CS 433 Automated Reasoning 2024

Lecture 22: Maxsat - an application of SAT oracle

Instructor: Ashutosh Gupta

IITB India

Compile date: 2024-04-17

Oracle

In computer science, oracle is an algorithm that can solve a hard problem.

Often complexity or security arguments depend on the availability or absence of such oracles.

SAT solver is the quintessential oracle used for many harder problems, *e.g.*, maxsat.

Topic 22.1

Maxsat

Maxsat

Input:

- A CNF fromula F
- A weight function $w : F \to \mathbb{N}$
  - maps each clause in F to a number

Output:

Find a model m such that the following sum is maximum.

$$\sum_{C \in F} w(C)m(C)$$

Commentary: $m(C)$ is 1 if $m \models C$, otherwise 0.

Partial maxsat

Input:

- A CNF fromula $Hard \land Soft$
- A weight function $w : Soft \to \mathbb{N}$

Output:

There may be no solutions.

Find a model m such that $m \models Hard$ and the following sum is maximum.

$$\sum_{C \in Soft} w(C)m(C)$$

Many interesting optimization problems can be encoded into maxsat problem.

Example: shortest path on a graph

Example 22.1

Consider an undirected graph (V, E). Find shortest path between two nodes $s, g \in V$

- We choose a Boolean variable p_v for each vertex v, indicating if v is visited.
- Hard constraints
  - $p_s \wedge p_g$ (source and goal must be visited)
  - $p_s \Rightarrow \sum_{v' \in E(s)} p_{v'} = 1$ (source has exactly one successor)
  - $\displaystyle\bigwedge_{v \in V - \{s,g\}} (p_v \Rightarrow \sum_{v' \in E(v)} p_{v'} = 2)$ (one neighbour to enter and the other to leave)
  - $p_g \Rightarrow \sum_{v' \in E(g)} = 1$ (goal has exactly one predecessor)
- Soft constraints $\bigwedge_{v \in V} \neg p_v$

Topic 22.2

Methods for maxsat

Methods for maxsat

There has been many proposed methods.

- Branch-and-bound
- Integer arithmetic solver based (IP)
- **SAT solvers based algorithms**
- Implicit hitting set algorithms (IP/Hybrid)

We will focus on only one class of them.

For further details: `https://www.cs.helsinki.fi/group/coreo/aaai16-tutorial/aaai16-maxsat-tutorial.pdf`

Unweighted Partial maxsat

In this lecture, we will only discuss the unweighted maxsat problem

Input:

- A CNF fromula $Hard \wedge Soft$

Output:

Find a model m such that $m \models Hard$ and the following sum is maximum.

$$\sum_{C \in Soft} m(C)$$

How a SAT solver based method works?

By setup initially, $\not\models Soft \wedge Hard$.
Iteratively, relax $Soft$ constraints until $\models Soft \wedge Hard$.

1. If $\not\models Hard$, return no maxsat solution
2. If $m \models Soft \wedge Hard$, return found optimal m
3. Relax $Soft$ so that more clauses **allowed to be** false in the original $Soft$
4. go to 2.

We will cover a few instances of the above design

- Iterative linear search
- Core based search

Blocking variables to allow false soft clauses

maxsat methods often use blocking variables to relax (block) soft clauses.

- In a soft clause $x_1 \lor .... \lor x_k$ we insert a new variable b

$$b \lor x_1 \lor ... \lor x_k$$

  b is fresh with respect to the formula.
- If $b = 0$ the soft clause has to be satisfied.
- If $b = 1$ the extended clause is already satisfied and the soft clause is blocked, i.e., no requirement to satisfy the soft clause.

