Journal of Universal Computer Science, vol. 26, no. 2 (2020), 220-243
submitted: 12/2/19, accepted: 27/2/20, appeared: 28/2/20 CC BY-ND 4.0

Improving WalkSAT for Random 3-SAT Problems

Huimin Fu
(School of Information Science and Technology, Southwest JiaoTong University
Chengdu 610031, China
fuhm6688@qq.com)

Yang Xu and Shuwei Chen *
(System Credibility Automatic Verification Engineering Lab of Sichuan Province
School of Mathematics, Southwest Jiaotong University, Chengdu 610031, China
xuyang@swjtu.edu.cn and swchen@home.swjtu.edu.cn)

Jun Liu
(School of Computing, Ulster University, Northern Ireland, UK
j.liu@ulster.ac.uk)

Abstract: Stochastic local search (SLS) algorithms are well known for their ability to efficiently find models of random instances of the Boolean satisfiability (SAT) problems. One of the most famous SLS algorithms for SAT is called WalkSAT, which has wide influence and performs well on most of random 3-SAT instances. However, the performance of WalkSAT lags far behind on random 3-SAT instances equal to or greater than the phase transition ratio. Motivated by this limitation, in the present work, firstly an allocation strategy is introduced and utilized in WalkSAT to determine the initial assignment, leading to a new algorithm called WalkSATvav. The experimental results show that WalkSATvav significantly outperforms the state-of-the-art SLS solvers on random 3-SAT instances at the phase transition for SAT Competition 2017. However, WalkSATvav cannot rival its competitors on random 3-SAT instances greater than the phase transition ratio. Accordingly, WalkSATvav is further improved for such instances by utilizing a combination of an improved genetic algorithm and an improved ant colony algorithm, which complement each other in guiding the search direction. The resulting algorithm, called WalkSATga, is far better than WalkSAT and significantly outperforms some previous known SLS solvers on random 3-SAT instances greater than the phase transition ratio from SAT Competition 2017. Finally, a new SAT solver called WalkSATlg, which combines WalkSATvav and WalkSATga, is proposed, which is competitive with the winner of random satisfiable category of SAT competition 2017 on random 3-SAT problem.

Keywords: 3-SAT; genetic algorithm; ant colony algorithm; WalkSAT; allocation strategy
Categories: D.1.6

1 Introduction

In computational complexity theory, the Cook-Levin theorem named after Cook [Cook, 1971] and Levin [Levin, 1984] states that Boolean satisfiability problem

* Corresponding author. Shuwei Chen, National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Southwest Jiaotong University, Chengdu, China. E-mail: swchen@home.swjtu.edu.cn

(SAT) is NP-complete. Thus, it is of great significance to find efficient SAT algorithms and apply them to engineering practice, which can improve productivity and promote social development [Marques-Silva, 2008; Xu, 2013].

SAT is the problem of deciding if there is an assignment for the variables in a propositional formula that makes the formula true [Chao, 1986; Faizullin, 2013, Zhang, 2015]. 3-SAT is a special case of SAT problem to target the Boolean formula of a particular form. There are many approaches to solve the SAT problem, which are mainly divided into two categories: one is complete, the other is stochastic local search (SLS) algorithms. Although the SLS algorithms are typically incomplete in the sense that they cannot prove an instance to be unsatisfiable, they often find solutions rather effectively [Cai, 2013a]. Moreover, some SLS algorithms are more effective than the state-of-the-art complete solvers on random 3-SAT problem.

SLS algorithms start by randomly generating a truth assignment of the variables of formula. Then it explores the search space to minimize the number of falsified clauses. To do this, it iteratively adopts some heuristics to select a variable to be flipped until it seeks out a solution or timeout. Genetic algorithm [Li, 2016; Canisius, 2016] and ant colony algorithm [Gao, 2007; Youness, 2015; Fu, 2018b] with global search are typically incomplete algorithms for solving SAT problem. Compared with the SLS algorithms, the incomplete algorithms with global search have a high time complexity, thus they are not widely utilized for solving SAT problem.

A family of SAT instances includes uniform random *k*-SAT [Achlioptas, 2009] and hard random SAT [Balyo, 2016]. In the last two decades, most SLS algorithms focus on solving uniform random *k*-SAT instances, refer to e.g., [Selman, 1994; Hoos, 2002; Kroc, 2010; Luo, 2012; Balint, 2012; Luo, 2013; Cai, 2017; Biere, 2017]. Moreover, substantial progress has been made in solving uniform random *k*-SAT with various clause-to-variable ratios.

Note that the hard instances of NP-hard problems are often associated with a phase transition. With SAT, there is a phase transition between satisfiability and unsatisfiability as the ratio of the number of clauses to variables in a problem is varied. The phase transition for SAT is therefore of considerable practical and theoretical importance. Solving hard random SAT remains a great challenge for all SLS algorithms including Dimetheus [Gableske, 2016], YalSAT [Biere, 2017] and $Score_2SAT$ [Cai, 2017]. Although uniform random *k*-SAT at the phase transition has been cited as the hardest track of SAT problems [Cai, 2013b; Luo, 2014], hard random SAT is even harder for SLS solvers. The main motivation for hard random SAT generated is to evaluate and improve SAT solvers (especially for SLS solvers) [Balyo, 2016]. It is worth noting that the hard random SAT problem is focused on 3-SAT instances greater than the phase transition ratio. Especially, most (nearly 65% of) instances on the benchmark of the random SAT track in SAT Competition 2018 are hard random SAT. Moreover, this direction has been a mainstream of SLS algorithms for SAT, which is witnessed by SAT competitions, where the instances of random track of recent SAT Competitions are composed of uniform random *k*-SAT instances and hard random SAT instances. However, the performance of existing SLS algorithms on hard random SAT instances is still rather unsatisfactory.

Among SLS algorithms for SAT problems, WalkSAT [Selman, 1994] stands out as one of the most influential algorithms. Moreover, extensive experiments have shown that the technique of WalkSAT is very suitable for random 3-SAT problem

[Liang, 1998]. Therefore, many scholars develop SLS algorithms based on WalkSAT for random SAT problem [Kroc, 2010; Luo, 2015]. For example, some researchers optimize WalkSAT algorithm to solve random k-SAT instances with k>3 [Luo, 2012; Luo, 2013]. However, the performance of WalkSAT lags far behind on random 3-SAT instances greater than the phase transition ratio, and the performance of WalkSAT at the phase transition still needs to be improved.

This present work aims to improve WalkSAT for random 3-SAT instances. One improvement is based on the allocation strategy, which was first introduced and utilized in [Fu, 2018a] to improve a greedy local search algorithm GSAT [Selman, 1992], resulting in an efficient local algorithm called AS, whose performance far exceeds GSAT for solving 3-SAT from STALAB library. The main advantage of allocation strategy is to handle the cycling problem. A combination of GSAT and genetic algorithm (as a global search algorithm), named GAGR [Fu, 2017], was proposed for solving 3-SAT instances with some competitive performance. Although WalkSAT is competitive with the state-of-the-art solvers for solving random 3-SAT problem, it is a SLS algorithm and easy to fall into the cycling problem. In addition, it is also easy to get stuck in the local optimum. This motivates us to optimize WalkSAT using the global search. Thus, some global search algorithms, e.g., genetic algorithm and ant colony algorithm, are considered to be incorporated into WalkSAT to further improve its performance on random 3-SAT instances equal to and greater than the phase transition ratio.

The remainder of our paper is organized as follows. Section 2 provides some preliminary definitions and notations, followed by a brief review of the allocation strategy for 3-SAT, the WalkSAT algorithm, genetic algorithm and ant colony algorithm. Section 3 summarizes main contributions of the present work. Section 4 proposes an improved WalkSAT algorithm using the allocation strategy, called WalkSATvav, along with its experimental results and analysis. Section 5 presents a new algorithm called WalkSATga, which utilizes a combination of an improved genetic algorithm and an improved ant colony algorithm into WalkSATvav, along with its experimental results and analysis. In Section 6, a combined algorithm called WalkSATlg is proposed by combining WalkSATvav and WalkSATga, and its performance is demonstrated by summarizing and analyzing the experimental results on random 3-SAT instances for SAT Competition 2017. Section 7 discusses the main differences between WalkSATvav and *AS* as well as the major differences between WalkSATga and GAGR. Finally, Section 8 concludes the paper with some future directions discussed.

