Overview

DPLL with look-ahead

The basic DPLL Algorithm:

```
DPLL(F)
  if F = 1 then return SAT
  (F, x) := LookAhead(F)
  if □ ∈ F then return UNSAT
     else if x = NULL then return DPLL(F)
  b := DirectionHeur(x, F)
  if DPLL(F[x := b]) = SAT
     then return SAT
     else return DPLL(F[x := (1 − b)])
```

The look-ahead procedure

$\textsc{LookAhead}(F)$

$P := \textsc{PreSelect}(F)$

$\texttt{for } x \in V(F) \texttt{for } x \in P$

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

$\texttt{if } \square \in F_0 \texttt{ and } \square \in F_1 \texttt{ then return } (F_0, \texttt{NULL})$
$\texttt{else if } \square \in F_0 \texttt{ then } F := F_1$
$\texttt{else if } \square \in F_1 \texttt{ then } F := F_0$

$\texttt{else } H(x) := \textsc{DecisionHeur}(F, F_0, F_1)$

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

$\texttt{return } (F, x_m)$

Decision heuristics

Decision heuristic is computed using a difference heuristic:

$\textsc{DecisionHeur}(F, F_0, F_1)$

$H_0 := \textsc{Diff}(F, F_0)$
$H_1 := \textsc{Diff}(F, F_1)$

`return` $\textsc{MixDiff}(H_0, H_1)$

Commonly used:

- $\textsc{MixDiff}(L, R) = L \cdot R$
- $\textsc{MixDiff}(L, R) = 1024 \cdot L \cdot R + L + R$

Difference heuristics

Heuristic based on reduction in variables:

$$\textsc{Diff}(F, F') = |V(F) \setminus V(F')|$$

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

F_k: the set of clauses of width k in F.

$$\textsc{Diff}(F, F') = \sum_{k \geq 2} \gamma_k \cdot |F'_k \setminus F|$$

γ_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

$h_k(a)$ = number of k-clauses in which a occurs

$$w(a) = \sum_{k \geq 2} \gamma_k \cdot h_k(a)$$

Weighted Binaries Heuristic (satz):

$$\textsc{Diff}_{WBH}(F, F') = \sum_{(a \vee b) \in F_2' \setminus F} w(\bar{a}) + w(\bar{b})$$

Weights are $\gamma_k = 5^{-(k+3)}$

Backbone Search Heuristic (kcnfs):

$$\textsc{Diff}_{BSH}(F, F') = \sum_{(a \vee b) \in F_2' \setminus F} w(\bar{a}) \cdot w(\bar{b})$$

Weights are $\gamma_k = 2^{-(k+3)}$

Direction heuristics

Heuristic based on $\textsc{Diff}$:

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

Heuristic based on occurrences:

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

Refined heuristic based on occurrences:

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

$b = 1$ iff $\#_{k_0}(x) > \#_{k_0}(\bar{x})$

Preselection heuristics

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

- At small branching depth, select $P := V(F)$.
- At larger branching depth, select

$$P := \left\{ x \in V(F) \,;\, h_2(x) > 0 \text{ and } h_2(\bar{x}) > 0 \right\}$$

Clause reduction approximation (march):

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

$$cra(x) := \Big(\sum_{(x \vee y) \in F} h_{>2}(y) \Big) \cdot \Big(\sum_{(\bar{x} \vee y) \in F} h_{>2}(y) \Big)$$

- Select the 10% variables x with largest $cra(x)$.

Further optimizations

Local learning:

During look-ahead on x unit y^ϵ detected:

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

Necessary assignments:

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

$\rightsquigarrow$ 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 γ
  assumption (or cube) is assignment γ
- Solve subproblems by CDCL solver

Cutoff heuristic determines when to pass subproblem to CDCL

Assumption γ passed as unit clauses

$\rightsquigarrow$ learned clauses useless for other subproblems

Better: use incremental CDCL solver

Incremental SAT solving

Incremental solver:

- called with F, an assumption γ and a set C of clauses
- clauses C added to database of learned clauses
- γ is set as assignments at level 0
  $\rightsquigarrow$ learned clauses valid under other asumptions
- 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 α, let $\alpha_{dec} \subseteq \alpha$ be the decisions in α
- At a conflict, add $\neg\alpha_{dec}$ to C
- When cutoff heuristic is triggered, add α_{dec} to A.

Incremental CDCL solver is called with clauses C and assumption γ, 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 θ by 30%
- Also reduce θ if α_{dec} gets too large.
- At each recursive call, increase θ by 5%

Solving Cubes in parallel

The calls of CDCL on different assumptions γ can be paralleleized

$\rightsquigarrow$ 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