Ramp: A Local Search Solver based on Make-positive Variables

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

Abstract

Many local search techniques for SAT can be adopted to solve MaxSAT. Yet often there are quite a few disjoint subset of clauses which are unsatisfiable. Sometimes only considering decreasing variables whose flip leads to less violated clause weights is not enough. So CCLS extend the notion of decreasing variables into make-positive variables whose flip turns at least one clause from unsatisfied to satisfied. Here we adopt the make-positive heuristic to develop a solver named Ramp. It adopts restarting techniques and uses age to break ties, but abandons the configuration checking techniques.

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 CCLS [1] adopts the configuration checking and the make-positive heuristics.

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 $cost(\alpha)$ to denote the total weight of unsatisfied clauses. We use $score(x)$ to denote the increase of satisfied clause weights. That is, $score(x) = cost(s) - cost(s_{-x})$. We use $make(x)$ to denote the number of unsatisfied clauses that will become satisfied by flipping x. We use MP to denote the set of variables whose $make$ is greater than zero. We use $age(x)$ to denote the number of flips that have been performed since last time x was flipped.

As to partial MaxSAT, previous researchers realize that the gap between hard clause weights and soft clause weights should be as small as possible. So we propose a novel heuristic to narrow the gap. Each time when we update the best solution, we decrease the weights of all hard clauses, as is shown below.

Algorithm 1: pickVar

```
1 if $MP \neq \emptyset$ & with probability $p$ then
2     $v \leftarrow$ a variable in $MP$ with the greatest $score$,
      breaking ties in favor of the greatest $age$;
3 else
4     $C \leftarrow$ a random unsatisfied clause;
5     $v \leftarrow$ a random variable in $C$;
6 return v;
```

The algorithms of our solver are described in Algorithm 1 and 2, where the parameter p is set to 0.6 in advance. Notice that we adopt a restart mechanism in our solver.

3. Usage

```
./Ramp instance
```

Algorithm 2: Ramp

```
1  $max\_flip \leftarrow 20 * |V|$ if $|V| < 10^5$;
2  $max\_flip \leftarrow +\infty$ otherwise;
3  weight$_{hard}$ ← hard clause weight;
4  **while** *elasp time* $<$ *cutoff* **do**
5      $flip\_bound \leftarrow max\_flip$;
6      $s \leftarrow$ a random generated assignment;
7      **while** *step* $<$ *flip_bound* **do**
8          **if** $cost(s) < cost^*$ & $cost(s) <$ *weight*$_{hard}$
           **then**
9              $s^* \leftarrow s$;
10             $cost^* \leftarrow cost(s)$;
11             weight$_{hard} \leftarrow cost^*$;
12             **foreach** *hard clause* $C_i$ **do**
13                 $w(C_i) \leftarrow$ weight$_{hard}$
14         $v \leftarrow pickVar()$;
15         flip $v$;
16     $max\_flip \leftarrow max\_flip * 4$;
```

References

[1] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su. CCLS: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Trans. Computers*, 64(7):1830–1843, 2015.