STAIRS 2014
U. Endriss and J. Leite (Eds.)

doi:10.3233/978-1-61499-421-3-11

A Two-Levels Local Search Algorithm for Random SAT Instances with Long Clauses

André ABRAMÉ, Djamal HABET and Donia TOUMI
Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, 13397, Marseille, France

Abstract. We present a local search algorithm exploiting two efficient solvers for SAT. The first one is based on the configuration checking strategy and the second one on an algorithm of the `Walksat` family. This new solver is dedicated to solve random k-SAT instances, such that $k \geq 4$. We have carried tests on the instances of the SAT Challenge 2012. The obtained results confirm the relevance of our approach.

Keywords. Configuration Checking, Novelty Heuristic, Random large k-SAT instances

1. Introduction

The satisfiability problem (SAT) consists of testing whether all clauses in a propositional formula F, in the Conjunctive Normal Form on a set of Boolean variables, can be satisfied by an assignment of truth values to its variables. The incomplete methods for SAT solving are generally based on the Stochastic Local Search (SLS). Starting by a randomly generated truth assignment of the variables of F, an SLS algorithm explores the search space by trying, at each step, to minimize the number of falsified clauses by flipping the truth value of a given variable. In dynamic local search algorithms [17], each clause has a dynamic weight and a variable is evaluated regarding its score. The score of a variable is the variation of the sum of the weights of the falsified clauses, if it is flipped. In the last years, the Configuration Checking (CC) strategy appears as a promising dynamic local search approach to solve SAT [4,6]. The main purpose of CC is to prevent cycling in local search by considering neighbors of variables. Two variables are neighbors if they appear in the same clause at least once. The configuration of a variable is the set of its neighbor variables and their corresponding truth values.

In this paper, we propose to improve the configuration checking strategy on random k-SAT such that $k \geq 4$. This improvement is carried out by using the `Novelty` heuristic with adaptive noise setting. `Novelty` is a powerful SLS algorithm from the `Walksat` family algorithms [11,16,19]. Our motivation is also to enhance the efficiency of diversification and intensification phases in CC-based solvers. The result of our work is a two-level SLS algorithm which we name `Ncca+`. The first stage in `Ncca+` consists of flipping a configuration changed decreasing variable (a variable with a positive score and hav-

ing the value of one of its neighbor variables changed since its last flip) [6], if it exists. Otherwise, `Ncca+` enters its second stage and applies a `Novelty+` like heuristic to select the variable to flip. Our algorithm is implemented on the basis of the powerful CC-based solver `CCASat` [4] which is the winner of the random SAT track of the SAT Challenge 2012[1]. We evaluate empirically `Ncca+` on the instances of this challenge regarding its robustness and its running time by a comparison to `CCASat`. We also compare `Ncca+` to other performing state-of-the-art SLS solvers. This empirical study confirms the efficiency of our solver. `Ncca+` also participated in the SAT competition 2013 [9] and won the bronze medal of the random SAT track[2].

The paper is organized as follows. Section 2 gives the necessary background and elements for the rest of the paper. Section 3 describes in details our contribution. Section 4 is dedicated to the experimental evaluation. Finally, we conclude in Section 5.

2. Preliminaries

2.1. Definitions and Notations

An instance F of the satisfiability problem (SAT) is defined by a pair $F = (X, C)$ such that $X = \{x_1 \cdots x_n\}$ is a set of n Boolean variables and $C = \{c_1 \cdots c_m\}$ is a set of m clauses. A clause $c_i \in C$ is a finite disjunction of literals and a literal is either a variable x_i or its negation $\neg x_i$. Two variables occurring in the same clause are neighbors. The set of the neighbors of x_i is denoted by $N(x_i)$. The size of a clause c_i is the number of its literals (denoted by $|c_i|$). If the size of each clause in C is equal to k then the instance is called $k-$SAT. An assignment is a mapping $I : X \rightarrow \{True, False\}$. I is complete if it maps all the variables of F. A clause $c_j \in C$ is satisfied by a complete assignment I iff it contains at least one satisfied literal, otherwise c_j is falsified. A model of F is an assignment that satisfies all the clauses of F. The satisfiability problem (SAT) consists of deciding if F has a model. If this is the case then F is satisfiable, otherwise F is unsatisfiable.

