Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Double Configuration Checking in Stochastic Local Search for Satisfiability

Chuan Luo[1] and **Shaowei Cai**[2,3*] and **Wei Wu**[1] and **Kaile Su**[1,4]
[1]Key Laboratory of High Confidence Software Technologies of Ministry of Education, Peking University, Beijing, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[3]Queensland Research Laboratory, NICTA, Brisbane, Australia
[4]Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
chuanluosaber@gmail.com; shaoweicai.cs@gmail.com; william.third.wu@gmail.com; k.su@griffith.edu.au

Abstract

Stochastic local search (SLS) algorithms have shown effectiveness on satisfiable instances of the Boolean satisfiability (SAT) problem. However, their performance is still unsatisfactory on random k-SAT at the phase transition, which is of significance and is one of the empirically hardest distributions of SAT instances. In this paper, we propose a new heuristic called DCCA, which combines two configuration checking (CC) strategies with different definitions of *configuration* in a novel way. We use the DCCA heuristic to design an efficient SLS solver for SAT dubbed DCCASat. The experiments show that the DCCASat solver significantly outperforms a number of state-of-the-art solvers on extensive random k-SAT benchmarks at the phase transition. Moreover, DCCASat shows good performance on structured benchmarks, and a combination of DCCASat with a complete solver achieves state-of-the-art performance on structured benchmarks.

Introduction

The Boolean satisfiability (SAT) problem is one of the most widely studied NP-complete problems, and is central to many areas of computer science and artificial intelligence (Kautz, Sabharwal, and Selman 2009). Given a formula with conjunctive normal form (CNF), the SAT problem is to decide whether there exists an assignment that satisfies all clauses in the formula.

A family of SAT instances is uniform random k-SAT (Achlioptas 2009). A well-known phase transition phenomenon occurs in the solubility of random k-SAT, and random k-SAT at the phase transition is one of the empirically hardest distributions of SAT instances (Selman 1995; Xu, Hoos, and Leyton-Brown 2012). In last two decades, great progress has been made in solving random k-SAT near the phase transition, for example, by a statistical physics algorithm called survey propagation (SP) (Braunstein, Mézard, and Zecchina 2005) and stochastic local search (SLS) algorithms, such as WalkSAT (Selman, Kautz, and Cohen 1994). However, solving random k-SAT at the phase transition remains a challenge for all kinds of algorithms including SP. Moreover, SLS seems to be the most promising method for such satisfiable instances.

SLS algorithms for SAT operate on complete assignments and try to find a solution by flipping the Boolean value of a variable chosen according to a function in each search step. We use *pickVar* to denote the function for choosing the variable to be flipped. Although SLS algorithms are typically incomplete, they usually find solutions for satisfiable instances surprisingly effectively (Hoos and Stützle 2004). SLS algorithms usually work in two different modes, i.e., the greedy (intensification) mode and the diversification mode. In the greedy mode, they prefer the variables whose flips can decrease the number of unsatisfied clauses; in the diversification mode, they tend to better explore the search space and avoid local optima.

SLS algorithms for SAT have been widely studied since the introduction of GSAT (Selman, Levesque, and Mitchell 1992). Particularly, numerous works are devoted to improving SLS algorithms on random k-SAT instances close to or at the phase transition due to their well-known hardness. Recent advances in this direction are mainly owed to a number of ideas, such as the promising decreasing variables exploiting strategy (Li and Huang 2005), probability distribution (Balint and Fröhlich 2010), satisfying time (Li and Li 2012), configuration checking (Cai, Su, and Sattar 2011) and subscore (Cai and Su 2013c). This direction has been a mainstream of SLS algorithms for SAT, which is also witnessed by SAT competitions[1], where the benchmarks of the random SAT track are composed of random k-SAT instances close to or at the phase transition. Especially, most (nearly 90% of) instances in the benchmark of the random SAT track in SAT Competition 2013 are the ones at the phase transition. However, the performance of existing SLS algorithms on random k-SAT instances at the phase transition is still unsatisfactory.

Inspired by the simplicity and the effectiveness of the configuration checking (CC) idea on random k-SAT at the phase transition, we concentrate on designing a CC-based heuristic to improve SLS algorithms. In the context of SAT, there are two different CC strategies, i.e., neighboring variables based configuration checking (NVCC) (Cai and Su 2013c) and clause states based configuration checking (CSCC) (Luo, Su, and Cai 2012). Previous CC-based algorithms (Luo, Su,

*Corresponding author

[1]http://www.satcompetition.org/

and Cai 2012; Luo et al. 2013; Cai and Su 2013b; 2013c; Li and Fan 2013; Habet, Toumi, and Abramé 2013) employ only one CC strategy, either NVCC or CSCC. However, further observations show an interesting relationship between NVCC and CSCC, and thus motivate us to combine them together to obtain a more flexible heuristic.

