Improved Exact Solver for the Weighted Max-SAT Problem

Adrian Kügel

Institute of Theoretical Computer Science, University of Ulm

July 10th, 2010

Motivation

- Max-SAT is NP-hard → no efficient algorithm known
- Applications
  - Bioinformatics → protein structure similarity
  - Electronic markets → combinatorial auctions
  - Sports scheduling → break minimization
  - Probabilistic reasoning → Most Probable Explanation (MPE)

Max-SAT Definition

- Given a list of clauses $C_1, \ldots, C_m$
- Clause consists of disjunction of literals $(l_1 \vee l_2 \vee \ldots \vee l_k)$
- Literal is either x_i or $\overline{x_i}$
- Find assignment of Boolean variables $x_1, \ldots, x_n$ that satisfies maximum number of clauses
- Clause is satisfied if at least one literal is assigned true

Instantiating a variable

- Assign a value to a variable x_i
- Remove literals from the clauses which have been assigned false
- Remove clauses which become satisfied

Example

$$C_1 = (x_1 \vee x_2 \vee x_6)$$
$$C_2 = (x_2 \vee \overline{x}_6)$$
$$C_3 = (\overline{x}_1 \vee x_3)$$
$$C_4 = (\overline{x}_1 \vee x_4)$$
$$C_5 = (\overline{x}_3 \vee \overline{x}_4)$$
$$C_6 = (x_1 \vee \overline{x}_2 \vee x_5)$$
$$C_7 = (x_1 \vee \overline{x}_2 \vee \overline{x}_5)$$

$\rightarrow$ assign $x_1 =$ true

Example

$$
\begin{aligned}
C_1 &= (\text{true} \vee x_2 \vee x_6) \\
C_2 &= (x_2 \vee \overline{x}_6) \\
C_3 &= (\text{false} \vee x_3) \\
C_4 &= (\text{false} \vee x_4) \\
C_5 &= (\overline{x}_3 \vee \overline{x}_4) \\
C_6 &= (\text{true} \vee \overline{x}_2 \vee x_5) \\
C_7 &= (\text{true} \vee \overline{x}_2 \vee \overline{x}_5)
\end{aligned}
$$

Example

$$C_2 = (x_2 \vee \overline{x}_6)$$
$$C_3 = (x_3)$$
$$C_4 = (x_4)$$
$$C_5 = (\overline{x}_3 \vee \overline{x}_4)$$

Branch and Bound for Max-SAT

- Search space is a binary tree with 2^n nodes
- Each inner node corresponds to partial assignment
- Leaf nodes correspond to complete assignments
- Branching:
    - Select unassigned variable
    - Assign value
    - Process remaining clauses recursively
    - Assign opposite value
    - Process remaining clauses recursively

Lower Bound

- Idea: (over)estimate the best possible value of a subtree
- If estimation is worse than best value found so far, subtree can be skipped
- For minimization problems: lower bound lb $\leq$ minimum value in subtree
- Simplest lower bound Max-SAT: number of clauses unsatisfied by partial assignment
- Better: calculate lower bound by finding disjoint inconsistent subformulas

Finding inconsistent subformulas by unit propagation

- Unit clauses can only be satisfied by satisfying the literal
- Propagate literals of unit clauses until empty clause is derived
- Reconstruct which clauses are needed to derive empty clause

Example

$$C_2 = (x_2 \vee \overline{x}_3)$$
$$C_3 = (x_3)$$
$$C_4 = (x_4)$$
$$C_5 = (\overline{x}_3 \vee \overline{x}_4)$$

$\rightarrow$ Assign x_3 = true

Example

$$C_2 = (x_2 \lor \text{false})$$
$$C_3 = (\text{true})$$
$$C_4 = (x_4)$$
$$C_5 = (\text{false} \lor \overline{x}_4)$$

Example

$C_2 = (x_2)$
$C_4 = (x_4)$
$C_5 = (\overline{x}_4)$
$\longrightarrow$ Assign x_4 = true

