CScoreSAT2013

Shaowei Cai
Griffith University
shaoweicai.cs@gmail.com

Chuan Luo
Peking University
chuanluosaber@gmail.com

Kaile Su
Griffith University
k.su@griffith.edu.au

***Abstract*—This note describes the SAT solver "CScore-SAT2013", which is a local search solver, especially designed for random instances.**

I. Introduction

Recently, we proposed a new variable property called $subscore$ [1], which shares the same spirit with the commonly used property $score$. While $score$ measures the increment of satisfied clauses by flipping a variable, $subscore$ does that of clauses with more than one true literal. Further, we design a scoring function called *comprehensive score* [2], which is a linear combination of $score$ and $subscore$. We also define a new type of "decreasing" variables namely *comprehensively decreasing* variables [2].

Based on the notions of comprehensive score and comprehensively decreasing variable, we develop an SLS algorithm called CScoreSAT (comprehensive score based SAT algorithm) [2]. The SAT solver CScoreSAT2013 adopts WalkSAT to solve random instances whose maximum clause length (denoted by k) is greater than 3, and adopts CScoreSAT to solve instances with $k > 3$.

II. Main Techniques

Main techniques in CScoreSAT include: configuration checking [3], [4] and comprehensive score [2].

A. Configuration Checking

To avoid blind search, we utilize the configuration checking (CC) strategy . The configuration checking (CC) strategy was proposed to handle the revisiting problem in local search [5], and has proved effective in SLS algorithms for SAT [4]. In the context of SAT, the CC strategy forbids flipping a variable if since its last flip, none of its neighboring variables has been flipped. A variable is configuration changed if since its last flip, at least one of its neighboring variables has been flipped.

B. Comprehensive Score and Comprehensively Decreasing Variables

We consider the number of true literals in a clause, which can be regarded as the degree of being satisfied of the clause. The more true literals a clause contains, the less likely it would become unsatisfied in the following flips.

Definition 1: Given a CNF formula F and an assignment α to its variables, the *satisfaction degree* of a clause C, is defined as the number of true literals in C under α. A clause with a satisfaction degree of δ is said to be a δ-satisfied clause.

Among satisfied clauses, 1-satisfied clauses are the most unstable, as they can become unsatisfied by flipping only one variable. It is beneficial for SLS algorithms to take into account the transformations between 1-satisfied and 2-satisfied clauses.

Based on the above considerations, the variable property $subscore$ is defined as follows.

Definition 2: For a variable x, $subscore(x)$ is defined as $submake(x)$ minus $subbreak(x)$, where $submake(x)$ is the number of 1-satisfied clauses that would become 2-satisfied by flipping x, and $subbreak(x)$ is the number of 2-satisfied clauses that would become 1-satisfied by flipping x.

When considering clause weights in DLS algorithms, $submake(x)$ measures the total weight of the 1-satisfied clauses that would become 2-satisfied by flipping x, and $subbreak(x)$ does that of the 2-satisfied clauses that would become 1-satisfied by flipping x.

Based on the above considerations, By combining $score$ and $subscore$, we design a scoring function named comprehensive score, which is formally defined as follows.

Definition 3: For a CNF formula F, the *comprehensive score* function, denoted by $cscore$, is a function for variables such that $cscore(x) = score(x) + \lfloor subscore(x)/d \rfloor$, where d is a positive integer parameter.

In the following, we define a new type of "deceasing" variables based on the $cscore$ function.

Definition 4: Given a CNF formula F and its $cscore$ function, a variable x is *comprehensively decreasing* if and only if $score(x) \geq 0$ and $cscore(x) > 0$.

Comprehensively decreasing variables are considered to be the flip candidates in the greedy search phases of our algorithm. We utilize the configuration checking (CC) strategy to identify the "good" comprehensively decreasing variables which are *configuration changed*. For convenience, such variables are further referred to as CDCC (Comprehensively Decreasing and Configuration Changed) variables.

III. The CScoreSAT Algorithm

This section presents the CScoreSAT algorithm, which utilizes two key notions: comprehensive score and comprehensively decreasing variable.

For the sake of diversification, CScoreSAT also employs the PAWS clause weighting scheme [6]. All clause weights are initiated as 1. When a local optimum is reached, with probability sp, for each satisfied clause whose weight is larger than one, its weight is decreased by one; with probability $(1 - sp)$, the weights of all unsatisfied clauses are increased by one.

