Improving stochastic local search for uniform k-SAT by generating appropriate initial assignment

Fu, H., Zhang, W., Wu, G., Xu, Y., & Liu, J. (2021). Improving stochastic local search for uniform k-SAT by generating appropriate initial assignment. *Computational Intelligence*, *37*(4), 1-39. Advance online publication. https://doi.org/10.1111/coin.12438

Link to publication record in Ulster University Research Portal

Published in:
Computational Intelligence

Publication Status:
Published online: 10/08/2021

DOI:
10.1111/coin.12438

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: 17/04/2024

Improving Stochastic Local Search for Uniform k-SAT by Generating Appropriate Initial Assignment

Huimin Fu, Wuyang Zhang, Guanfeng Wu*, Yang Xu and Jun Liu

Abstract

Stochastic Local Search (SLS) algorithms are well known for their ability to efficiently find models of random instances of the SAT problem, especially for uniform random *k*-SAT instances. Two processes affect most SLS solvers—the initial assignment of the variables and the heuristics that select which variable to flip. In the last few years, the work on generating the appropriate initial assignment has not been paid much attention or seen much progress, while most SLS solvers focused on the heuristic algorithm. The present work aims to improve SLS algorithms on uniform random *k*-SAT instances by developing effective methods for generating the initial assignment of variables in a controlled way. Firstly, the allocation strategy introduced recently for 3-SAT instances is extended to initialize the initial assignment on random *k*-SAT instances. Then a concept of an initial probability distribution of the clause-to-variable ratio of the instance is introduced to determine the parameters of the allocation strategy. This combined method is added to the beginning of six state-of-the-art SLS algorithms in order to generate initial assignments of variables in a controlled way instead of generating them randomly, resulting in six extended SLS algorithms named WalkSATlm_E, DCCASat_E, Score$_2$SAT_E, CSCCSat_E, Probsat_E, and Sparrow_E, respectively. They are then evaluated in terms of their capabilities and efficiency on uniform random *k*-SAT instance from the random track of SAT Competitions in 2016, 2017, and 2018. Experimental results show that these improved SLS solvers outperform their original performance, especially WalkSAT_E, Score$_2$SAT_E and CSCCSat_E outperform the winner of the random track of SAT competition in 2017. In addition, based on the initial probability distribution method, the present work proposes a parameter tuning and analysis of random 3-SAT instances and provides an additional comparative analysis with the state-of-the-art random SLS solvers based on large-scale experiments.

Keywords: Probability Distribution · Satisfiability (SAT) · Focused random walk (FRW) · Stochastic local search (SLS)

1 Introduction

The propositional satisfiability (SAT) problem is one of the most widely studied NP-complete problems and has been extensively studied in many domains of computer science and artificial intelligence due to its significant importance in both theory and applications[25,30]. Considering a propositional formula *F* in the Conjunctive Normal Form (CNF) defined on a set of Boolean variables, the SAT problem asks whether there exists a truth assignment to the variables of *F* that satisfies all clauses in *F*.

There are many optimization algorithms for solving SAT problems, which are divided into two main categories: one is complete, the other is incomplete. Although the incomplete SAT solvers cannot guarantee either to find the solutions or prove a given boolean formula unsatisfiable, some of them are surprisingly more effective than complete solvers on finding models of satisfiable formulae for random SAT instances[29]. The incomplete SAT solvers are mainly based on the Conflict Driven Clause Learning (CDCL) strategies[32,39], and the incomplete SAT solvers are mainly based on Stochastic Local Search (SLS) strategies, which are among the best-known methods for solving SAT instances[13,52].

An SLS algorithm starts a truth assignment of the variables of *F* by generating randomly. Then it explores the search space to minimize the number of falsified clauses. To do this, it iteratively flips the truth value of a variable selected according to some heuristic at each step[18-20,28]. An SLS algorithm for SAT switches between two different modes, which are greedy (intensification) mode and random (diversification) mode. The former aims at increasing the number of satisfied clauses, while the latter tends to explore the search space well and prevent the search from being stuck in the local minima.

* The corresponding author: System Credibility Automatic Verification Engineering Lab of Sichuan Province, Southwest Jiaotong University, Chengdu, 610031, China. E-mail: wgf1024@swjtu.edu.cn (F. Wu)

Random SAT instances are generated using random models, including uniform random *k*-SAT[2] or hard random SAT[9]. Hard random SAT instances exhibit some particularly statistical properties and are hard to solve by SLS algorithms, except for the complete algorithms like Sparrow2riss[8]. Uniform *k*-SAT instances remain very difficult for all algorithms, including SLS algorithms and CDCL algorithms. As reported in the literature[34], the performance of the SLS algorithms are often evaluated on uniform random *k*-SAT benchmarks. These benchmarks have a large variety of instances to test the robustness of algorithms, and by controlling the instance size and the clause-to-variable ratio, they provide adjustable hardness levels to assess the solving capabilities. Moreover, the performance of the algorithm is usually stable on uniform random *k*-SAT instances, either good or bad. Thus, we can easily recognize good heuristics by testing SLS algorithms on uniform random *k*-SAT instances, and these heuristics may be beneficial for solving application problems. Furthermore, uniform random *k*-SAT instances are widely used as benchmarks in the international SAT competitions, and uniform random *k*-SAT at the phase transition has been cited as one of the hardest track of SAT problems[37].

Although SLS is well known as the most effective approach for solving uniform random *k*-SAT instances, the performance of SLS algorithms on such instances has stagnated for a long time. Indeed, such instances remain challenging for all kinds of algorithms. The famous SLS algorithm WalkSAT has shown that mixed random walk is the superior strategy[45], and can solve random 3-SAT instances near the phase transition with one million variables[31]. The states-of-the-art in this direction SLS algorithms include Sparrow[3], Probsat[4,5], CCASat[13], FrwCB [34], WalkSATlm [21], respectively. They have shown good performance on random instances with long clauses ($k>3$). A few further progresses such as DCCASat[37], CSCCSat[35], dimetheus[20], Score$_2$SAT[22], and yalsat[10], have also been made in this direction. Sparrow and Probsat won the random track of SAT Competitions in 2011 and 2013, respectively. For random instances at the phase transition[1], DCCASat and WalkSATlm are good at solving random 5-SAT and 7-SAT instances; and for random instances near the phase transition, Sparrow and Probsat are good at solving random 5-SAT and 7-SAT instances. CCASat, CSCCSat, Score$_2$SAT, yalsat, and dimetheus have shown good performance on all uniform random *k*-SAT instances for $k\in\{3, 5, 7\}$. Especially, CCASat, dimetheus and yalsat won the random track of SAT Competitions in 2012, 2016, and 2017 respectively, and CSCCSat won the silver of the random track of SAT Competition in 2016. Score$_2$SAT won the bronze of the random track of SAT Competition in 2017. Note that CSCCSat shows better performance than yalsat on uniform random *k*-SAT instances at the threshold ratio of phase transition. WalkSATlm shows the best performance among these solves on random 3-SAT instances at the threshold ratio of phase transition.

Although current SLS solvers can solve effectively uniform random *k*-SAT instances, observing the champion solvers of the random track of SAT Competition in the last few years, we find that the best SLS algorithms have not solved nearly half of all uniform random *k*-SAT instances of SAT Competition. Therefore, there are still several spaces for further improvement in SLS solvers.

Note that two strategies that potentially affect the efficiency of SLS solvers are the initial assignment of the variables and the heuristics that select and flip a variable according to a variable selection heuristic iteratively until it seeks out a solution or timeout. In the last few years, the work on generating the appropriate initial assignment has not been paid much attention or no significant breaking method, and most SLS solvers focused on the heuristic algorithms. The present work aims to improve SLS algorithms on uniform random *k*-SAT instances by developing effective and efficient methods for generating the initial assignment of variables.

One of the concepts introduced by the same author team[26] places a good foundation on the present work, called the allocation strategy, which is based on the ratio of the number of occurrences of positive and negative literals for each variable in each instance and is utilized to generate the initial assignment on random 3-SAT instances. It is shown that the allocation strategy based on the initial information of the inherent properties of each instance has an obvious effect in improving the GSAT algorithm for random 3-SAT problems[46]. A natural question arises: *Whether the allocation strategy can also be used to improve the performance of SLS algorithms* for solving uniform random *k*-SAT problems?

The present work provides a positive answer to this question by extending the allocation strategy from uniform random 3-SAT to uniform random *k*-SAT instances. More specifically, a concept of initial probability distribution P(r) (r is the clause-to-variable ratio of the instance) is proposed to effectively determine the parameter setting for the

[1] The clause-to-variable ratio for which 50% of the uniform random formulas are satisfiable. For most algorithms, the closer a formula is generated near the threshold ratio, the harder it is to solve it

allocation strategy. The initial probability distribution is combined with the allocation strategy, which is added to the beginning of each algorithm to generate the initial assignment in a controlled way instead of generating an assignment randomly. This has resulted in six extended SLS algorithms on solving uniform random k-SAT instances and implemented in the corresponding six state-of-the-art SLS solvers - WalkSATlm, DCCASat, Score$_2$SAT, CSCCSat, Probsat, and Sparrow, respectively. The extended SLS solvers are named WalkSATlm_E, DCCASat_E, Score$_2$SAT_E, CSCCSat_E, Probsat_E, and Sparrow_E, respectively.

All those algorithms are evaluated through benchmarks in terms of their capabilities and efficiency on uniform random *k*-SAT instances from the random track of the SAT Competitions in 2016, 2017, and 2018, considering both the robustness and the runtime criteria. Experimental results show that the versions of improved SLS solvers outperform their original versions, and especially WalkSATlm_E, Score$_2$SAT_E, and CSCCSat_E outperform the winner of the random track of SAT competition in 2017, and CSCCSat _E outperforms the silver of the random satisfiable track of SAT competition in 2016, and CSCCSat_E and WalkSAT_E outperform the second-ranked solver probSAT among the SLS solvers in terms of capability for the SAT competition in 2018. Despite their simplicity, the proposed allocation strategy and initial probability distributions are effective for improving six SLS solvers on all instances. In addition, based on the introduced initial probability distribution method, the present work proposes a new hard and easy distribution of random 3-SAT instances and provides additional comparative analysis with random SLS solvers based on large-scale experiments.

The remaining of the paper is organized as follows. Section 2 introduces some required definitions, notations, and background. Section 3 presents the allocation strategy and initial probability distributions on uniform random *k*-SAT. Section 4 describes our contribution by detailing the WalkSATlm_E, DCCASat_E, Score$_2$SAT_E, CSCCSat_E, Probsat_E, and Sparrow_E, along with experimental evaluations and analyzes the effectiveness of the allocation strategy on uniform random k-SAT instances in Section 5. Section 6 presents parameter tuning and related analyses on random 3-SAT. Finally, we give some concluding remarks and future directions in Section 7.

2 Preliminaries

In this section, we first introduce some basic definitions and notation about SAT problems. Then, we briefly review the WalkSATlm, DCCASat, Score$_2$SAT, CSCCSat, Probsat, and Sparrow, and related works. Finally, we introduce the allocation strategy, which is also an important component in our improving algorithms.

2.1 Definitions and Notations

An instance *F* of the satisfiability problem (SAT) is defined by a pair $F = (X, C)$ such that $X = \{x_1, x_2, \cdots, x_n\}$is a set of *n* boolean variables (their values belong to the set {true, false}) and $C = \{c_1, c_2, \cdots, c_m\}$is a set of *m* clauses. A clause $c_i \in C$ is a disjunction of literals, and a literal is either a variable x_i (which is called positive literal) or its negation $\neg x_i$ (which is called negative literal). A clause can also be represented by the set of its literals. For a set of literals *L*, $var(L)$is the set of the variables in *L*. Accordingly, $var(c_i)$ is the set consisting of the variables appearing in c_i. The size of a clause c_i is the number of its literals, and it is denoted by$|c_i| = |var(c_i)|$. If the size of each clause in *C* is equal to *k* ($\forall c_i \in C, |c_i| = k$) then the instance is a *k*–SAT instance, and $r = m/n$ is its clause-to-variable ratio. An instance $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is a conjunction of clauses.

A satisfying assignment *I* for instance *F* is an assignment to its variables such that the instance evaluates to true. If x_i is true in *I* (or false) then $x_i \in I$ (or $\neg x_i \in I$). Given an instance *F*, the boolean satisfiability problem is to find a satisfying assignment or prove that none exists. A literal *l* is said to be satisfied by the current assignment in *I* if $l \in I$and falsified if $\neg l \in I$. A clause is satisfied in *I* if at least one of its literals is true in *I*, and falsified otherwise. A solution of *F* is an assignment that satisfies all the clauses of *F*.

A uniform random *k*-SAT instance is a formula which contains exactly *k* distinct variables. The uniform random *k*-SAT instances are generated for two different sizes: medium and huge on SAT Competitions. The clause-to-variable ratio of medium-sized instances is equal to the phase-transition ratio. The number of variables differs for all the instances. The huge random instances have a few million clauses. For the huge instances, the ratio ranges from far from the phase-transition ratio to relatively close, while for each *k* the number of variables is the same.

The popular variable properties used by SLS algorithms for SAT are *make*(*x*) and *break*(*x*), which are the number of clauses that would become satisfied and unsatisfied respectively if variable *x* was to be flipped, and *score*(*x*), which is defined as to *make*(*x*)-*break*(*x*). Usually, SLS algorithms for uniform random *k*-SAT instances select a variable to flip according to scoring functions. A scoring function can be a simple property or any mathematical expression with one or more properties. The second level properties consider the satisfaction degree of clauses, which is defined as the number of true literals in the clause. A clause with a satisfaction degree of δ is said to be a δ-*satisfied* clause. For a variable *x*, $score_2(x)$ [13, 15] for SAT is defined as $make_2(x)$-$break_2(x)$, where $make_2(x)$ is the number of 1-*satisfied* clauses that would become 2-*satisfied* by flipping *x*, and $break_2(x)$ is the number of 2-*satisfied* clauses that would become 1-*satisfied* by flipping *x*. The other level properties take into account *level make*, abbreviated as *lmake*, which incorporates *make*(*x*) and $make_2(x)$ properties of different levels. The level *make* has been used in SLS algorithms for SAT[16, 21]. The *age*(*x*) of a variable *x* is defined as the number of search steps that have occurred since *x* was last flipped. Two variables are *neighbors* if and only if they appear in the same clause. The set of *x* neighbors is denoted by *N*(*x*), which is the set of all neighboring variables of variable *x*. The states of all clauses in which *x* appears under the current assignment *I* is denoted by *CL*(*x*).

2.2 Configuration Checking for SAT

The Configuration Checking (CC) strategy was initially introduced for improving local search for the minimum vertex cover problem [11]. It aims at avoiding cycles in local search, i.e., revisiting the already visited assignments too early. It has been successfully used in some areas, such as MVC[13], SAT[1, 33] and so on.

In the SAT problems, the CC strategy (NVCC) considers the relationship between a variable x_i and its neighbors $N(x_i)$[13]. It defines the configuration $C(x_i)$ of a variable x_i by the subset of *I*, which is restricted to the variables of $N(x_i)$. In other words, we have$C(x_i) = I|N(x_i)|$. The Boolean array *NV_Changed* is used to implement NVCC. For a variable x_i, *NV_Changed*(x_i)=1 means that at least one variable in $N(xi)$ has been flipped since x_i's last flip. A variable x_i is neighboring-variables-based configuration changed decreasing (NVD) iff $score(x_i) > 0$and *NV_Changed*(x_i)=1. The notation *NVD_vars* is used to denote the set of all NVD variables. The other notion in aspiration is the significant decreasing (SD) variable. If $score(x_i) > \overline{w}$, where $\overline{w}$ is the averaged clause weight (over all clauses), the variable x_i is an SD variable. The notation *SD_vars* is used to denote the set of all SD variables.

