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 $\Rightarrow$ check only clauses in its occurence lists. Search for an unassigned literal in direction of head (tail):
  - Satisfied literal is encountered $\Rightarrow$ clause is already satisfied.
  - Unsatisfied literal is found which *is not* the head (tail) $\Rightarrow$ new tail (head).
  - Unsatisfied literal is found which *is* the head (tail) $\Rightarrow$ unit clause.
  - Else $\Rightarrow$ 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 $\Rightarrow$ check only clauses in its occurence lists. Search for an unassigned literal in direction of head (tail):
  - Satisfied literal is encountered $\Rightarrow$ clause is already satisfied.
  - Unsatisfied literal is found which *is not* the head (tail) $\Rightarrow$ new tail (head).
  - Unsatisfied literal is found which *is* the head (tail) $\Rightarrow$ unit clause.
  - Else $\Rightarrow$ 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 occurence 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 $\Rightarrow$ check only clauses in its occurence 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.
- Backtracking does not require any work!

Watched literals

Running example

$$c_1 = (\overset{\downarrow}{x_1} \vee x_2 \vee \overset{\downarrow}{x_3})$$

$$c_2 = (\overset{\downarrow}{\neg x_1} \vee x_2 \vee \overset{\downarrow}{\neg x_4})$$

$$c_3 = (\overset{\downarrow}{\neg x_1} \vee x_3 \vee \overset{\downarrow}{\neg x_4})$$

Watched occurences

x_1:	$\{c_1\}$	$\neg x_1$:	$\{c_2, c_3\}$
x_2:	$\{\}$	$\neg x_2$:	$\{\}$
x_3:	$\{c_1\}$	$\neg x_3$:	$\{\}$
x_4:	$\{\}$	$\neg x_4$:	$\{c_2, c_3\}$

Watched literals

Running example

Decide $x_1 \mapsto \mathit{False}$

$$c_1 = (x_1 \lor \downarrow x_2 \lor \downarrow x_3)$$

$$c_2 = (\downarrow \neg x_1 \lor x_2 \lor \downarrow \neg x_4)$$

$$c_3 = (\downarrow \neg x_1 \lor x_3 \lor \downarrow \neg x_4)$$

Watched occurences

x_1: $\{\}$	$\neg x_1$: $\{c_2, c_3\}$
x_2: $\{c_1\}$	$\neg x_2$: $\{\}$
x_3: $\{c_1\}$	$\neg x_3$: $\{\}$
x_4: $\{\}$	$\neg x_4$: $\{c_2, c_3\}$

Watched literals

Running example

Decide $x_2 \mapsto \mathit{False}$

$$c_1 = (x_1 \lor \overset{\downarrow}{x_2} \lor \overset{\downarrow}{x_3})$$

$$c_2 = (\overset{\downarrow}{\neg x_1} \lor x_2 \lor \overset{\downarrow}{\neg x_4})$$

$$c_3 = (\overset{\downarrow}{\neg x_1} \lor x_3 \lor \overset{\downarrow}{\neg x_4})$$

Watched occurences

x_1:	$\{\}$	$\neg x_1$:	$\{c_2, c_3\}$
x_2:	$\{c_1\}$	$\neg x_2$:	$\{\}$
x_3:	$\{c_1\}$	$\neg x_3$:	$\{\}$
x_4:	$\{\}$	$\neg x_4$:	$\{c_2, c_3\}$

Derive $x_3 \mapsto \mathit{True}$

$$c_1 = (x_1 \lor \overset{\downarrow}{x_2} \lor \overset{\downarrow}{x_3})$$

$$c_2 = (\overset{\downarrow}{\neg x_1} \lor x_2 \lor \overset{\downarrow}{\neg x_4})$$

$$c_3 = (\overset{\downarrow}{\neg x_1} \lor x_3 \lor \overset{\downarrow}{\neg x_4})$$

Watched occurences

x_1:	$\{\}$	$\neg x_1$:	$\{c_2, c_3\}$
x_2:	$\{c_1\}$	$\neg x_2$:	$\{\}$
x_3:	$\{c_1\}$	$\neg x_3$:	$\{\}$
x_4:	$\{\}$	$\neg x_4$:	$\{c_2, c_3\}$

Watched literals

Running example

Easy bactracking $\Rightarrow$ erase assignment, keep watched occurences.

$$c_1 = (x_1 \vee \overset{\downarrow}{x_2} \vee \overset{\downarrow}{x_3})$$

$$c_2 = (\overset{\downarrow}{\neg x_1} \vee x_2 \vee \overset{\downarrow}{\neg x_4})$$

$$c_3 = (\overset{\downarrow}{\neg x_1} \vee x_3 \vee \overset{\downarrow}{\neg x_4})$$