In this paper, we first explore the relationship between NVCC and CSCC by theoretical analyses, and then propose a new heuristic called DCCA (Double Configuration Checking with Aspiration), by combining NVCC and CSCC. Based on DCCA, we develop a new SLS solver for SAT dubbed DCCASat. To demonstrate the effectiveness of DCCASat, we compare it against numerous state-of-the-art solvers on extensive random k-SAT benchmarks at the phase transition. The experimental results show that DCCASat significantly outperforms its competitors on these instances.

More encouragingly, DCCASat shows effectiveness on a broad range of structured benchmarks, indicating its robustness. Further, we combine DCCASat with a preprocessor called CP3 (Manthey 2012) and a complete solver named Riss3g (Manthey 2013), and the resulting hybrid solver achieves the state-of-the-art performance on a broad range of structured instances.

The rest of the paper is organized as follows. Next section provides necessary preliminaries, following with a review on NVCC and CSCC. Then, we propose the DCCA heuristic. After that, we present the DCCASat solver. Experiments to evaluate DCCASat on random k-SAT benchmarks at the phase transition are illustrated subsequently. Afterwards, we evaluate DCCASat on structured instances. Finally, we conclude this paper and outline future directions.

Preliminaries

Given a set of n Boolean variables $V = \{x_1, x_2, \cdots, x_n\}$, a *literal* is a variable x or a negated variable $\neg x$, and a *clause* is a disjunction of literals. A propositional CNF formula $F = c_1 \wedge \cdots \wedge c_m$ is a conjunction of clauses, where the number of clauses is denoted m and c_i $(1 \leq i \leq m)$ is a clause, and $\alpha = m/n$ is the clause-to-variable ratio of formula F. We use $V(F)$ to denote the set of all variables appearing in formula F. Two different variables are neighbors when they appear simultaneously in at least one clause, and $N(x)$ is the set of all neighboring variables of variable x. We also denote $CL(x) = \{c \mid c \text{ is a clause which } x \text{ appears in}\}$. A mapping $s : V(F) \rightarrow \{0, 1\}$ is an *assignment*. If assignment s maps all variables to a Boolean value, it is *complete*. Given a complete assignment s, each clause has two possible *state*s: *satisfied* or *unsatisfied*; a clause is satisfied if at least one literal in that clause is true under s; otherwise, it is unsatisfied. Given a CNF formula F, the SAT problem is to find an assignment that satisfies all clauses in F.

A random k-SAT instance is a CNF formula where each clause is chosen uniformly and contains exactly k distinct variables. For random k-SAT, numerical simulations (Kirkpatrick and Selman 1994) suggest the existence of a phase transition at clause-to-variable ratio of α_c. This phase transition is particularly interesting because it turns out that the really difficult instances are those ones where α equals α_c. A wide range of state-of-the-art SAT solvers exhibit dramatically longer runtimes for instances at the phase transition (Xu, Hoos, and Leyton-Brown 2012).

In SLS algorithms which use clause weighting schemes, each clause c is associated with a weight $w(c)$, and, for a variable x, $score(x)$ is the increment in the total weight of satisfied clauses by flipping x. A clause is δ-satisfied if exactly δ literals in that clause are true. For a variable x, $subscore(x)$ is defined as $submake(x)$ minus $subbreak(x)$, where $submake(x)$ is the number of 1-satisfied clauses that would become 2-satisfied by flipping x, and $subbreak(x)$ is the number of 2-satisfied clauses that would become 1-satisfied by flipping x (Cai and Su 2013b); $age(x)$ is the number of steps that has occurred since x's last flip.

Previous Configuration Checking Strategies

The idea of CC is to forbid flipping a variable whose *configuration* has not been changed since its last flip. There are two different CC strategies for SAT, i.e., NVCC (Cai and Su 2013c) and CSCC (Luo, Su, and Cai 2012).

The NVCC Strategy

In NVCC, the *configuration* of a variable x refers to a vector consisting of Boolean values of $N(x)$ (x's all neighboring variables). The Boolean array $NVChanged$ is used to implement NVCC. For a variable x, $NVChanged(x) = 1$ means at least one variable in $N(x)$ has been flipped since x's last flip. A variable x is neighboring-variables-based configuration changed decreasing (NVD) iff $score(x) > 0$ and $NVChanged(x) = 1$. The notation $NVDvars$ is used to denote the set of all NVD variables.