Compared to the NVCC strategy, the CSCC strategy considers the relationship between a variable x_i and the states of all clauses in which *x* appears[40]. $C(x_i)$ of a variable x_i under the current assignment *I* is restricted to the variables of $CL(x_i)$. For a variable x_i, a change on any bit of $C(x_i)$ is considered as a change on the whole $C(x_i)$ vector. The Boolean array *CS_Changed* is used to implement CSCC. For a variable *x*, *CS_Changed*(x_i)=1 means that at least one clause in $CL(x_i)$ has changed its state (from unsatisfied to satisfied or from satisfied to unsatisfied) since x_i's last flip. A variable *x* is clause-states-based configuration changed decreasing (CSD) iff $score(x_i) > 0$and *CS_Changed*(x_i)=1. The notation *CSD_vars* is used to denote the set of all CSD variables. The other implementation of the CSCC strategy is to employ an integer array *ConfTimes* for variables. In the beginning, for each variable *x*, *ConfTimes*(*x*) is set to 1. Whenever a variable *x* is flipped, *ConfTimes*(*x*) is reset to 0. Then each clause $C \in CL(x_i)$ is checked whether its state is changed by flipping *x*. If this is the case, for each variable y ($y \neq x$) in *C*, *ConfTimes*(*y*) is increased by 1[34].

2.3 Stochastic Local Search for SAT

For a CNF formula *F*, initially, a basic SLS algorithm for SAT randomly generates a complete assignment *I*, which may falsify some clauses. Then, it attempts to find a solution by repeatedly iteratively this assignment by flipping the boolean value of one variable (changing its value from false to true, or true to false) according to several variable selection heuristics at a time until seeking out a solution (a solution is found) or timeout. SLS algorithms for SAT differ by the heuristic used in choosing which variable to flip (for examples, see the literature[12, 14, 23]). Below we briefly overview some popular SLS algorithms with some justifications why they were selected for further improvements.

WalkSATlm based heuristics: WalkSATlm algorithm utilizes the random walk and the multilevel *break* properties [41, 45]. Originally introduced in the literature[16], the multilevel *make* is added to WalkSAT for solving uniform random *k*-SAT with *k>3*. WalkSATlm applies the following variable selection scheme in each step[21]. First, WalkSATlm

randomly selects a falsified clause *C*, and then it sorts the variables of *C* according to their *break*. If there exist variables with a *break* value of 0 in clause C, one of such variables is flipped. If no such variable exists, then with a certain probability *p* (the noise parameter), one of the variables from *C* is randomly selected to flip; otherwise, one of the variables with the equal minimum *break* value from *C* is flipped by preferring the variable with the greatest *lmake* value; if there exist variables with the equal greatest *lmake* value, one of the variables is randomly selected to flip.

Recently, there has been increasing interest in WalkSATlm due to the improving WalkSAT of its great power on random long-clause instances, and WalkSATlm is very competitive with current SLS solvers on uniform random *k*-SAT instances. In 2015, Luo et al. observed that *lmake* was far more powerful than had been appreciated[38].

DCCASat based heuristics: DCCASat algorithm adopts the configuration checking strategy. Originally introduced in the literature[37], a novel concept named double configuration checking with aspiration (DCCA) heuristic combines NVCC and CSCC for solving uniform random *k*-SAT with *k>3*, and DCCA applies the following variable selection scheme in each step. First, if *CSD_vars* is not empty, DCCA selects a variable with the greatest score in CSD_vars; otherwise, if NVD_vars is not empty, it selects a variable with the greatest score in NVD_vars; then, if NVD_vars is empty, DCCA activates aspiration to select a variable in SD_vars with the greatest score.

DCCASat is one of the most influential SLS algorithms for SAT, and it is still competitive with the state-of-the-art SLS solvers in solving random long-clause instances. DCCASat algorithm serves as the basis of our algorithm in this work.

CSCCSat based heuristics: CSCCSat solver is a combination of two SLS solvers, FrwCB2014 developed based on FrwCB[34], which utilizes the *linear make* function to break ties, and DCCASat2014 improved based on the above DCCASat, which utilizes the algorithmic settings for random instances described in the literature[38]. The procedure of CSCCSat can be described as follows. CSCCSat first decides the type of this instance for solving a random SAT instance. Then based on the clause-to-variable ratio of the instance, CSCCSat picks either DCCASat2014 or FrwCB2014 to solve the instance.

CSCCSat is very competitive with the state-of-the-art SLS solvers on uniform random *k*-SAT instances. It won the '3rd Place Award' in the random SAT track of SAT Competition 2014 and the '2nd Place Award in the random SAT track of SAT Competition 2016.

Score$_2$SAT based heuristics: Score$_2$SAT won the '3rd Place Award' in the random SAT track of SAT Competition 2017. The Score$_2$SAT solver is a combination of DCCASat [37] and WalkSATlm[21]. The procedure of Score$_2$SAT can be described as follows. Score$_2$SAT first decides the type of this instance for solving a random SAT instance. Based on the clause-to-variable ratio of the instance, denoted as *r*, Score$_2$SAT selects either WalkSATlm or DCCASat to solve the instance. The procedures of Score$_2$SAT are described as follows. For random 3-SAT instances with $r \le 4.24$, WalkSATlm is called; for random 3-SAT instances with $r > 4.24$, DCCASat is called. For random 5-SAT instances with $r \le 20.1$, WalkSATlm is called; for random 5-SAT instances with $r > 20.1$, DCCASat is called. For random 7-SAT instances with $r \le 80$, WalkSATlm is called; for random 7-SAT instances with r > 80, DCCASat is called.

Sparrow based heuristics: Sparrow is the winner of the SAT Competition 2011 category random SAT. Sparrow is based on gNovelty+ [43], which is the winner of the SAT 2007 Competition category random. When Sparrow reaches a local minimum, it computes a probability distribution $p(x)$ defined as an exponential function of the $score(x)$ value on the variables from an unsatisfied clause *C*, and then selects a variable to flip according to this distribution as listed below[3]. The probability distributions is first presented in the SLS solver Sparrow [3].

$$p(x) = \frac{(c_1)^{score(x)} \cdot \left[\left(\frac{age(x)}{c_3}\right)^{c_2} + 1\right]}{\sum_{z \in C} (c_1)^{score(z)} \cdot \left[\left(\frac{age(z)}{c_3}\right)^{c_2} + 1\right]}, \quad 2\text{-}1)$$

where c_1 controls the influence of $score(x)$, and c_2 and c_3 specify, how quickly and how strongly $age(x)$ starts to affect the decision.

Probsat based heuristics: Probsat is the winner of the random track of SAT Competition 2013. Probsat is a simple and elegant SLS solver based on probability distributions[4]. Under the current assignment *I*, Probsat uses only the $make(x)$ and the $break(x)$ value of a variable *x* in the probability functions $f(x, I)$, which can have an exponential or a

polynomial shape as listed below. The *break*(*x*) *v*alue is more important than *make*(*x*) as discovered in the literature[4].

2-2） $$f(x,I)=\frac{(c_m)^{make(x,I)}}{(c_b)^{break(x,I)}}$$

2-3） $$f(x,I)=\frac{(make(x,I))^{c_m}}{(\varepsilon+break(x,I))^{c_b}}$$

where c_b and c_m are the parameters of the functions. Probsat applies the following variable selection scheme in each step. First, Probsat randomly selects a falsified clause *C*, and then it sorts random variable *x* of *C* according to the follow probability function.

2-4） $$f(x,I)\Big/\sum_{z\in C} f(z,I)$$

Although WalkSATlm, DCCASAT, CSCCSat, Score$_2$SAT, Sparrow, and Probsat are state-of-the-art SLS solvers, they solved only less than half of instances at and near phase transition of the random track of SAT Competition in the last three years. This result motivates our work toward improving these SLS solvers on uniform random *k*-SAT instances.

2.4 Hard and Easy Distributions of SAT Problems

Originally introduced in the literature[42], a detailed study of the average case of SAT testing for random instances has been carried out. A random SAT instance that is hard or easy to solve can be predicted in advance for Davis-Putnam (DP) [24] procedure, which is a complete algorithm.

Davis and Putnam explored the hardness of the random 3-SAT problems by DP. The hard and easy distributions were based on the number of instances solved by DP. The same variable instances with few clauses are under-constrained and have many satisfying assignments, and an assignment is likely to be found early in the search. Instances with many clauses are over-constrained (usually unsatisfiable), and thus contradictions are found easily so a full search can be completed quickly. Finally, instances in between hard and easy are much harder because they have relatively few satisfying assignments, and but the empty clause will only be generated after assigning values to many variables, resulting in a deep search tree.

However, large-scale experiments in satisfiability testing were based on quite easy instances. For example, the variable values were only 20 or 50, which is not convincing enough. In the last few years, random SAT instances are solved mainly by incomplete solvers--SLS solvers, and complete solvers have no advantage for solving random instances at and near the phase transition, which motivates our work toward predicting the hardness of random instances based on initial probability distributions, which is a new concept.

3 Allocation Strategy and Initial Probability Distribution on Uniform Random *k*-SAT

The allocation strategy[26] for random 3-SAT problem characterizes the greediness of assigning a variable at the beginning of the search, as it tends to decrease the number of falsified clauses, which is indeed the aim of the 3-SAT problem. It seems short-sighted to take the allocation strategy for solving the 3-SAT problem, and there is no careful analysis of the setting of parameters. To address this issue, first, we extend the allocation strategy from random 3-SAT to random *k*-SAT, and then we propose a new concept P(*r*) (*r* is the clause-to-variable ratio of the instance) to determine the parameters used in the allocation strategy for uniform random *k*-SAT instances with the same *r*.

3.1 Allocation Strategy on Uniform Random *k*-SAT

In this section, we first extend the allocation strategy from random 3-SAT to random *k*-SAT instances. We keep the same term “the allocation strategy” for simplicity. The reader can easily distinguish it from the random 3-SAT or random *k*-SAT. The idea of allocation strategy is to give an initial assignment for solving the *k*-SAT problem and aims to guide the trend of optimal truth assignment in advance to accelerate the finding of the optimal solution.

Definition 1. *Given a CNF formula F for a k-SAT instance, the Variable allocation degree function* $Vad(x_i)$ *of a*

variable $x_i \in X_n$ *is defined as*

$$3-1) \quad Vad(x_i) = \begin{cases} pad+1, & n_{x_i} = 0 \\ p_{x_i} / n_{x_i}, & otherwise \end{cases}$$

where n_{x_i}*and* p_{x_i} *are the number of occurrences of negative and positive literals for each variable* x_i *in each formula, and pad is a positive decimal parameter greater than 1.*

Note that when $Vad(x_i) \geq 1$, it indicates that the number of positive literals is more than the number of negative literals for variable x_i, and it is called the *positive allocation degree* (*pad*) of variable x_i. When $Vad(x_i) < 1$, it indicates that the number of positive literals is less than the number of negative literals for variable x_i, and it is called the *negative allocation degree* (*nad*) of variable x_i. Random SAT is symmetric when it comes to the roles of the truth values and the positive and negative occurrences of variables in clauses, thus we set **pad*nad≈1**.

This function is so simple that it can be computed with little overhead. It is the underlying function that leads to the following allocation strategy. Recall that a variable satisfies allocation strategy by the following definition.

Definition 2. *Given a CNF formula* F *for a k-SAT instance and its Vad*(*x*) *function, a variable* $x \in X_n$ *satisfies Allocation Strategy*(*AS*) *iff* $Vad(x) > pad$ *or* $Vad(x) < nad$.

In this work, we use the notation *ASvars* to denote the set of all variables which satisfies allocation strategy during the search. In the search process, our algorithm prefers to generate the initial assignment of variables in the greedy phase from *ASvars* set.

Definition 3. *Given a CNF formula* F *for a k-SAT instance and its Vad*(*x*) *function, a variable* $x \in X_n$ *is a Uniform Variable* (*UV*) *if* $nad \leq Vad(x) \leq pad$.

In this work, we use the notation *UVvars* to denote the set of all *UV* variables during the search. In the search process, our algorithms prefer to generate the initial assignment of variables randomly from the *UVvars* set.

Based on the above definitions, we design a variable allocation value function initializing an assignment. The resulting function is dubbed as *Vav*(*x*) and is given as follows.

Definition 4. *Given a CNF formula* F *for a k-SAT instance, and a function* $\chi(x_i)$*that randomly produces 0 or1 for each variable* x_i *, the Variable allocation value function* $Vav(x_i)$ *of a variable*$x_i \in X_n$*is defined as*

$$3-2) \quad Vav(x_i) = \begin{cases} 1, & Vad(x_i) > pad \\ 0, & Vad(x_i) < nad \\ \chi(x_i), & x_i \in UVvars \end{cases}$$

where $\chi(x_i)$ *randomly generates 0 and 1 and is presented in such a way* $rand(\quad)\%2$ *in a procedure (1 means true, and 0 means flase in Vav*(x_i)*) and pad is a positive decimal parameter greater than 1, and nad is a positive decimal parameter less than 1.*

Its simplicity allows its potential usage in solving *k*-SAT instances and perhaps other combinatorial search problems. The parameters *pad* and *nad* can be easily tuned. In our algorithms, when initializing an assignment, the algorithms make use of this function. We will show that the function is a better choice than the "traditional" strategy generating an assignment randomly.

A variable $x_i \in ASvars$ is initially assigned in a deterministic way, i.e., if $Vad(x_i) > pad$, the initial assignment of variable x_i is 1; if $Vad(x_i) < nad$, the initial assignment of variable x_i is 0. This is necessary, as the solution of *k*-SAT instance has randomness, and the SLS solvers initially generate an assignment randomly, which leads to unpredictable the objective at the beginning of search without any controlling mechanism. If the initial assignment is generated in a deterministic way, which maybe guide the next search process, and reduce the search space and speed up the search efficiency. Most SLS algorithms for SAT prefer to flip variables in the greedy search mode. The notation of variables of allocation strategy is a good alternative to be considered for the initial assignment in the greedy phase.

Based on the above definitions, we design a function taking account into ratio of the variables satisfying the allocation strategy. The resulting function is dubbed as *Sd*(*F*) and is given as follows.

Definition 5. *Given a CNF formula F for a k-SAT instance, the number of variables n, and a function $\phi(x_i, pad, nad)$that only produces 0 or 1 for each variable $x_i \in X_n$, the Satisfaction degree $Sd(F)$ of ASvars is defined as*

$$Sd(F) = \frac{\sum_{i=1}^{n} \phi(x_i, pad, nad)}{n} \quad (3-4)$$

where $\phi(x_i, pad, nad) = \begin{cases} 1, & x_i \in ASvars \\ 0, & otherwise \end{cases}$, *pad is a positive decimal parameter greater than 1, and nad is a negative decimal parameter less than 1.*

Based on the large-scale experiments on uniform *k*-SAT instances of the random track of SAT competition in 2016, 2017, and 2018, we found that when the parameters *pad* and *nad* are fixed, the *Sd* of *ASvars* differ by about 10% for different *k*-SAT instances with the same *r*, *k*, and *n* respectively. However, the *Sd* of *ASvars* is almost equal for uniform *k*-SAT instances with the same *r* and *k* and different *n*. For example, when *pad*=1.8 and *nad*=0.4, based on *k*-SAT instances of random track of SAT competition in 2017, the *Sd* of *ASvars* is 0.244 for a 3-SAT instance with *r*=4.3; the *S*d of *ASvars* are equal to 0.269 for other 3-SAT instances with *r*=4.3; the *Sd* of *ASvars* are about 0.240 for 40 different 3-SAT instances with $r \approx 4.267$ and $500 \leq n \leq 12800$; the *Sd* of *ASvars* are about 0.002 for 40 different 5-SAT instances with $r \approx 21.112$ and $200 \leq n \leq 590$; and the *Sd* of *ASvars* are all equal to 0 for 40 different 7-SAT instances with $r \approx 87.79$ and $90 \leq n \leq 180$.

