Decision Procedures and Verification

Martin Blicha

Charles University

12.3.2018
Efficient Data Structures for DPPL-based algorithm

Goal:
- Efficient unit propagation
- Efficient backtracking
Efficient Data Structures for DPPL-based algorithm

- **Goal:**
  - Efficient unit propagation
  - Efficient backtracking

- Be lazy! Don’t do unnecessary work!
Efficient Data Structures for DPPL-based algorithm

- Goal:
  - Efficient unit propagation
  - Efficient backtracking

- Be lazy! Don’t do unnecessary work!
  
- Lazy data structures
  - Head-tail lists (halfway there)
  - Two watched literals
Efficient Data Structures for DPPL-based algorithm

Head-tail lists

- First lazy data structure proposed for SAT; used in SATO solver, ‘97.
- Each clause maintains two references:
  - head and tail
- Each literal maintains two lists of clauses
  - where it is a head and where it is a tail
- When a literal is falsified ⇒ check only clauses in its occurrence lists. Search for an unassigned literal in direction of head (tail):
  - Satisfied literal is encountered ⇒ clause is already satisfied.
  - Unsatisfied literal is found which *is not* the head (tail) ⇒ new tail (head).
  - Unsatisfied literal is found which *is* the head (tail) ⇒ unit clause.
  - Else ⇒ conflict clause.
Efficient Data Structures for DPPL-based algorithm

Head-tail lists

- First lazy data structure proposed for SAT; used in SATO solver, ‘97.
- Each clause maintains two references:
  - head and tail
- Each literal maintains two lists of clauses
  - where it is a head and where it is a tail
- When a literal is falsified ⇒ check only clauses in its occurrence lists. Search for an unassigned literal in direction of head (tail):
  - Satisfied literal is encountered ⇒ clause is already satisfied.
  - Unsatisfied literal is found which is not the head (tail) ⇒ new tail (head).
  - Unsatisfied literal is found which is the head (tail) ⇒ unit clause.
  - Else ⇒ conflict clause.
- Backtracking requires recovering of the references.
Efficient Data Structures for DPPL-based algorithm

Two watched literals

- Improves head-tail lists
- Implemented in CHAFF SAT solver, ‘01.
- Each clause maintains two references:
  - watched literals
- Each literal maintains a list of clauses
  - where it is watched
- When a literal is falsified $\Rightarrow$ check only clauses in its occurrence list. Search for an literal which is not falsified.
  - Satisfied literal is encountered $\Rightarrow$ clause is already satisfied.
  - Unsatisfied literal is found which is not the other watched literal $\Rightarrow$ new watched literal.
  - Unsatisfied literal is found which is the other watched literal $\Rightarrow$ unit clause.
  - Else $\Rightarrow$ conflict clause.
Efficient Data Structures for DPPL-based algorithm

Two watched literals

- Improves head-tail lists
- Implemented in CHAFF SAT solver, ‘01.
- Each clause maintains two references:
  - watched literals
- Each literal maintains a list of clauses
  - where it is watched
- When a literal is falsified ⇒ check only clauses in its occurrence list. Search for an literal which is not falsified.
  - Satisfied literal is encountered ⇒ clause is already satisfied.
  - Unsatisfied literal is found which is not the other watched literal ⇒ new watched literal.
  - Unsatisfied literal is found which is the other watched literal ⇒ unit clause.
  - Else ⇒ conflict clause.
- Backtracking does not require any work!
Watched literals

Running example

\[
\begin{align*}
c_1 &= (x_1 \lor x_2 \lor x_3) \\
c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}
\]

Watched occurrences

\[
\begin{align*}
x_1 &: \{c_1\} \\
x_2 &: \{\} \\
x_3 &: \{c_1\} \\
x_4 &: \{\} \\
\neg x_1 &: \{c_2, c_3\} \\
\neg x_2 &: \{\} \\
\neg x_3 &: \{\} \\
\neg x_4 &: \{c_2, c_3\}
\end{align*}
\]
**Watched literals**

**Running example**

Decide $x_1 \mapsto False$

$\begin{align*}
c_1 &= (x_1 \lor x_2 \lor x_3) \\
c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}$

Watched occurrences

$\begin{align*}
x_1: \{\} \\
x_2: \{c_1\} \\
x_3: \{c_1\} \\
x_4: \{\} \\
\neg x_1: \{c_2, c_3\} \\
\neg x_2: \{\} \\
\neg x_3: \{\} \\
\neg x_4: \{c_2, c_3\}
\end{align*}$
Watched literals

Running example

Decide \( x_2 \mapsto False \)

\[
\begin{align*}
c_1 &= (x_1 \lor x_2 \lor x_3) \\
c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}
\]

Watched occurrences

\[
\begin{align*}
x_1: \{\} \\
x_2: \{c_1\} \\
x_3: \{c_1\} \\
x_4: \{\} \\
\neg x_1: \{c_2, c_3\} \\
\neg x_2: \{\} \\
\neg x_3: \{\} \\
\neg x_4: \{c_2, c_3\}
\end{align*}
\]