The CCA heuristic enhances NVCC with an aspiration mechanism (Cai and Su 2013c). A key notion in aspiration is the significant decreasing (SD) variable: A variable x is an SD variable iff $score(x) > \overline{w}$, where $\overline{w}$ is the averaged clause weight (over all clauses). The notation $SDvars$ is used to denote the set of all SD variables. The CCA heuristic works as follows. First, if $NVDvars$ is not empty, CCA picks a variable with the greatest $score$ in $NVDvars$; otherwise it picks an SD variable with the greatest $score$.

The CSCC Strategy

In CSCC, the *configuration* of a variable x refers to a vector consisting of clause states of $CL(x)$ (all clauses which x appears in). The Boolean array $CSChanged$ is used to implement CSCC. For a variable x, $CSChanged(x) = 1$ means at least one clause in $CL(x)$ has changed its state (from satisfied to unsatisfied or from unsatisfied to satisfied) since x's last flip. A variable x is clause-states-based configuration changed decreasing (CSD) iff $score(x) > 0$ and $CSChanged(x) = 1$. The notation $CSDvars$ is used to denote the set of all CSD variables.

The DCCA Heuristic

In this section, we first explore the relationship between the $NVDvars$ set and the $CSDvars$ set. Then, we propose the DCCA heuristic which combines NVCC and CSCC in a novel and effective way.

The $NVDvars$ Set and The $CSDvars$ Set

Intuitively, the forbidding strength of CSCC is stronger than that of NVCC, and the cardinality of $NVDvars$ is greater than that of $CSDvars$ (Li, Huang, and Xu 2013). In the following, we will show through theoretical analyses that indeed the $CSDvars$ set is a subset of the $NVDvars$ set.

Lemma 1. For a given variable x, if $CSChanged(x) = 1$, then $NVChanged(x) = 1$.

Proof. For a variable x, $CSChanged(x) = 1$ implies that there exists a clause $c \in CL(x)$ such that its state has been changed since x's last flip. On the other hand, to change c's state, at least one variable in c must be flipped, and we denote this variable as y. So, y has been flipped since x's last flip. Since both x and y appear in clause c, we have $y \in N(x)$. Therefore, we have $NVChanged(x) = 1$. □

Remark 1. The reverse of Lemma 1 is not necessarily true.

Proof. For a variable x, to make $NVChanged(x) = 1$, it suffices that one variable y in $N(x)$ has been flipped since x's last flip. Suppose no other variable in $N(x)$ has been flipped since x's last flip. In this case, to make $CSChanged(x) = 1$, the flip of y should be able to change the states of some clauses in $CL(x)$. However, suppose at the step just before y was flipped, all clauses in $CL(x) \cap CL(y)$ have more than one true literals, then flipping y does not change the state of any clause in $CL(x)$, and thus $CSChanged(x)$ is still 0. □

According to Lemma 1 and Remark 1, we can derive that the $CSDvars$ set is a subset of the $NVDvars$ set.

Details of the DCCA Heuristic

The main difference between DCCA and previous CC strategies is that it combines two different CC strategies, namely NVCC and CSCC, which have different forbidding strength.

The forbidding strength of a single CC strategy is either too weak or too strong. The forbidding strength of NVCC is too weak and becomes futile on dense instances such as random 6-SAT and 7-SAT instances near and at the phase transition (Cai and Su 2013c). On the other hand, the forbidding strength of CSCC is too strong as we can see from its definition, and it would forbid some good variables whose flips are of benefit. To support our arguments, the relatively poor results about using only either NVCC or CSCC can be found in Table 3 in Section 'Experiments on Random Benchmarks'.

Motivated by the relationship between NVCC and CSCC (as shown before), we propose a new heuristic which combines NVCC and CSCC, and also uses the aspiration mechanism. This new heuristic is thus called double configuration checking with aspiration (DCCA). The DCCA heuristic bridges NVCC and CSCC in terms of forbidding strength, and is more flexible for striking a better balance between intensification and diversification.

The DCCA heuristic is modified from the CCA heuristic and works as follows (shown in Algorithm 1). First, if $CSDvars$ is not empty, DCCA picks a variable with the greatest $score$ in $CSDvars$; otherwise, it picks a variable with the greatest $score$ in $NVDvars$ if $NVDVars$ is not empty; finally, if $NVDvars$ is empty, DCCA activates aspiration to pick an SD variable with the greatest $score$.