Different settings of the parameters *pad* and *nad* will directly affect the *Sd* of *ASvars* for SAT instances, and while different requirements of *Sd* of *ASvars* will indirectly lead to the setting of the parameters *pad* and *nad*. Therefore, the setting of the parameters *pad* and *nad* and *Sd* of *ASvars* have an interactive relationship so that the *Sd* of *ASvars* indirectly affects the performance of the algorithm.

Lemma 1. *Given a CNF formula F for a k-SAT instance, if satisfaction degree of ASvars is $Sd(F)$, then satisfaction degree of UVvars is $1 - Sd(F)$.*

Proof. For a variable *x*, *x* is a *UV* variable, meaning that $nad \leq Vav_d(x) \leq pad$. If *x* doesn't belong to *UV*vars, then $Vav(x) > pad$ *or* $Vav(x) < nad$, meaning that *x* belongs to *ASvars*, so $UVvars \cup EASvars = X_n$. As $UVvars = X_n - EASvars$, thus $|UVvars| = |X_n - EASvars|$, where $|\ |$ denote the number of elements in a set. Thus $\frac{|UVvars|}{|X_n|} = \frac{|X_n - EASvars|}{|X_n|} = \frac{|X_n|}{|X_n|} - \frac{|EASvars|}{|X_n|} = 1 - \frac{|EASvars|}{|X_n|} = 1 - Sd(F)$.

3.2 Initial Probability Distributions on Uniform random k-SAT

In this section, we introduce the initial probability distributions *P*(*r*) of random instances with the clause-to-variable *r* based on *Sd*(*F*) values of a uniform random *k*-SAT instance, which is formally defined as follows.

Definition 6. *Given a set of CNF formulas G for k-SAT instances, the clause-to-variable ratio r, and the total number of formulas W, the initial probability distributions of r, denoted by P(r), is defined as a function based on Sd(F) of ASvars such that*

$$P(r) = \frac{\sum_{F \in G} Sd(F)}{W} \quad (3-5)$$

From definition 6, we know that the bigger *Sd*(*F*) is, the bigger *P*(*r*) is, so they have a positive correlation. According to definition 5, $\phi(x_i, pad, nad)$ and *Sd*(*F*) have a positive correlation, and the number of elements in *ASvars* and $\phi(x_i, pad, nad)$ have a positive correlation, so the number of elements in *ASvars* and *P*(*r*) have a positive correlation. Based on definition 2, the number of elements in *ASvars* is related to *pad* and *nad*. The bigger the *nad* is, the bigger the number of elements in *ASvars* is; however, the smaller the *pad* is, the bigger the number of elements in *ASvars* is. So *nad* and *P*(*r*) have a positive correlation, and *pad* and *P*(*r*) have a negative correlation.

To get a picture of how the value of *P*(*r*) varies under different values of the parameters *pad* and *nad*, we have done a uniform sampling of *pad* ϵ [1.0, 3.0] and *nad* ϵ [0.1, 1.0] on all random 3-SAT instances with 6000 variables with

r≈4.267 from the random track of SAT competition in 2018[50] (see below).

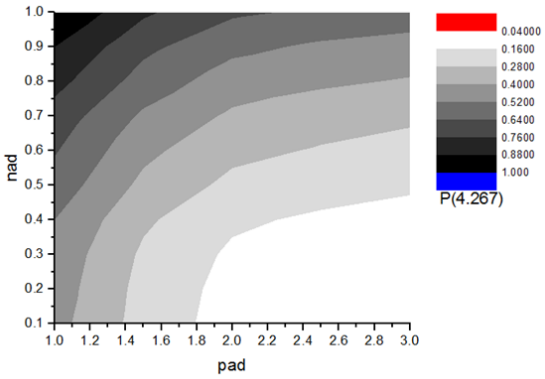

Fig. 1. Parameter space P(4.267) plot: The plot shows the relationship between parameters pad and nad under the same value of P(4.267). The brighter the area the bigger the value of P(4.267) with that parameter settings.

According to Fig.1, when the value of *P(4.267)* is fixed, the relationship between *pad* and *nad* is hyperbolic. As mentioned in Section 3.1, it turns out that *nad* and *P*(*r*) have a positive correlation, and *pad* and *P*(*r*) have a negative correlation, i.e., **pad*nad≈1**.

To generate balanced instances, Balint[6] has assumed the same positive and negative literals probability for the random track of SAT competition in 2018. In general, if we want to get the *P*(*r*) value of a certain interval and balance the number of variables satisfying *Vad*>1 and the number of variables satisfying *Vad* <1, we adjust *pad* and *nad* to make the difference between the number of variables that satisfy *Vav*>*pad* and the number of variables that satisfy *Vav*<*nad* as small as possible.

For example, if *pad*= 1.46 and *nad* =0.68, then the value of *P*(*21.117*) is 0.056 for 40 random 5-SAT instances from the random track of SAT competition in 2017[49]. We change the value of *P*(*21.117*) from 0.056 to 0.406, if we only adjust the value of *nad* and make the value of *nad* equal to 0.94, and then the value of *P*(*21.117*) is 0.406. However, if the allocation strategy removes the condition $Vav(x) > pad$, then the value of *P*(*21.117*) is 0.377 with *nad*=0.94; and if the allocation strategy removes the condition$Vav(x) < nad$, then the value of *P*(*21.117*) is 0.029 with *pad*=1.46. In *ASvars*, variables with more negative literals than positive literals occupy 92.9%, and variables with more positive literals than negative literals are only 7.1%, so the number of positive and negative literals in *ASvars* is seriously out of balance. That is to say that there is a huge difference in the number of variables that satisfy these two conditions. If *pad* =1.18 and *nad* = 0.85, then *P*(*21.117*)=0.404, which is almost close to 0.406. If the allocation strategy removes the condition $Vav(x) > pad$, then the value of *P*(*21.117*) is 0.205 with *nad*=0.85; and if the allocation strategy removes the condition $Vav(x) < nad$, then the value of *P*(*21.117*) is 0.199 with *pad*=1.18. The number of variables satisfying $Vav(x) < nad$ is almost equal to the number of variables satisfying $Vav(x) > pad$. Therefore, we could adjust *pad* and *nad* in a balanced way so that the number of variables satisfying a certain condition is not too large or too small.

In the SLS algorithms, before generating an initial assignment of variables, we can make use of the above-mentioned method to analyze the settings of parameters *pad* and *nad* and demonstrate the effectiveness of using this method to set parameters *pad* and *nad.*

3.3 Parameter Setting of the Allocation Strategy Based on Initial Probability Distribution

3.3.1 Influences of parameter setting on the performance

To get a picture of how the performance of the solver varies under different values of the parameters *pad* and *nad*, we have done a uniform sampling of *pad* ϵ [1.0, 4.0] and *nad* ϵ [0, 1.0] (see below). We have taken WalkSATlm solver as an example, and WalkSATlm is extended by using the allocation strategy (as detail in algorithm WalkSATlm_E in section 4**)** with the different parameter settings on a set of randomly generated 3-SAT instances with 1000 variables with r≈4.267. The cutoff limit was set to 10s.

In the case of 3-SAT, a good choice of the parameters is *nad >0* (as expected) and *pad*>1. For example, *pad* =1.8 and *nad* = 0.56 (see Fig. 2 left diagram and the survey in the following Table 1) are an optimal setting. In the interval *pad* ϵ

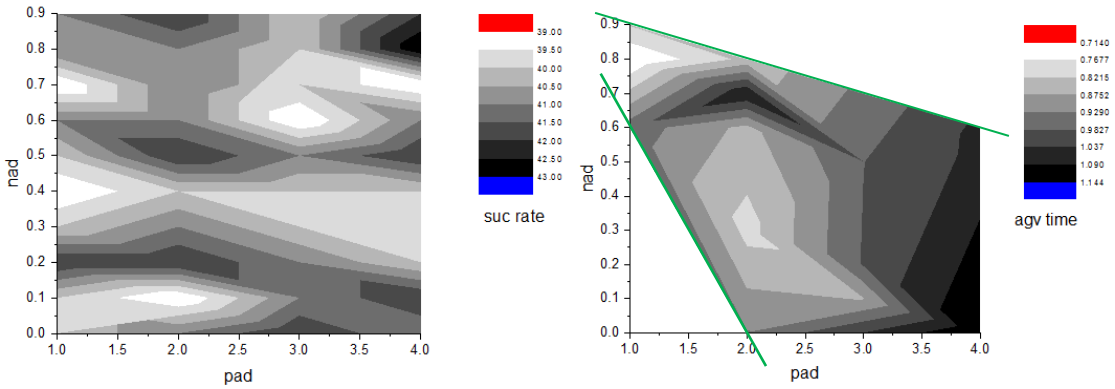

Fig. 2. Parameter space performance plot: The left and right plot show the performance of different combinations of parameters *pad* and *nad*. The darker the area the better the success rate (suc rate) of the solver with that parameter settings. The brighter the area the better the average runtime (agv time) of the solver with that parameter settings.

[1, 2.5] and *nad* ϵ [0.15, 0.25], the optimal choice of parameters can be described by a constant function: *nad=0.2*. In fact, in the interval *pad* ϵ [1.5, 2.5], the *nad* has three choices of intervals, [0.15, 0.25], [0.45, 0.58], and [0.85, 0.9], respectively. In the right plot of Fig. 2, the area between two green lines is the time cost for selecting the highest success rate on the left graphic data. From the right plot, in the interval *pad* ϵ [1, 2], the preferred interval for the parameter *nad* is between 0.75 and 0.9; in the interval *pad* ϵ [1.75, 3], the preferred interval for the parameter *nad* is about between 0.1 and 0.6. In the process of solving the 3-SAT instances, different parameters *nad* and *pad* have a direct impact on the performance of the algorithm. For the running time, it can be found through experiments that even a difference of more than 2 times may occur, and it can affect the success rate of the solution. As shown in Fig. 2, when *pad=3* and *nad=0.6*, the success rate is 39%. When *pad=1.8* and *nad=0.56*, the success rate is 43%. As mentioned in Section 3.1, it turns out that the relationship between *pad* and *nad* is *pad*nad*≈1, i.e., 1.8*0.56≈1.

It turns out that the parameters *pad* and *nad* should firstly satisfy two conditions. **The first condition is to ensure that the algorithm has the highest success rate. The second condition is to ensure that the algorithm takes the least time**. Therefore, combining the left and right plots, the preferred interval of the parameters *pad* and *nad* are [1.5, 2.5] and [0.5, 0.6] respectively. In our algorithms, under the two conditions, in order to balance the number of variables satisfying $Vav(x) < nad$ and $Vav(x) > pad$, **the parameters also should satisfy the third condition, which is to make the difference between the number of variables that satisfy *Vav*(x)>*pad* and the number of variables that satisfy *Vav*(x)<*nad* as small as possible, and the parameters should be set to satisfy *pad*nad*≈1**. Thus, we set parameters *pad*=1.8 and *nad*=0.56, and if the allocation strategy removes the condition $Vav(x) > pad$, the value of *P*(*4.267*) is 0.161 with *nad*=0.56, and if the allocation strategy removes the condition $Vav(x) < nad$, the value of *P*(*4.267*) is 0.153 with *pad*=1.8. Based on the parameters *pad*=1.8 and *nad*=0.56, we test the *P*(4.267) of the above instances which is 0.330, and *P*(4.267) is 0.328 on 3-SAT instances at the phase transition of random track of SAT competitions in 2016[48], 2017 and 2018.

3.3.2 Enhancing the parameter setting using initial probability distribution

According to the above method, the parameter setting of each track of *k*-SAT instances needs to test 40 sets of data. If the parameters of each track of *k*-SAT instances with the different clause-to-variable ratio of *r* are set according to the graphic analysis, it will take plenty of time to test. Here, we propose a new method to determine the parameters, and the implementation is simpler and has less overhead —if *P*(*r*) can be guaranteed to be between 0.33 and 0.40 for these parameters on random 3-SAT, 5-SAT, and 7-SAT with the clause-to-variable ratio of *r*, it is the optimal parameter setting. If *P*(*r*) is greater than 0.40, the number of positive and negative literals of each variable that satisfies the allocation strategy is almost equal. Then there is the initial assignment of more variables generated in a controlled way, which can be too greedy to determine. If *P*(*r*) is less than 0.33, most variables belong to *UVvars* that do not satisfy the allocation strategy. Then there is the initial assignment of more variables generated randomly, which cannot show the superiority of our method. Without loss of generality, we set *P*(*r*) as about 0.33, 0.37, and 0.40 for these parameters on random 3-SAT, 5-SAT, and 7-SAT with the clause-to-variable ratio of *r* respectively, and the obtained parameter setting could be optimal.

For different parameter settings, the satisfaction degree $Sd(F)$ of each instance F is likely to be the same, that is, initial probability distribution $P(r)$ on uniform random k-SAT instances with the clause-to-variable ratio of r is the same, so it is not necessary to test the success rate and the running time of each instance again (the success rate and average running time of the extended solvers based on the allocation strategy are almost the same, respectively). This can be illustrated using the following examples:

Example 1: considering a set of randomly generated 5-SAT instances with n=100 and r≈21.117, we found that $Sd(F)$ of each instance F with the parameters pad=2 and nad=0.7, is the same as that with the parameters pad=3 and nad=0.7, that is, $P(21.117)$ corresponding to these two parameters is the same, and then their success rate and average running time are 48%, 0.135 and 48%, 0.138, respectively. It follows that by checking the initial probability distribution for each type of uniform random k-SAT instances, we can avoid a lot of unnecessary testing and save plenty of time in analyzing the optimal parameter settings.

Example 2: If $P(21.117)$ is set to be about 0.37, the parameters pad and nad can be quickly adjusted into the new and better ones: pad=1.26 and nad=0.87, respectively. Under these settings, the $P(21.117)$ is 0.359 on 5-SAT instances at the phase transition of the random track of SAT competition 2016, 2017, and 2018. We have done a uniform sampling of $pad \in [1.0, 4.0]$ and $nad \in [0, 1.0]$ with the cutoff limit of 10s to verify that this is a set of better parameter settings by testing the above randomly generated 5-SAT instances with n=100 and r≈21.117. Then we get a picture of how the performance of improving WalkSATlm based on the allocation strategy varies for different values of the pad and nad in Fig.3.

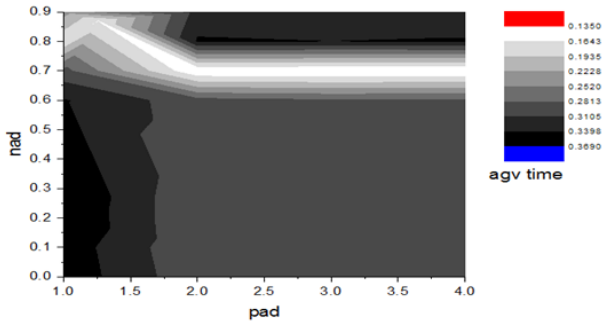

Fig. 3. Parameter space performance plot: The plot shows the performance of different combinations of parameters pad and nad. The brighter the area the better the average runtime (agv time) of the solver with that parameter settings.