2 Related Works

2.1 Some Basic Definitions and Notations

The symbol x_i, $i \in \{1, 2, ..., n\}$ represents a Boolean variable. Let $X_n = \{x_1, x_2, ..., x_n\}$ symbolizes a collection of Boolean variables, where the number of variables is denoted as n. Boolean variable x_i or the negation of Boolean variable $\neg x_i$ represents literal l_i, $i \in \{1, 2, ..., n\}$. A clause c_i is a disjunction of some literals, i.e., $c_i = l_1 \vee l_2 \vee ... \vee l_k$. A conjunctive normal form (CNF) F can be described as conjunction of some clauses, i.e., $F = c_1 \wedge c_2 \wedge ... \wedge c_m$, where the number of clauses in F is denoted as

m. If each clause contains *k* literals in a CNF formula, then the formula is a *k*-SAT problem. In this paper, we are only concerned with the 3-SAT problem. In a formula *F* we use *r=m/n* to denote clause-to-variable ratio of *F*, and p_{x_i} is the total number of the literal x_i appearing in *F*, and n_{x_i} is the total number of the literal $\neg x_i$ appearing in *F*.

The value of Boolean variable is either true or false. In this paper, 0 denotes false, and 1 denotes true. A mapping μ: $X_n \rightarrow \{0, 1\}$is called a complete assignment. Given a CNF formula *F*, the SAT problem is to decide whether all clauses are satisfied by a complete assignment in *F*. The *break* of a variable *x* is the number currently satisfied clauses that would become unsatisfied by flipping variable *x*. The *score* of a variable *x* is the increment of currently satisfied clauses by flipping variable *x*.

2.2 Allocation Strategy for Random 3-SAT Instances

The idea of allocation strategy [Fu, 2018a] is to determine a complete assignment as the initial solution for solving 3-SAT problem. It can guide the optimal assignment and accelerate to find the optimal solution.

Definition 1. *Given a CNF formula F, the Variable allocation degree Vad*(x_i) *of a variable* $x_i \in X_n$ *is defined as*:

$$2-1) \quad Vad(x_i) = \begin{cases} pad + 1, & n_{x_i} = 0 \\ p_{x_i} / n_{x_i}, & otherwise \end{cases}$$

where pad is a positive parameter greater than 1.

Note that for a variable x_i it is called the *positive Vad* of x_i if $Vad(x_i) \geq 1$; and it is called the *negative Vad* of x_i if $Vad(x_i) < 1$. In solving the 3-SAT problem, different settings of the parameter *pad* have a direct impact on the performance of the algorithm.

Definition 2. *Given a CNF formula F, and a random function* $\chi(x_i)$ *that only produces* 0 *or* 1, *the Variable allocation value Vav*(x_i) *of a variable* $x_i \in X_n$ *is defined as*:

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

where $\chi(x_i)$ *is a random number that can produce 0 and 1 and nad is a positive parameter less than 1.*

In fact, *Vav* of all variables is a complete assignment as the initial solution for solving a 3-SAT problem.

Remark: A variable x_i satisfies the allocation strategy if and only if $Vav(x_i) > pad$ or $Vav(x_i) < nad$.

2.3 WalkSAT Algorithm

WalkSAT is one of the most influential SLS algorithms for SAT. Its framework has been widely used, and it is still competitive with the state-of-the-art solvers for solving random 3-SAT instances. However, according to the experimental results of

SAT Competition 2017[1], the success rate of all solvers participating in the SAT Competition 2017 is still relatively small for solving the 3-SAT instances with r=4.267.

First, WalkSAT algorithm chooses a clause C randomly from the unsatisfied clauses. If there exist variables whose *breaks* are 0, one of such variables is flipped randomly; and if no such variable exists, then with a certain probability p (the noise parameter), WalkSAT algorithm selects a variable randomly from the clause C; otherwise, WalkSAT selects a variable with the minimum *break* to be flipped, and further breaks ties randomly.

2.4 Genetic Algorithm Overview

Genetic algorithms (GA) for solving 3-SAT problems mainly include three aspects [Fu, 2017]: problem transformation, chromosome encoding and genetic manipulation design. The core of problem transformation is how to define the fitness function f. Thus, SAT problems are transformed into an optimization problem of the corresponding fitness function. Using the binary string to represent a complete assignment is the most intuitive chromosome coding approach, which takes full advantage of the characteristics of SAT problems and is easy to calculate the fitness function and design a variety of genetic operations. For the SAT problems with n variables, the chromosome is represented by n as a binary string, which is directly corresponding to the assignment of the variables.

There are three kinds of genetic manipulation [any reference about GA]: selection operation, crossover operation and mutation operation. The selection operation is utilized to select groups of contemporary individuals and prepare for breeding the next generation. Crossover operation is used to the process of simulating biological reproduction by exchanging parts of two individual chromosomes and generate two new individuals. Since chromosomes are encoded by binary code, crossover operation can be accomplished by truncating and stitching binary strings. Mutation operation is adopted to simulate the mutation of a chromosome gene in the biological evolution and flip each chromosome at a certain probability, i.e. 0→1, 1→0.

As GA searches for different positions of the solution space, it significantly reduces possibility of getting into the local optimum. GA repeat these operations until the termination condition is met. The termination condition is to obtain a satisfiable solution or achieve the maximum operations. The pseudo code of GA for SAT problems is given in Algorithm 1.

GA has the following advantages: (1) the ability with a global search and independent of the problem domain; (2) search from group with potential parallelism and multi-value comparison as well as robustness; (3) the search using evaluation function enlighten and simple process; (4) using probability mechanism to iterate; (5) extensible and easy to combine with other algorithms. The disadvantage of GA is that it is not enough to make use of the feedback information in the system [Li, 2003]. In this paper, the improved GA for solving 3-SAT problems is based on the restart and greedy strategy [Fu, 2017].

[1] https://baldur.iti.kit.edu/sat-competition-2017/results/random.csv.

```
Algorithm 1: GA(F)
 Input: CNF-formula F
 Output: A satisfying assignment σ of F, or "no solution found"
 begin
1   t ←0 ;// t represents evolutionary generations
2   initialize (P(t)); //Initial population
3   evaluate (P(t)); // Fitness evaluation
4   keep_best (P(t));// Preserving the most chromosomes
5   while (not terminate condition) do
6    begin
7       P(t) ← secletion (P(t));// selection operation
8       P(t) ←crossover (P(t));// crossover operation
9       P(t) ←mutation (P(t));// mutation operation
10      t←t+1;
11      evaluate (P(t));
12      if( fitness of P(t)>best fitness)
13            Replace (best);// Replacing best with the best chromosome of P(t)
14   end
15 end
```

2.5 Ant Colony Algorithm Overview

Through long-term research, bionic scientists found that although ants have no vision, they can find a path by releasing pheromones on the path. The ant will release information about the path length on the path that is passed. The more information the path has, the more probability the path is selected. In this way, the role of pheromone makes the behavior of entire ant colony highly self-organizing. The ants exchange path information and eventually find the optimal path based on ant colony behavior. The fitness formula of each ant is the same as the fitness formula of each chromosome in GA.

The early success of ant colony algorithms (ACA) is to solve the famous TSP problem, and ant colony algorithms have excellent performance in solving various NP-hard problems [Wang, 2012]. ACA has the following advantages: (1) its principle is an enhanced learning system, which eventually converges to the optimal path through the continuous updating of pheromones; (2) it has a common property of SLS approaches, but the artificial ants are not a simple simulation of real ants, which is integrated into human intelligence; (3) it is a distributed optimization method, and not only suitable for the present serial computer, but also suitable for the future parallel computer; (4) it is a global optimization method, which not only can be used to solve the single objective optimization problem, but also can be used to solve the multi-objective optimization problem.

