Improving probability selection based weights for satisfiability problems

Fu, H., Liu, J., Wu, G., Xu, Y., & Sutcliffe, G. (2022). Improving probability selection based weights for satisfiability problems. *Knowledge-Based Systems*, *245*, 1-17. [108572]. https://doi.org/10.1016/j.knosys.2022.108572

Link to publication record in Ulster University Research Portal

Published in:
Knowledge-Based Systems

Publication Status:
Published (in print/issue): 07/06/2022

DOI:
10.1016/j.knosys.2022.108572

Document Version
Author Accepted version

General rights
Copyright for the publications made accessible via Ulster University's Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Ulster University's institutional repository that provides access to Ulster's research outputs. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact pure-support@ulster.ac.uk.

Download date: 25/03/2023

Improving Probability Selection Based Weights for Satisfiability Problems

Huimin Fu[1,*], Jun Liu[2], Guanfeng Wu[3,*], Yang Xu[3] , and **Geoff Sutcliffe[4]**

[1]Southwest University of Finance and Economics, Chengdu, China
[2]School of Computing, Ulster University, Northern Ireland, UK
[3]School of Mathematics, Southwest Jiaotong University, Chengdu 610031, China
[4]Department of Computer Science, University of Miami, USA.

[*] The corresponding author.
E-mail: fuhuimin@swufe.edu.cn (H. Fu) and wgf1024@swjtu.edu.cn (G. Wu);

Abstract

Boolean Satisfiability problem (SAT) plays a prominent role in many domains of computer science and artificial intelligence due to its significant importance in both theory and applications. Algorithms for solving SAT problems can be categorized into two main classes: complete algorithms and incomplete algorithms (typically stochastic local search (SLS) algorithms). SLS algorithms are among the most effective for solving uniform random SAT problems, while hybrid algorithms achieved great breakthroughs for solving hard random SAT (HRS) problem recently. However, there is a lack of algorithms that can effectively solve both uniform random SAT and HRS problems. In this paper, a new SLS algorithm named *SelectNTS* is proposed aiming at solving both uniform random SAT and HRS problem effectively. *SelectNTS* is essentially an improved probability selection based local search algorithm, the core of which includes new clause and variable selection heuristics: a new clause weighting scheme and a biased random walk strategy are utilized to select a clause, while a new probability selection strategy with the variation of configuration checking strategy is used to select a variable. Extensive experimental results show that *SelectNTS* outperforms the state-of-the-art random SAT algorithms and hybrid algorithms in solving both uniform random SAT and HRS problems effectively.

Introduction

Given a formula in clause normal form (CNF), the Boolean satisfiability (SAT) problem requires finding a Boolean assignment for the decision's variables that satisfies the formula. SAT has been widely studied as a canonical NP-complete problem. It has been extensively investigated in many domains of computer science and artificial intelligence due to its significance in both theory and applications [1]. Many practical problems in combinatorial optimization, statistical physics, circuit verification, and computing theory [44] can be converted into SAT problems, and thus SAT solvers have been widely used to solve real-world applications, such as computer algebra systems [9], scheduling [16], gene regulatory networks [17], core graphs [26], automated verification [38], machine induction [42], model-based diagnosis (MBD) [46], and scheduling [56]

Algorithms for solving SAT problems mainly include complete algorithms [25, 30, 36, 37, 39, 45] and incomplete algorithms [19, 24, 53]. Stochastic local search (SLS) algorithms among the incomplete algorithms are the most actively developing approaches [13, 21, 33].

In this work, we focus on the SLS algorithm. Although SLS algorithms are typically incomplete, they are often surprisingly effective in finding solutions to satisfiable random SAT problems. SLS algorithms are often evaluated on random SAT instances, including uniform random *k*-SAT problems [1] and hard random SAT (HRS) problems [3, 4]. Moreover, the class of random SAT instances is one of the three main tracks in the well-known SAT competitions [47]. The heuristics used by SLS solvers to solve random SAT problems are also potentially useful for solving real-world SAT problems [48].

In the beginning, an SLS algorithm generally generates an initial assignment of the variables of a SAT problem F in CNF. Then it explores the search space to minimize the number of unsatisfied clauses. To do this, it iteratively flips the truth value of a variable selected according to some heuristic at each step until it seeks out a solution or timeout [27-29]. Heuristics in SLS algorithms mainly differ from each other on the variable selection heuristics at each iteration.

The representative state-of-the-art SLS algorithms include the two-mode solvers containing the configuration checking (CC) algorithms (e.g., CCASat [14], Swqcc [50]), clause weighting algorithms (e.g., Pure Additive Weighting Scheme (PAWS) [41], Scaling and Probabilistic Smoothing (SAPS) [23] and Discrete Lagrangian Method (DLM) [43]), comprehensive score function algorithms (e.g., CScoreSAT [13] and DCCASat [31]) and focused random walk (FRW) solvers containing the tie-breaking algorithms (e.g., WalkSATlm [12], and FrwCBlm [33]), the algorithms based on the probability selection (e.g., ProbSAT [6,7] and YalSAT [8]), and hybrid algorithms (e.g., CCAnr [15], CSCCSat [34] Score$_2$SAT [10]) and other SAT solvers include in the literature [5, 28, 45].

No single SLS heuristic can be effective on all types of random SAT instances including HRS and uniform

random k-SAT instances with long clauses, which remain very difficult since different types of instances present different characteristics. Especially, as can be seen from the results of the random track of SAT Competition 2017 [54] and 2018 [55], all the participating solvers lost their power and effectiveness on several random SAT instances, especially for all HRS instances. One approach to designing an effective algorithm is to balance the search between intensification and diversification to guide the algorithm. Furthermore, it is also essential to solve different types of instances by developing new heuristics.

Following this spirit, we develop a new algorithm based on the basic framework of probability selection [6]. The general SLS algorithms based on probability selection for solving SAT include clause selection and variable selection which are two main factors affecting SLS algorithms. According to the features of the probability selection based algorithms, there are two important limitations: firstly, like general SLS solvers, it selects a clause from unsatisfied clauses randomly; secondly, using only probability selection may result in the same variable being selected in consecutive steps. These two observations constitute the main motivations of this work. We aim to improve the probability selection method based on the weights for random SAT by reinforcing the original algorithm ProbSAT [6], which remains an innovative and appealing approach due to its selecting feature and simplicity.

The main contribution of the present work is proposing the enhanced probability selection based SLS algorithm by using new weight concepts considering the feature of the random SAT problem and developing two weighting schemes, which can be summarized as follows:

1) Firstly, to distinguish the unsatisfied clauses and balance the number of times each clause is selected in the clause selection, we propose a new and global clause weighting scheme to guide the clause selection. The new clause weighting scheme, called *cNTS* (**c**lauses **N**umber of **T**imes **S**elected), counts the number of times a clause has been selected (in Section 4.1.1). *cNTS* is different from the existing clause weighting schemes which are usually updated according to a clause is satisfied or unsatisfied by flipping the value of a variable and only when the algorithms fall into local optimal [13, 14]. Based on *cNTS*, we define *hard satisfiable clauses* (*HSCs*) (in Section 4.1.2) to distinguish unsatisfied clauses. A biased random walk guided by *cNTS* and *HSCs* is adopted as a new clause selection heuristic (in Section 4.1.3).

2) Secondly, to avoid the same variable being selected in consecutive steps and balance the number of times each variable is selected in the variable selection, we adopt a variation of CC strategy based on a new and global variable weighting scheme called *vNTS* (**v**ariables **N**umber of **T**imes **S**elected), that counts the number of times a variable has been selected (in Section 4.2.1). Then we define a function called S_v, which is a linear combination of the commonly used *score* property and *vNTS*. Variable selection uses a probability selection method [7] with the variation of CC strategy based on S_v (in Section 4.2.2).

cNTS and *vNTS* play the key roles in the two new heuristics, and thus lead to a new SLS algorithm, called *SelectNTS* (**Select**ion based on the **N**umber of **T**imes clauses and variables are **S**elected) (in Section 4.3).

3) We assess the performance of the proposed *SelectNTS* algorithm on the instances of the well-known random track of SAT Competitions in 2017 and 2018 and generated by generators [1, 3]. Experimental results show that *SelectNTS* performs remarkably compared to the state-of-the-art SLS algorithms like ProbSAT [7], YalSAT [8], Sparrow [2], CscoreSAT [13] as well as Score$_2$SAT [10] and even a sophisticated hybrid algorithm called SparrowToRiss [5] on HRS instances. *SelectNTS* proves to be competitive even when it is compared to state-of-the-art SLS algorithms like ProbSAT, YalSAT, CscoreSAT, and Score$_2$SAT on uniform random k-SAT instances with long clauses.

4) Finally, we perform more empirical evaluations to analyze the effectiveness of the heuristics in *SelectNTS* algorithm and demonstrate its contribution to the performance of *SelectNTS* on HRS benchmarks, as well as the main difference between *SelectNTS* and *EPEFV* [59].

This paper is structured as follows: Section 2 provides some definitions, notations, and briefly

reviews some related heuristics. In Section 3, we review the general framework of the ProbSAT algorithm based on probability selection for solving random SAT. Section 4 presents our improved *SelectNTS* algorithm for random SAT. In Section 5, we illustrate some case studies. Section 6 presents the comparative experimental results and provides commentary. In Section 7, we summarize the main contributions of this work and suggest directions for future work.

2 Preliminaries

In this section, we introduce some of the definitions and related work to the SAT problem.

Definitions

A SAT problem F in CNF is constructed from a pair (V, C), where $V=\{v_1, v_2, ..., v_n\}$ is a set of n Boolean variables, and $C=\{c_1, c_2, ..., c_m\}$ is a set of m clauses. Each clause $c_i \in C$ is a disjunction of literals, and a literal is a variable v_i or its negation. $r = m/n$ is the clause-to-variable ratio. F is the conjunction of the clauses. An *assignment* for F is an assignment of truth values to its variables, and a *satisfying assignment* is an assignment that makes all the clauses true.

In SLS algorithms for SAT problems, for a variable v and assignment α, $score(v, \alpha)$ is the number of increase in satisfied clauses by flipping the assigned value of v. and $break(v, \alpha)$ is the number of satisfied clauses that become unsatisfied by flipping the assigned value of v.

SLS algorithms explore the search space aiming to minimize the number of unsatisfied clauses. To do this, it is natural to select a variable in an unsatisfied clause to flip. The heuristic factors are thus the clause and variable selection.

Related heuristics

Although our algorithm is different from the existing SLS SAT solver, *SelectNTS* inherits some excellent features of the previous algorithms. In this subsection, we review briefly the existing related heuristic algorithms and the variation approach of adopting these features into our *SelectNTS*.

Heuristics in SLS algorithms for SAT can be divided into two categories: two-mode SLS algorithms and focused random walk (FRW) algorithms.

Two-mode SLS algorithms include the greedy mode and the random mode. In the greedy mode, to increase the number of satisfied clauses, the algorithms prefer to select the greedy variables to be flipped. In random mode, to avoid local optimization, the algorithms randomly pick a variable to be flipped. For the two-mode SLS algorithms during the last ten years, the most significant development was perhaps the "configuration checking" strategy (CC) and "weights" strategy [13,18], leading to the effective CCASat [14], Swqcc [50], and DCCASat [31]. The CC idea takes the circumstance of the variable into account and prevents a variable from being flipped if its circumstance has not been changed since its last flip [14] (like the simple Tabu search strategy [49]). In SLS algorithms, the main idea of the weighting schemes is that greedily select the best variable to be flipped among the candidate variables (like the well-known GSAT [51]- the score function in GSAT actually is the weight of variables). Moreover, the weighting schemes have become the mainstream of optimizing SLS algorithms. *Our SelectNTS algorithm attempts to incorporate the ideas of both the inhibition of CC to avoid selecting the same variable in consecutive steps and intensification of variable weighting strategy to select the best variable to be flipped among the candidate variables*.

FRW algorithms always pick a variable to be flipped from an unsatisfied clause picked randomly in each step [33]. The WalkSAT [21] is the well-known the FRW algorithm. Cai *et al*. [12] introduced the tie-breaking strategy into the WalkSAT algorithm to prevent multiple candidate variables for solving uniform random k-SAT with long clauses. Cai *et al*. [53] improved the WalkSAT algorithm [12] by adopting the CC strategy. Luo *et al*. [33] adopted the tie-breaking strategy into FrwCB algorithm [53]. *Our SelectNTS algorithm borrows the idea of selecting an unsatisfied clause randomly to enhance its diversification capability and utilize the clause weighting strategy to distinguish unsatisfied clauses*.

The other direct improvement on WalkSAT is to extend it into a simple probability selecting strategy [20]. The ProbSAT [6] is obtained from WalkSAT by

associating a probability selecting method of the variables. YalSAT [8] and polypower1.0 [52] implemented several variants of ProbSAT's algorithm. The main principle of probability selection strategy for SAT in the literature [6-8] is that if a variable has the lowest *break* in an unsatisfied clause chosen randomly, then the variable is preferred to be selected. *We adapt this probability selecting strategy into our SelectNTS algorithm to enhance its robustness and utilize the variable weighting strategy to select the best variable to be flipped*.