Derive \( x_3 \mapsto True \)

\[
\begin{align*}
c_1 &= (x_1 \lor x_2 \lor x_3) \\
c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}
\]

Watched occurrences

\[
\begin{align*}
x_1: \{\} \\
x_2: \{c_1\} \\
x_3: \{c_1\} \\
x_4: \{\} \\
\neg x_1: \{c_2, c_3\} \\
\neg x_2: \{\} \\
\neg x_3: \{\} \\
\neg x_4: \{c_2, c_3\}
\end{align*}
\]
Watched literals

Running example

Easy backtracking ⇒ erase assignment, keep watched occurences.

\[ c_1 = (x_1 \lor x_2 \lor x_3) \]
\[ c_2 = (\neg x_1 \lor x_2 \lor \neg x_4) \]
\[ c_3 = (\neg x_1 \lor x_3 \lor \neg x_4) \]

Watched occurrences

\begin{align*}
  x_1 &: \{\} \\
  x_2 &: \{c_1\} \\
  x_3 &: \{c_1\} \\
  x_4 &: \{\} \\
  \neg x_1 &: \{c_2, c_3\} \\
  \neg x_2 &: \{\} \\
  \neg x_3 &: \{\} \\
  \neg x_4 &: \{c_2, c_3\}
\end{align*}
Decision heuristics

Decision heuristic

*Decision heuristic* in a SAT solver is a strategy by which the variables and the value given to them are chosen.

- Jeroslow-Wang
- Dynamic Largest Individual Sum
- Variable State Independent Decaying Sum
- Berkmin
- Clause-Move-To-Front
- ...
Jeroslow-Wang

- Idea: prefer literals that appear frequently in small clauses.
- Compute a score for each literal as $J(l) = \sum_{C \in \varphi, l \in C} 2^{-|C|}$.
- When making a decision choose a literal with highest score.
- Can be both static and dynamic:
  - static - Fast (single computation at the beginning), but does not reflect how the situation evolves.
  - dynamic - Makes better decisions but also imposes large overhead at the decision point.
Variable State Independent Decaying Sum (VSIDS)

- SAT solver Chaff, 2001
- *conflict driven* heuristic: Gives preferences to literals in newly learned clauses.
- Every literal has a score (based on how many occurrences there are).
- Score of literals in newly learned clauses increases.
- The score is periodically divided by $d > 1$. 
Berkmin

- member of a family of *clause-based* heuristics
- SAT solver \textbf{BerkMin}, 2002
- Score for every variable (divided) and for every literal (not divided)
- Keeps stack of conflict clauses.
- Picks a variable with highest score from unresolved clause from top of the stack.
- Assigns a polarity based on the literal score.
Random restarts

- Inspiration from stochastic search
- Few bad decisions at the beginning get SAT solver stuck in unperspective subtree
- Chance to do better (more informed) decisions at the beginning
- Keep (or not) the learned clauses
- Example of a strategy: Geometric sequence of number of conflicts after which a restart is performed
Preprocessing

- Tries to simplified the set of input clauses
- Applied before the input goes to CDCL algorithm
  - Also at a restart
- Trade-off between time spent in preprocessing and its effect
- Examples:
  - (bounded) variable elimination by clause distribution
  - blocked clause elimination
  - subsumption
  - self-subsumption
SAT solver toolbox overview

- DPLL algorithm
- clause learning
- two watched literals
- decision heuristics
- restarts
- preprocessing
Solvers based on stochastic (local) search
GSAT

- Selman et al., 1992
- Greedy traversing among complete valuations of variables with restarts
- Incomplete

1: procedure GSAT(\(\varphi\), MAX_TRIES, MAX_FLIPS)
2: \hspace{1em} for \(i = 1, 2, \ldots, \text{MAX\_TRIES}\) do
3: \hspace{2em} \(\alpha \leftarrow \) random full assignment
4: \hspace{2em} for \(j = 1, 2, \ldots, \text{MAX\_FLIPS}\) do
5: \hspace{3em} if \(\forall c \in \varphi : c\) is satisfied by \(\alpha\) then return TRUE
6: \hspace{3em} choose \(x \in \text{Var} (\varphi)\) such that flipping polarity of \(x\) leads to the highest number of satisfied clauses
7: \hspace{3em} flip polarity of \(x\) in \(\alpha\)
8: \hspace{1em} return FALSE
**WalkSAT**

- Selman et al., 1994
- Random walk with probability $p$
- Greedy step with probability $1 - p$

1: **procedure** `WalkSAT(ϕ, MAX_TRIES, MAX_FLIPS)`
2: for $i = 1, 2, \ldots, MAX\_TRIES$ do
3: \quad $α \leftarrow$ random full assignment
4: for $j = 1, 2, \ldots, MAX\_FLIPS$ do
5: \quad if $∀c ∈ ϕ : c$ is satisfied by $α$ then return `TRUE`
6: \quad choose $c ∈ ϕ$, random not satisfied clause
7: \quad if RAND($0, 1) < p$ then choose random $x ∈ Var(c)$
8: \quad else choose $x ∈ Var(c)$ such that flipping polarity of $x$ leads to the highest number of satisfied clauses
9: \quad flip polarity of $x$ in $α$
10: return `FALSE`
Message passing algorithms

- Iteratively change value of a variable according to effect of related clauses

**Factor graph** for a CNF formula $\varphi$ is $G_F(\varphi) = (V_F, E_F)$ where:

- $V_F = \text{Var}(\varphi) \cup I$,
- $E_F = \{\{x, i\} \mid x \in \text{Var}(\varphi) \land i \in I \land x \in \text{Var}(c_i)\}$, where $c_i$ is a clause of $\varphi$,

and $I = \{1, 2, \ldots, n\}$ indexes clauses of $\varphi$.