According to the experimental results, we found that the success rate of the solver under different combinations of pad and nad is the same, so we only show the average running time in Fig. 3. It can be seen that a good choice of the parameters is nad >0.6 (as expected) and pad>1.0 from Fig. 3. When the pad belongs to [1.8, 4.0], the optimal choice of parameters can be described by a constant function: nad=0.7. When the pad belongs to [1.25, 1.8], the parameter nad and pad are linear. $P(21.117)$ has been set to be about 0.37 in Example 2. If the parameters satisfy the third condition, then we can get pad=1.26 and nad=0.87, which is a good choice.

As mentioned, it turns out that the influence of initial probability distributions is rather strong, and thus is reasonable and still leads to a good approach for the parameter setting– also because the implementation is simpler and has less overhead. In detail, while WalkSATlm generates a random assignment, WalkSATlm_E does so by the allocation strategy based on the initial probability distribution to reduce blind random assignment. It is reasonable to generate an assignment using the inherent properties of each instance (ratio of the number of occurrences of positive and negative literals for each variable), which capture complementary information to random property.

4 Six Improved SLS Solvers for Uniform random *k*-SAT Instances

In this section, we detail the part of our contribution which consists of improving SLS solvers, which adopt the allocation strategy and initial probability distributions into their SLS algorithms to determine the initial assignments of the variable in a controlled way to guide the search in the greedy mode when handling uniform random k-SAT

instances. The resulted in six extended SLS solvers are named WalkSATlm_E, DCCASat_E, CSCCSat_E, Score$_2$SAT_E, Sparrow_E, and Probsat_E, respectively.

The extended SLS algorithms differ from the original SLS algorithms only in the method of determining the initial assignment of variables except for Score$_2$SAT_E. At the beginning of each extended SLS algorithm for each solver, the initial assignment is generated in a controlled way by using the allocation strategy and initial probability distributions instead of generating them randomly.

More specially, each extended SLS algorithm starts from computing the $Vad(x)$ of each variable x and then generates a complete assignment based on $Vav(x)$, which combines an assignment of variables satisfying the allocation strategy and using initial probability distribution generated in a controlled way with an assignment of other variables that do not satisfy the allocation strategy and generated randomly. If no variables satisfy the allocation strategy, each extended algorithm will generate a random assignment like most SLS algorithms. After initialization, each extended algorithm executes a loop until it finds a satisfying assignment or reaches a limited number of steps denoted by *maxSteps*.

For simplicity's purpose, we denote six extended SLS algorithms with the same names as the corresponding SLS solvers, that is WalkSATlm_E, DCCASat_E, CSCCSat_E, Score$_2$SAT_E, Sparrow_E, and Probsat_E, respectively. The pseudo-code of each algorithm is summarized below.

4.1 The WalkSATlm_E Algorithm

This section presents the WalkSATlm_E algorithm in the Algorithm 1, which adopts the allocation strategy based on the variable allocation value function to guide the search in the greedy mode when handling uniform random k-SAT instances. The difference essentially is without greedy initial assignment and some changes in WalkSATlm vs WalkSATlm_E.

Algorithm 1: WalkSATlm_E (F)

```
   Input: CNF-formula F, k, MaxTries, MaxSteps
1  Parameters Noise parameter p ϵ [0, 1]
   Output: A satisfying assignment σ of F, or "no solution found"
2  begin
3    k ←Maximum clause length;
4    for i = 1 to MaxTries do
5      σ ←a generated truth assignment for F by Variable Allocation value function;
6      for j = 1 to MaxSteps do
7        if σ satisfies F then Return σ;
8        C ←an unsatisfied clause chosen at random;
9        if ( k= =3 )
10         With probability p
11           v ←a random variable in C;
12         With probability 1 − p
13           v ← a variable in C with minimum number of clauses from true to false;
14       else
15         if ∃ variable x ϵ C with break(x) = 0 then v ← x, breaking ties by preferring the one with the greatest lmake value;
16         else
17           With probability p
18             v ←a random variable in C;
19           With probability 1 − p
20             v ←a variable in C with the minimum break, breaking ties by preferring one with the greatest lmake value;
21       σ:= σ with v flipped;
22     Return "no solution found";
23 end
```

4.2 The DCCASat_E Algorithm

Before getting into the details of the DCCASat_E algorithm, we first introduce three components employed in the algorithm.

***CW* weighting scheme**. For the sake of diversification, DCCASat_E employs the *CW* clause weighting scheme (Luo et al, 2013). When a local optimum is reached, the clause weights are updated as follows. If DCCASat_E solves random 3-SAT instances, *CW* increases clause weights of all unsatisfied clauses by one; further, if the averaged clause

weight $\overline{w}$ exceeds a threshold *w*, each clause weight is smoothed as $w(c_i) = \lfloor p \times w(c_i) \rfloor + \lfloor q \times \overline{w} \rfloor$, where $0 \le p \le 1$ and $0 \le q \le 1$. If DCCASat_E solves uniform random *k*-SAT (*k>3*) instances, *CW* adopts the PAWS scheme [47].

The weight of each clause is initiated as 1. With probability *sp*, where $0 \le sp \le 1$, for each satisfied clause whose weight is larger than one, its weight is decreased by one; with probability 1-*sp*, the weight of each unsatisfied clause is increased by one.

Function *FG*. The function which prefers to pick the variable *x* with the greatest $age(x)$ for random 3-SAT instances, and the function which prefers to select the variable *x* with the greatest $hscore_2(x) = score_2(x) + [age(x)/\gamma]$, where γ is a positive integer[18] for uniform random *k*-SAT instances (*k>3*).

Function *FR*. The function which prefers to select the variable *x* with the greatest $age(x)$ for random 3-SAT instances, and the function which prefers to select the variable *x* with the greatest $hscore(x) = score(x) + [score_2(x)/d] + [age(x)/\beta]$for uniform random *k*-SAT instances (*k>3*), where d and β are positive integers[18, 19].

The DCCASat_E algorithm is outlined in Algorithm 2, as described below. The difference essentially is without greedy initial assignment and some changes in DCCASat vs DCCASat_E.

Algorithm 2: DCCASat_E (*F*)

Input: CNF-formula *F*, *MaxTries*, *MaxSteps*
Output: A satisfying assignment σ of *F*, or "no solution found"

```
1  begin
2    σ ←a generated truth assignment for F by Variable Allocation value function;
3    for i = 1 to MaxTries do
4      for j = 1 to MaxSteps do
5        if σ satisfies F then Return σ;
6          if ∃x ∈ CSD_vars then v ←a variable x in CSD_vars with the greatest score(x), breaking ties by function FG;
7        else
8          if ∃x ∈ NVD_vars then v ←a variable x in NVD_vars with the greatest score(x), breaking ties by function FG;
9          else
10           if ∃x ∈ SD_vars then v ←a variable x in SD_vars with the greatest score(x), breaking ties by function FG;
11         else
12           activate clause weighting scheme CW;
13           C ←an unsatisfied clause chosen at random;
14           v ←a variable in C by function FR;
15       σ:= σ with v flipped;
16   Return "no solution found";
17 end
```

4.3 The CSCCSat_E Algorithm

Before getting into the details of the CSCCSat_E algorithm, we first introduce one technique employed in the algorithm.

FrwCB solver. The FrwCB scheme is based on FrwCB [34] and applies the following variable selection method in each step. FrwCB[35] first selects an unsatisfied clause *C* randomly and then it employs the FrwCB scheme to pick a variable from *C*. If *CSD_vars* in *C* is not empty, FrwCB selects a variable *x* with the greatest *score*(*x*) in *CSD_vars*, and break ties by preferring a variable *x* with the greatest *ConfTimes*(*x*) for random 3-SAT instances and the greatest *lmake* function for uniform random *k*-SAT instances (*k>4*); if *CSD_vars* in *C* is empty, with a probability *p*, FrwCB picks a variable *x* with the greatest *ConfTimes*(*x*) in the break minimum variable of clause *C* and break ties by preferring the least recently flipped one for random 3-SAT instances and the greatest *lmake* function for uniform random *k*-SAT instances (*k*>4); otherwise FrwCB selects a variable *x* in clause *C* with the greatest *ConfTimes*(*x*), and break ties by preferring the least recently flipped one and the greatest lmake function for uniform random k-SAT instances (*k*>4). The FrwCB_E algorithm is outlined in Algorithm 3, as described below. FrwCB_E differs from FrwCB only in the method of initial assignment.

The CSCCSat_E solver is a combination of FrwCB_E and DCCASat_E described in Algorithm 2. The procedure of CSCCSat_E can be described as follows. For solving an SAT instance, CSCCSat_E first decides the type of this instance. Then based on the properties of the instance, CSCCSat_E calls either FrwCB_E or DCCASat_E to solve the instance. For random 3-SAT with $r \le 4.24$, CSCCSat_E calls FrwCB_E; for random 3-SAT with $r > 4.24$,

CSCCSat_E calls DCCASat_E. For random 5-SAT with $r \leq 20.1$, CSCCSat_E calls FrwCB_E; for random 5-SAT with $r > 20.1$, CSCCSat_E calls DCCASat_E. For random 7-SAT with $r \leq 80$, CSCCSat_E calls FrwCB_E; for random 7-SAT with$r > 80$, CSCCSat_E calls DCCASat_E.

Algorithm 3: FrwCB_E (*F*)

Input: CNF-formula *F*, *MaxSteps*
Output: A satisfying assignment σ of *F*, or "no solution found"

```
1  begin
2      σ ←a generated truth assignment for F by Variable Allocation value function;
3      initialize ConfTimes(x) as 1 for each variable x;
4      for j = 1 to MaxSteps do
5          if σ satisfies F then Return σ;
6              C ←an unsatisfied clause chosen at random;
7          if ∃x ∈ CSD_vars then v ←a variable x in CSD_vars with the greatest score(x), breaking ties by preferring the one with the greatest ConfTimes(x);
8          else if With the fixed probability p then
9              v ←a variable x with the greatest score(x) in the break minimum variable x of clause C, breaking ties by preferring the least recently flipped one;
10         else
11             v ←a variable x with the greatest ConfTimes(x) in clause C, breaking ties by preferring the least flipped one;
12         σ:= σ with v flipped;
13         update ConfTimes(x);
14     Return "no solution found";
15 end
```

4.4 The Score$_2$SAT_E Algorithm

The Score$_2$SAT_E solver is a combination of DCCASat_E described in Algorithm 2 and WalkSATlm_E described in Algorithm 1. The procedure of Score$_2$SAT_E can be described as follows. For solving a random SAT instance, Score$_2$SAT_E first decides the type of each instance. Based on the properties of the instance, Score$_2$SAT_E calls either DCCASat_E or WalkSATlm to solve the instance. The procedures of Score$_2$SAT are described as follows. For random 3-SAT, Score$_2$SAT _E calls WalkSATlm_E. For random 5-SAT with $r \leq 20.1$, Score$_2$SAT_E calls WalkSATlm_E; for random 5-SAT with $r > 20.1$, Score$_2$SAT_E calls DCCASat_E. For random 7-SAT, random 7-SAT, Score$_2$SAT _E calls WalkSATlm_E.

In the beginning, based on the properties of the instance Score$_2$SAT_E calls either WalkSATlm_E or DCCASat_E, and then computes *Vad*(*x*) of each variable *x*, generates a complete assignment based on the *Vav*(*x*) of each variable in a controlled way rather than randomly.

Score$_2$SAT_E differs from Score$_2$SAT in the method of initial assignment and in which solver should be called based on the properties of the instance. In detail, based on the properties of the instance, Score$_2$SAT calls one solver at first. For random 5-SAT with $r > 20.1$, Score$_2$SAT_E calls DCCASat_E to solve the instance, and otherwise, Score$_2$SAT_E calls WalkSATlm_E.

4.5 The Sparrow_E Algorithm

The Sparrow_E algorithm is outlined in Algorithm 4, as described below. The difference essentially is without greedy initial assignment and some changes in Sparrow vs Sparrow_E.

Algorithm 4: Sparrow_E (*F*)

Input: CNF-formula *F*, *MaxTries*, *MaxSteps*
Output: A satisfying assignment σ of *F*, or "no solution found"

```
1  begin
2      for i = 1 to MaxTries do
3          σ ←a generated truth assignment for F by Variable Allocation value function;
4          for j = 1 to MaxSteps do
5              if σ satisfies F then Return σ;
6                  C ←an unsatisfied clause chosen at random;
7                  for x in C do
8                      compute probability distribution p(x);
9                  v ←x according to probability distribution p(x);
10             σ:= σ with v flipped;
11     Return "no solution found";
12 end
```

4.6 The Probsat_E Algorithm

Probsat_E is outlined in Algorithm 5, as described below. The difference essentially is without initial assignment and some changes in Probsat vs Probsat_E. The only difference between Probsat_E and Sparrow_E is that the probability distributions used to select variable to flip.

Algorithm 5: Probsat_E (*F*)
Input: CNF-formula *F*, *MaxTries*, *MaxSteps*
Output: A satisfying assignment σ of *F*, or "no solution found"

```
1  begin
2      for i = 1 to MaxTries do
3          σ ←a generated truth assignment for F by variable allocation value function;
4          for j = 1 to MaxSteps do
5              if σ satisfies F then Return σ;
6              C ←an unsatisfied clause chosen at random;
7              for x in C do
8                  compute f(x, a);
9              v ←random variable x according to probability f(x,σ) / Σ_{z∈C} f(z,σ);
10             σ:= σ with v flipped;
11     Return "no solution found";
12 end
```

5 Evaluations of six improving SLS solvers

This section is dedicated to the experimental evaluation of the six improving SLS solvers on uniform random *k*-SAT instances with $k \in \{3, 5, 7\}$. The evaluation is performed on 450 uniform random *k*-SAT instances (all satisfiable) at and near the phase transition, which are issued from the random track of SAT Competitions in 2016, 2017, and 2018. We select these instances because they had different sizes and difficulties (regarding the *r* values). First, we compare WalkSAT_E with WalkSAT, DCCASat_E with DCCASat, CSCCSat_E with CSCCSat, Score$_2$SAT_E with Score$_2$SAT, Sparrow_E with Sparrow and Probsat_E with Probsat on random 3-SAT, 5-SAT, and 7-SAT instances at the phase transition. Then, we compare WalkSAT_E, DCCASat_E, CSCCSat_E, Score$_2$SAT_E, and Probsat_E with SLS solvers on uniform random *k*-SAT with instances $k \in \{3, 5, 7\}$ at and near phase transition.

5.1 Benchmarks and Experiment Preliminaries

All the instances used in these experiments are generated according to the uniform random *k*-SAT generator[51]. Specifically, we adopt the following three benchmarks.

1) **SAT Competition 2016:** All huge random 3-SAT instances ($3.86 \leq r \leq 4.24$, *#var*=1000000, 20 instances, 1 for each size), and all medium random 3-SAT instances (*r*=4.267, $5000 \leq \#var \leq 12800$, 40 instances, 1 for each size), and all uniform random *k*-SAT instances with $k > 3$ from the random track of SAT Competition in 2016 (120 instances, 60 for each *k*-SAT, $k = 5, 7$), which vary in both size and ratio. 20 extremely huge-sized random 5-SAT instances ($16.0 < r < 19.8$, *#var*=250000, 1 for each size), and 40 medium-sized random 5-SAT instances (*r*=21.117, $200 \leq \#var \leq 590$, 1 for each size). 20 extremely huge-sized random 7-SAT instances ($55.0 < r < 74.0$, *#var*=5000, 1 for each size), and 40 medium-sized random 7-SAT instances (*r*=87.79, $90 \leq \#var \leq 168$, 1 for each size). These random instances occupy 60% of the random track of SAT Competition in 2016, indicating that the importance of uniform random *k*-SAT instances has been highly recognized by the SAT community.