Algorithm 1: The DCCA Heuristic

```
1 if CSDvars ≠ ∅ then return variable x ∈ CSDvars with greatest score;
2 if NVDvars ≠ ∅ then return variable x ∈ NVDvars with greatest score;
3 if SDvars ≠ ∅ then return variable x ∈ SDvars with greatest score;
```

We note that as with the CCA heuristic, the DCCA heuristic is used as a framework for the greedy mode. When using DCCA in SLS algorithms, one still needs to specify the diversification mode and the tie-breaking methods.

The DCCASat Solver

We utilize the DCCA heuristic to develop a new SLS solver for SAT dubbed DCCASat. To focus on the essential part of DCCASat, we only present the pseudo code of its *pickVar* function (outlined in Algorithm 2), as described below.

In each search step, the *pickVar* function first utilizes the DCCA heuristic to pick a variable, with ties broken by a function referred to as FG. If the DCCA heuristic fails to return a variable, then the *pickVar* function switches to the diversification mode, where it utilizes a clause weighting scheme (referred to as CW) to update clause weights and then picks a variable according to a function referred to as FR from a random unsatisfied clause.

To obtain the whole DCCASat solver, we need to specify three components, namely, clause weighting scheme CW, as well as functions FG and FR. As DCCASat is developed on the basis of CCASat (Cai and Su 2013c), it distinguishes two types of instances and employs different components for them as CCASat does. The first type of instances includes random 3-SAT and structured (non-random) instances, and the other one includes random k-SAT instances with $k > 3$.

Components for random 3-SAT and structured SAT:

- Scheme CW: the SWT scheme (Cai and Su 2013c). SWT increases clause weights of all unsatisfied clauses by one; further, if the averaged clause weight $\overline{w}$ exceeds a threshold w_t, each clause weight is smoothed as $w(c_i) = \lfloor p \times w(c_i) \rfloor + \lfloor q \times \overline{w} \rfloor$, where $0 \leq p \leq 1$ and $0 \leq q \leq 1$.
- Function FG: the function which prefers to select the variable x with greatest $age(x)$.
- Function FR: the function which prefers to select the variable x with the greatest $age(x)$ for random 3-SAT, and the greatest $score(x)$ for structured SAT.

Components for random k-SAT with $k > 3$:

- Scheme CW: the PAWS scheme (Thornton et al. 2004), which works as follows. With probability sp, where $0 \leq sp \leq 1$, for each clause whose weight is greater than one, its weight is decreased by one; with probability $1 - sp$, the weight of each unsatisfied clause is increased by one.

Algorithm 2: The *pickVar* Function of DCCASat

```
if CSDvars ≠ ∅ then return variable x ∈ CSDvars with greatest score, breaking ties by function FG;
if NVDvars ≠ ∅ then return variable x ∈ NVDvars with greatest score, breaking ties by function FG;
if SDvars ≠ ∅ then return variable x ∈ SDvars with greatest score, breaking ties by function FG;
activate clause weighting scheme CW;
pick an unsatisfied clause c randomly;
return variable x in clause c by function FR;
```

- Function FG: the function which prefers to select the variable x with the greatest $hscore_2(x) = subscore(x) + \lfloor age(x)/\gamma \rfloor$, where γ is a positive integer (Cai, Luo, and Su 2014).
- Function FR: the function which prefers to select the variable x with the greatest $hscore(x) = score(x) + \lfloor subscore(x)/d \rfloor + \lfloor age(x)/\beta \rfloor$, where d and β are positive integers (Cai and Su 2013b).

We introduce the algorithmic settings in DCCASat as follows. For random 3-SAT, w_t for SWT is set to $200 + \lfloor \frac{n+250}{500} \rfloor$, where n is the number of variables in the instance; p for SWT and q for SWT are set to 0.3 and 0.7, respectively. For random k-SAT with $k > 3$, d for $hscore$ is set to $13 - k$; β for $hscore$ and γ for $hscore_2$ are both set to 1000; sp for PAWS is set to 0.75 for 4-SAT, 0.8 for 5-SAT, 0.9 for 6-SAT and 0.92 for k-SAT with $k \geq 7$. For structured instances, DCCASat performs a simple unit propagation procedure before the search; w_t for SWT is set to 300; p for SWT is set to 0.3; q for SWT is set to 0 if $\alpha \leq 15$ and to 0.7 otherwise (α is the clause-to-variable ratio of the instance), inspired by the literature (Cai and Su 2013a).

Experiments on Random Benchmarks

We conduct experiments to evaluate DCCASat on extensive random k-SAT benchmarks at the phase transition.

The Benchmarks

We adopt two random benchmarks (in **bold**) at the threshold ratio of phase transition.

