HSGreedy: A Local Search MaxSAT Solver based on Dist

Yi Fan, Zongjie Ma, Kaile Su, Abdul Sattar
Institute for Integrated and Intelligent Systems
Griffith University
zongjie.ma@griffithuni.edu.au

Chengqian Li
Department of Computer Science
Sun Yat-sen University

Qingliang Chen
Department of Computer Science
Jinan University

Abstract

The Maximum Satisfiability (MaxSAT) problem is a well-known NP-hard problem which is important in both theory and applications. Local search is a promising approach to find optimal or near-optimal solutions for large instances. However, for industrial instances, the local search approach is inferior to other approaches like the core-guided ones. In this MaxSAT Evaluation, we submit a solver (named HSGreedy) that tries to narrow the gap in industrial instances. It is based on a state-of-the-art solver, Dist, and makes significant refinements. More specifically, during the execution it determines whether it should focus more on hard clauses or soft clauses.

1. Introduction

The Maximum Satisfiability (MaxSAT) problem is a well-known NP-hard problem. We use $x_1 \dots x_n$ to be the set of variables. A literal l is a variable or its negation, a clause is a disjunction of literals. Given a CNF formula $F = C_1 \wedge \dots \wedge C_m$, we associate C_i with a positive integer $w(C_i)$. Given a complete solution s, the cost of s is the total weight of the unsatisfied clauses, *i.e.*,

$$cost(s) = \sum_{C_i \text{ IS UNSATISFIED UNDER } s} w(C_i).$$

Given a weighted MaxSAT instance as above, the problem is to find a solution which minimize the cost.

There are three other types of MaxSAT problems which can all be encoded as a weighted MaxSAT problem. In unweighted MaxSAT problems, the weights associated with each clause are 1. In partial MaxSAT, clauses are distinguished into hard clauses and soft clauses, where each soft clause has weight 1 and each hard clause has a weight which is greater than the total weight of all soft clauses. In weighted partial MaxSAT, each soft clause can have a different weight.

2. Local Search for MaxSAT

The Dist [1] distinguishes hard clauses from soft clauses and calculate hard scores and soft scores independently. It adopts clause weighting for hard clauses, and uses the original weights for soft clauses.

Given an assignment s and a variable x, we use s_{-x} to denote the assignment obtained from s by flipping x. Given an assignment α, we use $hcost(\alpha)$ to denote the total weight of hard unsatisfied clauses, and use $scost(\alpha)$ to denote the total weight of soft unsatisfied clauses. We use $hscore(x)$ to denote the increase of satisfied hard clause weights, and use $sscore(x)$ to denote the increase of satisfied soft clause weights. That is, $hscore(x) = hcost(s) - hcost(s_{-x})$ and $sscore(x) = scost(s) - scost(s_{-x})$. Algorithm 1 is the algorithm of Dist, where

1. $p = 0.01$,
2. sp is set to 0.0001 for industrial instances with more than 2500 variables, and 0.001 for all other instances.

3. Our Improvements

Considering Lines 7 to 8, we find that it is too blind to select a variable randomly. We think that sometimes we should focus more on hard clauses while sometimes we should focus more on soft clauses. That is, sometimes we should be in favor of hard score and sometimes soft score. So we need a condition to determine whether hard or soft score should be cared. Then we replace these two lines as Algorithm 2 below.

In this algorithm, if $scost(s) < scost^*$, it seems that soft clauses have already been satisfied sufficiently, so we

Algorithm 1: Dist

$s \leftarrow$ a random generated truth assignment;
$scost^* \leftarrow +\infty$;
for $step \leftarrow 1$ **to** $maxSteps$ **do**
  if $hcost(s) = 0$ & $scost(s) < scost^*$ **then**
    $s^* \leftarrow s$;
    $scost^* \leftarrow scost(s)$;
  if $H \leftarrow \{x|hscore(x) > 0\} \neq \emptyset$ **then** /* */
    $v \leftarrow$ a random variable in H;
  else if $S \leftarrow \{x|hscore(x) = 0 \& sscore(x) > 0\}$ **then** /* */
    $v \leftarrow$ a variable in S with the greatest $sscore$;
  else with probability sp: for each hard satisfied clause with weight greater than 1, decrease its weight by 1;
  with probability $1 - sp$: for each hard unsatisfied clause, increase its weight by 1;
  if *there exists hard unsatisfied clause* **then**
    $C \leftarrow$ a random hard unsatisfied clause;
  else
    $C \leftarrow$ a random soft unsatisfied clause;
  With probability p: $v \leftarrow$ a variable in C with the greatest $sscore$;
  With probability $1 - p$: $v \leftarrow$ a random variable in C; /* */
  $s \leftarrow s_{-x}$;
return s^*;

Algorithm 2: HSG

if $H \leftarrow \{x|hscore(x) > 0\} \neq \emptyset$ **then**
  if $scost(s) < scost^*$ **then**
    $v \leftarrow$ a variable with the greatest $hscore$;
  else
    $v \leftarrow$ a variable with the greatest $sscore$;

should satisfy more hard clauses, *i.e.*, we should pick a variable with the greatest $hscore$. Otherwise, we should pick a variable with the greatest $sscore$.

4. Our Solver

We named our solver HSGreedy, and set p to 0.01. As to sp, we set it to 0.0001 for instances containing more than 2500 variables, and set it to 0.001 otherwise.

5. Usage

```
./HS-Greedy instance
```

References

[1] S. Cai, C. Luo, J. Thornton, and K. Su. Tailoring local search for partial maxsat. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2623–2629, 2014.