Disadvantage of ACA is lack of initial pheromone and low efficiency [Li, 2003]. In this paper, an improved ant colony algorithm is used to solve the 3-SAT problem, which is presented in Section 5.

3 Main Contributions

As the first contribution, the present work is to use the allocation strategy to generate a complete assignment as the initial solution for WalkSAT. Note that allocation strategy is based on *Vad* and *Vav*. *Vad* of a variable x is determined by the ratio of the numbers of positive literal x and negative literal $\neg x$ in a SAT instance, and *Vav* of a variable x is obtained according to *Vad* of x. The allocation strategy for SAT is utilized in the present work to improve WalkSAT, resulting a new algorithm called WalkSATvav. To demonstrate the effectiveness of WalkSATvav, we compare it with many state-of-the-art solvers on random 3-SAT instances at the phase transition. WalkSATvav outperforms YalSAT [Biere, 2017] (the winner of SAT Competition 2017), Score$_2$SAT [Cai, 2017] (won the bronze of the random track of SAT Competition 2017), CSCCSat [Luo, 2016] (won the silver of the random track of SAT Competition 2016), and DCCAlm [Luo, 2016] (won the bronze of the random track of SAT Competition 2016). However, WalkSATvav cannot compete with these solvers on solving random 3-SAT instances greater than the phase transition ratio. In our view, this is partially due to the fact that the WalkSATvav is a SLS algorithm that is easy to reach the local optimal solution instead of finding the global optimal solution.

The second contribution of this work is to improve the WalkSATvav for SAT by remedying its shortcoming as mentioned above. Accordingly, we combine the GA with the allocation strategy and utilize an improved ACA as well as in the WalkSAT, leading to a new algorithm called WalkSATga. In WalkSATga, there are two different priorities in the global search. GA and ACA complement each other and play an important role in guiding the search direction to find the solution. GA is adopted to generate the pheromone distribution, and ACA is used to generate the suboptimal solution. Then the suboptimal solution obtained is utilized as the initial assignment of WalkSAT to guide the future search, which plays an important role in the whole solution process. To demonstrate the effectiveness of WalkSATga, we compare it with many state-of-the-art solvers on random 3-SAT instances greater than the phase transition ratio. The experimental results show that WalkSATga has almost the same success rate as YalSAT, and far beyond CSCCSat and WalkSAT for random 3-SAT instances greater than the phase transition ratio.

The third contribution of this work is to combine WalkSATga with WalkSATvav, leading to a new algorithm called WalkSATlg, which, as illustrated through extensive experiments, outperforms YalSAT Score$_2$SAT, CSCCSat, DCCAlm and WalkSAT on all random 3-SAT instances from SAT Competition 2017.

4 WalkSATvav Algorithm for Random 3-SAT Instances at the Phase Transition

4.1 Rationality of Allocation Strategy

Different parameter settings could show different performances of the algorithms. Based on the 3-SAT instances with r=4.3 in the SATLAB library[2], we found that if the parameters *pad* and *nad* are tuned appropriately, the assignments of 90% of

[2]https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

variables based on the allocation strategy are the same as that of the corresponding variables of the optimal solution [Fu, 2018a]. The idea of allocation strategy is to generate a complete assignment as the initial solution for solving 3-SAT problem in order to guide the trend of optimal assignment in advance, therefore reduce search space and accelerate finding the optimal solution. In fact, it has an essential impact on the WalkSATvav algorithms.

Since the allocation strategy is an important component of WalkSATvav, we are interested in this question: *how often the allocation strategy is executed to initialize the assignment for variables*? We have carried out an experiment for WalkSATvav on random 3-SAT instances from SAT Competition 2017[3] to figure out how frequently the allocation strategy is performed in generating an initial assignment. There include all 120 hard random instances with r=4.3, 5.204<r<5.206, and r=5.5 (40 instances each ratio), as well as all 60 uniform random 3-SAT instances (20 instances with r<4.267, 40 instances with r=4.267).

The allocation strategy execution ratio (AS ratio) is calculated as *frequency_AS/n* (see below) and the experimental results are summarized in Table 1.

- *#frequency_AS*: denotes the frequency of executing the allocation strategy for variables which is the total number of variables that satisfy the following formula with the setting of *pad*=1.8 and *nad*=0.56.

$$4-1) \quad \frac{p_{x_i}}{p_{x_i}+n_{x_i}} >= pad \text{ or } \frac{p_{x_i}}{p_{x_i}+n_{x_i}} <= nad$$

- # *n*: the total number of variables that require the initial assignment.

Clause-to-variable ratio	r=4.3	r=5.5	5.205≤r≤5.206	r<4.267	r=4.267
Average AS ratio	0.3737	0.346	0.3075	0.3371	0.3285

Table 1: Average AS ratio of the variables executing the allocation strategy for each ratio from SAT Competition 2017.

As is demonstrated in Table 1, the allocation strategy is performed in about 30% of variables for 3-SAT instances with r≠4.3, and closes to 40% for 3-SAT instances with r=4.3. Therefore, the allocation strategy plays a substantial role in the WalkSATvav algorithm.

4.2 WalkSATvav Algorithm

WalkSATvav differs from WalkSAT only in generating an initial assignment. The pseudo code of WalkSATvav algorithm is outlined in Algorithm 2 below. More specifically, WalkSATvav generates a complete assignment σ by the allocation strategy as the initial solution. After initialization, WalkSATvav executes a loop until it finds a satisfying assignment or reaches the time limit.

[3] https://baldur.iti.kit.edu/sat-competition-2017/benchmarks/

4.3 Evaluation of WalkSATvav Algorithm

In this subsection, we first introduce the benchmarks, the competitors, and the experimental setup utilized in our experiments. Then, we report the experiments conducted on the random 3-SAT benchmarks to evaluate the efficiency of WalkSATvav.

Algorithm 2: WalkSATv*av* (*F*)

```
Input: CNF-formula F, MaxTries, MaxSteps
Output: A satisfying assignment σ of F, or "no solution found"
begin
1   σ ←a generated truth assignment for F by the allocation strategy;
2       for i = 1 to MaxTries do
3           for j = 1 to MaxSteps do
4               if σ satisfies F then Return σ;
5               C ←an unsatisfied clause chosen at random;
6               With probability p
7                   v ←a random variable in C;
8               With probability 1 − p
9                   v ← a variable in C with minimum break;
10              σ: = σ with v flipped;
11          end for
12      end for
13      Return "no solution found";
14  end
```

4.3.1 Benchmarks and experiment preliminaries

We evaluate WalkSATvav on all random 3-SAT instances at the phase transition from SAT Competition 2016 and 2017 (r=4.267, 5000 ≤n ≤12800, 80 instances, two instances each size).

WalkSATvav is implemented in *C* language and compiled by Dev-C++. For the two parameters *pad* and *nad* in WalkSATvav, we test all groups of *pad*=1.5, 1.6, ..., 3 (the performance of WalkSATvav degrades significantly when *pad* exceeds 3) and *nad*=0.3, ..., 0.6. The preliminary results show that, on solving random 3-SAT instances, *pad*=1.8 and *nad*=0.56 are the best setting. In order to find a more optimal parameters setting for WalkSATvav, we test the parameters near *pad*=1.8 and *nad*=0.56 in a higher degree of accuracy, but did not observe any noticeable improvement. This means that WalkSATvav is not so sensitive to its parameters.

We compare WalkSATvav with five SLS solvers including YalSAT and Score$_2$SAT [4] (the winder and the bronzer of the random track of SAT Competition 2017 respectively), CSCCSat and DCCAlm [5] (the silver and the bronzes respectively at the random track of SAT Competition 2016), as well as WalkSAT which is the most influential SLS algorithms for solving 3-SAT instances. Especially, Score$_2$SAT significantly outperformed other competitors on random 3-SAT instances, and CSCCSat and DCCAlm significantly outperformed other competitors on random 3-SAT instances at the phase transition of SAT Competition 2016.

