CCLS: Solver Description

Chuan Luo[1], Shaowei Cai[1,2], Zhong Jie[1], and Kaile Su[2]

[1] Key laboratory of High Confidence Software Technologies, Peking University, Beijing, China
[2] Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
chuanluosaber@gmail.com; shaoweicai.cs@gmail.com;
pkutcsj@gmail.com; k.su@griffith.edu.au

Abstract. This solver description introduces the basic ideas and the core strategies used in our solver called CCLS for solving weighted and unweighted MAX-SAT instances.

1 Introduction

Stochastic local search (SLS) algorithms for MAX-SAT usually work as follow. At the beginning of local search, an assignment mapping to all variables is generated randomly. Then the algorithm selects a variable to flip iteratively until timeout.

An important issue for local search is the cycling problem, i.e., revisiting a candidate solution that has been visited recently [5]. Recently, a new diversification strategy called configuration checking (CC) was proposed for the cycling problem. CC has been successfully used in minimum vertex cover (MVC) and SAT, resulting in two efficient algorithms namely EWCC [4] and Swcca [3], respectively. Morever, a local search solver called CCASat [1], which is improved from Swcca, won the random track of SAT Challenge 2012. The CC strategy for SAT forbids a variable to flip if all its neighboring variables have not changed their truth values since its last flip. A variable is said to be configuration changed if since its last flip, at least one of its neighboring variables has been flipped.

Based on the CC strategy, we propose a new heuristic called CCM (configuration checking with make) for weighted MAX-SAT. According to CCM, if there exist variables which are configuration changed and have a positive make value, CCM picks the one with greatest $score$ to flip; otherwise CCM picks a variable randomly from a random unsatisfied clause. We use the CCM heuristic to develop an SLS algorithm called CCLS.

2 Configuration Checking and CCM Heuristic

We first give some definitions and notations used in the CCM heuristic. We use $V(F)$ to denote the set of all variables appear in the formula F. Two different variables are neighbors when they appear in at least one clause simultaneously, and $N(x) = \{y \mid y \in V(F),\ y \text{ and } x \text{ are neighbors}\}$ is the set of all neighbors of variable x.

Given a weighted conjunctive normal form (CNF) formula F, the cost of an assignment α, denoted as $cost(F, \alpha)$, is the total weight of all unsatisfied clauses under α. For

a variable x, the property $make(x)$ is defined as the total weight of clauses that would become satisfied if the variable is flipped; the property $break(x)$ is the total weight of clauses that would become unsatisfied if the variable is flipped; the property $score(x)$ is the increment in the total weight of satisfied clauses if the variable is flipped, and can be understood as $make(x) - break(x)$. The CCM heuristic utilizes $make$ and $score$ to select the flipping variable.

The CC strategy records the circumstance information of each variable, and forbids flipping any variable whose circumstance information has not been changed since its last flip. The 'circumstance information' is formally defined as the concept of $configuration$.

As stated in [2], in the context of SAT, the $configuration$ of a variable x refers to a vector consisting of Boolean values of $N(x)$ (x's all neighboring variables). We have the formal definition of $configuration$ as follows.

Definition 1. *Given a CNF formula F and an assignment α to $V(F)$, the $configuration$ of a variable $x \in V(F)$ under α is a vector $configuration(x)$ consisting of truth values of all variables in $N(x)$ under α.*

An implementation of CC is to employ a Boolean array $ConfChanged$, whose size equals the number of variables in the formula. For each variable x, $ConfChanged(x)$ measures the frequency (i.e., the number of steps) that $configuration(x)$ has been changed since x's last flip. We maintain the array $ConfChanged$ as follows.

- Rule 1: At the beginning of the local search, all the variables' $ConfChanged$ values are set to 1.
- Rule 2: Whenever a variable x is flipped, $ConfChanged(x)$ is reset to 0. For each variable $y \in N(x)$, $ConfChanged(y)$ is set to 1.

Apparently, a variable x's $configuration$ has not been changed since x's last flip if and only if $ConfChanged(x)$ is 0.

Now we define the concept of Configuration Checking and Make Positive (CCMP) variables.

Definition 2. *Given a CNF formula F and an assignment α to $V(F)$, a variable x is defined configuration changed and make positive (CCMP) if and only if $make(x) > 0$ and $ConfChanged(x) > 0$.*

We use *CCMPvars* to denote the set of all CCMP variables during the search. The CCM heuristic can be briefly introduced as follows. If there exist CCMP variables, then the CCM heuristic picks the one with greatest $score$; otherwise it picks a variable randomly in an unsatisfied clause.

3 The CCLS Algorithm

The pseudo code of CCLS can be found in Algorithm 1. There is a parameter p in CCLS. For random weighted MAX-3-SAT instances, we set p to 0.42. For random weighted MAX-2-SAT instances, we set p to 0.37. For structured weighted MAX-SAT instances, we set p to 0.2. For unweighted ones, we set p to 0.1. These settings are based on preliminary experiments, and we believe the performance of CCLS could be improved after some tuning.

Algorithm 1: CCLS

Input: CNF-formula F, $maxSteps$
Output: An assignment α^* of F

```
1  begin
2      generate a random assignment α, α* ← α;
3      for step ← 1 to maxSteps do
4          if with the fixed probability p then
5              select an unsatisfied clause c randomly;
6              v ← a random variable for c;
7          else
8              if CCMPvars is not empty then
9                  v ← x with the greatest score in CCMPvars;
10             else
11                 select an unsatisfied clause c randomly;
12                 v ← a random variable for c;
13         α ← α with v flipped;
14         if cost(F, α) < cost(F, α*) then α* ← α;
15     return α*;
16 end
```

References

1. Cai, S., Luo, C., Su, K.: CCASat: Solver description. In: Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. pp. 13–14 (2012)
2. Cai, S., Su, K.: Local search with configuration checking for SAT. In: Proc. of ICTAI-11. pp. 59–66 (2011)
3. Cai, S., Su, K.: Configuration checking with aspiration in local search for SAT. In: Proc. of AAAI-12. pp. 434–440 (2012)
4. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. 175(9-10), 1672–1696 (2011)
5. Michiels, W., Aarts, E.H.L., Korst, J.H.M.: Theoretical aspects of local search. Springer (2007)