1. **SC13_Threshold:** all 250 random k-SAT instances from the threshold benchmark of the random SAT track of SAT Competition 2013[2] (50 instances for each k-SAT class with $k = 3, 4, 5, 6$ and 7), which vary in variables.
2. **Large_Threshold:** random k-SAT instances generated randomly at the threshold ratio of phase transition according to the random k-SAT generator[3] used in SAT Competition 2013 (totally 500 instances, 100 for each k-SAT class with $k = 3, 4, 5, 6$ and 7). The size of instances in this benchmark ($n = 15000, 2500, 600, 300$ and 150 for $k = 3, 4, 5, 6$ and 7, respectively) is larger than that in the SC13_Threshold benchmark.

[2]http://satcompetition.org/2013/files/sc13-benchmarks-random.tgz

[3]http://sourceforge.net/projects/ksatgenerator/

For the above two benchmarks, the clause-to-variable ratio (α) of each instance equals the threshold ratio of phase transition (α_c), which is reported in (Mertens, Mézard, and Zecchina 2006) and is indicated in Table 1 and Table 2, and thus a significant fraction of the instances are unsatisfiable.

The Competitors

Seven state-of-the-art competitors of DCCASat are listed as follows and indicated in **bold**.

CCASat[4] (Cai and Su 2013c) is based on the CCA heuristic and is the winner of the random SAT track of SAT Challenge 2012. **CCA2013*** (Li and Fan 2013) is an efficient implementation of CCASat with some enhancements. The **probSAT*** solver (Balint and Schöning 2012) is the winner of the random SAT track of SAT Competition 2013, and **Ncca+*** (Habet, Toumi, and Abramé 2013) is the best CC-based SLS solver in the same track. **CScoreSAT2013*** (Cai and Su 2013b) is based on NVCC and is the best SLS solvers for solving random k-SAT with $k > 3$ near the phase transition. **FrwCB2013*** (Luo et al. 2013) is based on CSCC and is the best SLS solver for solving huge random 3-SAT near the phase transition. **SP**[5] (Braunstein, Mézard, and Zecchina 2005) is a statistical physics approach which can solve random 3-SAT instances with up to 10^7 variables near the phase transition (Mézard 2003).

For CCASat on solving random benchmarks, we use the parameters which are reported in the literature (Cai and Su 2013c). The versions of the solvers marked with notation '*' are the ones submitted to SAT Competition 2013.

Experimental Preliminaries

The DCCASat solver is developed on the top of CCASat, and thus is implemented in C++.

We adopt the evaluation methodology used in SAT competitions: Each solver performs one run on each instance with a cutoff time of 5000 seconds. Note that the number of the instances and the cutoff time are enough to test the performance of the solvers. For each solver on each instance class (or each benchmark), we report the number of the solved instances ('#solv.') and the par10 run time ('par10 time') in seconds, where the run time of a failed run is penalized as 10 times as the cutoff time. The results in **bold** indicate the best performance for an instance class (or a benchmark). The rules at SAT competitions establish that the winner is the solver which solves the most instances, breaking ties by selecting the solver with the least par10 run time.

All experiments are carried out on a machine under GNU/Linux, using 2 cores of Intel Core i7 2.4GHz.

Experimental Results

Results on the SC13_Threshold benchmark:

Table 1 reports the comparative results of DCCASat and its state-of-the-art competitors on the SC13_Threshold benchmark. DCCASat stands out as the best solver for

[4]http://shaoweicai.net/Code/CCASat-Opensource.zip

[5]http://www.ictp.trieste.it/~zecchina/SP/sp-1.4b.tgz

Instance Class	Ratio ($\alpha = \alpha_c$)	SP #solv. par10 time	FrwCB2013 #solv. par10 time	CScoreSAT2013 #solv. par10 time	Ncca+ #solv. par10 time	CCA2013 #solv. par10 time	CCASat #solv. par10 time	probSAT #solv. par10 time	DCCASat #solv. par10 time
3-SAT	4.267	4 46001	13 37148	14 36271	16 34174	15 35365	17 33267	16 34251	**18** **32328**
4-SAT	9.931	0 50000	11 39155	11 39277	8 42113	11 39212	9 41102	**15** **35232**	**15** 35339
5-SAT	21.117	0 50000	8 42132	10 40226	12 38447	11 39221	11 39275	11 39108	**13** **37362**
6-SAT	43.37	0 50000	12 38356	12 38290	14 36345	17 33525	17 33366	14 36210	**21** **29524**
7-SAT	87.79	0 50000	23 27457	21 29178	24 26251	21 29087	22 28097	25 25441	**26** **24252**
Overall	N/A	4 49200	67 36850	68 36648	74 35466	75 35282	76 35022	81 34048	**93** **31761**

Table 1: Comparative results of DCCASat and its competitors on the SC13_Threshold benchmark.