2) **SAT Competition 2017:** All huge random 3-SAT instances ($3.86 \leq r \leq 4.24$,*#var* =1000000, 20 instances, 1 for each size), and all medium random 3-SAT instances (*r*=4.267, $5000 \leq \#var \leq 12800$, 40 instances, 1 for each size), and all uniform random *k*-SAT instances with $k > 3$ (120 instances, 60 for each *k*-SAT, $k = 5, 7$) from random track of SAT Competition in 2017. These random instances occupy 60% of the random track of SAT Competition in 2017. The medium-sized instances vary from 200 variables to 590 variables at *r* = 21.117, and the huge-sized instances vary from *r* =16.0 to *r* =19.8 at *n*=250000 for 5-SAT. The medium-sized instances vary from 90 variables to 168 variables at *r* = 87.79, and the huge-sized instances vary from *r* =55.0 to *r* =74.0 at *n*=5000 for 7-SAT.

3) **SAT Competition 2018:** all huge random 3-SAT instances ($3.86 \leq r \leq 4.24$,*#var* =1000000, 20 instances, 1 for

each size), and all medium random 3-SAT instances (r=4.267, n=6000, 10 instances), and all uniform random k-SAT instances with $k > 3$ (60 instances, 30 for each k-SAT, k =5, 7) from the random track of SAT Competition in 2018. The huge-sized instances vary from r=16.0 to r=19.8 at n=250000, and the medium-sized instances are the same variable 250 at r=21.117 for 5-SAT. The huge-sized instances vary from r=55.0 to r=74.0 at n=50000, and the medium-sized instances are same variable 120 at r=87.79 for 7-SAT.

Table 1: Parameter setting of six improving SLS solvers

Instances Class	Ratio (r)	Parameters pad and nad for improving SLS solvers
3-SAT	$r < 4.267$	*pad*=2, *nad*=0.5
	$r \geq 4.267$	*pad*=1.8, *nad*=0.56
5-SAT	$r < 17$	*pad*=1.275, *nad*=0.855
	$17 \leq r < 18$	*pad*=1.26, *nad*=0.865
	$18 \leq r < 19$	*pad*=1.25, *nad*=0.85
	$r \geq 19$	*pad*=1.26, *nad*=0.87
7-SAT	$r < 60$	*pad*=1.08, *nad*=0.9
	$60 \leq r < 66$	*pad*=1.07, *nad*=0.91
	$66 \leq r < 87.79$	*pad*=1.06, *nad*=0.92
	$r \geq 87.79$	*pad*=1.05, *nad*=0.92

The small-sized random 3-SAT instances from the random track of SAT Competitions in 2016, 2017, and 2018 are too easy for complete solvers. If we want to solve such instances, we can combine the complete algorithm with the SLS algorithm. However, the above instances that are difficult to solve by the complete algorithms and SLS algorithms are the key in our experiments. Thus small-sized satisfiable random 3-SAT instances are not included in our experiments.

Six improved SLS solvers are implemented in C/C++ and compiled by g++ with'-O3' option. The parameter setting of these solvers is reported in Table 1, which is based on some preliminary experiments and the initial probability distributions. We believe through more careful tuning, better settings can be found, and thus the performance of the six improved SLS solvers can be further improved.

Competitors: We consider six SLS solvers, i.e., Score$_2$SAT[22], Walksatlm[21], CSCCSat[35,36], DCCASat [37], Probsat[4] and Sparrow[3]. The corresponding six extended SLS solvers are compared with these original ones, plus an additional one of yalsat[10]. Sparrow and Probsat won the random track of the SAT competitions in 2011 and 2013 respectively, and Sparrow combined with a complete algorithm to form a new solver winning the random track of SAT competition in 2018. CSCCSat won the bronze medal and silver medal of the random SAT track of SAT Competitions in 2014 and 2016. yalsat and Score$_2$SAT won the gold medal and bronze medal of the random track of SAT competition in 2017 respectively.

The top three solvers of the random track of SAT competition in 2018 are the hybrid solvers composed of SLS and complete algorithm, but our solvers are all based on SLS algorithms. From the detailed results of the random track of SAT competition in 2018[2], we can see these solvers did not solve any uniform random instances, except that the champion solver can solve few instances, so we did not compare the top three solvers with our six solvers.

The latest binary version of Sparrow is downloaded from the webpage of SAT competition in 2018[3]. The binaries of Score$_2$SAT, yalsat, Walksatlm, and DCCASat are downloaded from the webpage of SAT competition in 2017[4], and the binary of CSCCSat is downloaded from the webpage of SAT competition in 2016[5], and the binary of Probsat is downloaded online[6].

All experiments are carried on a workstation under a 64-bit Ubuntu Linux Operation System, using 2 cores of Intel(R) Core (TM) i3-3240M 3.4 GHz CPU and 8 GB RAM. Each run terminates upon either finding a satisfying assignment

[2]http://sat2018.forsyte.tuwien.ac.at/index.php?cat=results
[3]http://sat2018.forsyte.tuwien.ac.at/solvers/random/
[4]https://baldur.iti.kit.edu/sat-competition-2017/solvers/
[5]https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=downloads
[6]https://www.satcompetition.org/2013/downloads.shtml

or reaching a given cutoff time, which is set to 5000 seconds for *k*-SAT benchmarks.

Evaluating Methodology: For *k*-SAT benchmarks, each solver is performed 20 times on each instance with a cutoff time of 5000 seconds from SAT Competition in 2018. In this way, each solver is performed 10 times on each instance with a cutoff time of 5000 seconds (the same as in SAT competitions) from SAT Competitions in 2016 and 2017. As for performance metrics, we report the success rate in which a satisfying assignment is found ('suc') and the penalized average run time ('par 2) (an unsuccessful run is penalized as 2 times cutoff time) for each instance class. The best results for each instance class are highlighted in bold font between the improved SLS solvers and their original versions. If a solver has no successful run for an instance class, the corresponding 'par 2' is marked with "-".

5.2 Experimental Results of Six Improving SLS Solvers

In this section, we present the comparative experimental results of six improved SLS solvers and their competitors on each benchmark. **For the SAT competition, the standard to rank the solver is using the PAR-2 scheme: The score of a solver is defined as the sum of all runtimes for solved instances + 2*timeout for unsolved instances, and the lowest score wins.** Thus, our experimental results mainly observe the average par 2 of the solvers.

5.2.1 Results on uniform random k-SAT Competition 2017

A. Comparison between the extended six SLS solvers with the original SLS solvers

Instances Class	WalkSATlm suc par 2	WalkSATlm_E suc par 2
3-SAT	46.7% 5519	50% **5392**
5-SAT	33.3% 6751	41.6% **6030**
7-SAT	55.0% **5065**	53.3% 5991

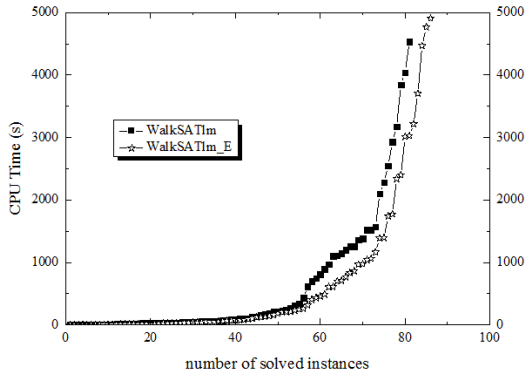

Fig. 4. Comparative results of WalkSATlm_E and WalkSATlm **on SAT Competition 2017 benchmark consisting of all uniform random k-SAT instances with *k* > 3 and random 3-SAT instances at and near phase transition**.

Instances Class	DCCASat suc par 2	DCCASat_E suc par 2
3-SAT	11.7% 8889	13.3% **8748**
5-SAT	38.3% 6345	43.3% **5947**
7-SAT	51.6% **5201**	51.6% 5211

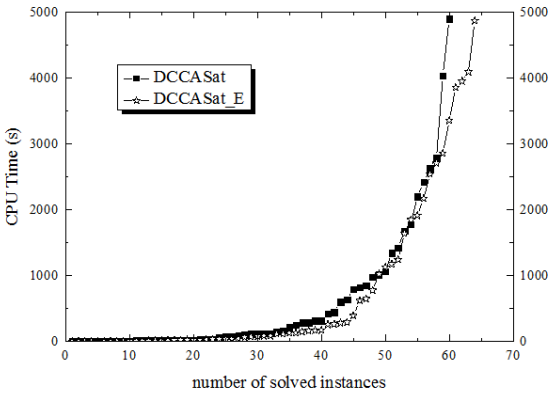

Fig. 5. Comparative results of DCCASat _E and DCCASat **on SAT Competition 2017 benchmark consisting of all uniform random k-SAT instances with *k* > 3 and random 3-SAT instances at and near phase transition**.

Figs. 4 to 9 show the comparative results between the extended six SLS solvers with the original SLS solvers on the uniform random *k*-SAT benchmark from SAT Competition in 2017. Each solver is performed 10 times on each instance class with a cutoff time of 5000 seconds. The left tables show a comparison of the success rate and the average runtime. The right plots show a comparison of average runtime distributions.

As is shown from Figs. 4 to 9, six improved SLS solvers show significantly better performance in terms of the

Instances Class	Score$_2$SAT suc par 2	Score$_2$SAT_E suc par 2
3-SAT	41.6% 5978	50% **5392**
5-SAT	40.0% 6137	45.0% **5763**
7-SAT	55.0% 4838	58.3% **4523**

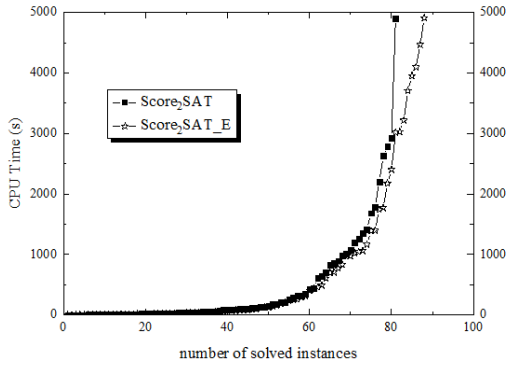

Fig. 6. Comparative results of Score$_2$SAT_E and Score$_2$SAT **on SAT Competition 2017 benchmark consisting of all uniform random k-SAT instances with $k > 3$ and random 3-SAT instances at and near phase transition**.

Instances Class	CSCCSat suc par 2	CSCCSat_E suc par 2
3-SAT	43.3% 5880	48.3% **5463**
5-SAT	45.0% **5768**	43.3% 5994
7-SAT	50.0% 5265	56.7% **4710**

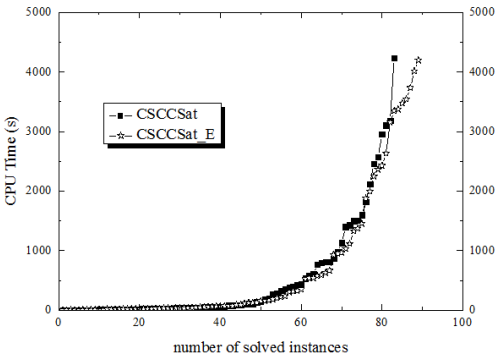

Fig. 7. Comparative results of CSCCSat_E and CSCCSat **on SAT Competition 2017 benchmark consisting of all uniform random k-SAT instances with $k > 3$ and random 3-SAT instances at and near phase transition**.

Instances Class	Probsat suc par 2	Probsat_E suc par 2
3-SAT	10.0% 9047	11.7% **8859**
5-SAT	40.0% **6207**	38.3% 6267
7-SAT	41.6% 6125	50.0% **5222**

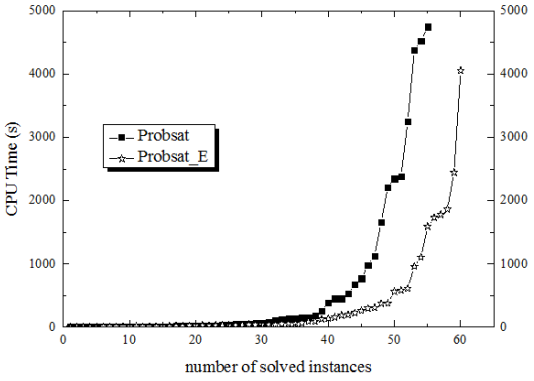

Fig. 8. Comparative results of Probsat_E and Probsat **on SAT Competition 2017 benchmark consisting of all uniform random k-SAT instances with $k > 3$ and random 3-SAT instances at and near phase transition**.

Instances Class	Sparrrow suc par 2	Sparrow_E suc par 2
3-SAT	25.0% 7921	33.3% **7166**
5-SAT	35.0% 6664	40.0% **6144**
7-SAT	56.7% **4876**	53.3% 4942

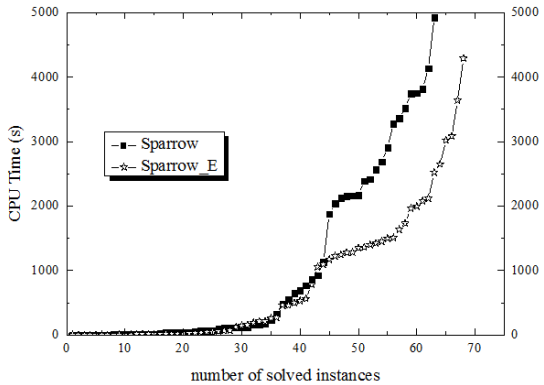

Fig. 9. Comparative results of Sparrow_E and Sparrow **on SAT Competition 2017 benchmark consisting of all uniform random k-SAT instances with $k > 3$ and random 3-SAT instances at and near phase transition**.

success rate, the average par 2, CPU time, and the number of solved instances than the original SLS solvers on the whole benchmark.

More specifically, WalkSATlm_E reduces over **2%** and **11%** of the average par 2 than WalkSATlm on uniform random 3-SAT and uniform random 5-SAT respectively. Although WalkSATlm_E performs worse than WalkSATlm on uniform random 7-SAT, it shows the best improvement over WalkSATlm on the overall uniform random *k*-SAT in terms of the success rate among the six improved SLS solvers.

DCCASat_E reduces over **2%** and **6%** of the average par 2 on uniform random 3-SAT and uniform random 5-SAT respectively. Although DCCASat_E performs worse than DCCASat on uniform random 7-SAT, it only increases over 0.2% of the average par 2 and shows improvement over DCCASat in terms of the success rate on the overall uniform random *k*-SAT.

Score$_2$SAT_E reduces over **10%**, **6%**, and **7%** of the average par 2 on uniform random 3-SAT, uniform random 5-SAT, and uniform random 7-SAT respectively, and shows improvement over Score$_2$SAT in terms of the success rate on the overall uniform random *k*-SAT.

Although CSCCSat_E performs worse than CSCCSat on uniform random 5-SAT, it reduces over **7%** and **11%** of the average par 2 on uniform random 3-SAT and uniform random 7-SAT respectively, and shows improvement over CSCCSat in terms of the success rate on the overall uniform random *k*-SAT.

Probsat_E reduces over **2%** and **15%** of the average par 2 on uniform random 3-SAT and uniform random 7-SAT respectively. Probsat_E reduces over **2%** and **15%** of the average par 2 on uniform random 3-SAT and uniform random 7-SAT respectively. Although Probsat_E performs worse than Probsat on uniform random 5-SAT, it only increases over 0.1% of the average par 2 and shows improvement over Probsat in terms of the success rate on the overall uniform random *k*-SAT.