Example 22.2

Recall the shortest path soft clauses $\bigwedge_{v \in V} \neg p_v$

Consider a clause $\neg p_x$ *for some node* $x \in V$*.*

Corresponding clause with blocking variable $(\neg p_x \lor b_x)$

Iterative linear search

1. Insert a blocking variable b_c in every $c \in Soft$.
2. $k := 0$
3. If $\models Hard \wedge Soft \wedge CNF(\sum b_c \leq k)$, return k
4. $k := k + 1$.
5. goto 3.

at most k soft clauses can be blocked

Exercise 22.1

Can we improve on the search of k?

Iterative linear search in the other direction

SAT $\rightarrow$ UNSAT

1. Insert a blocking variable b_c in every $c \in \mathit{Soft}$.
2. Get m such that $m \models \mathit{Hard}$
3. $k := \#$(of voilated clauses in Soft by m)-1
4. If there is a better $m \models \mathit{Hard} \wedge \mathit{Soft} \wedge \mathit{CNF}(\sum b_c \leq k)$, goto 3
5. return k.

after every iteration m violates fewer clauses in Soft

Exercise 22.2

Can we improve on the search of k like the previous algorithm?

Core based maxsat solving

In the last algorithm, we could guide our search using the model.

In case of unsatisfiability there is no guidance.

Definition 22.1

An unsat core *of an maxsat problem* $Hard \wedge Soft$ *is a subset* $F \subseteq Soft$ *such that* $Hard \wedge F$ *is unsatisfiable.*

We usually expect F to be significantly smaller than $Soft$.

Modern solvers can return an unsat cores in the case of unsatisfiability.

Iterative linear search with unsat core

1. $k := 0, BV = \{\}$
2. If $\models \mathit{Hard} \wedge \mathit{Soft} \wedge \mathit{CNF}(\sum_{b_c \in BV} b_c \leq k)$, return k
3. Otherwise, get unsat core K of $\mathit{Hard} \wedge \mathit{Soft} \wedge \mathit{CNF}(\sum_{b_c \in BV} b_c \leq k)$
4. For each $c \in K$ that has no blocking variable.
   4.1 Insert a blocking variable b_c in c and $BV = BV \cup \{b_c\}$.
5. $k := k + 1$.
6. goto 2.

The clauses that have participated in cores have blocking variables.

Now cardinality constraints are over far fewer variables.

Therefore, tighter relaxation and less wasteful search in satisfiability checks.

Exercise 22.3

a. Show that this algorithm obtains the maximum?

b. How can we incrementally construct cardinality constraints?

Example: core restricted cardinality constraints

Example 22.3

Recall the shortest path soft clauses $\bigwedge_{v \in V} \neg p_v$

Let us suppose we got two unsat cores $\{\neg p_x, \neg p_y\}$ and $\{\neg p_u, \neg p_v, \neg p_w\}$ in first two iterations.

In previous algorithm, we will insert blocking bits b_x, b_y, b_u, b_v, b_w as follows

- $\neg p_x \vee b_x$
- $\neg p_y \vee b_y$
- $\neg p_u \vee b_u$
- $\neg p_v \vee b_v$
- $\neg p_w \vee b_w$

We will add cardinality constraint $b_x + b_y + b_u + b_v + b_w \leq 2$.

We are still relaxing too much?

We are adding

$$b_x + b_y + b_u + b_v + b_w \leq 2$$

We are asking the solver to block at most any two of the five soft constraints.

It may end up blocking $\neg p_x$ and $\neg p_y$.

We already know that there is no solution for this blocking combination, since nothing is blocked from the other unsat core.

We can be more precise and add $b_x + b_y \leq 1 \wedge b_u + b_v + b_w \leq 1$

Exactly one soft clause is to be blocked from each core.

Overlapping cores

If cores overlap we can not add separate learned inequalities?

Example 22.4

Let us suppose we got two unsat cores $\{\neg p_x, \neg p_y\}$ *and* $\{\neg p_x, \neg p_v, \neg p_w\}$ *in first two iterations.*

We cannot add two constraints $b_x + b_y \leq 1 \wedge b_x + b_v + b_w \leq 1$.