Watched occurences

x_1: $\{\}$	$\neg x_1$: $\{c_2, c_3\}$
x_2: $\{c_1\}$	$\neg x_2$: $\{\}$
x_3: $\{c_1\}$	$\neg x_3$: $\{\}$
x_4: $\{\}$	$\neg x_4$: $\{c_2, c_3\}$

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 occurences 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 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

```
procedure GSAT(φ, MAX_TRIES, MAX_FLIPS)
    for i = 1, 2, ..., MAX_TRIES do
        α ← random full assignment
        for j = 1, 2, ..., MAX_FLIPS do
            if ∀c ∈ φ : c is satisfied by α then return TRUE
            choose x ∈ Var(φ) such that flipping polarity of x
                leads to the highest number of satisfied clauses
            flip polarity of x in α
    return FALSE
```

WalkSAT

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

procedure $\text{WalkSAT}(\varphi, \mathit{MAX\_TRIES}, \mathit{MAX\_FLIPS})$
    for $i = 1, 2, \ldots, \mathit{MAX\_TRIES}$ **do**
        $\alpha \leftarrow$ random full assignment
        for $j = 1, 2, \ldots, \mathit{MAX\_FLIPS}$ **do**
            if $\forall c \in \varphi : c$ is satisfied by α **then return** TRUE
            choose $c \in \varphi$, random not satisfied clause
            if $\mathit{RAND}(0, 1) < p$ **then** choose random $x \in \mathit{Var}(c)$
            else choose $x \in \mathit{Var}(c)$ such that flipping polarity of x leads to the highest number of satisfied clauses
            flip polarity of x in α
    return FALSE

Message passing algorithms

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

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

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

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

Variable occurrence indication

- $J_x^i = 1$ if $x \in Var(\varphi)$ has a *negative* occurrence in c_i
- $J_x^i = -1$ if $x \in Var(\varphi)$ has a *positive* occurrence in c_i
- $V(x) = \{i \mid i \in I \wedge x \in Var(c_i)\}$ for $x \in Var(\varphi)$
- $V(i) = \{x \mid x \in Var(\varphi) \wedge x \in Var(c_i)\}$ for $i \in I$
  - $V^+(x), V^-(x), V^+(i), V^-(i)$ defined analogically for positive and negative occurrences

Warning propagation

- $u_{i \to x} \in \{0, 1\}$ for $x \in Var(\varphi)$ and $i \in I$ is called a *warning*
  - $u_{i \to x} = 1$ . . . a message from c_i telling x to adopt the correct value
- Warning update rule: $u_{i \to x} = \prod_{y \in V(i) \setminus \{x\}} \Theta(-J_y^i \sum_{j \in V(y) \setminus \{i\}} J_y^j \, u_{j \to y})$,

  where $\Theta(r) = 0$ if $r \leq 0$ and $\Theta(r) = 1$ if $r > 0$.

procedure WARNING-PROPAGATION(φ, MAX_SWEEPS)
    let $G_F(\varphi) = (V_F, E_F)$ a factor graph for φ
    for $(x, i) \in E_F$ **do**
        $u_{i \to x}(0) \leftarrow 0$ or 1 with probability 0.5
    for $t = 1, 2, \ldots, MAX\_SWEEPS$ **do**
        for $(x, i) \in E_F$ **do**
            $u_{i \to x}(t) \leftarrow \prod_{y \in V(i) \setminus \{x\}} \Theta(-J_y^i \sum_{j \in V(y) \setminus \{i\}} J_y^j \, u_{j \to y}(t-1))$
        if $(\forall (x, i) \in E_F) u_{i \to x}(t) = u_{i \to x}(t-1)$ **then return** TRUE
    return FALSE

Warning inspired decimation

- Preferred value for variable x: $H_x = -\sum_{i \in V(x)} J_x^i \, u_{i \to x}$
- Contradiction indicator for variable x: $c_x = 1$ if $\left(\sum_{i \in V^+(x)} u_{i \to x}\right)\left(\sum_{i \in V^-(x)} u_{i \to x}\right) > 0$), $c_x = 0$ otherwise.

procedure $\textsc{Warning-inspired-decimation}(\varphi, \mathit{MAX\_SWEEPS})$
    while $\varphi \neq \mathit{True}$ **do**
        $\alpha = \emptyset$
        if not $\textsc{Warning-propagation}(\varphi, \mathit{MAX\_SWEEPS})$ **then return** UNKNOWN
        if $\exists x \in \mathit{Var}(\varphi) : c_x = 1$ **then return** UNSAT
        for $x \in \mathit{Var}(\varphi)$ **do**
            if $H_x > 0$ **then** $\alpha \leftarrow \alpha \cup \{x \mapsto 1\}$
            else if $H_x < 0$ **then** $\alpha \leftarrow \alpha \cup \{x \mapsto 0\}$
        $\varphi \leftarrow \varphi[\alpha]$
    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 φ is unsatisfiable, otherwise it is satisfiable.

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