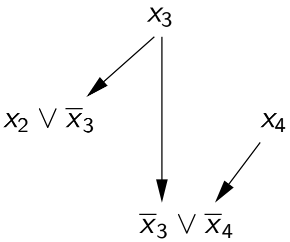
Example

$$C_2 = (x_2)$$
$$C_4 = (\mathrm{true})$$
$$C_5 = (\mathrm{false})$$

Example

$$C_2 = (x_2)$$
$$C_5 = ()$$

Implication graph



One step further: failed literal detection

- Add a unit literal l to the formula
- Use unit propagation to detect inconsistent subformula F'
- $S_l = F' \setminus l$ is resolution proof of $\bar{l}$
- l is called a failed literal
- If l and $\bar{l}$ are failed literals, $S_l \cup S_{\bar{l}}$ is inconsistent subformula

Improved algorithm

- Use failed literal detection during unit propagation to get new unit literals
- Whenever no unit literal is left, try to find a failed literal l
- If found
  - extract and store subformula S_l which is in conflict with l
  - add $\bar{l}$ to the formula and continue with unit propagation

Difference between algorithms

- improved algorithm starts with unit propagation and uses failed literal detection in simplified formula
- after finding a failed literal l, we do not stop if $\bar{l}$ is no failed literal, but continue in simplified formula after propagating $\bar{l}$
- improved algorithm can be seen as a restricted sat solver (no branches, only unit clause learning) with unsatisfiable core extraction

Example

$C_1 = (x_1 \vee x_2 \vee x_3)$
$C_2 = (x_2 \vee \overline{x}_3)$
$C_3 = (\overline{x}_1 \vee x_3)$
$C_4 = (\overline{x}_1 \vee x_4)$
$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$C_6 = (x_1 \vee \overline{x}_2 \vee x_5)$
$C_7 = (x_1 \vee \overline{x}_2 \vee \overline{x}_5)$
$\rightarrow$ no unit literal, starting failed literal detection

Example

$C_1 = (x_1 \vee x_2 \vee x_3)$
$C_2 = (x_2 \vee \overline{x}_3)$
$C_3 = (\overline{x}_1 \vee x_3)$
$C_4 = (\overline{x}_1 \vee x_4)$
$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$C_6 = (x_1 \vee \overline{x}_2 \vee x_5)$
$C_7 = (x_1 \vee \overline{x}_2 \vee \overline{x}_5)$
$C_8 = (x_1)$
$\rightarrow$ adding x_1 to the formula
$\rightarrow$ assign $x_1 =$ true

Example

$C_2 = (x_2 \vee \overline{x}_3)$
$C_3 = (x_3)$
$C_4 = (x_4)$
$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$\rightarrow$ ... continue as in the example before

Example

$S_{x_1} = \{(x_3), (x_4), (\overline{x}_3 \vee \overline{x}_4)\}$
$\longrightarrow$ add $\overline{x}_1$ to the formula

Example

$$
\begin{aligned}
C_1 &= (x_1 \vee x_2 \vee x_3) \\
C_2 &= (x_2 \vee \overline{x}_3) \\
C_3 &= (\overline{x}_1 \vee x_3) \\
C_4 &= (\overline{x}_1 \vee x_4) \\
C_5 &= (\overline{x}_3 \vee \overline{x}_4) \\
C_6 &= (x_1 \vee \overline{x}_2 \vee x_5) \\
C_7 &= (x_1 \vee \overline{x}_2 \vee \overline{x}_5) \\
C_8 &= (\overline{x}_1)
\end{aligned}
$$

$\rightarrow$ assign $x_1 =$ false

Example

$C_1 = (x_2 \vee x_3)$
$C_2 = (x_2 \vee \overline{x}_3)$
$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$C_6 = (\overline{x}_2 \vee x_5)$
$C_7 = (\overline{x}_2 \vee \overline{x}_5)$
$\longrightarrow$ failed literal detection, add x_2 to the formula

Example

$$C_1 = (x_2 \vee x_3)$$
$$C_2 = (x_2 \vee \overline{x}_3)$$
$$C_5 = (\overline{x}_3 \vee \overline{x}_4)$$
$$C_6 = (\overline{x}_2 \vee x_5)$$
$$C_7 = (\overline{x}_2 \vee \overline{x}_5)$$
$$C_8 = (x_2)$$

$\rightarrow$ assign $x_2 =$ true

Example

$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$C_6 = (x_5)$
$C_7 = (\overline{x}_5)$
$\longrightarrow$ assign $x_5 =$ true