In order to take advantage of both two-mode heuristics and FRW heuristics during the search process to guide the algorithm, Cai *et al*. [34] proposed a hybrid algorithm called CSCCSat, which is a combination of DCCASat [31] and FrwCB [53], and the hybrid algorithm called Score$_2$SAT [10] is a combination of the DCCASat [31] and WalkSATlm [12]. To enhance the performance of SLS algorithm, Cai *et al*. [15] applied to preprocess technology to the SLS algorithms, and Balint and Manthey [5] combined the SLS algorithm and complete algorithm.

3 Probability Selection Method for SAT

The probability selecting method proposed in the literature [6] is a general framework for solving SAT problems. In this section, we briefly review the probability selecting method [6], which is the basis of our algorithm. The ProbSAT algorithm [6] has a wide influence among current SLS algorithms and attracted increasing interest for solving SAT benchmarks in the last few years.

ProbSAT uses only the *break* values of a variable v and assignment α in a probability function $f(v, \alpha)$, including a polynomial or exponential shape as listed below:

$$f(v, \alpha))=(0.9+break(v, \alpha)))^{-cb1} \quad (1)$$

$$f(v, \alpha))=(cb_2)^{-break(v, a)} \quad (2)$$

where cb_1 and cb_2 are decimal parameters.

The ProbSAT algorithm is designed for solving random SAT instances. The pseudo-code of ProbSAT is described in the following Algorithm 1 and can be found in the literature [6, 7].

To apply the probability selection method to SAT problem, four processes need to be attended. The algorithm generates a complete assignment α randomly as the initial assignment (line 3 in Algorithm 1). During the search process, the algorithm picks an unsatisfied clause randomly (line 6 in Algorithm 1). During the probability updating process (lines 7-10 in Algorithm 1), the probability is updated by the *break* values of variables, while the probability is computed by the polynomial function in Eq. (1) for 3-SAT problems, and the probability is computed by the exponential function in Eq. (2) for the remaining problems. During the probability selecting process (line 11 in Algorithm 1), the algorithm based on the probability

$$\frac{f(x,a)}{\sum_{z\in C} f(z,a)} \quad (3)$$

tries to select a variable to be flipped.

In order to explore the search space to minimize the number of unsatisfied clauses, ProbSAT algorithm picks a variable to be flipped by some heuristics.

Algorithm 1 The ProbSAT Algorithm

Input: CNF-formula *F*, *MaxTries*, *MaxSteps*
Output: An assignment α of *F*, or *Unknown*
1: **for** *try:* = 1 to *MaxTries* **do**
2: α := a randomly generated truth assignment;
3: **for** *step*:= 1 to *MaxSteps* **do**
4: **if** α *satisfies F* **then return** α;
5: *C* := a random unsatisfied clause;
6: **for** *v* in *C* **do**;
7: compute $f(v, \alpha)$;
8: **end for**
9: $v := x \in C$ selected randomly with probability $\frac{f(x,\alpha)}{\sum_{z\in C} f(z,\alpha)}$;
10: $\alpha := \alpha$ with *v* flipped;

```
11:     end for
12:   end for
13:  return Unknown;
14: end
```

The variable selection heuristic of ProbSAT mainly depends on two factors, including clause selection strategy and variable selection strategy. Thus, the heuristic focusing on selecting clauses or variables is suggested in the EHC heuristic [35]. However, EHC may not be suitable for SAT problems. Since there are no hard clauses and soft clauses (all clauses in a weighted partial CNF formula are divided into hard ones and soft ones, and each soft clause is associated with a positive integer as its weight) [35] in SAT problems and the selecting strategy decides the direction of the search, it is reasonable for us to employ a heuristic focusing on the selecting strategy to solve SAT problems. In order to further improve SLS algorithms for SAT, we propose two new selection heuristics focusing on the selecting clauses or variables, which are detailed in subsequent Sections 4.

4 Improving Probability Selection Based Weights for SAT

In this section, we introduce our algorithm called *SelectNTS* and select the algorithm [6] based on the probability selection heuristic as the basic framework. The *SelectNTS* includes two important components - clause selection heuristic-based new clause weighting scheme and variable heuristic-based new variable weighting scheme.

4.1 The new clause selection heuristic

In this subsection, we define a new clause selection heuristic, composed of three components: a clause weighting scheme called *cNTS*, a notion of *hard satisfiable clauses*, and a biased random walk.

The strategy of picking an unsatisfied clause is known to be successful for general SAT solving [6, 7]. Indeed, the condition that the selected clause is unsatisfied is necessary, as selecting a satisfied clause may lead to a local optimum [51]. However, selecting from the unsatisfied clauses with equal probability does not provide enough guidance for SLS algorithms, especially for SAT problems. The number of times an unsatisfied clause is selected is an indication of how difficult it is to satisfy the clause. Based on **this observation** (in Section 5), we propose a new clause weighting scheme to distinguish unsatisfied clauses. In order to fully use the information of each clause in the SAT problem, we propose a new clause selection heuristic to balance the number of times each clause is selected.

4.1.1 New and global clause weighting scheme

Clause weighting schemes such as DLM [43], SAPS [23], PAWS [41], and SWT [31] have been utilized successfully in SLS algorithms for general SAT solving. However, according to the results of the random track of SAT Competition 2017 [54] and 2018 [55], we can see that these weighting schemes are not always effective for solving different types of SAT problems. This motivated us to design a new clause weighting scheme called *cNTS*, which counts the number of times each clause has been selected.

***Definition* 1** *For a clause c, in search step s, cNTS*(c, s) *is the number of steps that c has been chosen up to step s.*

Intuitively, clauses with larger *cNTS* values are easier to keep unsatisfied in the search process. Thus, it is beneficial for SLS algorithms to prefer satisfying these clauses, and we use *cNTS* to guide clause selection. The **major differences** between *cNTS* and existing clause weighting schemes are that although *cNTS* is clause weight, the scoring function (in Section 4.2.2) does not consider the clause weights, and *cNTS* is updated in the total search process, while previous schemes are updated only when the algorithms fall into local optimum [2, 11, 13, 14, 31].

4.1.2 Hard satisfiable clauses

Based on *cNTS*, we define the notion of Hard Satisfiable Clauses (*HSCs*).

***Definition* 2** *For a clause c, in search step s, and given a positive parameter* β, *c is called an HSC in step s if c is unsatisfied and* $cNTS(c, s) \geq \beta$.

HSCs(s,β) denotes the set of all *HSCs* in step s for the given β. *HSCs* are regarded as good candidates for selection, especially when solving HRS problems.

4.1.3 The biased random walk

An important component of most SLS algorithms is a random walk [40]. However, a standard random walk [6, 7, 8, 12] might not be suitable for HRS problems. *HSCs* are given higher priority in our algorithm by using a biased random walk [35] as follows: At each step s, if *HSCs* (s, β) is not empty, our algorithm picks an *HSC* randomly, otherwise, it chooses an unsatisfied clause randomly.

4.2 The new variable selection heuristic

In this subsection, we define a new variable selection strategy that is composed of two components: a new variable weighting scheme called *vNTS* and a variation of CC strategy based on a new scoring function.

In many SLS algorithms [11, 32, 35], the strategy for picking a variable to be flipped in each step is guided by the *score* property, which maximizes the number of satisfied clauses. In contrast, we differentiate between variables in unsatisfied clauses by probability selection method and then consider a combination of the *score* property and *the number of times each variable has been selected*. This is quite intuitive, as the more times a variable has been selected, the less likely it is that all clauses containing the variable are satisfied after subsequent variable flips. We also have used **this observation** (in Section 5) as the basis for a new variable weighting scheme, and in order to fully use the information of each variable in the SAT problem, we propose a new variable selection heuristic to balance the number of times each variable selected.

4.2.1 The new and global variable weighting scheme

The new variable weighting scheme is defined as follows.

***Definition* 3** *For a variable v, in the search step s, vNTS*(v, s) *is the number of steps that v has been chosen up to step s.*

Intuitively, clauses containing variables with larger *vNTS* are easier to keep unsatisfied in the search process, and we use *vNTS* to guide variable selection.

4.2.2 A variation of CC strategy

The *score* property tends to increase the number of satisfied clauses in a greedy search mode, and *vNTS* can be regarded as a heuristic for greedy search as its use tends to reduce *HSCs*(s,β) by flipping the variables of an *HSC*. To combine *score* and *vNTS* in a greedy search, we define a scoring function that is a linear combination of *score* and *vNTS*, inspired by the concept of a comprehensive score [13]. The new scoring function, named S_v, for a variable v at step s when the assignment is α, is defined as follows:

***Definition* 4** For a variable v, in the search step s, a complete assignment α, and given a positive integer parameter γ, $S_v(v, s, \alpha) = score(v, \alpha) + vNTS(v, s)/\gamma$.

In our algorithm, the variable is picked to be flipped firstly by the probability selection method [6]. However, using only the probability selection method may lead to the same variable to be selected in consecutive steps. To avoid this, we employ a variation of CC strategy. If the variable selected by probability at step s is the same as the variable flipped in step s - 1, a different variable with the greatest S_v value is selected instead. The CC strategy has proved effective in the SLS algorithm for random SAT problems, but it is technically significantly harder to track and utilized to select a variable from all variables in a SAT problem. Thus, as the variation of CC strategy, S_v can be computed with little overhead.

This variation of CC strategy is inspired by the idea in the literature [14] but is essentially different. In our algorithm, a variable is selected to be flipped from an unsatisfied clause selected randomly, and there is no need to select candidate variables from all variables.

4.3 The *SelectNTS* Algorithm

In this section we present the *SelectNTS*, whose pseudo code is shown in Algorithm 2. *SelectNTS* has an outer loop that (re)starts with a randomly generated truth assignment, looping maximally *MaxTries* times (lines 1-2). *bestVar* is utilized to record which variable was flipped in the last step (line 3). Within that outer loop, the inner loop searches for a satisfying assignment with up to *MaxSteps* variable flips (line 4). If the current assignment satisfies all clauses of F, then *SelectNTS*

returns the assignment (line 5). Otherwise *SelectNTS* proceeds with the biased random walk: if *HSCs*(s, β) is not empty (line 6), an *HSC* is selected randomly (line 7), and otherwise, an unsatisfied clause is selected randomly (line 8). *cNTS* is updated for the selected clause (line 9). Then *SelectNTS* picks a variable from the selected clause by the probability selection method (line10), and if the variable is the same as *bestVar* (line 11), then *SelectNTS* selects a variable with the greatest S_v value (line 12). *vNTS* is updated for the selected variable (line 14). Then *SelectNTS* flips the assignment value of the chosen variable (line 15) and starts the next search step. If *MaxTries* is reached, *SelectNTS* reports *Unknown* (line 16).

Algorithm 2 The *SelectNTS* Algorithm

Input: CNF-formula F, *MaxTries*, *MaxSteps*, γ, β
Output: A compelete assignment α of F, or *Unknown*
1: **for** *try:* = 1 to *MaxTries* **do**
2: α := a randomly generated truth assignment;
3: *bestVar* := null;
4: **for** *step:*= 1 to *MaxSteps* **do**
5: **if** α *satisfies* F **then return** α;
6: **if** *HSCs*(*step*,β) ≠ Ø **then**
7: C := a *HSC* selected randomly;
8: **else** C := an unsatisfied clause selected randomly;
9: update *cNTS*;
10: $v := x \in C$ selected with probability $\frac{f(x,\alpha)}{\sum_{z\in C} f(z,\alpha)}$;
11: **if** v :== *bestVar* **then**
12: *bestVar* := $x \in C, x \neq v,$ with greatest $S_v(x,s,\alpha)$;
13: **else** *bestVar* := v;
14: update *vNTS*;
15: $\alpha := \alpha$ with *bestVar* flipped;
16: **return** *Unknown*;

5 Case Studies

In Section 4.1 and Section 4.2, we introduce the definitions of *cNTS* and *vNTS* and propose the new clause selection heuristic and variable selection heuristic respectively.

Different types of SAT problems may provide different distributions of *cNTS* and *vNTS* respectively. We have studied a large number of SAT instances with the aim of determining the distribution of each of them. In this section, we present two case studies (at HRS and uniform random k-SAT from SAT Competition 2017 [54]) about distributions of *cNTS* and *vNTS* within 10^5 steps in both the original algorithm ProbSAT based probability selection [7] and our *SelectNTS* algorithm (parameter settings in Section 6.2) respectively.

5.1 Case 1: fla-qhid-540-5

In this first case, we study a HRS instance which distributions correspond to the *cNTS* and *vNTS* within 10^5 steps in both ProbSAT and our algorithm *SelectNTS* respectively. Table 1 exhibits several important information of the HRS instance fla-qhid-540-5 from the random track of SAT Competition 2017.

Table 1 Statistical description of HRS instance fla-qhid-540-5.