Instance Class	Ratio ($\alpha = \alpha_c$)	probSAT		DCCASat	
		#solv.	par10 time	#solv.	par10 time
3-SAT-v15000	4.267	2	49014	**6**	**47078**
4-SAT-v2500	9.931	**5**	**47545**	**5**	47644
5-SAT-v600	21.117	2	49005	**9**	**45671**
6-SAT-v300	43.37	0	50000	**11**	**44663**
7-SAT-v150	87.79	28	36314	**43**	**28856**
Overall	N/A	37	46375	**74**	**42783**

Table 2: Comparative results of DCCASat and probSAT on the Large_Threshold benchmark.

this benchmark, and outperforms its competitors on all instance classes except for the 4-SAT one. On the 4-SAT instance class, although probSAT performs slightly faster than DCCASat, DCCASat solves the same number of instances as probSAT does, and significantly outperforms other CC-based SLS solvers as well as SP. Overall, DCCASat solves 93 instances, while the second best solver namely probSAT solves 81 instances, indicating the efficiency of DCCASat. More encouragingly, DCCASat solves an instance which is not solved by all submitted solvers in the random SAT track of SAT Competition 2013. This also confirms the superiority of DCCASat over its state-of-the-art competitors.

Results on the Large_Threshold benchmark:

To measure the superiority of DCCASat on random k-SAT instances at the phase transition more accurately, we directly compare DCCASat with probSAT on large random instances at the phase transition (Large_Threshold), as DCCASat and probSAT are the best two solvers for SC13_Threshold in Table 1. According to Table 2, it is clear that DCCASat performs much better than probSAT on all instance classes (but the 4-SAT-v2500 one, where DCCASat solves the same number of instances as probSAT does). Particularly, on 6-SAT-v300, DCCASat solves 11 instances, while probSAT fails to solve any one. Overall, DCCASat solves 74 instances, while probSAT only solves 37 instances.

The effectiveness of the DCCA heuristic:

Recalling that the DCCA heuristic combines NVCC and CSCC, to demonstrate the effectiveness of DCCA, we evaluate two alternative versions of DCCASat namely DCCASat_alt1 (without CSCC, i.e., deleting line 1 in Algorithm 2) and DCCASat_alt2 (without NVCC, i.e., deleting line 2 in Algorithm 2) on the SC13_Threshold benchmark, and the related results are presented in Table 3, where the parameters used in DCCASat_alt1 and DCCASat_alt2 are identical to those used in DCCASat. It is apparent that DCCASat obviously outperforms its two alternative versions, proving that the effectiveness of the DCCA heuristic is mainly due to the combination of NVCC and CSCC.

Benchmark	DCCASat_alt1 #solv. par10 time	DCCASat_alt2 #solv. par10 time	DCCASat #solv. par10 time
SC13_Threshold	75 35253	59 38402	**93** **31761**

Table 3: Comparative results of DCCASat and its two alternative versions on the SC13_Threshold benchmark.

Experiments on Structured Benchmarks

We perform further empirical analyses to present the robustness of DCCASat on extensive structured benchmarks.

Empirical Setup

We set up three structured benchmarks (in **bold**).

1. **SC13_HC:** all 150 satisfiable instances from the hard-combinatorial SAT track of SAT Competition 2013[6], which covers a broad range of structured types.
2. **CBMC:** all 39 satisfiable testing instances generated by a bounded model checking tool.
3. **SWV:** all 75 satisfiable testing instances generated by the Calysto static checker (Babic and Hu 2008).

The CBMC and SWV benchmarks[7] are real world verification problems. Further, they are currently remarkable chal-

[6]http://satcompetition.org/2013/files/sc13-benchmarks-combinatorial.tgz

[7]https://cs.uwaterloo.ca/~dtompkin/papers/sat10-dave-instances.zip

Benchmark	#instances	CCASat		gNovelty+GCwa		Sattime2013		Sparrow2013HC		DCCASat	
		#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time
SC13_HC	150	62	29469	65	28471	73	25840	71	26396	**77**	**24476**
CBMC	39	25	18375	**39**	39	25	18230	28	14203	**39**	**4**
SWV	75	24	34200	37	25338	33	28092	24	34075	**38**	**24696**

Table 4: Comparative results of DCCASat and its competitors on structured benchmarks.