Example

$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$C_7 = ()$
$\longrightarrow$ empty clause found
$\longrightarrow$ x_2 is failed literal
$\longrightarrow$ $S_{x_2} = \{(\overline{x}_2 \vee x_5), (\overline{x}_2 \vee \overline{x}_5)\}$

Example

$C_1 = (x_2 \vee x_3)$
$C_2 = (x_2 \vee \overline{x}_3)$
$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$C_6 = (\overline{x}_2 \vee x_5)$
$C_7 = (\overline{x}_2 \vee \overline{x}_5)$
$C_8 = (\overline{x}_2)$
$\rightarrow$ add $\overline{x}_2$ to the formula
$\rightarrow$ assign $x_2 =$ false

Example

$C_1 = (x_3)$
$C_2 = (\overline{x}_3)$
$C_5 = (\overline{x}_3 \vee \overline{x}_4)$
$\longrightarrow$ assign $x_3 =$ true

Example

$C_2 = ()$
$C_5 = (\overline{x}_4)$
$\longrightarrow$ empty clause found
$\longrightarrow$ derive inconsistent subformula with implication graph

Optimizations

- Assign priority order to variables for failed literal detection
- Decrease priority of variables for which a failed literal has been found
- For each variable, check only the literal which occurs in more clauses
- Sometimes more than one failed literal can be detected by only one failed literal detection $\longrightarrow$ use failed literal l for which S_l is smallest.

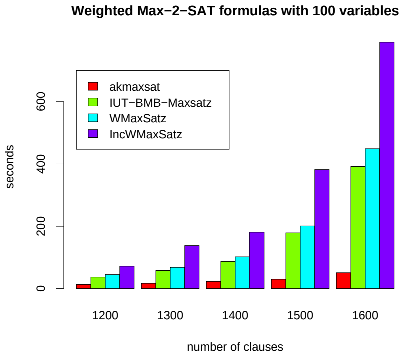
Example

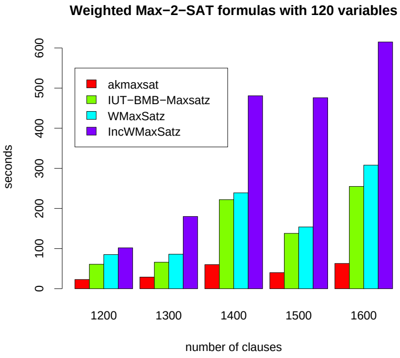
$C_1 = \overline{x}_1 \vee x_2$
$C_2 = \overline{x}_2 \vee x_3$
$C_3 = \overline{x}_3 \vee x_4$
$C_4 = \overline{x}_3 \vee x_5$
$C_5 = \overline{x}_4 \vee \overline{x}_5$
C_3, C_4, C_5 are resolution proof of x_3
$\rightarrow x_1$, x_2 and x_3 are failed literals

Data structure

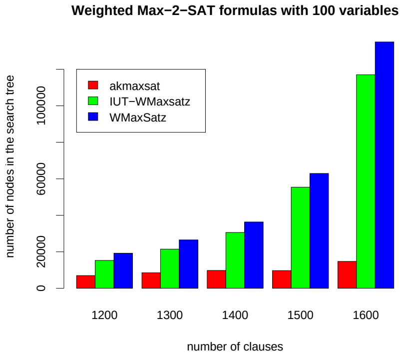
- for each literal keep list of clause pointers to clauses where literal occurs
- support lazy deletion of clause pointers
- each clause has a delete flag and deletion timestamp
- clause pointers are removed during traversal of clause pointer list when delete flag of clause is set to true
- when backtracking, it can be checked in constant time if clause pointer was deleted or not

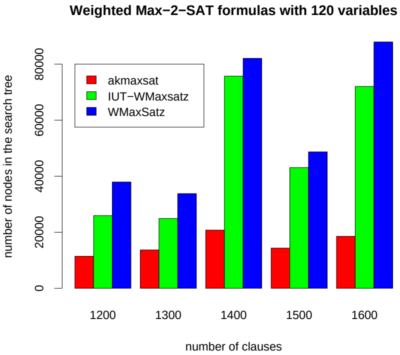
Comparison of runtimes





Comparison of traversed nodes of search tree





Conclusions

- New propagation algorithm improves lower bound
- Smaller part of the search tree needs to be traversed
- Data structure improves runtime for high clauses-to-variables ratio

Any questions?

Thank you for your attention!