Since $b_x = 1$ *and* $b_y = 1$ *removes both cores and is not satisfied by the above constraints.*

If overlap we have to create inequalities that is combined.

Two solutions

- Fresh blocking variable for each core (Fu-malik)
- Maintain disjoint sets of clauses of overlapping cores

Fu-Malik: fresh blocking variable for each core

1. $k := 0$,
2. If $\models \mathit{Hard} \land \mathit{Soft}$, return k
3. Otherwise, get unsat core K of $\mathit{Hard} \land \mathit{Soft}$
4. $BV = \{\}$
5. For each $c \in K$.
   5.1 Insert a fresh blocking variable b_c in c and $BV = BV \cup \{b_c\}$.
6. $\mathit{Hard} := \mathit{Hard} \land \mathit{CNF}(\sum_{b_c \in BV} b_c \leq 1)$
7. $k := k + 1$.
8. goto 2.

A clause may get multiple blocking variable.

Since we compare with 1 only, simpler constraints!!

Overlapping cores

Example 22.5

Let us again consider two unsat cores $\{\neg p_x, \neg p_y\}$ *and* $\{\neg p_x, \neg p_v, \neg p_w\}$ *in first two iterations.*

A blocking bit for each clause in each core.

- $\neg p_x \lor b_{x1} \lor b_{x2}$
- $\neg p_y \lor b_y$
- $\neg p_u \lor b_u$
- $\neg p_v \lor b_v$
- $\neg p_w \lor b_w$

Gets two blocking variables, since $\neg p_x$ *occurs in two cores.*

Adds too many new variables with symmetric roles. Burden on future iterations.

Maintain disjoint core covers

We can keep that records of cores that do not overlap.

Definition 22.2

A cover is a set of overlapping cores and two covers do not have cores that overlap with each other.

We maintain a set *Covers* of covers.

- Let $F[Covers] := \bigwedge_{Cover \in Covers} CNF(\sum_{b_c \in K \in Cover} b_c \leq |Cover|)$.

Number of cores in the cover

- Let $Covers' := mergeCover(Covers, K)$ insert core K in *Covers* as follows
  1. $Covers' = \{\}$, $NewCover = \{K\}$
  2. For each $Cover \in Covers$
     - If $\exists K' \in Cover.\ K' \cap K \neq \emptyset$, $NewCover := NewCover \cup Cover$
     - Otherwise, $Covers' := Covers' \cup \{Cover\}$
  3. return $Covers' \cup \{NewCover\}$

Example: cover

Example 22.6

Consider unsat cores after five iterations.

- $\{\neg p_x, \neg p_y\}$
- $\{\neg p_x, \neg p_z\}$
- $\{\neg p_u, \neg p_v\}$
- $\{\neg p_a, \neg p_w\}$
- $\{\neg p_a, \neg p_v, \neg p_c\}$

$$
\begin{aligned}
Covers := \{ & \\
& \{\ \ \{\neg p_x, \neg p_y\},\ \ \{\neg p_x, \neg p_z\}\ \ \}, \\
& \{\ \ \{\neg p_u, \neg p_v\},\ \ \{\neg p_a, \neg p_w\},\ \ \{\neg p_a, \neg p_v, \neg p_c\}\ \ \} \\
\} &
\end{aligned}
$$

$$F[Covers] := b_x + b_y + b_z \leq 2 \wedge b_u + b_v + b_w + b_a + b_c \leq 3$$

Maxsat via core covers

1. $k := 0, Covers = \{\}$
2. If $\models Hard \land Soft \land F[Covers]$, return k
3. Otherwise, get unsat core K of $Hard \land Soft \land F[Covers]$
4. $Covers := mergeCovers(Covers, K)$.
5. $k := k + 1$.
6. goto 2.

Covers is an equivalence class over clauses

LP methods

Topic 22.3

Problems

End of Lecture 22