[4] https://baldur.iti.kit.edu/sat-competition-2017/solvers/
[5] https://baldur.iti.kit.edu/sat-competition-2016/solvers/

In the subsequent sections, all experiments run on a machine with a 3.4 GHZ Intel Core i3-3240 CPU and 8 GB RAM under Windows. Each solver is performed for 10 runs for each instance with a cutoff time of 5000 seconds. We report the number of averaged successful runs ("num"), i.e., the number of total successful runs divided by 10, and "all" represents the total number of average successful runs, i.e. , the sum of "num", as well as the success rate ("suc rate"), i.e. , the number of successful runs divided by the number of total runs.

4.3.2 Comparing WalkSATvav with the state-of-the art SLS solvers on random 3-SAT instances at the phase transition

For the 80 instances from random track of SAT Competition 2016 and 2017, Table 2 and Table 3 only show some instances which can be solved by these algorithms mentioned in this paper. However, other instances that are not shown in Table 2 or Table 3 cannot be solved by the algorithms mentioned in this paper.

Table 2 summarizes the performance of WalkSATvav on the random 3-SAT at the phase transition from SAT Competition 2017. The experimental results show that WalkSATvav significantly outperforms the above five SLS solvers on these instances. In particular, WalkSATvav solves **2** instances more than WalkSAT, 5 instances more than YalSAT on the 3-SAT instances with r=4.267. A well-known hardest distribution of SAT instances is at the phase transition [Xu, 2012]. Thus, although one instance of solving success is increased, it is enough to show the better performance

Instance Class	YalSAT num	Score$_2$SAT num	CSCCSat num	DCCAlm num	WalkSAT num	WalkSATv*av* num
v5400	1	1	1	1	1	1
v6400	0	0	0	0	0	**1**
v7400	1	1	1	1	1	1
v7600	0	0	0	1	1	1
v8000	1	0	1	0	1	1
v8200	1	1	1	1	1	1
v9400	0	0	0	0	1	1
v9600	1	1	1	1	1	1
v10200	0	0	0	0	1	1
v11000	1	1	1	1	1	1
v11200	1	1	1	1	1	1
v11600	0	1	1	1	0	**1**
All	7	7	8	8	10	**12**
suc rate	17.5%	17.5%	20%	20%	25%	**30%**

Table 2: Experimental results on the 3-SAT benchmark based on 10 runs for each instance, with a cutoff time of 5000s. Instances are at the phase transition from SAT Competition 2017.

Instance Class	YalSAT num	Score2SAT num	CSCCSat num	DCCAlm num	WalkSAT num	WalkSATv*av* num
v5800	0	0	0	0	**1**	**1**
v9000	1	1	1	1	1	1
All	1	1	1	1	**2**	**2**
suc rate	2.5%	2.5%	2.5%	2.5%	**5%**	**5%**

Table 3: Experimental results on the 3-SAT benchmark based on 10 runs for each instance, with a cutoff time of 5000s. Instances are at the phase transition from SAT Competition 2016.

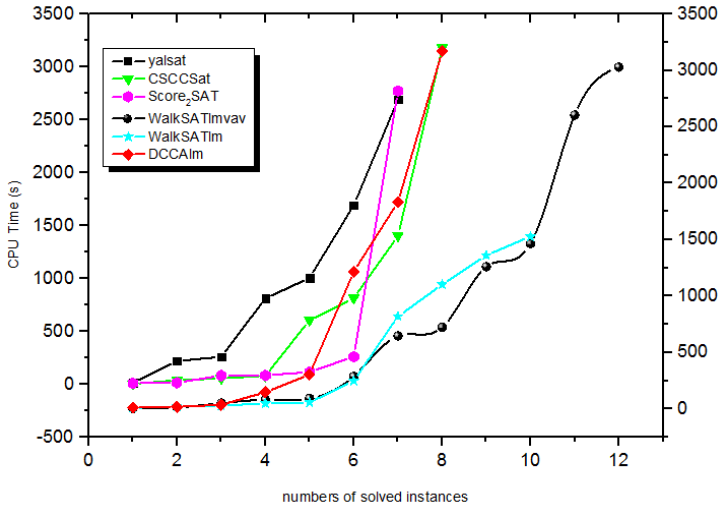

Figure 1: Comparing averaged CPU time distributions for SAT solvers in the random track of SAT Competition 2017 on random 3-SAT instances at the phase transition, where the cutoff time is 5000 s.

of WalkSATvav. Moreover, the good performance of WalkSATvav on the SAT Competition 2017 is clearly illustrated by Figure 1, which summarizes the run time distributions of the solvers on this benchmark.

5 Improving WalkSATvav Algorithm on Random 3-SAT

Section 4 above shows the good performance of WalkSATvav on random 3-SAT at the phase transition. However, the performance of WalkSATvav degrades on the 3-SAT instances greater than the phase transition ratio, e.g., WalkSATvav is worse than other state-of-the-art SLS solvers such as YalSAT and Score$_2$SAT, which are the top three solvers in the satisfiable random category of SAT Competition 2017.

Although the allocation strategy shows its effectiveness in the local search algorithms for solving random 3-SAT instances at the phase transition, it is still in its infancy. We consider the allocation strategy on random 3-SAT instances is too greedy for variable allocation value. The initial assignment largely determines the direction of the search. If an initial assignment is generated according to the allocation strategy, the restart strategy will occur when the number of flipping exceeds the search step limit in WalkSATvav, i.e., WalkSATvav would restart to obtain a new initial assignment based on the allocation strategy, and then to find an optimal solution. Since the allocation strategy is too greedy, WalkSATvav may result in a new initial assignment by restarting. However, the differences between the new initial assignment and the previous initial assignment may be small and even equal. Thus, WalkSATvav may perform the same search process as the previous one after restarting strategy activates. Especially, WalkSATvav could fall into a cycle, and waste a lot of time in a certain extent. Thus, this lack of differentiation is a serious disadvantage for WalkSATvav in our opinion.

To overcome this drawback, we combine the improved GA (the GA using the allocation strategy) with the improved ACA (detailed in Section 5.1 below) that has global search capabilities in order to improve WalkSATvav further. According to the advantages and disadvantages of both GA and ACA, we make full use of their advantages in this work so that they can complement each other. The improved GA is used to generate the pheromone distribution for the subsequent improved ACA, which is then used to generate the suboptimal solution as the initial assignment of WalkSAT to guide the future search, which plays an important role in the whole search process. It is worth noting that in the improvement process, the allocation strategy is utilized to the GA instead of WalkSAT.

5.1 Improved ACA

Considering an example of a formula: assume that the number of variables is n and the number of clauses is m. Suppose the variable set is $X_n=\{x_1, x_2, \ldots, x_n\}$ and the clause set is $C_m=\{c_1, c_2,\ldots, c_m\}$. Construct a structure diagram as shown in Figure 2. There are two types of values for each variable x_i, $i \in \{1, 2, \ldots, n\}$, i.e. 0 or 1, and the structure diagram has $2*n$ edges $\{(x_i, x_{i+1})^1, (x_i, x_{i+1})^0 \mid i \in \{1, 2, \cdots$. $(x_i, x_{i+1})^1$ indicates that the assignment of the variable x_i takes 1, and $(x_i, x_{i+1})^2$ means that the assignment of the variable x_i takes 0. An ant needs to travel n edges to get a set of assignments [Fu, 2018b].

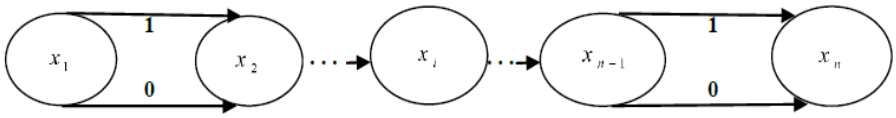

Figure 2: n variables structure diagram.

