Overview

Introduction

Tractable cases

DPLL algorithms

CDCL solvers

Probabilistic algorithms

Lookahead-based solvers
  Lookahead solvers
  Cube-and-Conquer

Applications
DPLL with look-ahead

The basic DPLL Algorithm:

DPLL($F$)

1. if $F = 1$ then return SAT

2. $(F, x) := \text{LookAhead}(F)$

3. if $\Box \in F$ then return UNSAT

4. else if $x = NULL$ then return DPLL($F$)

5. $b := \text{DirectionHeur}(x, F)$

6. if DPLL($F[x := b]$) = SAT

7. then return SAT

8. else return DPLL($F[x := (1 - b)]$)
The look-ahead procedure

\textbf{LookAhead}(F)

\[ P := \text{PreSelect}(F) \]

\text{for } x \in V(F) \text{ for } x \in P

\[ F_0 := \text{UnitProp}(F[x := 0]) \]
\[ F_1 := \text{UnitProp}(F[x := 1]) \]

if $\Box \in F_0$ and $\Box \in F_1$ then return $(F_0, \text{NULL})$
else if $\Box \in F_0$ then $F := F_1$
else if $\Box \in F_1$ then $F := F_0$
else $H(x) := \text{DecisionHeur}(F, F_0, F_1)$

\[ x_m := \arg \max \{ H(x) ; y \in V(F) P \} \]

return $(F, x_m)$
Decision heuristics

Decision heuristic is computed using a difference heuristic:

\[
\text{DECISIONHEUR}(F, F_0, F_1)
\]

\[
H_0 := \text{DIFF}(F, F_0)
\]
\[
H_1 := \text{DIFF}(F, F_1)
\]

return MIXDIFF\((H_0, H_1)\)

Commonly used:

- \(\text{MIXDIFF}(L, R) = L \cdot R\)
- \(\text{MIXDIFF}(L, R) = 1024 \cdot L \cdot R + L + R\)
Difference heuristics

Heuristic based on reduction in variables:

\[ \text{DIFF}(F, F') = |V(F) \setminus V(F')| \]

Heuristic based on reduced clauses, weighted by width (OKsolver):

\[ F_k: \text{ the set of clauses of width } k \text{ in } F. \]

\[ \text{DIFF}(F, F') = \sum_{k \geq 2} \gamma_k \cdot |F_k' \setminus F| \]

\( \gamma_k \) are weights resulting from experimental optimization.

\[ \gamma_2 = 1, \quad \gamma_3 = 0.2, \quad \gamma_4 = 0.05, \quad \gamma_5 = 0.01, \quad \gamma_6 = 0.003, \ldots \]
Difference heuristics based on new 2-clauses

\[ h_k(a) = \text{number of } k\text{-clauses in which } a \text{ occurs} \]
\[ w(a) = \sum_{k \geq 2} \gamma_k \cdot h_k(a) \]

Weighted Binaries Heuristic (satz):
\[ \text{Diff}_{WBH}(F, F') = \sum_{(a \lor b) \in F_2' \setminus F} w(\overline{a}) + w(\overline{b}) \]
Weights are \( \gamma_k = 5^{-(k+3)} \)

Backbone Search Heuristic (kcnfs):
\[ \text{Diff}_{BSH}(F, F') = \sum_{(a \lor b) \in F_2' \setminus F} w(\overline{a}) \cdot w(\overline{b}) \]
Weights are \( \gamma_k = 2^{-(k+3)} \)
Direction heuristics

Heuristic based on $\text{DIFF}$:

$$b = 1 \text{ iff } \text{DIFF}(F, F[x = 1]) > \text{DIFF}(F, F[x = 0])$$

Heuristic based on occurrences:

$$b = 1 \text{ iff } \#(x) > \#(\overline{x})$$

Refined heuristic based on occurrences:

$$k_0 := \min\{w(C) ; C \in F\}$$

$$b = 1 \text{ iff } \#_{k_0}(x) > \#_{k_0}(\overline{x})$$
Preselection heuristics

Heuristic based on occurrence in binary clauses (satz, kcnfs)

▶ At small branching depth, select \( P := V(F) \).

▶ At larger branching depth, select

\[
P := \{ x \in V(F); h_2(x) > 0 \text{ and } h_2(\bar{x}) > 0 \}
\]

Clause reduction approximation (march):

▶ For every \( x \in V(F) \), compute a score

\[
cra(x) := \left( \sum_{(x \lor y) \in F} h_{\geq 2}(y) \right) \cdot \left( \sum_{(\bar{x} \lor y) \in F} h_{\geq 2}(y) \right)
\]

▶ Select the 10\% variables \( x \) with largest \( cra(x) \).
Further optimizations

Local learning:

During look-ahead on $x$ unit $y^c$ detected:

- due to binary clause $\bar{x} \lor y^c$: direct implication
- otherwise: add binary clause $\bar{x} \lor y^c$
- has to be removed on backtracking

Necessary assignments:

Variable $y$ assigned $[y \leftarrow \epsilon]$ during look-ahead on $x$ and during look-ahead on $\bar{x}$

$\Rightarrow$ necessary assignment $[y \leftarrow \epsilon]$ kept.
Combining Look-Ahead and CDCL: Cube-and-Conquer

Idea for combination:

- Use DPLL/Look-Ahead to split formula into smaller subproblems
- Subproblem = formula $F$ plus assumption $\gamma$
  assumption (or cube) is assignment $\gamma$
- Solve subproblems by CDCL solver

Cutoff heuristic determines when to pass subproblem to CDCL

Assumption $\gamma$ passed as unit clauses
$\rightarrow$ learned clauses useless for other subproblems

Better: use incremental CDCL solver
Incremental SAT solving

Incremental solver:

- called with $F$, an assumption $\gamma$ and a set $C$ of clauses
- clauses $C$ added to database of learned clauses
- $\gamma$ is set as assignments at level 0
  - $\leadsto$ learned clauses valid under other assumptions
- when UNSAT, returns subset $\gamma' \subseteq \gamma$ sufficient for unsatisfiability plus some learned clauses $C'$
- $C'$ and $\neg \gamma$ are passed as clauses to next iteration
Creating Cubes

Look-Ahead procedure returns set $A$ of assumptions and a set $C$ of clauses.

- For the current assignment $\alpha$, let $\alpha_{\text{dec}} \subseteq \alpha$ be the decisions in $\alpha$
- At a conflict, add $\neg \alpha_{\text{dec}}$ to $C$
- When cutoff heuristic is triggered, add $\alpha_{\text{dec}}$ to $A$.

Incremental CDCL solver is called with clauses $C$ and assumption $\gamma$, for each $\gamma \in A$. 
Cutoff heuristic

Simple cutoff heuristics:

- number of variables assigned $|\alpha|$
- number of decisions $|\alpha_{dec}|$

Better, dynamic heuristic:

- Cutoff when $|\alpha_{dec}| \cdot |\alpha| > \theta$
- Initially set e.g. $\theta := 1000$
- At a conflict, reduce $\theta$ by 30%
- Also reduce $\theta$ if $\alpha_{dec}$ gets too large.
- At each recursive call, increase $\theta$ by 5%
The calls of CDCL on different assumptions $\gamma$ can be paralleleized $\Rightarrow$ no benefit from learned clauses.

For very hard problems, a two-stage approach is used:

- First stage partitions into subproblems that are solved in parallel
- Each of these is again solved by Cube-and-Conquer
- I.e., each subproblem further partitioned by Look-Ahead
- The obtained sub-subproblems are solved by sequential, incremental CDCL