U; k; \bot)$

```
while (any rule applicable) do
    ifrule (Conflict(S)) then
        while (Skip(S) ‖ Resolve(S)) do
            update VSIDS on resolved literals;
        update VSIDS on learned clause, Backtrack(S);
        if (forget heuristic) then
            Forget(S), Restart(S);
        else
            if (restart heuristic) then
                Restart(S);
    else
        ifrule (!Propagate(S)) then
```

Implementation: Data Structures

Propagate $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (ML^{C \vee L}; N; U; k; \top)$

provided $C \vee L \in (N \cup U)$, $M \models \neg C$, and L is undefined in M

Conflict $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (M; N; U; k; D)$

provided $D \in (N \cup U)$ and $M \models \neg D$

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Data Structures

Idea: Select two literals from each clause for indexing.

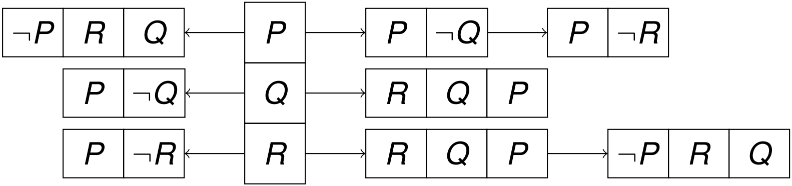
Data Structures

Idea: Select two literals from each clause for indexing.

2.10.1 Invariant (2-Watched Literal Indexing)

If one of the watched literals is false and the other watched literal is not true, then all other literals of the clause are false.

$$N = \{P \lor \neg R,\ P \lor \neg Q,\ R \lor Q \lor P,\ \neg P \lor R \lor Q\}$$



VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit *d* on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit *d* on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, . . .
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \& -u_n) = v_n ? (u_n + 1, 1) : (u_n, 2 * v_n))$

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, ...
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \,\&\, -u_n) = v_n ? (u_n + 1, 1) : (u_n, 2 * v_n))$

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, . . .
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \,\&\, -u_n) = v_n ? (u_n + 1, 1) : (u_n, 2 * v_n))$

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, ...
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \mathbin{\&} - u_n) = v_n ?(u_n + 1, 1) : (u_n, 2 * v_n))$

Memory Matters: SPASS-SATT

Forget-Start	800	108800
Restarts	412	369
Conflicts	153640	133403
Decisions	184034	159005
Propagations	17770298	15544812
Time	11	23
Memory	16	41