5.1.1 Edge selection rules

In the search process, the ants calculate the transition probability based on the amount of information of each path. Then the ant goes from vertex x_i to vertex x_{i+1}, $i \in \{1, 2, ..., n\}$. There are two paths: 0 and 1, and the probability of x_i taking 0 or 1 is the following two formulas respectively:

5-1) $$P(x_i)^1 = \frac{\left(\tau_{x_i}^{\ 1}\right)^{\alpha} * \left(\eta_{x_i}^{\ 1}\right)^{\beta}}{\left(\tau_{x_i}^{\ 1}\right)^{\alpha} * \left(\eta_{x_i}^{\ 1}\right)^{\beta} + \left(\tau_{x_i}^{\ 0}\right)^{\alpha} * \left(\eta_{x_i}^{\ 0}\right)^{\beta}};$$

5-2) $$P(x_i)^0 = \frac{\left(\tau_{x_i}^{\ 0}\right)^{\alpha} * \left(\eta_{x_i}^{\ 0}\right)^{\beta}}{\left(\tau_{x_i}^{\ 1}\right)^{\alpha} * \left(\eta_{x_i}^{\ 1}\right)^{\beta} + \left(\tau_{x_i}^{\ 0}\right)^{\alpha} * \left(\eta_{x_i}^{\ 0}\right)^{\beta}}.$$

When the ACA runs, it calculates the probability value according to Eqs. 5-1) and 5-2), and then chooses an edge according to the roulette rules. In Eqs. 5-1) and 5-2), α is a heuristic factor of pheromone, and β is the expected heuristic factor, and α and β reflect the relative importance of the information accumulated in the process of the ant colony algorithm and the heuristic information in the ant selection path, respectively. Here $\tau_{x_i}^{\ 1}$ is the pheromone value of x_i, and $\tau_{x_i}^{\ 0}$ is the pheromone value of $\neg x_i$, and $\eta_{x_i}^{\ 1}$ is the heuristic information value of x_i, and $\eta_{x_i}^{\ 0}$ is the heuristic information value of $\neg x_i$.

5.1.2 Pheromone update rules

As discussed in Section 3, we know that the disadvantage of GA is that it is not enough to make use of the feedback information, and the disadvantage of ACA is the lack of initial pheromone and low efficiency. Thus, we perform the improved GA, and then activate the ACA to select an individual with the maximum fitness value in THE restricted populations by the improved GA, i.e., a set of assignments with the maximal fitness value become the initial information of ACA.

Suppose that the suboptimal individual obtained by the improved GA is X, we regard it as an ant X. If X has traversed n edges from x_1 to x_n, X starts to initialize information according to the following two equations 5-3) and 5-4). Then ACA selects an ant X_m with the maximal fitness value, after each ant gets a set of assignments. Finally, the ant X_m updates the information according to the following two equations 5-5) and 5-6):

5-3) $$\tau_{x_i}^{\ 1} = \begin{cases} (m - f(X))/f(X) & , \quad (x_i, x_{i+1})^1 \\ f(X)/m & , \quad (x_i, x_{i+1})^0 \end{cases};$$

5-4) $$\tau_{x_i}^{\ 0} = \begin{cases} f(X)/m & , \quad (x_i, x_{i+1})^1 \\ (m - f(X))/f(X) & , \quad (x_i, x_{i+1})^0 \end{cases};$$

5-5) $$\tau_{x_i}^{\ 1} = \begin{cases} \tau_{x_i}^{\ 1} * (1-p) + Q/f(X_m) & , \quad (x_i, x_{i+1})^1 \\ \tau_{x_i}^{\ 1} * (1-p) & , \quad (x_i, x_{i+1})^0 \end{cases};$$

$$5-6) \quad \tau_{x_i}{}^0 = \begin{cases} \tau_{x_i}{}^0 * (1-p) + Q/f(X_m) & , \quad (x_i, x_{i+1})^0 \\ \tau_{x_i}{}^0 * (1-p) & , \quad (x_i, x_{i+1})^1 \end{cases};$$

where p is the pheromone residue factor, $p \in [0, 1]$. If $\tau_{x_i}{}^1$ or $\tau_{x_i}{}^0$ exceeds the maximal pheromone $t_{\max}$, then $\tau_{x_i}{}^1 = t_{\max}$ or $\tau_{x_i}{}^0 = t_{\max}$; if $\tau_{x_i}{}^1$ or $\tau_{x_i}{}^0$ is less than the minimal pheromone $t_{\min}$, then $\tau_{x_i}{}^1 = t_{\min}$ or $\tau_{x_i}{}^0 = t_{\min}$. After each pheromone updates, ACA sets the pheromone between the maximal pheromone value $t_{\max}$ and the minimal pheromone value $t_{\min}$. Q is a constant related to the pheromone quantity released by ants.

5.1.3 Heuristic update rules

The heuristic information reflects the degree of inspiration of adjacent two variables, and the values of heuristic information obtained by the GA are not changed in the whole operation of ACA. The heuristic information of a variable $x_i \in X_n$ is given as follows, where p_{x_i} and n_{x_i} are described in Section 2.

$$5-7) \quad \eta_{x_i}{}^1 = \frac{p_{x_i}}{p_{x_i} + n_{x_i}}, \eta_{x_i}{}^0 = \frac{n_{x_i}}{p_{x_i} + n_{x_i}}.$$

5.1.4 Improved ACA

All formulas from Eqs. 5-3) to Eqs. 5-7) are proposed based on the characteristics of random 3-SAT problem greater than the phase transition ratio. A new variable assignment heuristic strategy (*varassign*) named *IAS* is introduced in the ACA. The pseudo-code of *IAS* is given in Algorithm 3 below. The improved ACA applies crossover operation and mutation operation like the GA, where *maxGenerations* represents the maximum limit of evolutionary generation.

Algorithm 3: *varassign*-heuristic IAS

Input: CNF-formula F, *maxAnts*= $n/2$, *maxGenerations*, α, β, p, Q

Output: A satisfying assignment σ of F, or "Ultimately the best assignment σ "

```
begin
    σ : = a better truth assignment generated by GA;
    Initialize heuristic information;
    Initialize pheromone by σ;
    for step : = 1 to maxGenerations do
        Calculates the transition probability;
        for step : = 1 to maxAnts do
            P: = Get a new truth assignment σ by roulette rules according to
                transition probability;
            if σ satisfies F then return σ;
        end for
        P1←Crossover operation on P;
        P2←Mutate operation on P1;
        for step : = 1 to maxAnts do
            if σ∈P2 satisfies F then return σ;
        end for
        Update pheromone;
    end for
    Find the ant σ with the greatest fitness;
    return "Ultimately the best assignment σ";
end
```

5.2 WalkSATga Algorithm

We employ the improved GA and the improved ACA as detailed above into WalkSATvav, resulting in a new algorithm called WalkSATga, which integrates the global search algorithm with some local search strategies. The pseudo-code of WalkSATga algorithm is outlined in Algorithm 4 below.

WalkSATga utilizes two global search schemes including the improved GA and the improved ACA. It is different from the WalkSATvav algorithm which is only based on the local search. Firstly, WalkSATga algorithm utilizes the improved GA to obtain an optimal assignment which serves as an initial pheromone of *IAS*, and then *IAS* is executed to get a suboptimal assignment which serves as an initial assignment of WalkSAT. This initial assignment determines the search direction of WalkSAT. The simply flow chart of WalkSATga is given in Figure 3 below.

Algorithm 4: WalkSATga(*F*)

Input: CNF-formula *F* , *MaxTries*, *MaxSteps*
Output: A satisfying assignment σ of *F*, or "no solution found"
begin
for *i* = 1 to *MaxTries* **do**
  σ: = **a better truth assignment generated by GA**;
  if σ *satisfies F* **then Return** σ;
  σ : = **a suboptimal truth assignment generated by *varassign*-heuristic IAS**;
  if σ *satisfies F* **then Return** σ;
    for *j* = 1 to *MaxSteps* **do**
      C ←an unsatisfied clause chosen at random;
      With probability *p*
       v ←a random variable in *C*;
       With probability 1 − *p*
      v ← a variable in *C* with the minimum *break*;
       Flip *v* in σ;
       if σ *satisfies F* **then Return** σ;
    end for
  end for
  Return "no solution found";
end

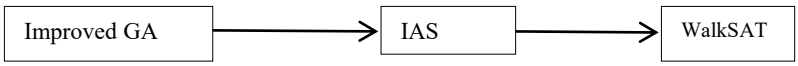

Figure 3: Flow chart of WalkSATga

Since GA and ACA are not suitable for solving large SAT problem, we call WalkSATga for solving small SAT problem; otherwise, we call WalkSAT for solving SAT problem.