We first introduce the two scoring functions used in CScoreSAT. For the greedy search, CScoreSAT adopts the $cscore$ function. When reaching a local optimum, CScoreSAT makes use of a hybrid scoring function (denoted by $hscore$), which combines $cscore$ with the diversification property age: $hscore(x) = cscore(x) + \lfloor age(x)/\beta \rfloor$, where β is a (relatively large) positive integer parameter.

Algorithm 1: CScoreSAT

```
   Input: CNF-formula $F$, $maxSteps$
   Output: A satisfying assignment $\alpha$ of $F$, or "unknown"
1  begin
2      $\alpha$ := randomly generated truth assignment;
3      for $step := 1$ to $maxSteps$ do
4          if $\alpha$ satisfies $F$ then return $\alpha$;
5          if $\exists$ CDCC variables then
6              $v$ := the CDCC variable with the greatest $cscore$,
               breaking ties in favor of the oldest one;
7          else
8              update clause weights according to PAWS;
9              pick a random unsatisfied clause $C$;
10             $v$ := the variable in $C$ with the greatest $hscore$,
               breaking ties in favor of the oldest one;
11         $\alpha := \alpha$ with $v$ flipped;
12     return "unknown";
13 end
```

CScoreSAT works in two modes, i.e., the greedy mode or the diversification mode. If there exist CDCC variables, CScoreSAT works in the greedy mode. It picks the CDCC variable with the greatest $cscore$ value to flip, breaking ties by preferring the oldest one. If no CDCC variable is present, which means a local optimum is identified, then CScoreSAT switches to the diversification mode. It first updates clause weights according to the PAWS scheme. Then it randomly selects an unsatisfied clause C, and picks the variable from C with the greatest $hscore$ value to flip, breaking ties by favoring the oldest one.

IV. Main Parameters

We combine the WalkSAT and CScoreSAT algorithms, leading to an SLS solver also called CScoreSAT2013, which adopts WalkSAT to solve instances with $k \leq 3$, and adopts CScoreSAT to solve instances with $k > 3$.

WalkSAT has one parameter, namely the noise parameter wp. In CScoreSAT2013, wp is set to 0.567 when $r \leq 4.22$, 0.777-0.05r when $r \in (4.22, 4.23]$, 1.553-0.23r when $r \in (4.23, 4.26)$ and 2.261-0.4r when $r \geq 4.26$, where r is the clause-to-variable ratio.

CScoreSAT has three parameters, namely d, β and sp. Fortunately, d is simply defined as $13 - k$, and β is a constant (2000) for any instance. The sp parameter for PAWS is set to 0.62 for $k = 4$, $0.045r - 0.29$ for $k = 5$, 0.9 for $k = 6$, and 0.92 for $k > 6$.

V. Implementation Details

CScoreSAT2013 is implemented in C++. The CScore-SAT algorithm is implemented based on the codes of CCASat solver [7], which can be downloaded from www.shaoweicai.net/research.html, while the WalkSAT algorithm is implemented from scratch.

VI. SAT Competition 2013 Specifies

CScoreSAT2013 is submitted to "Core solvers, Sequential, Random SAT" and "Core solvers, Parallel, Random SAT" tracks. It is compiled by g++ with the 'O2' optimization option. It is a 32-bit binary.

Its running command is:

CScoreSAT2013 <instance file name> <random seed>.

References

[1] S. Cai and K. Su, "Local search for boolean satisfiability with configuration checking and subscore," *submitted to Artif. Intell.*, 2013.

[2] ——, "Comprehensive score: Towards efficient local search for sat with long clauses," in *Proc. of IJCAI-13*, 2013, p. to appear.

[3] ——, "Local search with configuration checking for SAT," in *Proc. of ICTAI-11*, 2011, pp. 59–66.

[4] ——, "Configuration checking with aspiration in local search for SAT," in *Proc. of AAAI-12*, 2012, pp. 334–340.

[5] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artif. Intell.*, vol. 175, no. 9-10, pp. 1672–1696, 2011.

[6] J. Thornton, D. N. Pham, S. Bain, and V. F. Jr., "Additive versus multiplicative clause weighting for SAT," in *Proc. of AAAI-04*, 2004, pp. 191–196.

[7] S. Cai, C. Luo, and K. Su, "CCASat: Solver description," in *Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions*, 2012, pp. 13–14.