Benchmark	#instances	Sparrow+CP3		DCCASat+CP3		Glucose		SparrowToRiss		DCCASatToRiss	
		#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time
SC13_HC	150	78	24085	82	22740	109	13991	131	6751	**134**	**5733**

Table 5: Comparative results of DCCASat+CP3, DCCASatToRiss and their competitors on the SC13_HC benchmark.

lenges for SLS solvers, and have been extensively studied in literature (Hutter et al. 2009; Tompkins and Hoos 2010; Tompkins, Balint, and Hoos 2011; Duong et al. 2013).

We compare DCCASat with four efficient SLS solvers, which are listed as follows and indicated in **bold**. The **gNovelty+GCwa*** solver (Duong et al. 2013) is one of the most recent SLS solvers tested on CBMC and SWV benchmarks. **Sattime2013*** (Li and Li 2012) is an improved version of Sattime2012, which is the best SLS solver for hard-combinatorial track in SAT Challenge 2012. **Sparrow2013HC*** (Balint and Fröhlich 2010) is the core component of Sparrow+CP3, which is the best SLS solver for hard-combinatorial instances in SAT Competition 2013. We also include **CCASat** as the baseline solver.

For CCASat on solving structured benchmarks, we use the parameters which are reported in the literature (Cai and Su 2013c). For the gNovelty+GCwa solver on solving the CBMC and SWV benchmarks, we use the parameters which are reported in the literature (Duong et al. 2013). The versions of the solvers marked with notion '*' are the ones submitted to SAT Competition 2013.

We note that the evaluation methodology on structured benchmarks is the same as the one on random benchmarks.

Empirical Analyses

The robustness of the DCCASat solver:

Table 4 illustrates the empirical results of DCCASat and its efficient SLS competitors on structured benchmarks (including SC13_HC, CBMC and SWV). The results show that DCCASat achieves the best performance among all competing SLS solvers on structured benchmarks. On the SC13_HC benchmark, the number of solved instances of DCCASat is 77, while those of CCASat, gNovelty+GCwa, Sattime2013 and Sparrow2013HC are 62, 65, 73 and 71, respectively. On CBMC and SWV, DCCASat performs better than gNovelty+GCwa, which is one of the best SLS solvers for solving CBMC and SWV, and also DCCASat dramatically outperforms other competitors. The experimental results indicate the robustness of DCCASat.

Combining DCCASat with other techniques:

Inspired by the success of the Sparrow+CP3 solver, which equips Sparrow (Balint and Fröhlich 2010) with a preprocessor called CP3 (Manthey 2012) and is the best SLS solver for SC13_HC, and the SparrowToRiss solver, which combines Sparrow+CP3 with a complete solver named Riss3g (Manthey 2013) and exhibits the best performance (although not officially ranked) for SC13_HC in SAT Competition 2013 (Balint and Manthey 2013), we combine DCCASat with CP3 and Riss3g, and evaluate the resulting hybrid solvers on hard-combinatorial instances. By replacing Sparrow with DCCASat in both the Sparrow+CP3 and SparrowToRiss solvers, we obtain two new solvers namely DCCASat+CP3 and DCCASatToRiss, respectively. Table 5 shows the comparative results of five solvers, namely, DCCASat+CP3, DCCASatToRiss, Sparrow+CP3, SparrowToRiss and Glucose (Audemard and Simon 2013) which is the official winner of the hard-combinatorial SAT track of SAT Competition 2013, on the SC13_HC benchmark. The results on SC13_HC show that DCCASat+CP3 outperforms Sparrow+CP3, and also present that DCCASatToRiss achieves the state-of-the-art performance, indicating that DCCASat can cooperate well with other techniques.

Conclusions and Future Work

This work took a significant step towards improving performance of SLS solvers for solving random k-SAT at the phase transition. We explored the relationship between NVCC and CSCC via theoretical analyses, and then proposed a new heuristic called DCCA, which combines NVCC and CSCC in a novel way. We used the DCCA heuristic to develop a new SLS solver for SAT dubbed DCCASat. The experiments present that DCCASat significantly outperforms state-of-the-art solvers on extensive random k-SAT benchmarks at the phase transition. Moreover, experiments on structured benchmarks demonstrate the robustness of DCCASat.

The strong experimental results suggest that this work opens a promising direction for improving SLS solvers, which focuses on the relationship and the combination of different forbidding strategies. Therefore, for future work, we would like to do more deep work along this direction.

Acknowledgments

This work is supported by 973 Program 2010CB328103, ARC Grant FT0991785, and NSFC 6137007. The authors would like to thank the anonymous reviewers for their helpful comments.

References

Achlioptas, D. 2009. Random satisfiability. In *Handbook of Satisfiability*. 245–270.