Benchmark's name	fla-qhid-540-5
Number of clauses	2970
Number of variables	540
ratio	5.5
Generator seed	5

Fig. 1 and Fig. 2 illustrate the distributions of *cNTS* and *vNTS* for the instance within 10^5 steps on ProbSAT. Fig. 1 shows the number of times each clause is selected is quite different. Within 10^5 steps, the maximum value of *cNTS* is close to 2500. From Fig. 2, the distribution of *vNTS* is uneven, and the maximum value of *vNTS* is close to 2000 within 10^5 steps.

The intuition is that the larger *cNTS* of a clause is an indication of how difficult it is to keep the clause satisfied, and as the larger *vNTS* of a variable, the less likely it is that all clauses containing the variable are satisfied after subsequent variable flips. In order to fully

use the information of each clause and variable, the new clause selection heuristic and the new variable selection heuristic are used to balance the number of times each clause selected and each variable selected respectively. The distributions of *cNTS* and *vNTS* for the instance within 10^5 steps on *SelectNTS* are presented in Fig. 3 and Fig. 4 respectively.

Comparing Fig. 3 with Fig. 1, and Fig. 4 with Fig. 2, the distributions of *cNTS* and *vNTS* are relatively more **uniform** on *SelectNTS* than of that on ProbSAT respectively. Specially, the maximum value of *cNTS* on *SelectNTS*, which is about **0.4** times as large as on ProbSAT, is close to 1000. The maximum value of *vNTS* on *SelectNTS*, which is about **0.5** times as large as on ProbSAT, is close to 1000. It follows that the new clause selection heuristic and variable selection heuristic dramatically balances the values of *cNTS* and *vNTS* for this HRS instance respectively. Thus, the new clause selection heuristic and the new variable selection heuristic play important roles in *SelectNTS*.

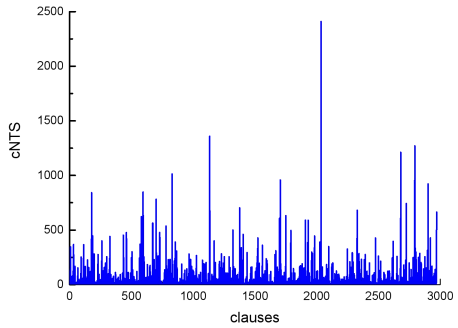

Fig. 1. fla-qhid-540-5 Distribution of *cNTS* within 10^5 steps on ProbSAT.

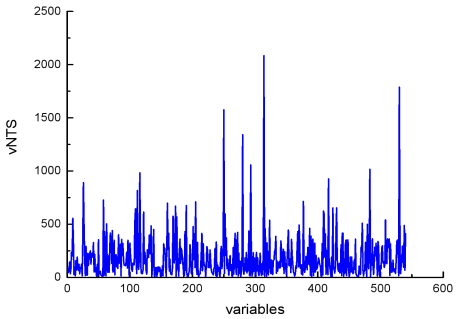

Fig. 2. fla-qhid-540-5 Distribution of *vNTS* within 10^5 steps on ProbSAT.

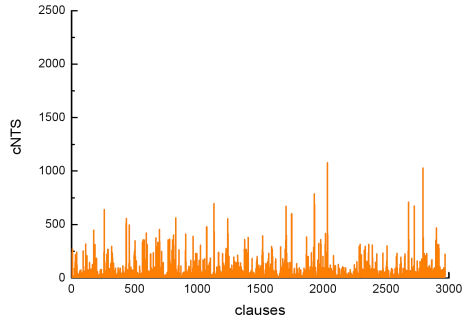

Fig. 3. fla-qhid-540-5 Distribution of *cNTS* within 10^5 steps on *SelectNTS*.

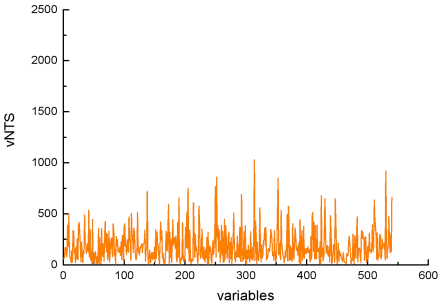

Fig. 4. fla-qhid-540-5 Distribution of *vNTS* within 10^5 steps on *SelectNTS*.

5.2 Case 2: unif-k5-r21.117-v540-c11403

In this second case, we study a uniform random *k*-SAT instance in which distributions correspond to the *cNTS*and *vNTS* within 10^5 steps in both ProbSAT and *SelectNTS* respectively.

Table 2 summarizes the most important information of the *unif-k5-r21.117-v540-c11403* instance from the random track of SAT Competition 2017.

Table 2 Statistical description of the unif-k5-r21.117-v540-c11403.

Benchmark's name	unif-k5-r21.117-v540-c11403
Number of clauses	11403
Number of variables	540
ratio	21.117
Generator seed	5955214796121725857

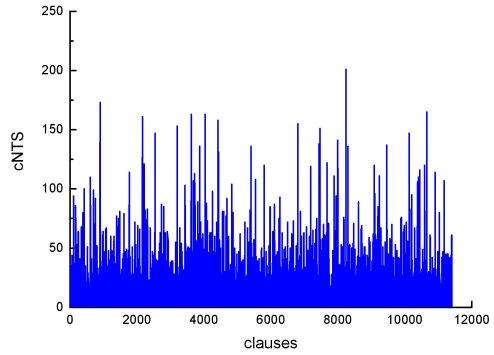

Fig. 5. unif-k5-r21.117-v540-c11403 Distribution of *cNTS* within 10^5 steps on ProbSAT.

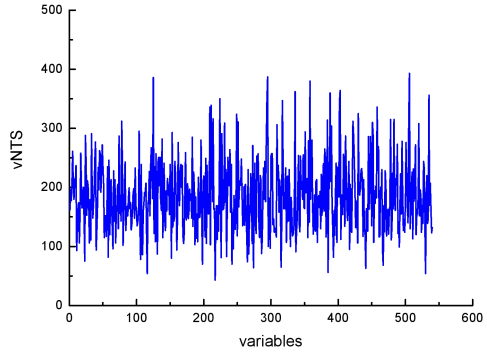

Fig. 6. unif-k5-r21.117-v540-c11403 Distribution of *vNTS* within 10^5 steps on ProbSAT.

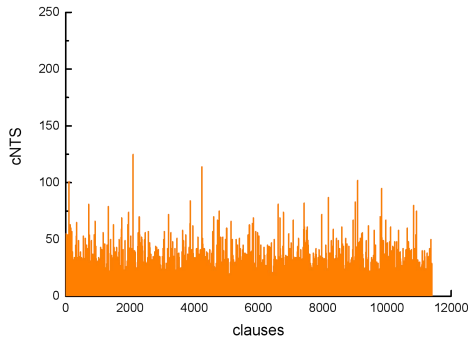

Fig. 7. unif-k5-r21.117-v540-c11403 Distribution of *cNTS* within 10^5 steps on *SelectNT*

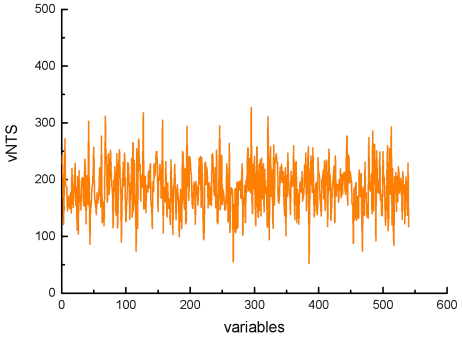

Fig. 8. unif-k5-r21.117-v540-c11403 Distribution of *vNTS* within 10^5 steps on *SelectNTS*.

Fig. 5 and Fig. 6 illustrate the distributions of *cNTS* and *vNTS* for the instance within 10^5 steps on ProbSAT. Fig. 7 and Fig. 8 illustrate the distributions of *cNTS* and *vNTS* for the instance within 10^5 steps on *SelectNTS*.

Comparing Fig. 5 with Fig. 7, and Fig. 6 with Fig. 8 respectively, the distributions of *cNTS* and *vNTS* are relatively more **uniform** on *SelectNTS* than of that on ProbSAT respectively. Specially, the maximum value of *cNTS* is about 200 on ProbSAT from Fig. 5, while the maximum value of *cNTS* is close to 150 on *SelectNTS* from Fig. 7, i.e., the maximum value of *cNTS* on *SelectNTS* is about **0.6** times as large as on the one on ProbSAT. The maximum value of *vNTS* is about 400 on ProbSAT from Fig. 6, while the maximum value of *cNTS* is close to 350 on *SelectNTS* from Fig. 8, i.e., the maximum value of *vNTS* on *SelectNTS* is about **0.8** times as large as on the one on ProbSAT. It follows that our algorithm *SelectNTS* can also dramatically balance the values of *cNTS* and *vNTS* for this uniform random SAT instance respectively.

Comparing Fig. 1 with Fig. 3, Fig. 2 with Fig. 4, Fig. 5 with Fig. 7, and Fig. 6 with Fig. 8 respectively, the distributions of *cNTS* and *vNTS* on the HRS instances change more obviously than those on the uniform random SAT instance respectively. Thus, comparing Case 1 with Case 2, the new clause selection heuristic and the new variable selection heuristic on distributions of *cNTS* and *vNTS* for the HRS instances have more influence than those for the uniform random SAT instances under the same variable size. We conjecture that the performance of *SelectNTS* in solving different random SAT problems may be related to the degree of

influencing distributions of *cNTS* and *vNTS* on the *SelectNTS* compared to the ProbSAT.

6 Experimental Evaluations

We carried out extensive experiments to evaluate *SelectNTS* on random SAT problems. For each class, we compared *SelectNTS* with the state-of-the-art SLS solvers and a hybrid solver.

6.1 Benchmarks

All the HRS problems utilized in our experiments were generated by the HRS tool [3], and all the uniform random k-SAT problems utilized were generated by a generator [56]. Specifically, we used the following problems:

- **SAT Competition 2017:** SAT problems taken from the random track of the SAT Competition 2017. All random HRS instances (120 instances, 40 for each ratio r, r=4.3, 5.206, 5.5, 15 instances for each variable n = 400, 420, 440, …, 540), which vary in both size and ratio. All uniform random k-SAT instances with k>3 (120 instances, 60 for each k-SAT, k=5, 7), which vary in both size and ratio. The uniform random 5-SAT instances vary from 200 variables at the threshold ratio of phase transition $r \approx 21.117$ to 590 variables, from 16.0 ratio at n=250000 to 19.8 ratio (r<21.117). The uniform random 7-SAT instances vary from 90 variables at the threshold ratio of phase transition r≈87.79 to 168 variables, from 55.0 ratio at n=50000 to 74.0 ratio (r<87.79). These HRS and uniform random instances occupy 80% of the random track in SAT Competition 2017, indicating the importance of these instances.
- **HRS Random 5.206**: HRS problems generated by the HRS tool. r≈5.206, n=600, 700, ···, 1000 (1000 instances, 200 instances for each size).
- **HRS Random 5.5**: HRS problems generated by the HRS tool. r = 5.5, n = 600, 700, ⋯, 1000 (1000 instances, 200 instances for each size).
- **HRS Random 5.699**: HRS problems generated by the HRS tool. r = 5.699, n = 200, 300, ⋯, 1000 (900 instances, 100 instances for each size).
- **HRS Random 7.821**: HRS problems generated by the HRS tool. r = 7.821, n = 200, 300, ⋯, 1000 (900 instances, 100 instances for each size).
- **Uniform random 5-SAT:** Random 5-SAT problems generated by the k-SAT generator [56] (250 instances). *Medium 5-SAT instances* at the threshold ratio of phase transition (r≈ 21.115, 100 instances, n=200, 250, 300, 350, 400, 20 instances for each size). *Huge 5-SAT instances* at r<21.117 (n=250000, 150 instances, r=18.0, r=18.2, r=18.4, 50 instances for each ratio).
- **Uniform random 7-SAT:** Random 7-SAT problems generated by the k-SAT generator (250 instances). *Medium 7-SAT instances* at the threshold ratio of phase transition (r≈87.79, 100 instances, n=100, 110, 120, 130, 140 20 instances for each size). *Huge 7-SAT instances* at r< 87.79 (n=50000, 150 instances, r=65.0, r=66.0, r=67.0, 50 instances for each ratio).
- **SAT Competition 2018**: the benchmark from the random track of SAT Competition in 2018. The HRS instances and uniform random k-SAT instances with k>3 have various sizes and ratios. These instances occupy 88.2% of the random benchmark in SAT Competition 2018.

6.2 Experimental preliminaries

According to Section 5.2, we know the influence of *SelectNTS* on the distribution of *cNTS* and *vNTS* when solving HRS problems are more obvious than that when solving uniform random problems, so we share the insight on how the clause selection and variable selection change when the parameters β and γ take different values for solving fla-qhid-540-5 instance (described in Section 5).