**Variable occurrence indication**

- $J^i_x = 1$ if $x \in \text{Var}(\varphi)$ has a negative occurrence in $c_i$
- $J^i_x = -1$ if $x \in \text{Var}(\varphi)$ has a positive occurrence in $c_i$
- $V(x) = \{i \mid i \in I \land x \in \text{Var}(c_i)\}$ for $x \in \text{Var}(\varphi)$
- $V(i) = \{x \mid x \in \text{Var}(\varphi) \land x \in \text{Var}(c_i)\}$ for $i \in I$
  - $V^+(x), V^-(x), V^+(i), V^-(i)$ defined analogically for positive and negative occurrences
Warning propagation

- \( u_{i \rightarrow x} \in \{0, 1\} \) for \( x \in \text{Var}(\varphi) \) and \( i \in I \) is called a warning.
- \( u_{i \rightarrow x} = 1 \) ... a message from \( c_i \) telling \( x \) to adopt the correct value.
- Warning update rule: 
\[
    u_{i \rightarrow x} = \prod_{y \in V(i) \setminus \{x\}} \Theta(-J^i_y \sum_{j \in V(y) \setminus \{i\}} J^j_y u_{j \rightarrow y}(t - 1)),
\]
where \( \Theta(r) = 0 \) if \( r \leq 0 \) and \( \Theta(r) = 1 \) if \( r > 0 \).

1: procedure \textsc{Warning-propagation}(\varphi, MAX\_SWEEPS)
2: let \( G_F(\varphi) = (V_F, E_F) \) a factor graph for \( \varphi \)
3: for \((x, i) \in E_F\) do
4: \( u_{i \rightarrow x}(0) \leftarrow 0 \) or 1 with probability 0.5
5: for \( t = 1, 2, \ldots, \text{MAX}\_\text{SWEEPS} \) do
6: for \((x, i) \in E_F\) do
7: \( u_{i \rightarrow x}(t) \leftarrow \prod_{y \in V(i) \setminus \{x\}} \Theta(-J^i_y \sum_{j \in V(y) \setminus \{i\}} J^j_y u_{j \rightarrow y}(t - 1)) \)
8: if \( (\forall (x, i) \in E_F) u_{i \rightarrow x}(t) = u_{i \rightarrow x}(t - 1) \) then return TRUE
9: return FALSE
Warning inspired decimation

- Preferred value for variable $x$: $H_x = - \sum_{i \in V(x)} J^i_x u_{i \rightarrow x}$

- Contradiction indicator for variable $x$: $c_x = 1$ if $(\sum_{i \in V^+(x)} u_{i \rightarrow x})(\sum_{i \in V^-(x)} u_{i \rightarrow x}) > 0)$, $c_x = 0$ otherwise.

1: \textbf{procedure} WARNING-INSPIRED-DECIMATION($\varphi$, MAX_SWEEPS)
2: \hspace{1em} \textbf{while} $\varphi \neq \text{True}$ \textbf{do}
3: \hspace{2em} $\alpha = \emptyset$
4: \hspace{2em} \textbf{if not} WARNING-PROPAGATION($\varphi$, MAX_SWEEPS) \textbf{then}
5: \hspace{3em} \textbf{return} UNSAT
6: \hspace{2em} \textbf{if} $\exists x \in \text{Var}(\varphi): c_x = 1$ \textbf{then return} UNSAT
7: \hspace{2em} \textbf{for} $x \in \text{Var}(\varphi)$ \textbf{do}
8: \hspace{3em} \textbf{if} $H_x > 0$ \textbf{then} $\alpha \leftarrow \alpha \cup \{x \mapsto 1\}$
9: \hspace{3em} \textbf{else if} $H_x < 0$ \textbf{then} $\alpha \leftarrow \alpha \cup \{x \mapsto 0\}$
10: \hspace{2em} $\varphi \leftarrow \varphi[\alpha]$
11: \hspace{1em} \textbf{return} SAT
Properties of message passing

Convergence
If the factor graph of a formula is a tree, then warning propagation converges after $|Var(\varphi)|$ iterations. If $c_x = 1$ for some $x \in Var(\varphi)$ then $\varphi$ is unsatisfiable, otherwise it is satisfiable.

▶ Other algorithms based on message passing
  ▶ Belief propagation (BP)
  ▶ Survey propagation / survey inspired decimation