Although Sparrow_E performs worse than Sparrow on uniform random 7-SAT, it reduced over **10%** and **8%** of the average par 2 on uniform random 3-SAT and uniform random 5-SAT respectively and shows improvement over Sparrow in terms of the success rate on the overall uniform random *k*-SAT.

Therefore, the proposed method enhances all six SLS solvers on the whole benchmark in the performance, which shows its robustness, indicating the effectiveness of allocation strategy using the initial probability distribution on the improved six SLS algorithms.

B. Comparison on random 3-SAT instances at phase transition from SAT Competition in 2017

We report the number of solved instances and the average par 2 for the extended six SLS solvers and their original ones, plus another SLS solvers yalsat and Dimetheus on random 3-SAT instances at phase transition from SAT Competition in 2017 in Table 2.

Table 2: Comparative results on all medium random 3-SAT instances at phase transition from SAT Competition 2017 based on 10 runs for each instance, with a cutoff time of 5000 seconds.

		3-SAT 5000-6800	3-SAT 7000-8800	3-SAT 9000-10800	3-SAT 11000-12800
Sparrow	suc	10%	20%	10%	20%
	par 2	9001	8162	9150	8270
Sparrow_E	suc	20%	20%	10%	30%
	par 2	**8431**	**8032**	**9004**	**7402**
Probsat	suc	10%	19%	0%	9%
	par 2	9151	8357	-	9237
Probsat_E	suc	10%	20%	0%	10%
	par 2	**9001**	**8065**	-	**9056**
DCCASat	suc	10%	20%	10%	28%
	par 2	9151	8026	9008	7299
DCCASat_E	suc	10%	30%	10%	30%

	par 2	**9002**	**7224**	**9001**	**7409**
WalkSATlm	suc	10%	35%	29%	20%
	par 2	9001	6721	7322	8031
WalkSATlm_E	suc	17%	40%	30%	30%
	par 2	**8557**	**6580**	**7677**	**7302**
CSCCSat	suc	10%	27%	10%	30%
	par 2	9000	7646	9004	7230
CSCCSat_E	suc	10%	40%	26%	30%
	par 2	9000	**6629**	**7742**	**7124**
Score$_2$SAT	suc	10%	20%	10%	28%
	par 2	9002	8026	9008	7479
Score$_2$SAT_E	suc	17%	38%	30%	30%
	par 2	**8557**	**6751**	**7677**	**7302**
Dimetheus	suc	10%	30%	10%	30%
	par 2	9001	7482	9030	7331
yalsat	suc	10%	30%	10%	20%
	par 2	9001	7216	9081	9370

Sparrow_E stands out as the best solver and dramatically outperforms others and solves **2** times as many instances as Sparrow, CSCCSat, Dimetheus, yalsat, and WalkSATlm respectively did on the instances with $5000 \leq \#var \leq 6800$. WalkSATlm_E and Score$_2$SAT_E solve **1.7** times as many instances as WalkSATlm, Score$_2$SAT, yalsat, and Dimetheus did on the instances with $5000 \leq \#var \leq 6800$. WalkSATlm_E and CSCCSat_E get the best **40%** success rate, but WalkSATlm_E has the lowest average par 2 on the instances with $7000 \leq \#var \leq 8800$. WalkSATlm_E and Score$_2$SAT_E significantly outperform their competitors on medium-sized instances with $9000 \leq \#var \leq 10800$ and $11000 \leq \#var \leq 12800$. The improved SLS solvers reduce the average par 2 compared with the original ones on medium random 3-SAT instances at phase transition from SAT competition in 2017.

C. Comparison between the extended SLS solvers with the state-of-the-art SLS solvers on uniform random k-SAT instances from SAT Competition 2017

To investigate the performance of six improving SLS solvers on uniform random *k*-SAT instances with various *k*, we compare them with SLS solvers on SAT Competition 2017 benchmark.

Table 3: Comparative results on all uniform random k-SAT instances at and near phase transition, different r and variable numbers from SAT Competition 2017

clause-to-variable ratio		*r*<4.267	*r*=4.267	*r*<21.117	*r*=21.117	*r*<87.79	*r*=87.79	Over All
variable number		10^6	5000-12800	250000	200-590	50000	90-168	
instance type		3-SAT		5-SAT		7-SAT		180
instance number		20	40	20	40	20	40	
Probsat	suc	10%	10%	50%	35%	45%	40%	31%
	par 2	9010	9066	**5139**	9242	5506	8936	8238
Probsat_E	suc	15%	10%	45%	35%	50%	50%	33%
	par 2	**8592**	9066	5522	**9134**	**5006**	**5330**	**7353**
Sparrow	suc	45%	15%	0%	35%	55%	57%	35%
	par 2	6626	8569	-	9165	**4732**	**4994**	6858
Sparrow_E	suc	60%	20%	0%	40%	50%	53%	38%
	par 2	**5065**	**8217**	-	**6144**	5084	5072	**6504**
DCCASat	suc	0%	18%	40%	38%	45%	55%	34%
	par 2	-	8286	6244	6373	5597	**4924**	6801
DCCASat_E	suc	0%	20%	50%	40%	55%	50%	36%

	par 2	-	**8159**	**5258**	**6370**	**4769**	5248	**6655**
WalkSATlm	suc	90%	25%	45%	28%	55%	55%	45%
	par 2	1194	7630	5603	7283	4754	5196	5774
WalkSATlm_E	suc	90%	30%	55%	35%	55%	53%	48%
	par 2	**1106**	**7256**	**4636**	**6728**	**4608**	**5023**	**5374**
CSCCSat	suc	90%	20%	55%	40%	55%	48%	46%
	par 2	1242	8154	4720	6294	4778	5082	5638
CSCCSat_E	suc	90%	28%	55%	38%	55%	58%	49%
	par 2	**1117**	**7492**	**4683**	**5649**	**4760**	**4623**	**5121**
Score$_2$SAT	suc	90%	18%	45%	38%	55%	55%	46%
	par 2	1194	8286	5603	6373	4754	4924	5600
Score$_2$SAT_E	suc	90%	30%	55%	40%	60%	58%	49%
	par 2	**1106**	**7257**	**4636**	**6370**	**4118**	**4681**	**5164**
yalsat	suc	40%	18%	60%	33%	55%	43%	42%
	par 2	6117	8372	4147	6790	4520	5909	5950

Table 3 reports the average par 2 for each solver on each *k*-SAT instance class. As shown from Table 3, six improved SLS solvers show significantly better performance in terms of the success rate and the average par 2 than the original SLS solvers on the whole benchmark.

Probsat_E reduces **418** of the average par 2 and increases **5%** of a success rate than Probsat on uniform random 3-SAT with r<4.267. Although Probsat_E performs worse than Probsat on uniform random 5-SAT with r<21.117, it reduces **108** of the average par 2 than Probsat on uniform random 5-SAT with r=21.117. Moreover, Probsat_E reduces **500** of the average par 2 and increases **5%** of a success rate than Probsat on uniform random 7-SAT with r<87.79, and reduces **3606** of the average par 2 and increases **10%** of a success rate than Probsat on uniform random 7-SAT with r=87.79. Especially, Probsat_E reduces **885** of the average par 2 and increases **2%** of a success rate than Probsat on the overall results.

Sparrow_E reduces **1561** of the average par 2 and increases 1**5%** of a success rate than Sparrow on uniform random 3-SAT with r<4.267, and reduces **352** and increases **5%** on uniform random 3-SAT with r=4.267. Sparrow_E reduces **3021** and increases **5%** on uniform random 5-SAT with r=21.117. Although Sparrow_E performs worse than Sparrow on uniform random 7-SAT, it reduces **354** and increases **3%** on the overall results.

DCCASat_E performs better than DCCASat on all uniform random *k*-SAT except for uniform random 7-SAT with r=87.79 in terms of success rate and average par 2. Indeed, DCCASat_E reduces **127** and increased **2%** on uniform random 3-SAT with r=4.267, and reduces **986** and increases **10%** on uniform random 5-SAT with r<21.117, and reduces **3** and increases **2%** on uniform random 5-SAT with r=21.117, and reduces **828** and increased **10%** on uniform random 7-SAT with r<87.79. Although DCCASat_E performs worse than DCCASat on uniform random 7-SAT with r=87.79, it reduces **146** and increased **2%** on the overall results.

WalkSATlm_E performs better than WalkSATlm on all uniform random *k*-SAT except for uniform random 7-SAT with r=87.79. In fact, WalkSATlm_E solves only 2% of success rate less than WalkSATlm on uniform random 7-SAT with r=87.79. However, WalkSATlm_E reduces **88**, **374**, **967**, **555**, **146**, and **173** of the average par 2 than WalkSATlm on all uniform random *k*-SAT respectively, and reduces **400** and increases **3%** on the overall results. Moreover, WalkSATlm_E shows significantly better performance than other solvers on the whole uniform random 3-SAT and the uniform random *k*-SAT with k>3 near the phase transition.

CSCCSat_E performs better than CSCCSat on all uniform random *k*-SAT except for uniform random 5-SAT with r=21.117. Although CSCCSat_E solves only 2% of success rate less than CSCCSat on uniform random 5-SAT with r=21.117, CSCCSat_E reduces **645** of the average par 2 than CSCCSat on uniform random 5-SAT with r=21.117. Moreover, CSCCSat_E reduces **662** and increased **8%** on uniform random 3-SAT with r=4.267, and reduces **125** on uniform random 3-SAT with r<4.267, and reduces **37** on uniform random 5-SAT with r<21.117, and reduces **459** and increases **10%** on uniform random 7-SAT with r=87.79, and reduces **18** on uniform random 7-SAT with r<87.79. Especially, CSCCSat_E solves **3%** of the success rate more than CSCCSat and reduces **512** of the average par 2 than

CSCCSat on the overall results. Furthermore, CSCCSat_E shows significantly better performance than other solvers on the uniform random *k*-SAT with *k*>3 at the phase transition and on the overall results.

Score$_2$SAT_E performs better than Score$_2$SAT on all uniform random *k*-SAT. Specifically, Score$_2$SAT_E reduces **88**, **1029**, **967**, **3**, **636**, and **243** of the average par 2 than Score$_2$SAT, and solves **0%**, **12%**, **10%**, **2%**, **5%**, and **3%** of the success rate more than Score$_2$SAT on all uniform random *k*-SAT respectively. Moreover, Score$_2$SAT_E solves **3%** of the success rate more than Score$_2$SAT and reduces **436** of the average par 2 than Score$_2$SAT on the overall results. Especially, Score$_2$SAT_E shows significantly better performance than other solvers on the uniform random *k*-SAT with *k*<7 near the phase transition.

Overall, it is worth noting that Score$_2$SAT_E and CSCCSat_E solve six instances and seven instances respectively, which are not solved by all submitted solvers from the random track of SAT Competition in 2017. Table 3 also shows that CSCCSat_E solves most *k*-SAT instances for each *k*, which illustrates its robustness. These results confirm the superiority of CSCCSat_E over its competitors.

The good performance of WalkSATlm_E, CSCCSat_E, and Score$_2$SAT_E is also clearly illustrated by Fig.10 on the SAT Competition 2017 benchmark. CSCCSat_E solves more instances than all the other competitors. Overall, CSCCSat_E solves **89** instances, while the best original solver named CSCCSat solves 82 instances, indicating the efficiency of CSCCSat_E. Indeed, it solves 13 instances more than yalsat, 8 instances more than Score$_2$SAT, 6 instances more than CSCCSat, 28 instances more than DCCASat, 8 instances more than WalkSATlm, 26 instances more than Sparrow, and 34 instances more than Probsat. WalkSATlm_E and Score$_2$SAT_E are also better than SLS competitors. Score$_2$SAT_E solves 2 instances more than WalkSATlm_E, 12 instances more than yalsat, 7 instances more than Score$_2$SAT, 5 instances more than CSCCSat, 27 instances more than DCCASat, 7 instances more than WalkSATlm, 25 instances more than Sparrow, and 33 instances more than Probsat.

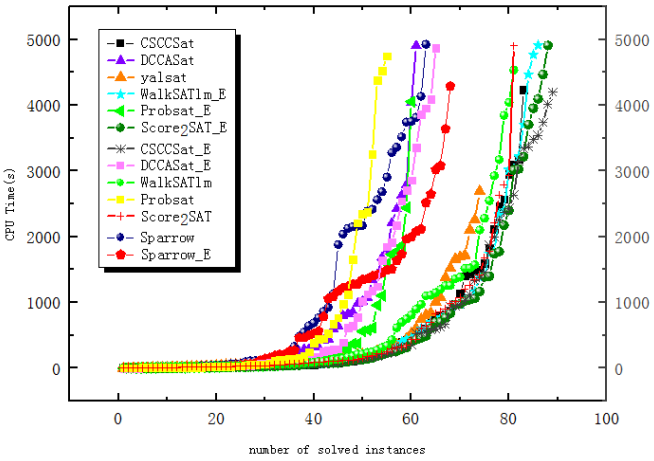

Fig. 10: Comparison of run time distributions on the SAT Competition 2017 benchmark consisting of all random 3-SAT instances with and all uniform random k-SAT instances with k > 3 with a cutoff time of 5000 seconds.

5.2.2 Results on uniform random k-SAT Competition 2016

Table 4: Comparative results on all uniform random k-SAT instances at and near phase transition, different r and variable numbers from SAT Competition 2016

clause-to-variable ratio		*r*<4.267	*r*=4.267	*r*<21.117	*r*=21.117	*r*<87.79	*r*=87.79	Over All
variable number		10^6	5000-12800	250000	200-590	50000	90-168	
instance type		3-SAT		5-SAT		7-SAT		180
instance number		60		60		60		
Probsat	par 2	9026		6789		6020		7278
Probsat_E	par 2	**8765**		**6391**		**5887**		**7016**
Sparrow	par 2	8678		7914		5682		7424

Sparrow_E	par 2	**7881**	**7890**	**5153**	**6975**
CSCCSat	par 2	6936	**6141**	5688	6255
CSCCSat_E	par 2	**6653**	6166	**5557**	**6125**
DCCASat	par 2	9744	6394	**5982**	7373
DCCASat_E	par 2	**9712**	**6104**	6014	**7277**
WalkSATlm	par 2	6932	**6019**	5569	6173
WalkSATlm_E	par 2	**6869**	6120	**5516**	**6168**
Score$_2$SAT	par 2	6932	5960	5661	6184
Score$_2$SAT_E	par 2	**6869**	**5901**	**5516**	**6095**
yalsat	par 2	5500	5950	5866	6359

Table 4 reports the average par 2 for each solver on each *k*-SAT instance class at and near phase transition, where each solver is performed 10 runs for each instance with a cutoff time of 5000 seconds.

Probsat_E performs better than Probsat on all uniform random *k*-SAT. Specifically, Probsat_E reduces **261**, **398**, and **133** of the average par 2 than Probsat on all uniform random *k*-SAT respectively. Moreover, Probsat_E reduces **262** of the average par 2 to Probsat on the overall results.

Sparrow_E performs better than Sparrow on all uniform random *k*-SAT in terms of average par 2. Specifically, Sparrow_E reduces **797**, **24**, and **529** of the average par 2 than Sparrow on all uniform random *k*-SAT respectively. Moreover, Sparrow_E reduces **449** of the average par 2 to Sparrow on the overall results.

CSCCSat_E reduces **283** on uniform random 3-SAT and **131** on uniform random 7-SAT. Although Sparrow_E performs worse than Sparrow on uniform random 5-SAT, it reduces **130** on the overall results.

DCCASat_E performs better than DCCASat on all uniform random *k*-SAT except for uniform random 7-SAT in terms of average par 2. Although DCCASat_E performs worse than DCCASat on uniform random 7-SAT, DCCASat_E reduces **32** on uniform random 3-SAT and **290** on uniform random 5-SAT. Moreover, DCCASat_E reduces **96** of the average par 2 to DCCASat on the overall results.