Fig. 9 and Fig. 10 below illustrate the distributions of *cNTS* and *vNTS* for the instance within 10^6 steps on *SelectNTS* under different parameter β or γ. 'YES' indicates a successful run within 10^6 steps under parameters settings, and 'No' indicates an unsuccessful run within 10^6 steps under parameters settings.

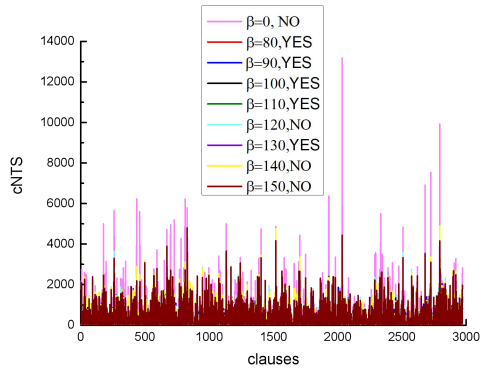

Fig. 9. fla-qhid-540-5 Distribution of *cNTS* within 10^6 steps on *SelectNTS* under γ=1200 and different parameter $\boldsymbol{\beta}$.

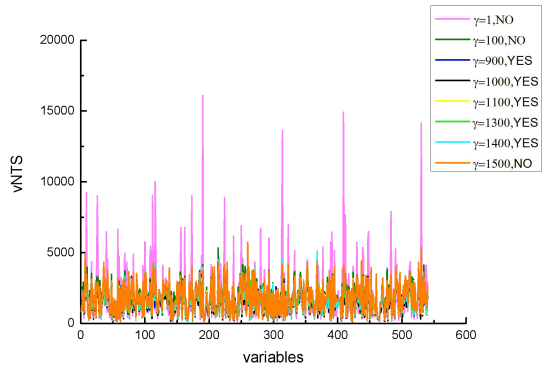

Fig. 10. fla-qhid-540-5 Distribution of *vNTS* within 10^6 steps on *SelectNTS* under $\boldsymbol{\beta}$=110 and different parameter $\boldsymbol{\gamma}$.

According to Fig. 9, if an β setting is marked 'NO', within 10^6 steps, the maximum value of *cNTS* is greater than 4000 for solving fla-qhid-540-5 instance; and if a β setting is marked 'YES', within 10^6 steps, the maximum value of *cNTS* is less than 4000 for solving fla-qhid-540-5 instance.

According to Fig. 10, if a γ setting is marked 'NO', within 10^6 steps, the maximum value of *vNTS* is greater than 5000 for solving fla-qhid-540-5 instance; and if a γ setting is marked 'YES', within 10^6 steps, the maximum value of *vNTS* is less than 5000 for solving fla-qhid-540-5 instance.

The purpose of our method is to **distinguish the unsatisfied clauses** and **balance the number of times each clause being selected in the clause selection** as highlighted in Section 1. Some further clarifications about parameter β and γ are summarized below:

If β is less than a threshold, there may be many clauses that belong to HSC, so it will not be able to effectively distinguish the unsatisfied clauses and balance the number of times each clause being selected in the clause selection.

If β is greater than a threshold, there may be less clauses that belong to HSC, so it will also not effectively distinguish the unsatisfied clauses.

If γ is less than a threshold, it can clearly distinguish the variables, but it will be too greedy and lose randomness for solving SAT instances, and it is easy to fall into local optimal.

If γ is greater than a threshold, there may be many variables that have the same function S_v, so it will not effectively distinguish the variables, so it will be too random for solving SAT instances.

However, different instances with different numbers of variables or clauses have different threshold, we performed an extensive number of experiments to find optimal parameters (i.e., threshold) according to our experience.

Implementation: *SelectNTS* is implemented in C. The the parameters β and γ of *SelectNTS* are showed in Table 3 and Table 4 according to our experience. For cb_1 and cb_2, we use the default parameter setting tuned in the literature [7].

Table 3 Parameter settings of *SelectNTS* for HRS instances.

	r=4.3	r=5.206	r=5.5/5.699	r=7.821
$n\leq600$		β = 80	β = 110	
	β = 10	γ =300	γ =1200	β = 400
$n>600$	γ=1200	β = 60	β = 110	γ =300
		γ =800	γ =900	

Table 4 Parameter settings of *SelectNTS* for uniform *k*-SAT.

	5-SAT	7-SAT
medium	β=5000000	β=700000
instances	γ =500000	γ=500000
huge	β = 700	β = 2000
instances	γ =600	γ =4000

Competitors: In the following, we use RSC to denote the **R**andom track of the **S**AT **C**ompetition. In order to evaluate the performance of *SelectNTS*, we compare *SelectNTS* with the following solvers including one hybrid solver and 4 state-of-the-art SLS SAT solvers:

- SparrowToRiss (denoted by STR in the result tables) [5]: a hybrid algorithm is the 1st place in RSC 2018.
- ProbSAT [7]: The algorithm based on the probability selection is the 2nd place among the SLS algorithms in RSC 2018 and 1st place in RSC 2013.

- YalSAT [8]: The algorithm based on the probability selection is the 1st place in RSC 2017.
- Score$_2$SAT [10]: The algorithm with CC strategy and clause weighting scheme is the 2nd place in RSC 2016.
- CScoreSAT [13]: The algorithm with CC strategy, clause weighting scheme and complex scoring functions.

ProbSAT and SparrowToRiss are available from the web site of the SAT Competition 2018 [55]. YalSAT and Score$_2$SAT are available from the web site of the SAT Competition 2017 [54]. CScoreSAT is available from the web site of SAT Competition 2013 [57].

Evaluation Methodology: The experiments are carried out on an Intel(R) Core (TM) i7-6700M 3.4 GHz CPU with 16GB RAM, running the 64-bit Ubuntu Linux operating system. The CPU time limit is 600 seconds for the HRS Random 5.206, 5.5, 5.699, 7.821 problem sets (as in the literature [4]), and 5000 seconds for remaining benchmarks (as in the recent SAT competitions).

For all benchmarks, we run each solver 10 times for each instance. For performance metrics, we report the number of averages solved instances at ten-run "AverS" and the penalized average run time "PAR-2" (an unsuccessful run is penalized two times the time limit) as in the competitions. Note PAR-2 is the average CPU time taken for cases of 100% success rate. The best results for each instance class are highlighted in **bold**.

Note: we do not compare our solver against the state-of-the-art complete solvers, such as the top 3 solvers from recent SAT Competitions. We know that when evaluating on random SAT instances, complete solvers are not suitable for solving random SAT problems according to the literature [59].

6.3 Experimental results

In this subsection, we present the experimental results of *SelectNTS* and its competitors on each instance set.

Results for SAT Competition 2017

Table 5 presents the experimental results on random SAT instances from SAT Competition in 2017. Since *SelectNTS* is developed based on ProbSAT, we first compare these two solvers. From Table 5, although ProbSAT spends a little less time than *SelectNTS* for the HRS instances with r=4.3 and the uniform 7-SAT instances with r≈87.79, ProbSAT and *SelectNTS* solve the same number of instances. For remaining instance classes, *SelectNTS* solves more instances than ProbSAT. Overall, ProbSAT solves 102 instances on average, while *SelectNTS* solves 177 instances on average, which is 1.74 times as many as ProbSAT does.

SelectNTS solves more instances than its competitors. Overall, *SelectNTS* solves 177 instances on average, compared to 158 for SparrowToRiss on average and 102 for ProbSAT on average and 101 for both YalSAT and Score$_2$SAT on average, and 100 for CScoreSAT on average. Further observation shows that for HRS instances classes, *SelectNTS* significantly outperforms SparrowToRiss [55] and for solving uniform random k-SAT instances with k>3, *SelectNTS* significantly outperforms CScoreSAT, YalSAT, Score$_2$SAT and ProbSAT which are among the most successful solvers for solving uniform random k-SAT instances in the literature.

Table 5 Computational results on the **SAT Competition 2017** benchmark.

Random SAT	Ratio	STR		CScoreSAT		Score$_2$SAT		YalSAT		PobSAT		*SelectNTS*	
		AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
HRS	4.3	40	0.117	40	**0.009**	40	0.020	40	0.017	40	0.062	40	0.134
	5.206	40	5.709	0	-	0	-	0	-	0	-	40	**0.032**
	5.5	40	151.0	6	8500	9	7750	9	7750	9	7750	40	**0.113**
Uniform	<21.117	4	8083	10	5250	8	6231	12	4147	11	4526	**13**	**3741**
	21.117	9	7760	15	**6476**	14	6655	13	6880	13	6829	14	6661
	<87.79	9	5602	11	4839	11	5756	10	5067	11	4514	**12**	**4045**
	87.79	16	6035	18	5931	**19**	**5582**	17	5957	18	5552	18	5800
Overall/240		158	3466	100	5992	101	5997	101	5866	102	5775	**177**	**2733**

Table 6 Computational results on the **HRS Random 5.206** benchmark.

	n=600	n=700	n=800	n=900	n=1000

	AverS PAR-2	AverS PAR-2	AverS PAR-2	AverS PAR-2	AverS PAR-2
STR	200 5.812	140 363.6	40 960.4	80 722.2	0 -
ProbSAT	40 960.0	40 960.0	40 960.0	40 960.0	0 -
YalSAT	40 960.0	40 960.0	40 960.0	40 960.0	0 -
Score$_2$SAT	40 960.0	40 960.0	40 960.0	0 -	0 -
CScoreSAT	0 -	0 -	0 -	0 -	0 -
SelectNTS	200 **0.179**	**200** **0.217**	**200** **0.258**	**200** **0.339**	**200** **0.314**

In order to investigate the detailed performance of*SelectNTS* on HRS instances with $r\approx5.206$, 5.5, 5.699, and 7.821, we utilize the HRS tool to generate larger size HRS instances. Then we test *SelectNTS* and its competitors on these HRS Random benchmark.

Results for HRS Random 5.206

Table 6 suggests that the difficulty of such HRS instances with $r\approx5.206$ increases significantly with a relatively large increment of the size. According to Table 6, the results show *SelectNTS* dramatically outperforms its competitors. For example, all competitors fail in all instance classes, and SparrowToRiss solves 200, 200, 160, 120, 120 instances for each instance class on average respectively, while *SelectNTS* solves all instances for each instance class on average. Given the good performance of *SelectNTS* on HRS Random 5.206 instances with 1000 variables, it is potential to solve larger size instances with r=5.206.

Results for HRS Random 5.5

The experimental results are showed in Table 7. It is clear that *SelectNTS* presents significantly better performance than Score$_2$SAT, CScoreSAT, YalSAT, ProbSAT, and SparrowToRiss on the whole benchmark. Also, *SelectNTS* significantly outperforms its competitors in terms of par 2, which is more obvious as the instance size increases. In particular, on the HRS instance with n=900, Score$_2$SAT and CScoreSAT fail in all instances, and the other competitors solve less than 80 instances on average, while *SelectNTS* solves all instances (200) on average. Finally, *SelectNTS* is the only solver which solves the total HRS Random 5.5 benchmark on average, which illustrates its robustness.

Table 7 Computational results on the **HRS Random 5.5** benchmark.

	n=600 AverS PAR-2	n=700 AverS PAR-2	n=800 AverS PAR-2	n=900 AverS PAR-2	n=1000 AverS PAR-2
STR	200 11.25	200 11.81	160 273.0	120 505.2	120 504.5
ProbSAT	0 -	0 -	0 -	0 -	0 -
YalSAT	0 -	0 -	0 -	0 -	0 -
Score$_2$SAT	0 -	0 -	0 -	0 -	0 -
CScoreSAT	0 -	0 -	0 -	0 -	0 -
SelectNTS	200 **0.038**	200 **0.073**	**200** **0.088**	**200** **0.065**	**200** **0.145**

Table 8 Computational results on the **HRS Random 5.699** benchmark.

Variables	STR		CScoreSAT		Score$_2$SAT		YalSAT		PobSAT		*SelectNTS*	
	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
n=200	100	42.01	0	-	0	-	0	-	0	-	100	**0.023**
n=300	100	94.28	0	-	0	-	0	-	0	-	100	**0.037**
n=400	100	213.5	0	-	0	-	0	-	0	-	100	**0.067**
n=500	100	228.2	0	-	0	-	0	-	0	-	100	**0.080**
n=600	80	444.1	0	-	0	-	0	-	0	-	**100**	**0.102**
n=700	40	855.9	0	-	0	-	0	-	0	-	**100**	**0.128**
n=800	0	-	0	-	0	-	0	-	0	-	**100**	**0.157**
n=900	0	-	0	-	0	-	0	-	0	-	**100**	**0.185**
n=1000	0	-	0	-	0	-	0	-	0	-	**100**	**0.197**

Results for HRS Random 5.699

We conduct further evaluations of *SelectNTS* with its competitors and a hybrid solver on HRS Random 5.699 benchmark [3].