WalkSATga algorithm has two important advantages: it makes full use of the advantages of GA and ACA, and also employ the advantages of both local search and global search.

5.3 Evaluations of WalkSATga Algorithm

5.3.1 Benchmarks and experiment preliminaries

To evaluate WalkSATga, we set up the following three benchmarks:

1. **3-SAT *r*=4.3**: 40 hard random 3-SAT instances with *r*=4.3 from SAT Competition 2017 (400 ≤*n* ≤540, 5 instance each size).
2. **3-SAT *r*=5.5**: all random 3-SAT instances with *r*=5.5 from SAT Competition 2017 (400 ≤*n* ≤540, 5 instance each size, 40 instances in all).
3. **3-SAT *r*=4.3**: 500 random generated instances with r=4.3 (200≤*n*≤600 variables, 100 instances each size).

WalkSATga is implemented in *C* and compiled by Dev-C++. For the five parameters in WalkSATga, we test all groups of *maxGenerations*=2, 4,…, 40 (the performance of WalkSATga degrades significantly when maxGenerations exceeds 40), α=0.5, 1, …, 5, β=0, 1, 2, …, 5, p=0.1, 0.2, …, 0.9, and Q=1, 2, 5, 10, 15, 20,…, 100 for solving random 3-SAT instances with r= 4.3 from SATLAB library. The experimental results show that the parameters α, p and Q have a very small impact on the algorithm, i.e., when they were set differently, there is no big changes on the success rate of the algorithm, but maxGenerations and β have a great impact on the algorithm, thus, they are the main parameters to be set, and other three parameters (α, p and Q) can be fixed. The group of parameters *maxGenerations*=20, α=2, β=1, p=0.6 and Q=5 is the best for 3-SAT instances with r=4.3. In order to find a more optimal setting, we tried different setting for α, p and Q with *maxGenerations*=20 and β=1 in a higher degree of accuracy but did not observe any noticeable improvement. This means WalkSATga is not so sensitive to those parameters α, p and Q.

We compared WalkSATga with five SLS solvers including YalSAT, Score$_2$SAT, CSCCSat, WalkSAT and WalkSATvav. Especially, YalSAT outperforms other competitors on random 3-SAT instances with r=4.3 and r=5.5 for SAT Competition 2017. For the first two benchmarks, each solver performs 10 runs on each instance. For the generated random instances, each solver is performed for one run on each instance. The cut off time of each run is set to 5000 CPU seconds.

5.3.1 Comparing WalkSATga with well-known SLS solvers on random 3-SAT instances greater than the phase transition ratio

Tables 4, 5 and 6 summarize the performance of WalkSATga on the random 3-SAT instances greater than the phase transition ratio compared with other well-knowns SLS solvers. The results showed that WalkSATga performs far beyond CSCCSat and WalkSAT on these instances. Compared with WalkSAT and CSCCSat, the "All" increases by **30** respectively. WalkSATvav outperforms WalkSAT for the instances with r=4.3 and r=5.5. Thus, WalkSATvav works toward the improvement of

Instance class	YalSAT num	Score$_2$SAT num	CSCCSat num		WalkSAT num	WalkSATvav num	WalkSATga num
v400	5	5	5		5	4	5
v420	5	5	3		3	1	5
v440	5	5	1		1	3	5
v460	5	5	1		1	3	5
v480	5	5	0		0	3	5
v500	5	5	0		0	2	5
v520	5	5	0		0	3	5
v540	5	5	0		0	2	5
All	**40**	**40**	10		10	21	**40**
suc rate	**100%**	**100%**	25%		25%	52.5%	**100%**

Table 4: Experimental results on the 3-SAT benchmark with r=4.3 from SAT Competition 2017, based on 10 runs for each instance.

Instance class	YalSAT num	Score$_2$SAT num	CSCCSat num	WalkSAT num	WalkSATvav num	WalkSATga num
v400	1	1	1	1	1	1
v420	2	2	1	1	2	1
v440	0	0	0	0	0	0
v460	1	1	0	0	1	1
v480	1	1	0	0	1	1
v500	2	2	0	0	2	2
v520	1	1	0	0	1	1
v540	1	1	0	0	1	0
All	**9**	**9**	2	2	**9**	7
suc rate	**22.5%**	**22.5%**	5%	5%	**22.5%**	17.5%

Table 5: Experimental results on random 3-SAT based on 100 runs for each instance, with a cutoff time of 5000 s. Instances with r=5.5 are from SAT Competition 2017.

Instance class	YalSAT num	Score$_2$SAT num	CSCCSat num	WalkSAT num	WalkSATvav num	WalkSATga num
v200	100	100	100	100	100	100
v300	100	100	73	72	80	100
v400	100	100	25	23	32	100
v500	100	100	20	0	22	100
v600	100	100	0	0	16	100

Table 6: Experimental results on the 3-SAT benchmark with r=4.3 from generated instances, based on one run for each instance, with a cutoff time of 5000 s.

WalkSAT for these instances. However, the performance of WalkSATga is far better than that of WalkSAT and WalkSATvav, which indicates that WalkSATga has improved the success rate of WalkSAT on the instances with r=4.3. The experimental results show that the proposed heuristics have played a great role in the improvement of WalkSAT. Moreover, WalkSATga is highly competitive with that of YalSAT and Score$_2$SAT.

It was noted in Section 3 that the total number of variables satisfying the allocation strategy is different based on different parameters, which have a great impact on the proposed algorithms for solving the SAT problems. Thus, compared to solving random 3-SAT instances with r=4.3, the success rate for solving random 3-SAT instances with r=5.5 is lower. If parameter settings are tuned based on random 3-SAT instances with r= 5.5, the success rate of the solution for the random 3-SAT instances with r=5.5 can be further improved.

6 WalkSATlg Solver and Results on Random 3-SAT instances for SAT Competition 2017

As we can clearly see from Section 4 and Section 5 that WalkSATvav and WalkSATga have their own advantages, this section presents a new SAT solver called WalkSATlg, which is a combination of WalkSATvav and WalkSATga, along with

experimental study to evaluate WalkSATlg on random 3-SAT instances at and near the phase transition.

6.1 Experimental Result on Random 3-SAT instances

The SAT Competition in 2017 is a competitive event for SAT solvers. All random 3-SAT instances (180 instances) from SAT Competition 2017 are generated randomly. In this subsection, we carry out experiments to evaluate the performance of WalkSATlg on random 3-SAT instances with various ratio.

6.1.1 Benchmarks and experiment preliminaries

To evaluate WalkSATlg, we set up four benchmarks:

1. **3-SAT 3.86 ≤ *r* ≤ 4.24:** all 20 random 3-SAT instances with $3.86 \leq r \leq 4.24$ from SAT Competition 2017 (n=1000000, one instance each ratio).
2. **3-SAT *r*=4.267:** all 40 random 3-SAT instances with r=4.267 from SAT Competition 2017 (n=5000, 5200, ..., 12800, one instance each size).
3. **3-SAT *r*=4.3:** all 40 random 3-SAT instances with r=4.3 from SAT Competition 2017 (n=400, 420, ..., 540, 5 instances each size).
4. **3-SAT *r*=5.5**: all 40 random 3-SAT instances with r=5.5 from SAT Competition 2017 (n=400, 420, ..., 540, 5 instances each size).

The instances with $5.205 \leqslant r \leqslant 5.206$ for SAT Competition 2017 are too hard for all incomplete solvers so that they are not included in our experiments.

WalkSATlg is implemented in *C* and compiled by Dev-C++. All the parameters are set in the same way as those in Section 4 and Section 5. Firstly, we compare WalkSATlg with five SLS solvers including YalSAT, Score$_2$SAT, CSCCSat, DCCAlm and WalkSAT. Particularly, Score$_2$SAT significantly outperformed other competitors on random 3-SAT instances of SAT Competition 2017. Secondly, we compare *AS* [Fu *et al*. 2018a], WalkSATvav, GAGR [Fu *et al*. 2017] and WalkSATga. Each solver performs ten runs on each instance with a cutoff time of 5000 seconds.