WalkSATlm_E performs better than WalkSATlm on all uniform random *k*-SAT except for uniform random 5-SAT in terms of average par 2. Although WalkSATlm_E performs worse than WalkSATlm on uniform random 5-SAT, WalkSATlm_E reduces **63** on uniform random 3-SAT and **53** on uniform random 7-SAT. Moreover, WalkSATlm_E shows significantly better performance than other solvers on the uniform random 7-SAT. Score$_2$SAT_E performs better than Score$_2$SAT on all uniform random *k*-SAT in terms of average par 2. Specifically, Score$_2$SAT_E reduces **63**, **59**, **145**, and **89** of the average par 2 than Score$_2$SAT on all uniform random *k*-SAT respectively. Moreover, Score$_2$SAT_E reduces **89** of the average par 2 than Score$_2$SAT on the overall results. And Score$_2$SAT_E shows significantly better performance than other solvers on the uniform random *k*-SAT with $k>3$.

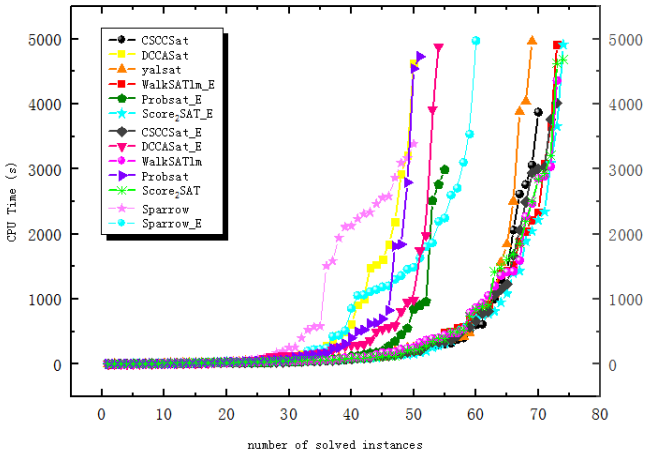

Fig. 11: Comparison of run time distributions on the SAT Competition 2016 benchmark consisting of all random 3-SAT instances with $r \leq 4.267$ and all uniform random k-SAT instances with k > 3 with a cutoff time of 5000 seconds

Overall, although yalsat has the lowest average par 2, CSCCSat_E stands out as the best solver on uniform random 3-SAT in terms of success rate. Actually, all the competitors become ineffective on the 3-SAT instances at the phase transition, while CSCCSat_E significantly reduces the average par 2 for this instance class. Score$_2$SAT_E stands out as the best solver on the uniform random *k*-SAT instance with *k*>3 in terms of average par 2. Six improved SLS solvers show significantly better performance than original solvers on the overall results, which shows its robustness, indicating the effectiveness of allocation strategy based on the initial probability distribution on the improved six SLS algorithms. The good performance of six improved SLS solvers is also clearly illustrated by Fig. 11, which summarizes the run time distributions of the solvers on the SAT Competition 2016 benchmark.

5.2.3 Results on uniform random k-SAT Competition 2018

Table 5 reports the comparative results of six improved SLS solvers and their original two plus yalsat and probSAT[7] in terms of average par 2. Table 5 indicates the better performance of the improved SLS on an instance class in bold font.

Table 5: Comparative results on all uniform random k-SAT instances at and near phase transition, different r and variable numbers from SAT Competition 2018

clause-to-variable ratio		*r*<4.267	*r*=4.267	*r*<21.117	*r*=21.117	*r*<87.79	*r*=87.79	Over All
variable number		10^6	6000	250000	250	50000	120	
instance type		3-SAT		5-SAT		7-SAT		90
instance number		30		30		30		
Probsat	par 2	**7266**		4750		4786		5600
Probsat_E	par 2	8167		**4084**		**4380**		**5543**
Sparrow	par 2	5666		7422		3294		5453
Sparrow_E	par 2	**5526**		**7410**		**3191**		**5375**
CSCCSat	par 2	3200		4203		**4207**		3903
CSCCSat_E	par 2	**3061**		**4008**		4334		**3737**
DCCASat	par 2	**9110**		4999		4972		6329
DCCASat_E	par 2	9186		**4530**		**4510**		**6075**
WalkSATlm	par 2	3392		4262		**3901**		3852
WalkSATlm_E	par 2	**3282**		**3959**		4024		**3755**
Score$_2$SAT	par 2	**3340**		4120		4266		3941
Score$_2$SAT_E	par 2	3392		**3948**		**4024**		**3756**
yalsat	par 2	4859		3425		4900		4362
probSAT	par 2	3653		3845		4043		3881

For the 3-SAT class, CSCCSat_E solves the most instances. Actually, all the competitors become ineffective on the 3-SAT instances at the phase transition, while CSCCSat_E reduces an average par 2 of **4%** on this instance class. For the 5-SAT class, although yalsat exhibits the best performance, the extended SLS solvers show a great improvement. Especially, Probsat_E reduces an average par 2 of **14%** than Probsat, and CSCCSat_E reduces an average par 2 of **5%** than CSCCSat, and DCCASat_E reduces an average par 2 of **9%** than DCCASat, and WalkSATlm_E reduces an average par 2 of **7%** than WalkSATlm, and Score$_2$SAT_E reduces an average par 2 of **4%** than Score$_2$SAT on this instance class. For the 7-SAT class, Sparrow_E exhibits the best performance. Although CSCCSat_E and WalkSATlm_E perform worse than their original versions, CSCCSat_E reduces **166** of the average par 2 than CSCCSat, and WalkSATlm_E reduces **97** of the average par 2 than WalkSATlm on the overall results. Moreover, the extended SLS solvers show significantly better performance than the original ones on the overall results. Indeed, Probsat_E reduces **57** of the average par 2 to Probsat, and Sparrow_E reduces **78** of the average par 2 to Sparrow, and DCCASat_E reduces **254** of the average par 2 than DCCASat, and Score$_2$SAT_E reduces **185** of the average par 2 than Score$_2$SAT on the overall results.

Six improved SLS solvers show significantly better performance than original solvers on the overall results, which

shows its robustness, indicating the effectiveness of allocation strategy based on the initial probability distribution on the improved six SLS algorithms.

5.3 Experimental Analyses of the allocation strategy based on the initial probability distribution

To demonstrate the effectiveness of the allocation strategy based on the initial probability distribution, we carry out extensive experiments to evaluate WalkSATlm and WalkSATlm_E on random *k*-SAT instances $k \in \{3, 5, 7\}$ at and near phase transition about the average number of the unsatisfied clauses with fixed steps and then we compare WalkSATlm_E with WalkSATlm as well as SLS solvers on the following uniform random *k*-SAT instances generated randomly with $k = 3, 5, 7$ at and near the phase transition.

1) **3SAT Huge:** 3-SAT instances generated randomly according to the random *k*-SAT model (r=4.1, n=10^6, 100 instances)
2) **5SAT Huge:** 5-SAT instances generated randomly according to the random *k*-SAT model (r=17.0, n=250000, 100 instances)
3) **7SAT Huge:** 7-SAT instances generated randomly according to the random *k*-SAT model (r=62, n=50000, 100 instances)
4) **3SAT Medium:** 3-SAT instances generated randomly according to the random *k*-SAT model (r=4.267, n=7000, 100 instances)
5) **5SAT Medium:** 5-SAT instances generated randomly according to the random *k*-SAT model (r=21.117, n=300, 100 instances)
6) **7SAT Medium:** 7-SAT instances generated randomly according to the random *k*-SAT model (r=87.79, n=160, 100 instances)

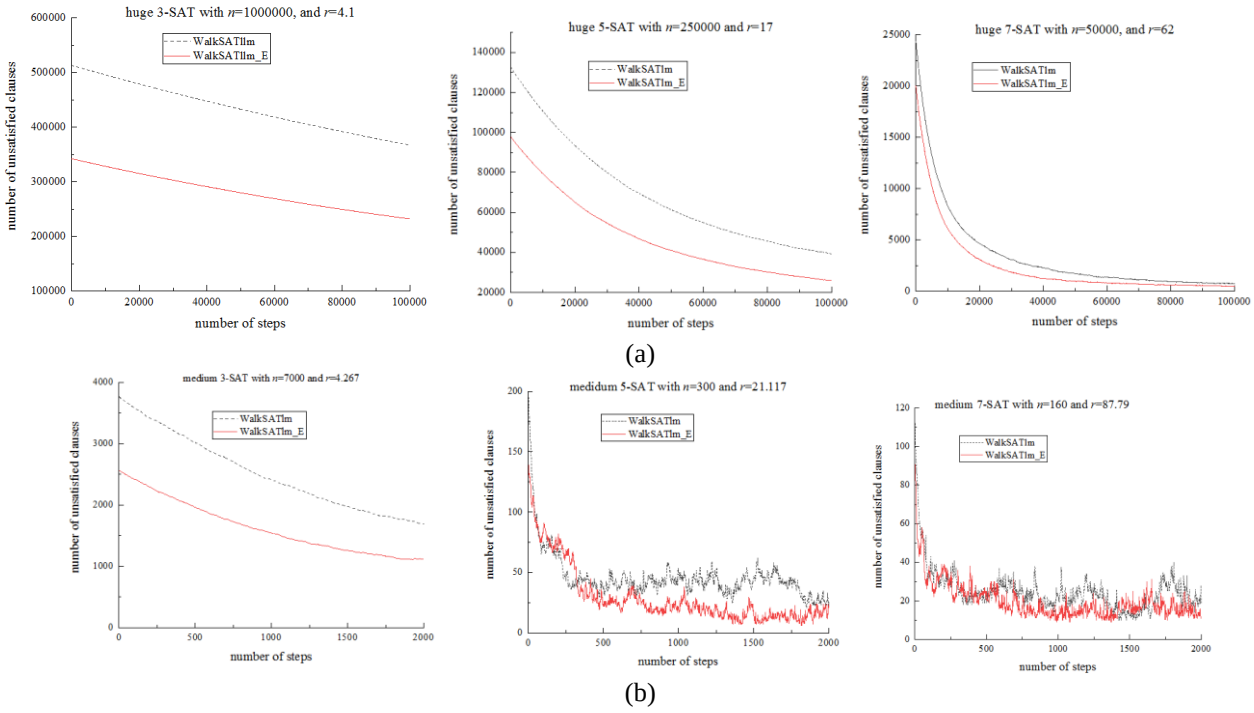

Fig. 12: **The average number of unsatisfied clauses variation plot.** The (a) plots show the comparative variation of the average number of unsatisfied clauses with 100000 steps between WalkSATlm and WalkSATlm_E on all generated uniform huge random *k*-SAT instances near the phase transition. The (b) plots show the comparative variation of the average number of unsatisfied clauses with 2000 steps between WalkSATlm and WalkSATlm_E on all generated uniform medium random *k*-SAT instances at the phase transition.

Table 6: Experimental results on the *k*-SAT Huge and Medium benchmark. There are 600 instances in each class and each solver is executed one time on each instance with a cutoff time of 2000 seconds.

clause-to-variable ratio	r=4.1	r=4.267	r=17	r=21.117	r=62	r=87.79

variable number		10^6	9000	250000	300	50000	160
instance type		3-SAT		5-SAT		7-SAT	
instance number		100	100	100	100	100	100
WalkSATlm	par 2	227	3258	61	2461	257	3344
WalkSATlm_E	par 2	**223**	**3117**	**59**	2452	252	**2901**
Dimetheus	par 2	224	3278	375	5453	119	2914
ProbSAT	par 2	380	3159	63	**1994**	**17**	3654
yalsat	par 2	417	3308	69	2469	69	3333
Score$_2$SAT	par 2	227	3154	61	2366	257	2922

The comparative results of WalkSATlm_E and WalkSATlm are displayed in Fig. 12 and Table 6. According to Fig. 12, the average number of unsatisfied clauses of WalkSATlm_E decreases faster than that of WalkSATlm within the same steps on random instances. The performance of WalkSATlm_E is significantly better than that of WalkSATlm from Table 6, which indicates that the allocation strategy based on the initial probability distribution is suitable for SLS algorithms on generating appropriate initial assignments for solving random 3-SAT, 5-SAT, and 7-SAT. More careful observations show that WalkSATlm_E stands out as the best solver and significantly performs better than its competitors on random 3-SAT instances, 5-SAT instances with r=17, and 7-SAT instances at the phase transition. Although WalkSATlm_E performs worse than ProbSAT on random 5-SAT instances at the phase transition and random 7-SAT instances with r=62, WalkSATlm_E does show better performance than Dimetheus (it is the first place among the SLS algorithms in SAT Competition 2018 and the winner of random satisfiable track in SAT Competition 2016) on random 5-SAT instances at the phase transition.

6 Parameter Tuning and Analyses on Random 3-SAT for SAT competition

In this section, we report results of a large-scale experiment on random 3-SAT instances from SAT competitions in 2016, 2017, and 2018. Based on the concept of the initial probability distribution, we propose a new method that easy and hard an instance class (for a particular SLS SAT procedure, anyway) are predictable in advance.

6.1 Brief about Parameter Tuning and Analyses on 3-SAT Instances

Phase transition[27] is an important feature of SAT instances. For most algorithms, the formula generated closer to the phase-transition ratio are harder to solve. The hard and easy distribution was achieved by large-scale experiments in satisfiability testing based on completed DP. In the current years, random SAT instances are solved mainly by SLS solvers, while complete solvers have no advantage for solving random instances at and near the phase transition. This motivates our work toward predicting the hardness of random 3-SAT instances by large-scale experiments in satisfiability testing based on SLS solvers.

SLS solvers are good choices for two reasons. Firstly, they are effective and popular methods in random SAT problems. Secondly, almost all empirical work on random SAT instances has used one or another refinement of these methods, which facilitates comparison. Besides, our experiment is not based on the clause-to-variable ratio of random SAT instances as a criterion for predicting instances, but the initial probability distribution of random SAT instances as a criterion for predicting instances.

6.2 Benchmarks and Experiment Preliminaries

In this section, we only consider different initial probability distributions of 3-SAT instances and focus on analyzing an instance class that is easy or hard in SAT competitions. All benchmarks are standard based on all random 3-SAT instances from SAT competitions in 2016, 2017, and 2018. Although 3-SAT instances of different sizes have different variables, we mainly take the initial probability distribution as the criterion. Specifically, we adopt the following six benchmarks:

1) **3-SAT r=4.3:** random 3-SAT instances, from SAT Competition in 2017 ($400 \le n \le 540$, 80 instances, 10 for each

size).

2) **3-SAT r=5.5:** random 3-SAT instances, from SAT Competition in 2017 ($400 \le n \le 540$, 80 instances, 10 for each size).
3) **3-SAT $5.205 \le r \le 5.206$:** random 3-SAT instances, from SAT Competition in 2017 ($400 \le n \le 540$, 80 instances, 10 for each size).
4) **3-SAT r=4.267:** random 3-SAT instances, from SAT Competitions in 2016, 2017 ($5000 \le n \le 12800$, 80 instances, 10 for each size) and 2018 (n =6000, 10 instances).
5) **3-SAT $3.86 \le r \le 4.24$:** random 3-SAT instances, from SAT Competitions in 2016, 2017 and 2018 (n=10^6, 60 instances, 3 for each size).
6) **3-SAT $4.40<r<4.42$:** random 3-SAT instances, from SAT Competition in 2016 ($350 \le n \le 600$, 60 instances, 10 for each size).