The comparative results are presented in Table 8. For HRS instances with *n*=200, 300, 400, 500 classes, *SelectNTS* solves the same number of instances on average as SparrowToRiss, but *SelectNTS* spend less accumulative run time. For HRS instances with *n*=600, 700, 800, 900, and 1000 classes, *SelectNTS* succeeds in the most instances. Especially, *SelectNTS* shows dramatically superior performance than its competitors on HRS instances with *n*=800, 900, 1000 classes, where it solves all instances on average for each class, while its competitors fail to solve an instance for any of these instance classes.

Results for HRS Random 7.821

Table 9 shows the experimental results of *SelectNTS* and its competitors on the HRS Random 7.821 benchmark. It is promising to see the performance of *SelectNTS* remains surprisingly good on this benchmark, where its competitors present rather poor performance, especially for SLS solvers. For the experimental results, Score$_2$SAT, CScoreSAT, YalSAT, ProbSAT fail to solve an instance for the whole benchmark, while *SelectNTS* solves all instances on average. Although *SelectNTS* solves the same number of instances on average as SparrowToRiss for the whole benchmark, *SelectNTS* is over 138 times faster than SparrowToRiss on average in the whole HRS Random 7.821 instances indicating that *SelectNTS* is the comprehensive best algorithm on this benchmark, indicating that the *SelectNTS* algorithm achieves state-of-the-art performance on HRS instances with *r*=7.821.

In order to investigate the detailed performance of *SelectNTS* on uniform medium and huge random *k*-SAT instances with *k*>3, we utilize the *k*-SAT generator to generate some these types of instances, which size increases more fast than the one on medium uniform random *k*-SAT with *k*>3. Then we additionally test *SelectNTS* and its competitors on these uniform random benchmarks.

Table 9 Computational results on the **HRS Random 7.821** benchmark.

Variables	STR		CScoreSAT		Score$_2$SAT		YalSAT		PobSAT		*SelectNTS*	
	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
n=200	100	202.6	0	-	0	-	0	-	0	-	100	**0.224**
n=300	100	213.8	0	-	0	-	0	-	0	-	100	**0.423**
n=400	100	219.1	0	-	0	-	0	-	0	-	100	**0.657**
n=500	100	236.6	0	-	0	-	0	-	0	-	100	**0.874**
n=600	100	252.9	0	-	0	-	0	-	0	-	100	**1.029**
n=700	100	253.9	0	-	0	-	0	-	0	-	100	**1.468**
n=800	100	279.8	0	-	0	-	0	-	0	-	100	**1.550**
n=900	100	277.3	0	-	0	-	0	-	0	-	100	**1.911**
n=1000	100	314.9	0	-	0	-	0	-	0	-	100	**2.271**

Table 10 Computational results on the **Uniform random 5-SAT** benchmark.

Ratio	Variable	STR		CScoreSAT		Score$_2$SAT		YalSAT		PobSAT		*SelectNTS*	
		AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
Medium instances													
	n=200	11	4516	11	4506	11	4523	11	4513	11	4513	11	**4501**
	n=250	9	5582	10	5069	9	5502	10	5247	10	5142	10	**5022**
r=21.117	n=300	3	8525	8	6298	9	6078	10	**5283**	8	6122	9	5993
	n=350	8	6091	12	4166	13	3749	13	3734	13	3734	13	**3703**
	n=400	1	9510	3	8667	3	8728	2	9216	3	**8613**	3	8703
Huge instances													
r=18.0		0	-	0	-	50	3768	50	354.2	48	831.0	50	**476.5**
r=18.2	n=2.5×10^5	0	-	0	-	0	-	46	1626	0	-	**50**	**1558**
r=18.4		0	-	0	-	0	-	0	-	0	-	**50**	**3900**

Results for Uniform Random 5-SAT

We present the experimental results of *SelectNTS* for uniform medium and huge 5-SAT instances in Table 10. Table 10 indicates our *SelectNTS* algorithm performs quite well on these uniform random 5-SAT instances. Especially, for all 8 instance classes, *SelectNTS* shows the best performance for 6 instance classes, while YalSAT and PrboSAT show the best performance only for one instance class and the remaining three algorithms SparrowToRiss, CScoreSAT, and Score$_2$SAT cannot show the best performance for any instance classes.

Especially, for medium 5-SAT instances with n=200, 250, 350, *SelectNTS*, YalSAT, and PrboSAT solve the same number of instances on average, but *SelectNTS* spend less run time. For medium 5-SAT instances with n=300, *SelectNTS* has a similar performance as the best solver YalSAT, solving only one less instance on average. For medium 5-SAT instances with n=400, although ProbSAT has less run time, *SelectNTS* solves the same number of instances on average as ProbSAT. The huge 5-SAT instances with a few million clauses and the ratio from far from the phase-transition ratio to relatively close, are as large as some of the application benchmarks. As can be seen from Table 10, *SelectNTS* is based on ProbSAT, while *SelectNTS* solves more instances than ProbSAT. Overall, ProbSAT solves 48 (out of 150) instances on average, while *SelectNTS* solves all instances, which is 3 times as many as ProbSAT does. *SelectNTS* solves more instances than SparrowToRiss, CScoreSAT, YalSAT, and Score$_2$SAT. Especially, *SelectNTS* solves 150 instances on average, and YalSAT solves 96 (out of 150) instances on average, and Score$_2$SAT solves 50 (out of 150) instances on average, and while SparrowToRiss and CScoreSAT have difficulty in solving these huge random 5-SAT instance classes. In sum, this experiment further confirms the efficiency of *SelectNTS* for solving the general uniform medium and huge random 5-SAT problems

Table 11 Computational results on the **Uniform random 7-SAT** benchmark.

Ratio	Variable	STR		CScoreSAT		Score$_2$SAT		YalSAT		PobSAT		*SelectNTS*	
		AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
Medium instances													
	n=100	12	**4013**	12	4043	12	4037	12	4044	12	4044	12	4042
	n=110	10	5087	10	5141	11	4592	11	4749	11	4559	11	**4558**
r=87.79	n=120	9	5626	9	5780	10	5248	10	5451	9	5969	**11**	5442
	n=130	10	5123	10	5518	13	**3981**	13	4412	12	4380	12	4324
	n=140	10	5087	11	4829	13	**4048**	10	5397	10	5597	13	4566
Huge instances													
r=65.0		0	-	40	4860	41	5234	0	-	47	778.2	**50**	**268.2**
r=66.0	n=5×10^4	0	-	0	-	0	-	0	-	0	-	**50**	**1444**
r=67.0		0	-	0	-	0	-	0	-	0	-	**18**	**7484**

Results for Uniform Random 7-SAT

In Table 11, we show our experimental results on the uniform medium and huge 7-SAT instances. Table 11

shows that *SelectNTS* is competitive with its competitors for these medium 5-SAT instances. Specifically, *SelectNTS* obtains the best performance for 2 medium instance classes. Score$_2$SAT reaches the best performance for 2 medium instance classes. SparrowToRiss gives the best performance for only one medium instance class. However, the three other algorithms ProbSAT, YalSAT, and CScoreSAT perform worse than *SelectNTS* on these uniform medium random 7-SAT instance classes. These results demonstrate that our *SelectNTS* algorithm is quite competitive for solving this medium random 7-SAT problems.

According to the experimental results in Table 11, we can find *SelectNTS* significantly outperforms its competitors. Compared to the competitors whose performance descends sharply as the instance ratio increases, *SelectNTS* shows good scalability. For example, for the huge 7-SAT instances with r=66, 67, all competitors fail in solving all instances, while *SelectNTS* solves 50 and 18 huge 7-SAT instances with r=66, 67 on average respectively, which confirms the good performance of *SelectNTS* on these huge 7-SAT instances.

Results for SAT Competition 2018

Table 12 shows the experimental results of our *SelectNTS* algorithm and SparrowToRiss, CScoreSAT, Score$_2$SAT, YalSAT, and ProbSAT on all HRS instances and uniform random k-SAT instances with k>3 from SAT Competition in 2018 [55]. *SelectNTS* presents the best performance for all random SAT instances except for the HRS instances with r=4.3, and especially it solves more uniform random k-SAT instances with k>3 than its competitors. For the HRS instances with r=4.3, *SelectNTS* solves the same number of instances as Score$_2$SAT and YalSAT, but PAR 2 is a little more than Score$_2$SAT and YalSAT's. Overall, *SelectNTS* solves 209 instances on average, and SparrowToRiss, Score$_2$SAT, ProbSAT, YalSAT, and CScoreSAT solve 192, 137, 117, 111, and 106 instances on average respectively, indicating that the *SelectNTS* algorithm achieves state-of-the-art performance on random SAT instances.

Summary for HRS and uniform random k-SAT with k>3

According to Tables 5-12, the experimental results present that *SelectNTS* consistently outperforms CScoreSAT, YalSAT, ProbSAT, Score$_2$SAT, and SparrowToRiss on solving HRS instances with various ratios and sizes except for the HRS instances with r=4.3, and shows the efficiency and the robustness on solving all HRS instances with up to 1000 variables. **The reason** why the existing state-of-the-art solvers cannot effectively solve HRS instances is that the HRS instances are generated based on the purpose of making incomplete solvers like ProbSAT and complete solvers like Glucose difficult to solve [3]. Thus, the existing SLS solvers are hard to solve HRS problems. However, *SelectNTS* adds a biased random walk strategy and a variation of CC strategy based on the original SLS framework, breaking the original SLS framework. And this experiment demonstrates that the superiority of *SelectNTS* becomes more significant over its competitors as the size of HRS instances increases.

Table 12 Computational results on the **SAT Competition 2018** benchmark.

Random SAT	Ratio	STR		CScoreSAT		Score$_2$SAT		YalSAT		PobSAT		*SelectNTS*	
		AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
HRS	4.3	55	0.052	55	0.009	55	**0.001**	55	**0.001**	55	0.013	55	0.032
	5.206	55	1.020	8	8591	33	4000	9	8387	12	7858	55	**0.017**
	5.5	55	136.4	11	8000	12	7818	12	7818	12	7818	55	**0.060**
Uniform	<21.117	5	7614	9	5706	11	4683	11	4540	11	4524	**13**	**3821**
	21.117	7	3111	8	2495	7	3015	8	2326	8	2409	**9**	**1741**
	<87.79	9	5583	10	5129	11	4720	9	5510	11	4522	**13**	**3748**
	87.79	8	2261	5	5224	8	2453	7	3880	8	2967	**9**	**1696**
Overall/225		194	1445	106	5362	137	3968	111	5130	117	4875	**209**	**825.6**

Table 13 Comparison among *SelectNTS* and its alternative degenerating versions on the six benchmarks. Each solver is performed 10 times on each instance with the cutoff time of 600 seconds, and the results in bold indicate the best performance for each class.

Benchmarks	*SelectNTS*		*SelectNTS_alt1*		*SelectNTS_alt2*		*SelectNTS_alt3*		*SelectNTS_alt4*		*SelectNTS_alt5*	
	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
SAT Competition 2017	120	**0.093**	49	710.0	120	0.117	107	130.1	119	10.14	49	710.0
HRS Random 5.206	1000	0.082	0	-	967	39.67	1000	**0.067**	1000	0.077	0	-
HRS Random 5.5	1000	**0.261**	80	1104	1000	0.246	1000	864.2	1000	0.348	160	1008
HRS Random 5.699	900	**0.108**	0	-	900	0.110	900	0.143	900	0.135	0	-
HRS Random 7.821	900	**1.156**	0	-	900	1.160	900	1.696	900	1.645	0	-
SAT Competition 2018	165	**0.036**	71	683.7	164	7.318	150	109.2	165	0.064	79	625.5

Moreover, *SelectNTS* is quite competitive for solving uniform random *k*-SAT with long clauses, i.e., the performance of *SelectNTS* in solving HRS instances is better than of that in solving uniform random *k*-SAT with $k>3$. Based on the case study in Section 5, we conjecture that for random SAT problems, if the maximum value of *cNTS* and *vNTS* on *SelectNTS* is less than or equal to **0.5** times as large as on ProbSAT within 10^5 steps respectively, *SelectNTS* is more effective to solve these problems than other random SAT instances which the maximum value of *cNTS* and *vNTS* on *SelectNTS* is more than **0.5** times as large as on ProbSAT within 10^5 steps respectively.

7. Discussions

Some further discussions are given below to clarify some issues and highlight some important cases.

7.1 Effectiveness of the *SelectNTS* components

In this section, we present a detailed discussion on each underlying component of the *SelectNTS* algorithm, namely biased random walk strategy, *cNTS*, *vNTS*, the variation of CC strategy, and the *score* property. According to the experimental results of Section 6.3, we can conclude that the performance of *SelectNTS* in solving HRS instances obviously exceeds that in solving uniform random *k*-SAT with $k>3$. Thus, we conducted extensive experiments for five alternative versions on all HRS benchmarks respectively. The computing environments in this section are the same as those utilized for experiments in Section 6.2.

Effectiveness of the clause weighting scheme *cNTS*

In order to demonstrate the effectiveness of the clause weighting scheme *cNTS* in the *SelectNTS* algorithm, we did experiments to compare *SelectNTS* with its an alternative version called *SelectNTS_alt1*, which does not utilize the *cNTS*, i.e., removing biased random walk based on the *cNTS* of lines 6-7 and removing update *cNTS* of line 9 in Algorithm 2. For the parameter setting of *r*, we utilize the default value of *SelectNTS*. Table 13 shows the comparative results on the six benchmarks.