6.1.2 Experimental results on random 3-SAT for SAT competition 2017

Table 7 shows experimental results on the random 3-SAT instances. From Table 7, it is apparent that WalkSATlg dramatically outperforms other solvers. Although WalkSATlg and its competitors solves the same number of instances with r<4.267 expect for YalSAT, WalkSATlg solves most of instances. Overall, WalkSATlg solves **39** instances more than WalkSAT, **41** instances more than CSCCSat, **7** instances more than YalSAT, **5** instances more than Score$_2$SAT, and **4** instances more than DCCAlm, respectively.

It is very competitive for all kinds of algorithms on solving the random 3-SAT instances. Although the YalSAT solver received a gold medal and the Score$_2$SAT won the bronze medal on random track of SAT Competition 2017, it can be seen from Table 7 that Score$_2$SAT performs better than YalSAT on random 3-SAT instances. DCCAlm won the bronze award and CSCCSat won the silver award on random track of SAT Competition 2016. However, the performance of the DCCAlm on random 3-

SAT instances is better than CSCCSat. Score$_2$SAT had the best performance on random 3-SAT instances for SAT competition 2017, and DCCAlm had the best performance on random 3-SAT instances for SAT competition 2016. However, it is worth noting that WalkSATlg significantly outperforms DCCAlm and Score$_2$SAT. Therefore, WalkSATlg shows the state-of-the-art performance on random 3-SAT instances.

Instance class	YalSAT num	Score$_2$SAT num	CSCCSat num	DCCAlm num	WalkSAT num	WalkSATlg num
r<4.267	16	18	18	18	18	18
r=4.267	7	7	8	8	10	**12**
r=4.3	40	40	10	40	10	**40**
r=5.5	9	9	2	9	2	9
All	72	74	38	75	40	**79**
suc rate	51.4%	52.9%	27.1%	53.6%	28.6%	**56.4%**

Table 7: Experimental results on random 3-SAT instances based on 10 runs for each instance, with a cut off time of 5000 seconds for SAT Competition 2017.

Instance class	AS num	WalkSATvav num	GAGR num	WalkSATga num
r<4.267	0	18	0	18
r=4.267	0	12	0	10
r=4.3	14	21	0	40
r=5.5	0	9	0	9
All	14	60	0	77
suc rate	10%	42.8%	0%	55%

Table 8: Experimental results on random 3-SAT instances based on 10 runs for each instance, with a cutoff time of 5000 seconds for SAT Competition 2017.

According to Table 7 and Table 8, WalkSATlg gives the best performance on random 3-SAT instances. WalkSATlg solves 79 instances; WalkSATga solvers 77 instances, WalkSATvav solves 60 instances; AS solves 14 instances; GAGR solves 0 instances.

7 Comparative Discussions

In this section, we present a discussion compared with some related work to further show and clarify the originality and novelty of the proposed algorithms. More specifically, we discuss the major differences between *AS* [Fu et al. 2018a] and WalkSATvav, as well as the main differences between GAGR [Fu et al. 2017] and WalkSATga.

7.1 Main Differences between WalkSATvav and AS

Although WalkSATvav is related to the *AS* [Fu et al. 2018a], there exist major differences between WalkSATvav and *AS*, which are summarized as below:

1) Variable selection mechanism: the WalkSATvav algorithm prefers to select the variable with the minimum *break* to be flipped, i.e., the variable selection of WalkSATvav

is based on the WalkSAT algorithm, while the *AS* algorithm uses the general *score* property to choose the variable to be flipped, i.e., the variable selection of *AS* is based on GSAT algorithm.

2) Empirical performance on random 3-SAT benchmarks: as can be clearly seen from the experiments illustrated in Table 8 in Section 6, the WalkSATvav algorithm performs much better than the *AS* algorithm on random 3-SAT benchmarks.

7.2 Main Differences between WalkSATga and GAGR

The main differences between GAGR [Fu et al. 2017] and WalkSATga are summarized below:

1) Basic framework: the WalkSATga algorithm is based on the WalkSAT algorithm which has been applied successfully to solving random 3-SAT, while the GAGR algorithm is developed following the *GSAT* algorithm and the *GA* algorithm which is a global search algorithm which is not suitable for solving large SAT instances.

2) Initial assignment strategy: the WalkSATga algorithm utilizes a combination of GAGR and the allocation strategy, as well as the heuristic *IAS* to generate a complete assignment as the initial solution, while the GAGR algorithm does not use the heuristic *IAS* and the allocation strategy, and simply generates an initial assignment based on the *GA* algorithm.

3) Variable selection mechanism: the WalkSATga algorithm prefers to select the variable with the minimum *break* to be flipped, i.e., the variable selection of WalkSATga is based on the WalkSAT algorithm, while the GAGR algorithm uses the general *score* property to choose the variable to be flipped, i.e., the variable selection of GAGR is based on the *GSAT* algorithm.

4) Empirical performance on random 3-SAT benchmarks: according to the experimental results presented in Table 8 in Section 6, it can be clearly observed that WalkSATga significantly outperforms GAGR on random 3-SAT benchmarks. Since the GAGR algorithm is not suitable for solving large 3-SAT instances, the WalkSATga utilizes WalkSAT framework to solve the large 3-SAT benchmarks.

8 Conclusions

The present work focused on improving the performance of a well-known SLS SAT solver WalkSAT on solving random 3-SAT instances equal to and greater than the phase transition ratio. Three effective algorithms have been proposed respectively for this purpose, namely WalkSATvav, WalkSATga, and WalkSATlg. WalkSATvav applied the idea of the allocation strategy on random 3-SAT instances at the phase transition. WalkSATga further enhanced WalkSATvav to target on random 3-SAT instances greater than the phase transition ratio by combining an improved GA and an improved ACA, while WalkSATlg was an integration of WalkSATga and WalkSATvav to target for all random 3-SAT instances.

The key ideas centralized in the present work was summarized as follows: the allocation strategy was utilized to generate an initial assignment for solving 3-SAT instances, and to guide the trend of optimal assignment in advance, and to accelerate finding the optimal solution. Then the allocation strategy is further adopted by the GA and the ACA, resulting in a new heuristic namely *IAS*. According to *IAS* heuristic,

there are two different priorities in the global search while the improved GA and the improved ACA are combined to complement each other and overcome their shortcomings.

The experimental studies on random 3-SAT instances at the phase transition indicated that WalkSATvav outperformed the state-of-the-art SLS solvers YalSAT, Score$_2$SAT, CSCCSat, DCCAlm and WalkSAT, where YalSAT and Score$_2$SAT were the gold winner and the bronzer on the random track of SAT Competition 2017 respectively, and CSCCSat and DCCAlm won the silver and the bronzer respectively on the random track of SAT Competition 2016. In addition, the experimental studies on random 3-SAT instances greater than the phase transition ratio indicated that the performance of WalkSATga is far better than that of CSCCSat, WalkSAT and WalkSATvav, and WalkSATga is highly competitive with that of YalSAT and Score$_2$SAT. Finally, WalkSATlg significantly outperformed the existing SLA algorithms according to the experiments on random 3-SAT instances at and near the phase transition from SAT Competition 2017.

As for the future work, we would like to design ehanced functions on pheromone update and heuristic update. We will then further improve the state-of-the-art SLS algorithms using enhanced functions on random 3-SAT instances. Moreover, we're going to generate extensive random 3-SAT instances with r=5.5 by a random generator, and then the parameters of WalkSATlg are trained to find a set of parameters adapting to such instances. Furthermore, we would like to apply the allocation strategy, the improved GA, as well as the improved ACA to other combinatorial 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.2682017ZT12, 2682016 CX119).

References

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

[Balint, 2012] Balint, A., & Schöning, U.: Choosing probability distributions for stochastic local search and the role of make versus break. In Pro. 15th Int. Conf. on Theory and Applications of Satisfiability Testing, Trento, Italy, 17–20, June, 2012.