The size of random 3-SAT instances with $r \ge 4.3$ from the random track of SAT Competition 2018 is too small for experimental comparison. To make the gap of all variable sizes as small as possible, these instances are not included in our experiments.

The parameters of the initial probability distribution are also *pad* and *nad*. In order to make the results comparable, let *pad* = 1.8 and *nad* =0.56 (Here the parameters can be set casually except that they cannot be set to zero at the same time, but the parameters of all the instances must be set the same). The computing environments for these experiments are the same as those used for experiments in the Section 5. The initial probability distributions of different instances classes are reported in Table 7. If there is no instance class in the random track of SAT competition, the corresponding initial probability distribution is marked with "-".

Table 7: The initial probability distribution on random 3-SAT instances of different ratios from SAT competitions in 2016, 2017 and 2018.

ratio	r=5.5	$5.205 \le r \le 5.206$	$4.40<r<4.42$	r=4.3	r=4.267	$3.86 \le r \le 4.24$
2016	-	-	0.336	-	0.328	0.34
2017	0.351	0.299	-	0.377	0.328	0.34
2018	0.354	0.294	-	0.378	0.328	0.34
Over all	0.353	0.296	0.336	0.377	0.328	0.34

6.3 Experimental Results

In this section, we present the relationship between the initial probability distributions and hard and easy distributions (average success rate) of solving random 3-SAT instances from the random track of SAT Competitions in 2016, 2017, and 2018 by yalsat, Score$_2$SAT, CSCCSat, WalkSATlm, and Probsat. Each solver performs 20 runs on each instance. The cutoff time is set to 5000 seconds.

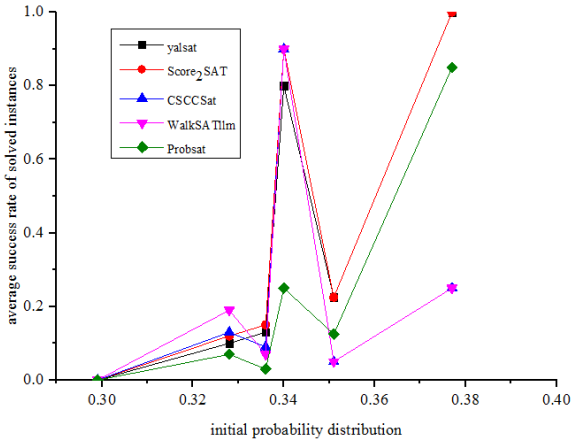

Fig. 13: The distributions between Average success rate of solved instances and the initial probability distributions on the random 3-SATinstances from SAT Competitions in 2016, 2017, 2018. Each solver is executed 20 runs on each instance with a cutoff time of 5000 seconds

The results are presented in Fig 13. The variation intervals of all variables are almost the same under different initial probability distributions except that the initial probability distributions are 0.34 and 0.328 (according to Section 6.2). Fig 13 gives a rough description of the hard and easy distribution based on the initial probability distributions on random 3-SAT instances from the random track of SAT competitions.

It can be seen from Fig. 13 that the trend of curves under different solvers is almost identical. Although the variables of four classes of satisfiable instances with an initial probability distribution of about 0.298, 0.328, 0.336 and 0.351 are much smaller than the variables of satisfiable instances with an initial probability distribution of about 0.34, the success rate of four classes of instances is far less than the class of instances with an initial probability distribution of 0.34. Although the variables of the class of satisfiable instances with an initial probability distribution of 0.328 are larger than the variables of the class of instance with an initial probability distribution of about 0.298, the success rate of the class of satisfiable instance is larger than the variables of the class of instance with an initial probability distribution of about 0.298 for SLS solvers. Because the variables of the class of satisfiable instances with an initial probability distribution of 0.377 are much smaller than 10^6, we cannot summarize that the success rate of the class of instances is better than that of the class of instances with the initial probability distribution of 0.34. The variables of three classes of satisfiable instances with an initial probability distribution of 0.299, 0.351, and 0.377 have the same variables. We conjecture that (1) since the class of satisfiable instances with an initial probability distribution of 0.377 is under-constrained for SLS solvers, an assignment is easy to be found, (2) the class of satisfiable instances with a small initial probability distribution of 0.299 are over-constrained for SLS solvers, and thus an assignment is hard to be found in the search. It is possible that our results on hard and easy areas can generalize to all SAT procedures, but this remains to be seen.

7 Conclusions and Further work

In this paper, we proposed six improved SLS solvers by determining the initial assignments of variables in a controlled way on solving uniform random *k*-SAT instances. The resulting six SLS algorithms named Score$_2$SAT_E, CSCCSat_E, WalkSATlm_E, DCCASat_E, Probsat_E, and Sparrow_E are evaluated through benchmarks in terms of their capabilities and efficiency when treating uniform random *k*-SAT instance with medium and huge variables, e.g., benchmarks from the random track of SAT Competitions in 2016, 2017 and 2018 respectively. Experimental results showed that these improved SLS solvers outperform their original versions.

The method of determining the initial assignments of variables in a controlled way is based on two concepts: the allocation strategy and initial probability distribution. We derived the allocation strategy, which is a greedy mode to improve random initial assignment from random 3-SAT to uniform random *k*-SAT. The allocation strategy is used to generate a greedy initial assignment instead of a random initial assignment on solving *k*-SAT instances. The idea of this strategy is utilized to guide the trend of optimal truth assignment in advance and to accelerate the finding of the optimal solution. Then we proposed the initial probability distribution, which is devoting to tune the parameters of the allocation strategy. The allocation strategy and initial probability distribution were used to develop Score$_2$SAT_E, CSCCSat_E, WalkSATlm_E, DCCASat_E, Probsat_E, and Sparrow_E algorithms.

Experiments on uniform random *k*-SAT instances at and near phase-transition threshold from the random track of SAT competitions in 2016, 2017, and 2018 show that Score$_2$SAT_E, CSCCSat_E, and WalkSATlm_E significantly improve Score$_2$SAT, CSCCSat, and WalkSATlm respectively and outperform state-of-the-art SLS algorithms, and Probsat_E and Sparrow_E significantly have a great improvement for Probsat and Sparrow on uniform random *k*-SAT instances respectively, which show the generality and robustness of the proposed method in determining initial assignment effectively to significantly improve the performance of some state-of-the-art solvers for uniform random *k*-SAT instances.

Moreover, according to extensive random instances generated, we analysis the variation of the average number of unsatisfied clauses between WalkSATlm_E and WalkSATlm with the same steps and compared WalkSATlm_E with state-of-the-art SLS algorithms. Experiments on random *k*-SAT instances show that WalkSATlm_E significantly improves WalkSATlm and outperforms state-of-the-art SLS algorithms on random 3-SAT instances, 5-SAT instances near the phase transition, and 7-SAT instances at the phase transition.

Further, the initial probability distribution of each class of instances and the success rate of solvers are taken as the objects of analysis. According to the extensive experiments based on the state-of-the-art SLS solvers on random 3-SAT instances of the random track of SAT competitions in 2016, 2017, and 2018, we obtain a new hard and easy distribution as initial probability distribution changes.

As for future work, we plan to apply our method to other SLS algorithms, especially those focused on random walk ones. Furthermore, the notions in this work are simple that they can be easily applied to other problems, such as constrained satisfaction and graph search problems.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant No.61673320) and the Fundamental Research Funds for the Central Universities (Grant No.2682019ZT16, 2682020CX59).

Data Availability Statement Information

All data generated or analyzed during this study are included in this article.

References

1. Abramé, A., Habet, D., & Toumi, D. (2014). Improving configuration checking for satisfiable uniform random k-SAT instances. In: Pro. of ISAIM-14, pp. 5-24.
2. Achlioptas, D. (2009). Random satisfiability. In Handbook of Satisfiability, 245–270.
3. Balint A. and Fröhlich A. (2010). Improving Stochastic Local Search for SAT with a New Probability Distribution. In: Pro. of SAT-10, pp. 10-15.
4. Balint, A., & Schöning, U. (2012). Choosing probability distributions for stochastic local search and the role of make versus break. In: Pro. of SAT-12, pp. 16-29.
5. Balint, A., Schöning, U. (2016). Engineering a lightweight and efficient local search sat solver. from book Algorithm Engineering: Selected Results and Surveys, pp. 1-18.
6. Balint, A. (2018). Uniform random k-SAT q-planted solutions: Benchmark Descriptions. In: Pro. of SAT- 2018, pp. 64.
7. Balint, A., & Schöning, U. (2018). probSAT: Solver and Benchmark Descriptions. In: Pro. of SAT- 2018, pp. 35.
8. Balint, A., & Manthey, N. (2013) "Boosting the Performance of SLS and CDCL Solvers by Preprocessor Tuning," in Pragmatics of SAT, 2013.
9. Balyo, T. (2016). Using algorithm configuration tools to generate hard random satisfiable benchmarks. In Pro. of SAT-2016, 60–62.
10. Biere, A. (2017). CADICAL, LINGELING, PLINGELING, TREENGELING and YALSAT: Solver description. In: Pro. of SAT-2017, pp. 14-15.
11. Cai, S., Su, K., Sattar, A. (2011) Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175, 1672–1696.
12. Cai, S., Luo, C. & Su, K. (2012). CCASat: Solver description. In: Pro. of SAT- 2012, pp. 13-14.
13. Cai, S., & Su, K. (2013). Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence*, 204, 75-98.
14. Cai, S. & Su, K. (2013). CCAnr : Solver description. In: Pro. of SAT- 2013, pp. 16-17.
15. Cai, S. and Su, K. (2013). Comprehensive Score: Towards Efficient Local Search for SAT with Long Clauses. In: Pro. of IJCAI 13, pp. 489-495.
16. Cai, S., Su, K. and Luo, C. (2013) Improving WalkSAT for Uniform random k-SAT isfiability Problem with k > 3. Proc. In: Pro. of AAAI-13, pp. 145-151.
17. Cai, S., Su, K., Luo, C., & Sattar, A. (2013). NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46, 687–716
18. Cai, S., Luo, C., & Su, K. (2014). Scoring functions based on second level score for *k*-SAT with long clauses. *Journal of Artificial Intelligence Research*, 51, 413-441.
19. Cai, S.; Luo, C.; and Su, K. (2014). New scoring functions for uniform random k-SAT with long clauses. Technical report. *http://shaoweicai.net/Paper/new-scoring-functions.pdf*.
20. Gableske, O. (2016), dimetheus: Solver description. In: Pro. of SAT- 2013, pp. 37-38.

21. Cai, S., Luo, C., & Su, K (2015). Improving WalkSAT by effective tie-breaking and efficient implementation. *The computer Journal*, 58, 2864-2875.
22. Cai, S. & Luo, C. (2017). Score$_2$SAT: Solver description. In: Pro. of SAT-2017, pp. 34.
23. Abramé, A., Habet, D., & Toumi, D. (2017). Improving configuration checking for satisfiable uniform random k-SAT instances. *Annals of Mathematics and Artificial Intelligence*, 79(1-3), pp.5-24.
24. Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM* (*JACM*), 7, 201-215.
25. Fu H. M., Xu Y., Wu G. F. & Ning X. R. (2017). An Improved Genetic Algorithm for Solving 3-SAT Problems Based on Effective Restart and Greedy Strategy. In: Pro. of ISKE-17, pp. 1-6.
26. Fu H. M., Xu Y., He X. X. & Ning X. R. (2018). GSAT Algorithm Based on Task Allocation and Scheduling for 3-SAT Problem. Chinese Journal of Computer engineering& Science, 40, 1366-1374.
27. Gent, I.P. and Walsh, T. (1994). The SAT phase transition. *In ECAI. PITMAN*, pp. 105-109,
28. Hoos, H. H. (2002). An adaptive noise mechanism for WalkSAT. In: Pro. of AAAI-02, pp. 655-659.
29. Hoos, H. H., & Stützle, T. (2004). Stochastic local search: Foundations and applications. *Elsevier*.
30. Kautz, H. A.; Sabharwal, A.; & Selman, B. 2009. Incomplete algorithms. In *Handbook of Satisfiability*, pp. 185–203.
31. Kroc, L., Sabharwal, A., & Selman, B. (2010). An empirical study of optimal noise and runtime distributions in local search. In: Pro. of SAT-10, pp. 346-351.
32. Liang, J. H., Oh, C., Ganesh,V., Czarnecki, H. & Poupart, P.(2017). MapleCOMSPS LRB VSIDS and MapleCOMSPS CHB VSIDS. In: Pro. of SAT-2017, pp. *20-21*.
33. Li, C. M., Huang, C., & Xu, R. (2014). Balance between intensification and diversification: a unity of opposites. In: Pro. of SAT-2014, pp. 10-11.
34. Luo, C., Cai, S., Wu, W., & Su, K. (2013). Focused random walk with configuration checking and break minimum for satisfiability. In International Conference on Principles and Practice of Constraint Programming, pp. 481-496.
35. Luo, C., Cai, S., Wu, W., & Su, K. (2014). CSCCSat2014: Solver description. In: Pro. of SAT-2014, pp. 25-26.
36. Luo, C., Cai, S., Wu, W., & Su, K. (2016). CSCCSat2014: Solver description. In: Pro. of SAT-2016, pp. 10-11.
37. Luo, C., Cai, S., Wu, W., & Su, K. (2014). Double configuration checking in stochastic local search for satisfiability. In: Pro. of AAAI-14, pp. 2703-2709.
38. Luo, C., Cai, C, Su, K. & Wu, W. (2015). Clause states based configuration checking in local search for satisfiability. *IEEE Transactions on Cybernetics*, 45, 1014–1027.
39. Luo, M., Li, C. M., Xiao, F., Manyà, F., & Lü, Z. (2017). An effective learnt clause minimization approach for CDCL SAT solvers. In: Pro. of AAAI-17, pp. 703-711.
40. Luo, C. Su, K. & Cai, S. (2012). Improving local search for random 3-SAT using quantitative configuration checking. In: Pro. of ECAI 2012, pp. 570-575.
41. McAllester, D., Selman, B. & Kautz, H. (1997). Evidence for invariants in local search. In: Pro. of AAAI-97, pp. 321-326.
42. Mitchell, D. G., Selman, B., & Levesque, H. J. (1992). Hard and Easy Distributions of SAT Problems. In: Pro. of AAAI-92, pp. 459-465.
43. Pham, D.N., Thornton, J.R., Gretton, C. & Sattar, A. (2007). Advances in local search for satisfiability. In: Pro. of AAAI -07, pp. 213-222.
44. Russell, R., & Holden, S. B. (2016). Survey propagation applied to weighted partial maximum satisfiability. *No. UCAM-CL-TR-883*. *University of Cambridge, Computer Laboratory*.
45. Selman, B., Kautz, H.A.& Cohen, B. (1994). Noise strategies for improving local search. In: Pro. of AAAI -94, pp. 337-343.
46. Selman, B., Levesque, H. J. & Mitchell, D. G. (1992). A New Method for Solving Hard Satisfiability Problems. In: Pro. of AAAI-92, pp. 440-446.
47. Thornton, J.; Pham, D. N.; Bain, S.; and Ferreira Jr., V. (2004). Additive versus multiplicative clause weighting for SAT. In: Pro. of AAAI -04, pp. 191-196.
48. SAT Competition 2016. https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=downloads
49. SAT Competition 2017. https://baldur.iti.kit.edu/sat-competition-2017/index.php?cat=downloads
50. SAT Competition 2018. http://sat2018.forsyte.tuwien.ac.at/index689b.html?cat=downloads
51. Uniform random k-SAT generator. https://sourceforge.net/projects/ksat generator/
52. Balint, A., Biere, A., Fröhlich, A., & Schöning, U. (2014). Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses. In: Pro. of SAT- 2014, pp. 302-316.