The experimental results show that *SelectNTS* obviously outperforms *SelectNTS_alt1*. Specifically, *SelectNTS_alt1* fails to solve any instances with $r=5.206$, $r=5.699$, and $r=7.821$, which indicates the importance of the clause weighting scheme *cNTS*.

Effectiveness of the variable weighting scheme *vNTS*

By removing the variable weighting scheme *vNTS* in the *SelectNTS* algorithm, i.e., replacing *Sv* with the only *score* in line 12 in Algorithm 2, an alternative version called *SelectNTS_alt2* is obtained. For the parameter settings of β, it adopts the default value of *SelectNTS*. The experimental results are showed in Table 13.

The performance of *SelectNTS* significantly outperforms that of *SelectNTS_* alt6 on all HRS benchmarks in terms of metrics. Especially, *SelectNTS_alt2* succeeds in solving 33 instances on average for HRS Random 5.206 benchmark, whereas *SelectNTS_alt2* and *SelectNTS* succeed in solving 967 instances and 1000 instances on average for HRS Random 5.206 benchmark respectively, which indicates that the importance of the variable weighting scheme *vNTS*.

Effectiveness of *the variation of CC strategy*

In this subsection, we conducted additional experiments to analyze the effectiveness of the variation of *CC* strategy in the *SelectNTS* algorithm. We did not utilize the variation of *CC* strategy, i.e.,

removing lines 11-13 in Algorithm 2. An alternative version named *SelectNTS_alt3* is obtained and allows the same variable to be flipped in successive steps. For the parameter settings of β, it uses the default value of *SelectNTS*.

According to the results of Table 13, *SelectNTS* presents significantly better performance than *SelectNTS_alt3* on all HRS benchmarks except for HRS Random 5.206. Particularly, on the HRS from SAT Competition 2017, HRS Random 5.5, and HRS from SAT Competition 2018 benchmarks, the runtime of *SelectNTS* is about 1399 times, 3311 times, and 3033 times less than of *SelectNTS_alt3* respectively, which confirm the effectiveness of the variation of *CC* strategy as does in *SelectNTS* on solving HRS instances.

Effectiveness of the scoring function *score*

This alternative version of *SelectNTS* utilizes the variation of *CC* strategy, but the *Sv* function only uses the *vNTS* (i.e., removing the scoring function *score*, i.e., replacing the *Sv* function of line 12 in Algorithm 2, with the *vNTS*). Thus, this alternative version called *SelectNTS_alt4* is obtained. For the parameter setting, it uses the default value of *SelectNTS*.

From Table 13, it is clear that *SelectNTS* significantly performs better than *SelectNTS_alt4* on all HRS benchmarks except for HRS Random 5.206, which indicates that if the *score* property is not utilized in *SelectNTS*, the algorithm performs much worse than *SelectNTS*.

Effectiveness of the weighting schemes and the scoring function

This alternative version of *SelectNTS* does not use the *cNTS*, *vNTS*, and the variation of *CC* strategy, i.e., removing the biased random walk strategy and the *Sv* (i.e., removing lines 6-7, 9, and 11-14 in Algorithm 2, i.e., only using the probability and standard random walk). This alternative version named *SelectNTS_alt5* is obtained and has no need of parameter setting. Table 13 presents that *SelectNTS_alt5* fails to solve any instances with r=5.206, r=5.699, and r=7.821 even on the HRS Random 5.699 and HRS Random 7.821 benchmarks including instances with n=200. The *SelectNTS* obviously perform better than *SelectNTS_alt5*, which conform significance of *cNTS* and the variation of *CC* strategy.

7.2 Main differences between *SelectNTS* and *EPEFV*

EPEFV [59] is the first solver that can effectively solve both uniform random k-SAT and HRS. *EPEFV* also includes clause selection and variable selection. In this subsection, we discuss the main differences between *EPEFV* and *SelectNTS* for solving SAT instances. Before getting into the details of discussion, we first introduce two weighting schemes and a scoring function U_v in the *EPEFV* algorithm:

- **UnsatT weighting scheme** [59]**:** The weight of each clause is a positive integer, and under initial assignment a, if a clause c is unsatisfied, then the weight of c is initiated as 1; otherwise, the weight of c is initiated as 0. The clause weights are updated as follows. If a clause c is unsatisfied and contains the flipping variable under the current assignment, then c'weight is large than one; otherwise, c'weight remains unchanged.
- ***v*UnsatT weighting scheme** [59]**:** The weight of each variable is a positive integer, and under initial assignment a, if a variable v is in t unsatisfied clauses, then the weight of v is initiated as t; otherwise, the weight of v is initiated as 0. The variable weights are updated as follows: If a variable v is in m_v unsatisfied clause while containing the flipped variable under the current assignment, then v'weight is large than m_v; otherwise, c'weight remains unchanged.
- ***U*$_v$ Scoring function** [59]**:** $U_v(v,s)=score(v,s) + vUnsatT(v,s)/\gamma$ (γ is a positive integer parameter and s is the number of step).

In the following, we summarize these major differences between *EPEFV* and *SelectNTS* as follows:

- **Clause weighting scheme**: *SelectNTS* employs a new clause weighting scheme that only works on the clauses selected, while *EPEFV* works on the unsatisfied clauses containing the flipping variable.

- **Table 14** Comparison results of different solvers on HRS benchmark

Benchmark	Ratio	*EPEFV*		*EPEFV*_S		*SelectNTS*		*SelectNTS*_E	
		AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
SAT	4.3	40	**0.023**	40	0.068	40	0.134	40	0.566
Competition	5.206	40	0.039	34	1500	40	**0.032**	40	0.071
2017	5.5	40	0.662	17	5750	40	**0.113**	40	1.594
SAT	4.3	55	0.007	55	**0.005**	55	0.032	55	0.022
Competition	5.206	55	0.019	50	909.1	55	**0.017**	55	0.035
2018	5.5	55	0.156	23	5818	55	**0.060**	55	0.810
HRS Random	5.206	1000	**0.079**	720	2800	1000	0.082	1000	0.135
HRS Random	5.5	1000	1.298	280	9001	1000	**0.261**	1000	3.843
HRS Random	5.699	900	0.768	560	3778	900	**0.098**	900	1.193
HRS Random	7.821	260	7111	260	7111	900	**1.041**	900	1.257

- **Clause selection component**: The *SelectNTS* algorithm applies a biased random walk strategy based on the *cNTS*, while the *EPEFV* algorithm uses the biased random walk strategy depending on the *UnsatT* for selecting a clause.
- **Variable selection mechanism**: If the current variable selected based on probability is the same as the last flipped variable, the *SelectNTS* algorithm prefers to select a variable to be flipped depending on the U_v, while the *EPEFV* relies on the *vUnsatT*.
- **Empirical performance on random SAT benchmarks**: Overall, as can be seen clearly from the extensive experiments illustrated in Table 14 and Table 15, the *SelectNTS* algorithm generally performs much better than the *EPEFV* algorithm on a wide range of HRS benchmarks, and *SelectNTS* and *EPEFV* have their own advantages on the uniform random benchmarks, indicating that the significant performance improvements of *SelectNTS* over *EPEFV* are due to the above major differences between these two SLS algorithms.

In order to test the sensitivity of the algorithm under different parameter settings and make the comparison results more convincing, this subsection compares two alternative versions of the solvers with the *EPEFV* algorithm and the *SelectNTS* algorithm for solving all HRS benchmarks reported in Section 6.1. The two alternative versions are named *EPEFV*_S and *SelectNTS*_E respectively. *EPEFV*_S is still *EPEFV* algorithm, but the parameter settings of *EPEFV*_S and *SelectNTS* are exactly the same. Similarly, *SelectNTS*_E is still the *SelectNTS* algorithm, but the parameter settings of *SelectNTS*_E and *EPEFV* are exactly the same.

For all benchmarks, we run each solver 10 times for each instance. For performance metrics, we report the number of averages solved instances at ten-run "AverS" and "PAR-2". The best results for an instance class are highlighted in bold. If the performance of *EPEFV*_S is better than *EPEFV* in a certain benchmark, the result will be marked in red; if the performance of *SelectNTS*_E is better than *SelectNTS*, the result will be marked with underline.

It can be seen from Table 14 that the performance of *SelectNTS* outperforms than *EPEFV* in almost all HRS benchmarks. In the HRS instances with r=4.3 from the random track of SAT Competition in 2018, the performance of *EPEFV*_S slightly exceeds *EPEFV*, and the performance of *SelectNTS*_E is slightly better than that of *SelectNTS*. For the rest of HRS benchmarks, the performance of *EPEFV*_S is worse than that of *EPEFV*, but *SelectNTS*_E and *SelectNTS* have the similar performance. It indicates that under different parameter settings, the sensitivity of the *SelectNTS* algorithm is weaker than that of the *EPEFV* algorithm in the HRS benchmarks.

- **SAT Competition 2016**: the uniform random benchmark with k>3 from the random track of SAT Competition in 2016.

It can be seen from Table 15 that there are different algorithms with the best performance on different uniform random instances. Thus, each algorithm has its own advantages.

Table 15 Comparison results of different solvers on uniform random benchmark from the random track of SAT competitions in 2016, 2017 and 2018

Benchmark	Ratio	*EPEFV*		*EPEFV*_S		*SelectNTS*		*SelectNTS*_E	
		AverS	PAR-2	AverS	PAR-2	AverS	PAR-2	AverS	PAR-2
SAT Competition 2016	<21.117	13	3675	13	**3623**	13	3688	13	3691
	21.117	15	6445	12	7073	13	6801	16	6174
	<87.79	12	**4025**	12	4035	12	4084	12	4030
	87.79	18	5759	15	6405	15	6427	14	6743
SAT Competition 2017	<21.117	13	3667	13	**3646**	13	3761	13	3684
	21.117	**17**	**6074**	12	7011	15	6389	15	6389
	<87.79	12	**4015**	12	4067	12	4030	12	4045
	87.79	20	5415	15	6443	18	5817	18	5819
SAT Competition 2018	<21.117	13	**3673**	12	4122	13	3821	13	3701
	21.117	8	2462	8	2427	9	1741	9	**1571**
	<87.79	12	4021	12	4055	**13**	**3748**	12	4031
	87.79	7	3092	8	2619	**9**	**1696**	8	2362

Although *EPEFV*_S performs better than *EPEFV* on four classes of instances, and *SelectNTS*_E show rather better performance than *SelectNTS* on five classes of instances, *SelectNTS* and *SelectNTS*_E solve the same number of instances on average, while *EPEFV* solves 15 more instances than *EPEFV*_S. Thus, it indicates that under different parameter settings, the sensitivity of the *SelectNTS* algorithm is also weaker than that of the *EPEFV* algorithm in the uniform benchmarks.

For a SAT formula F, for the relationship between *UnsatT* and *cNTS*, we have the following conclusions.

Theorem 1. *For a CNF formula F, when algorithm runs to step s, for a clause c, if the cNTS weight of c is* $cNTS(c, s)$ *and the UnsatT weight of c is* $UnsatT(c, s)$ *, then* $UnsatT(c, s) \geq cNTS(c, s)$.

Proof. We will prove it by induction. Suppose that in a CNF formula F, suppose F includes m clauses.

When s=0, the *UnsatT* values of all unsatisfied clauses are set to 1, and the *UnsatT* values of all satisfied clauses are set to 0. However, the *cNTS* values of all clauses are set to 0. Thus,

$$UnsatT(c, 0) \geq cNTS(c, 0) \quad (1).$$

When s=1, we have to considee the following cases:

(**1**) suppose the first clause selected by the algorithm from F is c_t. Then

$$cNTS(c_t, 1) = cNTS(c_t, 0)+1 \quad (2).$$

Before the clause selection, c must be unsatisfied, and after the clause selection, c must be satisfied. Obviously,

$$UnsatT(c_t, 0) = UnsatT(c_t,1)=1 \quad (3),$$

and thus

$$cNTS(c_t,1) = UnsatT(c_t,1) \quad (4).$$

(**2**) Suppose the unsatisfied clause containing the flipped variable is c_j, whose value of *UnsatT* is large than one, i.e.,

$$UnsatT(c_j,1) = UnsatT(c_j, 0)+1 \quad (5),$$

while $cNTS(c_j,1) = cNTS(c_j,0)=0$, and thus

$$UnsatT(c_j, 1) > cNTS(c_j, 1) \quad (6).$$

(**3**) For the rest of the clauses c_i ($i \neq t, j$ and i=0, 1, …, m-1), the values of *UnsatT* and *cNTS* do not change, and obviously, $UnsatT(c_i,1) > cNTS(c_i,1)$. Based on the conditions (1), (2) and (3), we can get that

$$UnsatT(c,1) \geq cNTS(c,1) \quad (7).$$

When $s=k$. Suppose

$$UnsatT(c_i, k) > cNTS(c_i,k) (i=0,1,\cdots,m-1) \quad (8),$$

holds. Then we can obtain

$$UnsatT(c, k) \geq cNTS(c, k) \quad (9).$$

When $s=k+1$. Suppose the clause selected by the algorithm from F is c_p. Then

$UnsatT(c_p,k+1)= cNTS(c_p,k+1)<UnsatT(c_p,k)+1$ (10).