[Balyo, 2016] Balyo, T. : Using algorithm configuration tools to generate hard random satisfiable benchmarks. In Pro. 19th Int. Conf. on Theory and Applications of Satisfiability Testing, Bordeaux, France, 60–62, July, 2016.

[Biere, 2017] Biere, A.: CADICAL, LINGELING, PLINGELING, TREENGELING and YALSAT. In Pro. 20th Int. Conf. on Theory and Applications of Satisfiability Testing, Melbourne, Australi, 14, August, 2017.

[Cai, 2017] Cai, S., & Luo, C. Score$_2$SAT Solver Description. In Pro. 20th Int. Conf. on Theory and Applications of Satisfiability Testing, Melbourne, Australi, 34, August, 2017.

[Cai, 2013a] Cai, S., & Su, K.: Local search for boolean satisfiability with configuration checking and subscore. Artificial Intelligence, 204, 75-98, 2013.

[Cai, 2013b] Cai, S., Su, K., & Luo, C.: Improving WalkSAT for random k-satisfiability problem with k> 3. In Pro. 27th AAAI Conference on Artificial Intelligence, 145-151, 2013.

[Canisius, 2016] Canisius, G., Wilson, K., Zhang, Y., Chenhui, Y., & Xin, H.: A genetic-based local search method for SAT problem. In Pro. On Information Technology, Networking, Electronic & Automation Control Conference, IEEE, Chongqing, 20-22, May, 2016.

[Chao, 1986] Chao, M. T., & Franco, J.: Probabilistic analysis of two heuristics for the 3-satisfiability problem. SIAM Journal on Computing, 15(4), 1106-1118, 2986.

[Cook, 1971] Cook, S. A.: The complexity of theorem-proving procedures. In Pro. of the third annual ACM symposium on Theory of computing, USA, 151-158, May, 1971.

[De Jong, 1989] De Jong, K. A., & Spears, W. M.: Using genetic algorithms to solve NP-complete problems. In Pro. Int. Conf. on Genetic algorithms, USA, 124-132, 1989.

[Faizullin, 2013] Faizullin, R. T., Dulkeyt, V. I., & Ogorodnikov, Y. Y. E.: Hybrid method for the approximate solution of the 3-satisfiability problem associated with the factorization problem. Trudy Instituta Matematiki i Mekhaniki UrO RAN, 19(2), 285-294, 2013.

[Fu, 2018a] Fu H. M., Xu Y., He X. X. and Ning X. R.: GSAT Algorithm Based on Task Allocation and Scheduling for 3-SAT Problem. Chinese Journal of Computer engineering&

Science, 40(08): 1366-1374, 2018.

[Fu, 2018b] Fu H. M., Xu Y., Ning X. R. and Zhang W. Y.: The empirical study of improved genetic algorithm combined with ant colony algorithm based on 3-SAT problem, 2017 International Conference on Fuzzy Theory and Its Applications (iFUZZY), Britain,733-739, 2018.

[Fu, 2017] Fu H. M., Xu Y., Wu G. F. & Ning X. R.: An Improved Genetic Algorithm for Solving 3-SAT Problems Based on Effective Restart and Greedy Strategy. In Pro. 12th Int. Conf. on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, 1-6, Nov., 2017.

[Gableske,2016] Gableske, O.: Sat solving with message passing. *Ph.D. dissertation*, *Ulm University*, *Germany*, 2016.

[Gao, 2007] Gao S., Jiang X. Z. & Tang K. Z.: Hybrid Algorithm Combining Ant Colony Optimization Algorithm with Genetic Algorithm. In Proc. Int. Conf. on Chinese Control, Harbin, 701-704, Aug., 2006.

[Hoos,2002] Hoos, H. H.: An adaptive noise mechanism for WalkSAT. In Proc. 8th Int. Conf. on Artificial intelligence, Canada, 655–660, July, 2002.

[Kroc, 2010] Kroc, L., Sabharwal, A., & Selman, B.: An empirical study of optimal noise and runtime distributions in local search. In Pro. 13th Int. Conf. on Theory and Applications of Satisfiability Testing, Edinburgh, Scotland, 34, July, 2010.

[Levin, 1984] Levin, L.: A survey of Russian approaches to perebor (brute-force searches) algorithms. Annals of the History of Computing. 6 (4): 384–400. (Translated by Trakhtenbro，B. A.)

[Li, 2003] Li, D. J., Qiang, C. Z., & Zhi, Y. Z.: On the Combination of Genetic Algorithm and Ant Algorithm. Journal of Computer Research and Development, 40(7): 273-275, 2003.

[Ling, 2005] Ling, Y. B., Wu, X. J., & Jiang, Y. F.: Genetic algorithm for solving SAT problems based on learning clause weights. Chinese Journal of computers -Chinese Edition, 2005(09), 1476-1482.

[Li, 2016] Li, B., & Zhang, Y. A.: A hybrid genetic algorithm to solve 3-SAT problem. In Proc. 12th Int. Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) , Changsha, 476-480, Aug., 2016.

[Liang, 1998] Liang, D. M., Wu, U., & Ma, S. H.. An efficient local search algorithm for structured SAT problems. Chiness Journal of Computers, 1998(S1): 92-97, 1998.

[Luo, 2013] Luo, C., Cai, S., Wu, W., & Su, K.: Focused random walk with configuration checking and break minimum for satisfiability. In Pro. on Principles and Practice of Constraint Programming. Berlin Heidelberg, 481-496, 2013.

[Luo, 2012] Luo, C., Su, K., & Cai, S.: Improving local search for random 3-SAT using quantitative configuration checking. In Pro. 20th European Conf. on Artificial Intelligence, France, 570-575, Aug., 2012.

[Luo, 2014] Luo, C., Cai, S., Wu, W., & Su, K.: Double Configuration Checking in Stochastic Local Search for Satisfiability. In Pro. 28th AAAI Conference on Artificial Intelligence, Canna, 27-31, July, 2014.

[Luo, 2016] Luo, C, Cai, S., Wu, W. & Su K.: CSCCSat. In Pro. 19th Int. Conf. on Theory and Applications of Satisfiability Testing, France, 10, July, 2016.

[Luo, 2015] Luo, C., Cai, S., Su, K., & Wu, W.: Clause states based configuration checking in local search for satisfiability. IEEE transactions on cybernetics, 45(5), 1028-1041, 2015.

[Marques-Silva, 2008] Marques-Silva, J.: Practical applications of Boolean satisfiability. In Proc. 9th Int. Workshop on Discrete Event Systems, Sweden, 74-80, Aug., 2008.

[Selman, 1994] Selman, B., Kautz, H. A., & Cohen, B.: Noise strategies for improving local search. In Pro. 20th Int. Cont. on Artificial intelligence, USA, 337-343, 1994.

[Selman, 1992] Selman B, Levesque H J, Mitchell D G.: A New Method for Solving Hard Satisfiability Problems. In Pro. 10th Int. Conf. on Artificial intelligence, California, 440-446, July, 1992.

[Wang, 2012] Wang, F., Zhou, Y. R., & Ye, L.: Ant Colony Algorithm Combined with Survey Propagation for Satisfiability Problem. Computer Science, 39(4), 227-231, 2012.

[Xu, 2012] Xu, L., Hoos, H. H., and Leyton-Brown, K.: Predicting satisfiability at the phase transition. In Proc. In Pro. 26th AAAI Conference on Artificial Intelligence, 584–590, 2012.

[Xu, 2013] Xu, L., Yu, J. P.: Improved Bounded Model Checking on Verification of Valid ACTL Properties. Computer Science,40(S1):99-102, 2013.

[Youness, 2015] Youness, H., Ibraheim, A., Moness, M., & Osama, M.: An efficient implementation of ant colony optimization on GPU for the satisfiability problem. In Pro. 23rd Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing, Finland, 230-235, April, 2015.

[Zhang, 2015] Zhang Y. A. & LI B. F.: The Empirical Study of the Schema Theory of Genetic Algorithm Based on 3-satisfiability Problem. In Pro. Joint International Mechanical, Electronic and Information Technology Conference, 448-453, 2015.