Audemard, G., and Simon, L. 2013. Glucose 2.3 in the SAT 2013 competition. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 42–43.

Babic, D., and Hu, A. J. 2008. Calysto: scalable and precise extended static checking. In *Proc. of ICSE 2008*, 211–220.

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

Balint, A., and Manthey, N. 2013. Sparrow+CP3 and SparrowToRiss. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 87–88.

Balint, A., and Schöning, U. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In *Proc. of SAT 2012*, 16–29.

Braunstein, A.; Mézard, M.; and Zecchina, R. 2005. Survey propagation: An algorithm for satisfiability. *Random Struct. Algorithms* 27(2):201–226.

Cai, S., and Su, K. 2013a. CCAnr. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 16–17.

Cai, S., and Su, K. 2013b. Comprehensive score: Towards efficient local search for SAT with long clauses. In *Proc. of IJCAI 2013*, 489–495.

Cai, S., and Su, K. 2013c. Local search for Boolean satisfiability with configuration checking and subscore. *Artif. Intell.* 204:75–98.

Cai, S.; Luo, C.; and Su, K. 2014. New scoring functions for random k-SAT with long clauses. Technical report. http://shaoweicai.net/Paper/new-scoring-functions.pdf.

Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artif. Intell.* 175(9-10):1672–1696.

Duong, T.-T. N.; Pham, D. N.; Sattar, A.; and Newton, M. A. H. 2013. Weight-enhanced diversification in stochastic local search for satisfiability. In *Proc. of IJCAI 2013*, 524–530.

Habet, D.; Toumi, D.; and Abramé, A. 2013. Ncca+: Configuration checking and Novelty+ like heuristic. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 61.

Hoos, H. H., and Stützle, T. 2004. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)* 36:267–306.

Kautz, H. A.; Sabharwal, A.; and Selman, B. 2009. Incomplete algorithms. In *Handbook of Satisfiability*. 185–203.

Kirkpatrick, S., and Selman, B. 1994. Critical behavior in the satisfiability of random Boolean expressions. *Science* 264:1297–1301.

Li, C., and Fan, Y. 2013. CCA2013. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 14–15.

Li, C. M., and Huang, W. Q. 2005. Diversification and determinism in local search for satisfiability. In *Proc. of SAT 2005*, 158–172.

Li, C. M., and Li, Y. 2012. Satisfying versus falsifying in local search for satisfiability. In *Proc. of SAT 2012*, 477–478.

Li, C. M.; Huang, C.; and Xu, R. 2013. Balance between intensification and diversification: two sides of the same coin. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 10–11.

Luo, C.; Cai, S.; Wu, W.; and Su, K. 2013. Focused random walk with configuration checking and break minimum for satisfiability. In *Proc. of CP 2013*, 481–496.

Luo, C.; Su, K.; and Cai, S. 2012. Improving local search for random 3-SAT using quantitative configuration checking. In *Proc. of ECAI 2012*, 570–575.

Manthey, N. 2012. Coprocessor 2.0 - a flexible CNF simplifier. In *Proc. of SAT 2012*, 436–441.

Manthey, N. 2013. The SAT solver Riss3g at SC 2013. In *Proc. of SAT Competition 2013: Solver and Benchmark Descriptions*, 72–73.

Mertens, S.; Mézard, M.; and Zecchina, R. 2006. Threshold values of random *K*-SAT from the cavity method. *Random Struct. Algorithms* 28(3):340–373.

Mézard, M. 2003. Passing messages between disciplines. *Science* 301:1685–1686.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proc. of AAAI 1994*, 337–343.

Selman, B.; Levesque, H. J.; and Mitchell, D. G. 1992. A new method for solving hard satisfiability problems. In *Proc. of AAAI 1992*, 440–446.

Selman, B. 1995. Stochastic search and phase transitions: AI meets physics. In *Proc. of IJCAI 1995*, 998–1002.

Thornton, J.; Pham, D. N.; Bain, S.; and Ferreira Jr., V. 2004. Additive versus multiplicative clause weighting for SAT. In *Proc. of AAAI 2004*, 191–196.

Tompkins, D. A. D., and Hoos, H. H. 2010. Dynamic scoring functions with variable expressions: New SLS methods for solving SAT. In *Proc. of SAT 2010*, 278–292.

Tompkins, D. A. D.; Balint, A.; and Hoos, H. H. 2011. Captain Jack: New variable selection heuristics in local search for SAT. In *Proc. of SAT 2011*, 302–316.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2012. Predicting satisfiability at the phase transition. In *Proc. of AAAI 2012*, 584–590.