We have to consider the following two cases:

(**1**) If $cNTS(c_p, k)< UnsatT(c_p, k)$, since *UnsatT* and *cNTS* is a nonnegative integer, then

$$cNTS(c_p, k+1)= cNTS(c_p, k)+1\leq UnsatT(c_p, k) \leq UnsatT(c_p, k+1) \quad (11).$$

(**2**) If $cNTS(c_p, k)= UnsatT(c_p, k)$, suppose that the algorithm runs from step q to step k, $cNTS(c_p, q)= cNTS(c_p, q+1)=\ldots= cNTS(c_p, k)$, and suppose that algorithm runs from step w to step k, $UnsatT(c_p, w)= UnsatT(c_p, w+1)\ldots= UnsatT(c_p, k)$. Since $UnsatT(c_i, k) \geq cNTS(c_i, k)$ (i=0, 1, …, m-1), thus $w\leq q$. Obviously, the algorithm runs from step q+1 to Step k, clause c_p is not selected once, and in Step q, c_p must be satisfied after the variable is flipped. Since clause c_p is selected in Step k+1, there is at least some step between Step q and Step k+1, and in some step, the state of c_p changes from satisfied to unsatisfied after the variable is flipped. Suppose that in step h, the state of c_p changes from satisfied to unsatisfied after the variable is flipped, then c_p includes the flipped variable. Obviously,

$$UnsatT(c_p, h)= UnsatT(c_p, h-1)+1 \quad (12),$$

and $w\leq q< q+1< h\leq k$, and we can get

$$UnsatT(c_p, h) \geq cNTS(c_p, h-1)+1 = cNTS(c_p, h)+1 \quad (13),$$

and thus we can obtain the result

$$UnsatT(c_p, h)> cNTS(c_p, h)+1 \quad (14),$$

but the result and $cNTS(c_p,q)= UnsatT(c_p,q)= cNTS(c_p,q+1)= UnsatT(c_p,q+1)=\ldots= cNTS(c_p,k)= UnsatT(c_p,k)$ are contradictory, and obviously, assuming $cNTS(c_p,k)= UnsatT(c_p,k)$ does not hold. If the clause selected is c_p in Step k+1, then $cNTS(c_p,k)<UnsatT(c_p,k)$, and otherwise, for the rest of clauses c_g (g=0,1, …, m-1 and $g\neq p$), we can obtain $cNTS(c_g,k+1)= cNTS(c_g,k) \leq UnsatT(c_g,k) \leq UnsatT(c_g,k+1)$. Based on the cases (1) and (2), we can get that

$$UnsatT(c, k+1)\geq cNTS(c, k) \quad (15).$$

In summary, for any step s and clause c,

$$UnsatT(c, s)\geq cNTS(c, s) \quad (16).$$

Suppose under the same parameter β, a clause c is called HSC-UT if the c'weight is larger than β in *EPEFV* algorithm [59], and a clause c is called HSC if the c'weight is larger than β in *SelectNTS* algorithm.

For the relationship between HSC-UT and HSC in F, we have the following conclusions.

Lemma 1. *For a CNF formula F, given a clause c, when algorithm runs to step s, if c is HSC, then c is an HSC-UT.*

Proof. For a clause c, if c is HSC, then $cNTS(c, s)> \beta$. Based on the **Theorem 1**, we can obtain $UnsatT(c, s)\geq cNTS(c, s)$. Obviously, $UnsatT(c, s)> \beta$, and thus c is HSC-UT.

Note 1. The inverse of **Lemma 1** does not necessarily hold. For example, if a clause c is HSC-UT, then $UnsatT(c, s)> \beta$. Based on **Theorem 1**, we can obtain $UnsatT(c, s)\geq cNTS(c, s)$. If $UnsatT(c, s)= cNTS(c, s)$, then $cNTS(c, s)> \beta$, and thus c is HSC. If $UnsatT(c, s)>cNTS(c, s)> \beta$, then c is HSC. However, if $UnsatT(c, s)> \beta>cNTS(c, s)$, then c is not HSC.

We summarize that based on the **Theorem 1**, **Lemma 1** and **Note 1**, under the same parameter setting β, the set of HSC is also a subset of HSC-UT, indicating that if a clause c is taboo by the biased random walk strategy based on the *UnsatT*, then c must be taboo by the biased random walk strategy based on *cNTS*. However, the inverse is not necessarily correct. Thus, the biased random walk strategy based on *cNTS* is stronger than the clause selection of the biased random walk strategy based on *UnsatT*.

According to Table 13 and the results of *EPEFV* algorithm in the literature [59], the main role in the *SelectNTS* algorithm and the *EPEFV* algorithm is clause selection strategy based on the biased random walk. From the Table 14, we can see the performance of *SelectNTS* is better than that of *EPEFV* algorithm for solving HRS benchmarks. We conjecture that the stronger the taboo of clause selection is, the more refined the candidate clauses

are, and the better the algorithm is for solving HRS problem.

For a HRS formula F, for the relationship between *vUnsatT* and *vNTS*, we have the following conclusions.

Theorem 2. *For a CNF formula F, when algorithm runs to step s, for a variable v, if the vNTS weight of v is vNTS(v, s) and the vUnsatT weight of v is vUnsatT(v, s), then vUnsatT(v, s)≥ vNTS(v, s).*

Proof. According to the update methods of *vUnsatT* and *vNTS*, on the one hand, the *vNTS* values of all variables are initialized as 0, while if there is a variable v which appears in the unsatisfied clauses, then the *vUnsatT* value of v is initialized as greater than 0, thus

$$vUnsatT(v,0)\geq vNTS(v, 0) \quad (17).$$

On the other hand, each variable appears at least once in F, and suppose the variable v appears in t clauses denoted as $g_1, g_2, \ldots, g_t$. Since when a clause with v is selected in step s, v is not necessarily selected, and thus

$$vNTS(v, s) \leq cNTS(g_1, s)+ cNTS(g_2, s)+\ldots + cNTS(g_t, s) \quad (18).$$

Based on **Theorem 1** above and Theorem 1 in the literature [59], we can obtain

$$cNTS(g_1, s)+ cNTS(g_2, s)+\ldots+ cNTS(g_t, s) \leq UnsatT(g_1, s)+ UnsatT(g_2, s)+\ldots+ UnsatT(g_t, s) = vUnsatT(v, s) \quad (19).$$

Thus,

$$vUnsatT(v, s) \geq vNTS(v, s) \quad (20).$$

If the variable selected based on probability in step s is the same as the flipped variable in step s-1, the *EPEFV* algorithm and *SelectNTS* algorithm prefer to select a variable to be flipped by preferring the variable with the greatest scoring function. When the algorithm runs to Step s, the scoring function U_v is $score(v, s) + vUnsatT(v, s)/\gamma$ in *EPEFV*, and the scoring function S_v is $score(v, s) + vNTS(v, s)/\gamma$ in *SelectNTS*.

Suppose under the same parameter γ, for the relationship between U_v and S_v, we have the following conclusion.

Lemma 2. *For a CNF formula F, given a variable v in a clause c, when algorithm runs to step s, if the variable with the greatest value of U_v is v among all variables in c, then the variable with the greatest value of S_v is not necessarily v in c.*

Proof. In the clause c, suppose there are n variables denoted as $v_1, v_2, \ldots, v_n$. Without loss of generality, if the variable with the greatest value of U_v is v_1 and v_2 is the same as the last flipped variable, then we can get that $vUnsatT(v_1, s) \geq \{ vUnsatT(v_3, s), vUnsatT(v_4, s), \ldots, vUnsatT(v_n, s)\}$. Based on **Theorem 2**, we can obtain that $vUnsatT(v_1, s) \geq vNTS(v_1, s)$, $vUnsatT(v_3, s) \geq vNTS(v_3, s)$, …, $vUnsatT(v_n, s) \geq vNTS(v_n, s)$. However, the relationship of $vNTS(v_1, s)$, $vNTS(v_3, s)$, …, $vNTS(v_n, s)$ is uncertain. Thus, the variable with the greatest value of S_v is not necessarily v in c.

Lemma 3. *For a CNF formula F, given a variable v in a clause c, when algorithm runs to step s, if the variable with the greatest value of S_v is v among all variables in c, then the variable with the greatest value of U_v is not necessarily v in c.*

Proof. In the clause c, suppose there are n variables denoted as $v_1, v_2, \ldots, v_n$. Without loss of generality, if the variable with the greatest value of S_v is v_1 and v_2 is the same as the last flipped variable, then we can get that $vNTS(v_1, s) \geq \{ vNTS(v_3, s), vNTS(v_4, s), \ldots, vNTS(v_n, s)\}$. Based on **Theorem 2**, we can obtain that $vUnsatT(v_1, s) \geq vNTS(v_1, s)$, $vUnsatT(v_3, s) \geq vNTS(v_3, s)$, …, $vUnsatT(v_n, s) \geq vNTS(v_n, s)$. However, the relationship of $vUnsatT(v_1, s)$, $vUnsatT(v_3, s)$, …, $vUnsatT(v_n, s)$ is uncertain. Thus, the variable with the greatest value of U_v is not necessarily v in c.

Based on **Lemma 2** and **Lemma 3**, we can summarize that in a clause c, although the U_v value of each variable is greater than or equal to its S_v value, if a variable is the greatest variable based on U_v, it is not necessarily the greatest variable based on S_v, and if a variable is the

Table 16 The main differences between *SelectNTS* and the FRW algorithms

Solvers	Clause selection strategy	Variable selection strategy	Whether to use weighting scheme
SelectNTS	Biased random walk	Probability selection strategy and the variation of CC strategy	Yes
FRW algorithms	Standard random walk	Probability selection strategy like ProbSAT or variable property like WalkSAT	No

greatest variable based on S_v, it is not necessarily the greatest variable based on U_v. Thus, the scoring functions based on U_v and S_v belong to two different strategies without any inclusion relationship.

7.3 Approximate implementation of SelectNTS

Before introducing the approximate implementation of *SelectNTS*, we describe the main differences between *SelectNTS* and the FRW algorithms. The *SelectNTS* belongs to the FRW algorithm. Although the *SelectNTS* is conceptually related to the ProbSAT based on the probability selection strategy, there exist major differences between *SelectNTS* and FRW algorithms. We summarize these major differences in Table 16.

Compared with ProbSAT, the biased random walk strategy, the *cNTS* updating procedure, the variation of *CC* strategy, and the *vNTS* updating procedure are the only steps in *SelectNTS*.

In this discussion, we assume that a random k-SAT instance F includes n variables and m clauses (r=m/n). Thus, each clause c contains k variables, i.e., $E(|c|) = k$. We use $F(s)$ to denote the number of unsatisfied clauses in step s, thus $E(|F(s)|) < m$.

(1) The biased random walk strategy is the clause selection heuristic. The maintenance of accurate implementation of the biased random walk strategy is described as follows. If there exists HSC, a clause is randomly picked from HSC; and otherwise, a clause is randomly selected from the unsatisfied clauses. HSC is updated by the unsatisfied clauses at each step. Therefore, the complexity of ProbSAT adding the HSC updating procedure in each step is equal to that on ProbSAT. For the accurate implementation of the biased random walk strategy, the worst-case time complexity of picking an unsatisfied clause at step s is $O(E(|F(s)|))$. Therefore, the complexity of ProbSAT using the biased random strategy to replace the standard random walk in each step is less than and equal to that on ProbSAT.

(2) Whenever a clause is selected in each step, the *cNTS* is updated. The complexity of the *cNTS* updating procedure in each step is $O(1)$. Therefore, the complexity of ProbSAT adding the *cNTS* updating procedure in each step is equal to that on ProbSAT.

(3) The variation of *CC* strategy is the variable selection heuristic, but it is not executed at each step. When the variable selected by the probability selection method at Step s is equal to the flipping variable at Step s-1, the variation of *CC* strategy is executed. The complexity of the variation of *CC* strategy in each step is $O(k)$. Therefore, the complexity of ProbSAT adding the variation of *CC* strategy in each step is equal to that on ProbSAT.

(4) Whenever a variable is selected in each step, the *vNTS* is updated. The complexity of the *vNTS* updating procedure in each step is $O(1)$. Therefore, the complexity of ProbSAT adding the *vNTS* updating procedure in each step is equal to that on ProbSAT.

The literature [58] has shown that all the time complexities of ProbSAT in each step are about $O(k*r)$. According to the above (1)~(4), the worst-case time complexity of *SelectNTS* is equal to that of ProbSAT. Thus, all the time complexities of the approximate implementation of *SelectNTS* in each step are about $O(k*r)$.

The existing probability selection strategy is ineffective for solving HRS problems, while the *cNTS* property and the variation of *CC* strategy based on the *vNTS* property show effectiveness when applying to probability selection strategy, and the related empirical analyzes have been shown in Sections 7.1. The possible reason is that the *cNTS* property and the variation of *CC* strategy help the algorithms based on the probability

selection strategy to deal with local search and thus lead algorithms to the appropriate search space.

8. Conclusions and Future Works

In this work, we presented an enhanced probability selection based SLS algorithm which could work effectively for both the well-known HRS problem and uniform random k-SAT problem with k>3. This work has opened up a new direction for effective SLS algorithms. The first enhancement improved the probability selection strategy of the original ProbSAT [6] by using a new and global clause weighting scheme called *cNTS* to distinguish unsatisfied clauses and adopting the biased random walk to prefer satisfying several unsatisfied clauses hard to keep satisfied. The second enhancement concerns the variation of CC strategy, which is more powerful than the algorithm based on the probability selection method, and utilized a new and global variable weighting scheme called *vNTS* to distinguish variables and then proposes a linear function named S_v which combined *vNTS* and *Score*, to avoid selecting the same variable in consecutive steps. As the variation of CC strategy, S_v is simple compared to the CC strategy.

The enhanced probability selection based local search method is called *SelectNTS*, whose effectiveness has been demonstrated on random SAT problems from the SAT Competitions in 2017 and 2018, and on randomly generated HRS and uniform k-SAT with k>3 problems. The results have shown that *SelectNTS* outperformed the state-of-the-art SLS solvers and the best hybrid solver in most cases. Moreover, *SelectNTS* can effectively be applied to solve both uniform random k-SAT problems and HRS problems. The SAT instances encoded from real-world applications may be of large size. As our *SelectNTS* algorithm is able to solve large HRS instances quickly with up to 1000 variables within five seconds, therefor it may be beneficial to solve cryptography instances, and thus we believe the experimental results of *SelectNTS* on HRS instances and URS instances may provide support for solving problems from the application domain.

As future work, based on the *cNTS* property and *vNTS* property, we strive to develop more properties, which are to improve SLS algorithms for structured problems, constrained satisfaction problems, and graph search problems, by using the new heuristics.

Acknowledgments

This work is partially supported by National Natural Science Foundation of China (Grant No: 62106206), and Sichuan Science and Technology Program (Grant No. 2020YJ0270), and the Fundamental Research Funds for the Central Universities (Grant No. 2682017ZT12, 2682016CX119, 2682019ZT16, 2682020CX59).

References

[1] Achlioptas, D. Random satisfiability. In Handbook of Satisfiability, 2009, pp. 245–270.

[2] Balint, A., and Fröhlich, A. Improving stochastic local search for SAT with a new probability distribution. In Proc. of SAT 2010, pp,10–15.

[3] Balyo, T. Using algorithm configuration tools to generate hard random satisfiable benchmarks. In Proc. of SAT 2016: Solver and Benchmark Descriptions, pp, 60–62, https://helda.helsinki.fi/bitstream/handle/10138/164630/sc2016_ proceedings.pdf?sequence=1&isAllowed=y

[4] Balyo, T., & Chrpa, L. Using Algorithm Configuration Tools to Generate Hard SAT Benchmarks. In Proc. of SoCS 2018, pp.133-137.

[5] Balint, A. and Manthey, N. SparrowToRiss. In Proc. of SAT 2018: Solver and Benchmark Descriptions, pp. 38-39, 2018.

[6] Balint, A. and Schöning, U. (2012). Choosing probability distributions for stochastic local search and the role of make versus break. In Pro. of SAT-2012, pp. 16–29.

[7] Balint, A. and Schöning, U. 2018. ProbSAT. In Proc. of SAT 2018: Solver and Benchmark Descriptions, pp, 35.

[8] Biere A. Cadical, Lingeling, Llingeling, Treengeling and Yalsat entering the SAT Competition 2017. In Proc. of SAT 2017: Solver and Benchmark Descriptions, pp, 14-15.

[9] Bright, C., Ilias, K. and G. Vijay. Applying computer algebra systems with SAT solvers to the Williamson conjecture. Journal of Symbolic Computation, 100(2020) 187-209.

[10] Cai, S. and Luo, C. Score$_2$SAT. In Proc. of SAT 2017: Solver and Benchmark Descriptions, pp, 34.

[11] Cai, S., Luo, C., Lin, J., & Su, K. New local search methods for partial MaxSAT. Artificial Intelligence, 240 (2016)1-18.

[12] Cai, S., Luo, C., & Su, K. Improving walksat by effective tie-breaking and efficient implementation. Computer Journal, 2014, 58(11)2864-2875.

[13] Cai, S., & Su, K. Comprehensive score: Towards efficient local search for SAT with long clauses. In Proc. of IJCAI 2013, pp, 489-495, 2013.

[14] Cai, S., and Su, K. Local search for Boolean satisfiability with configuration checking and subscore. Artificial Intelligence, 204(2013)75–98.
[15] Cai, S., & Su, K. (2013c). CCAnr. In Pro. of SAT-2013: Solver and Benchmark Descriptions, pp. 16–17, https://helda.helsinki.fi/bitstream/handle/10138/40026/sc2013_proceedings.pdf?sequence=2&isAllowed=y
[16] Coelho, J. & Vanhoucke, M.. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. European Journal of Operational Research, 2011, 213(1), 73-82.
[17] Deshpande, A. and Layek, R. K.. Fault detection and therapeutic intervention in gene regulatory networks using SAT solvers. BioSystems, 179(2019)55-62.
[18] Duong, T. T. N., Pham, D. N., Sattar, A., & Newton, M. H. (2013). Weight-enhanced diversification in stochastic local search for satisfiability. In Proc. of IJCAI 2013, pp.524-530.
[19] Braunstein, A., Mézard, M. & Zecchina, R.. Survey propagation: an algorithm for satisfiability. Random Struct. Algorithms, 2005, 27(2) 201–226.
[20] S. Liu and A. Papakonstantinou. Local search for hard sat formulas: the strength of the polynomial law. In Proc. of AAAI 2016, pp. 732-738
[21] Hoos, H.H. An adaptive noise mechanism for WalkSAT. In Proc. of AAAI 2002, pp, 655–660.
[22] Hutter, F., Hoos, H.H. and Leyton-Brown, K. Sequential model based optimization for general algorithm configuration. In Proc. of the LION 2011, pp, 507–523.
[23] Hutter, F., Tompkins, D. A., & Hoos, H. H. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In Proc. of CP 2002, pp, 233-248.
[24] KhudaBukhsh, A.R., Xu, L., Hoos, H.H., and Leyton-Brown, K. SATenstein: Automatically building local search SAT solvers from components. Artificial Intelligence, 232(2016)20–42.
[25] Kochemazov, S., Zaikin, O., Kondratiev, V. and Semenov, A. MapleLCMDistChronoBT-DL, duplicate learnts heuristic -aided solvers at the SAT Race 2019. In Proc. of SAT 2019: Solver and Benchmark Descriptions, pp, 24, https://helda.helsinki.fi/bitstream/handle/10138/306988/sr2019_proceedings.pdf?sequence=1&isAllowed=y
[26] König, B., Maxime, N. and N. Dennis. CoReS: A Tool for Computing Core Graphs via SAT/SMT Solvers. In: Proc. of Graph Transformation,2018, 37-42.
[27] Kroc, L., Sabharwal, A., & Selman, B.. An empirical study of optimal noise and runtime distributions in local search. In Proc. of SAT-2010, pp. 346–351.
[28] Li, C. & Li Y.. Satisfying versus falsifying in local search for satisfiability - (poster presentation). In Proc. of SAT-2012, pp. 477–478.
[29] Li, C. & Huang, W.. Diversification and determinism in local search for satisfiability. In Pro. of SAT-2005, pp. 158–172.
[30] Liang, J.H., Ganesh, V., Poupart, P. and Czarnecki, K. An empirical study of branching heuristics through the lens of global learning rate. In Proc. of SAT 2010, pp, 119-135,.
[31] Luo, C., Cai, S., Wu, W., and Su, K. Double configuration checking in stochastic local search for satisfiability. In Pro. of AAAI 2014, pp 2703-2709.
[32] Luo, C., Su, K., and Cai, S. More efficient two-mode stochastic local search for random 3-satisfiability. Applied intelligence, 2014, 41(3) 665-680.
[33] Luo, C., Cai, S., Su, K. and Wu, W. Clause states based configuration checking in local search for satisfiability. IEEE transactions on Cybernetics, 2015, 45 (5) 1028-1041.
[34] Luo, C., Cai, S., Wu, W., and Su, K. CSCCSat. In Proc. of SAT 2016: Solver and Benchmark Descriptions, pp, 10, https://helda.helsinki.fi/bitstream/handle/10138/164630/sc2016_ proceedings.pdf?sequence=1&isAllowed=y
[35] Luo, C., Cai, S., Su, K. and Huang, W. CCEHC: An Efficient Local Search Algorithm for Weighted Partial Maximum Satisfiability (Extended Abstract). In Proc. of IJCAI 2017, pp, 5030 – 5034.
[36] Marques-Silva, JP. & Sakallah KA. (1999). Grasp: A search algorithm for propositional satisfiability. IEEE Trans. Comput. 48(5), pp. 506–521.
[37] Moskewicz, M., Madigan, C., *et al*. (2001). Chaff: Engineering an Efficient SAT Solver, In Proc. of Design Automation Conference, pp. 530-535.
[38] Ouimet, M., and Lundqvist, K.. Automated verification of completeness and consistency of abstract state machine specifications using a sat solver. Electronic Notes in Theoretical Computer Science, 2007, 190(2), 85-97.
[39] Ryvchin, V. and Nadel, A. Ma-ple_LCM_Dist_ ChronoBT. In Proc. of SAT 2018: Solver and Benchmark Descriptions, pp, 29.
[40] Selman, B., Kautz, H. A., and Cohen, B. Noise strategies for improving local search. In Proc. of AAAI 1994, pp, 337–343.
[41] Thornton, J. Clause weighting local search for SAT. Journal of Automated Reasoning, 2015, 35(1-3)97-142.
[42] Ulyantsev, V. andTsarev, F.. Extended finite-state machine induction using SAT-solver. IFAC Proceedings Volumes, 2012, 45(6), 236-241.
[43] Wu, Z., & Wah, B. W. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In Proc. of AAAI/IAAI, 2000, pp, 310-315.
[44] Yin L, He F, Hung WNN, Song X, Gu M (2012) Maxterm covering for satisfiability. IEEE Trans Comput 61(3):420–426.
[45] Zha, A. GluHack. In Proc. of SAT 2018: Solver and Benchmark Descriptions, pp, 26.
[46] Zhao, X., Zhang, L., Ouyang, D. and Jiao, Y.. Deriving all minimal consistency-based diagnosis sets using SAT solvers. Progress in Natural Science, 2009, 19(4), 489-494.
[47] Heule, M. J., Generating the uniform random benchmarks," in Proc. SAT competition 2018, pp. 80.
[48] Mavrovouniotis, M., Müller, F. M. and Yang, S.. Ant colony optimization with local search for dynamic

traveling salesman problems, IEEE Trans. Cybern, 2017, 47 (7)1743-1756.

[49] Mazure, B. , Saïs, L. and Grégoire, E.. Tabu search for SAT, In Proc. of the AAAI- 97, 1997, pp. 281–285.

[50] Luo, C., Su, K. and Cai, S., Improving local search for random 3-SAT using quantitative configuration checking, in Proc. of ECAI 2012, 2012, pp. 570–575

[51] Selman, B., Mitchell, D. and Levesque, H.. A new method for solving hard satisfiability problems, In Proc. of the AAAI-92, 1992, pp. 440–446.

[52] Liu, S. and Papakonstantinou, A.. Local search for hard sat formulas: the strength of the polynomial law, In Proc. of the AAAI-2016, 2016, pp. 732-738.

[53] Luo, C., Cai, S., Wu, W., & Su, K.. Focused random walk with configuration checking and break minimum for satisfiability. In Proc. of CP-2013, pp. 481–496.

[54] SAT Competition 2017. https://baldur.iti.kit.edu/sat-competition-2017/.

[55] SAT Competition 2018. http://sat2018.forsyte. tuwien. ac.at

[56] k-SAT generator. https://sourceforge.net/projects /ksat generator/

[57] SAT Competition 2013. http://satcompetition.org/edacc/SATCompetition2013/experiment/23/solver-configurations/854.

[58] Fu, H., Liu J., Xu, Y.. Focused Random Walk with Probability Distribution for SAT with Long Clauses. Applied Intelligence, 2020, 50(12): 4732-4753.

[59] Fu, H. Xu, Y., Wu, G., Liu J., Chen, S., He, X.. Emphasis on the Flipping Variable: Towards Effective Local Search for Hard Random Satisfiability, Information Sciences, 2021, 566 (2021) : 118-139.