2.2. Stochastic Local Search for SAT

For a given CNF formula F, a basic Stochastic Local Search (SLS) algorithm for SAT starts by randomly generating a complete assignment I which may falsify some clauses. Hence, it attempts to minimize the number of falsified clauses by repeatedly repairing this assignment by flipping the value of one variable at once (changing its value from $false$ to $true$, or $true$ to $false$) until satisfying all clauses or reaching a cutoff time. In a dynamic SLS algorithm for SAT, a positive weight $w(c_j)$ is associated to each clause c_j in C and the evaluation of an assignment I is the sum of the weights of the clauses falsified under I, which we denote by $E(I)$. The score of a variable x_i is defined by $score(x_i) = E(I) - E(I_{x_i})$ where I_{x_i} is the complete assignment obtained by flipping x_i in I. If $score(x_i) > 0$ then x_i is called a decreasing variable.

SLS algorithms for SAT differ by their employed heuristic to choose the variable to flip. In this paper, we are interested in the heuristic used in `Novelty`, which is based on the `Walksat` algorithm architecture [16,19]. `Novelty`(p) selects randomly a falsified

[1] `baldur.iti.kit.edu/SAT-Challenge-2012/`

[2] `www.satcompetition.org/2013/`

clause c_j. Then it sorts the variables of c_j according to their scores breaking ties in favor of the least recently flipped variable and considers the two best variables under this sorting. If the best variable is not the most recently flipped one in c_j then `Novelty`(p) selects it for flipping. Otherwise, with probability p, it picks the second best one, and with probability $1-p$, it picks the best variable. When `Novelty`(p) gets stuck in local minima, a diversification phase is introduced in `Novelty+`(p,wp) [10] and `Novelty++`(p,dp) [12] to escape from such regions of the search space. With a probability wp, `Novelty+`(p,wp) picks randomly a variable from c_j and with probability $1-wp$ it does like `Novelty`(p). With a probability dp, `Novelty++`(p,dp) picks the least recently flipped variable in c_j and with probability $1-dp$ it acts like `Novelty`(p). In the adaptive versions of these algorithms, the values of p, wp and dp are adjusted depending on the evolution of the search. This adaptive noise setting was first introduced by Hoos [11] and has been applied in other works (see for instance [12]).

2.3. Configuration Checking for SAT

The Configuration Checking (CC) strategy for SAT considers during the search the relations between variables and their neighbors. It defines the configuration $C(x_i)$ of a variable x_i by a subset of I which is restricted to the variables of $N(x_i)$. If a variable in $C(x_i)$ has been flipped since the last flip of x_i then $C(x_i)$ is *changed*.

A typical CC-based algorithm for SAT follows the general scheme of an SLS algorithm. Its variable selection heuristic attempts first to flip those decreasing variables (with positive scores) that have their configurations changed [6]. Such variables are called *Configuration Changed Decreasing* (CCD). X_{CCD} is the set of CCD variables. Hence, a CC-based algorithm forbids the flip of a variable x_i if its configuration $C(x_i)$ has not changed since the last flip of x_i. If there is no CCD variable ($X_{CCD} = \emptyset$), an aspiration criterion is employed. The one defined by Cai and Su [6] selects a *significant decreasing variable* (SD) to be flipped. A variable x_i is SD if $score(x_i)$ is greater than some threshold g, $g > 0$. In practice, the value of g is equal to the average of clause weights in C and denoted by $\overline{w}$. We use X_{SD} to denote the set of the SD variables.

One of the most powerful algorithms based on the CC strategy is `CCASat` [7]. It selects the variable to flip as follows: if $X_{CCD} \neq \emptyset$ then it selects the CCD variable with the highest score. Else, if $X_{SD} \neq \emptyset$ then it chooses the SD variable with the highest score. In these two cases, `CCASat` is in an intensification/greedy phase. Otherwise (both X_{CCD} and X_{SD} are empty), it enters in the diversification phase which consists of choosing the least recently flipped variable appearing in a falsified clause selected randomly. We name such variables *focused random ones*.

3. The Ncca+ Algorithm for Random k-SAT Instances with $k \geq 4$

In this section, we start by giving some experimental observations on one of the most powerful CC-based solver, `CCASat` [7]. These observations have been performed on 480 k-SAT instances (120 instances for each k value in $\{4 \ldots 7\}$) of the SAT Challenge 2012. We have limited the running time of `CCASat` to 300 seconds per instance. We are interested in the nature of the flipped variables (CCD or SD). We have measured the average rates of flips of CCD variables (S_{CCD}), SD ones (S_{SD}) and focused random ones (S_{FR}).

The results are summarized in Table 1. On the whole, we can observe that 77.63% of flips are done on X_{CCD} variables and only 1.5% of flips are done on X_{SD} variables (this rate is only 0.25% for the 7-SAT instances). Finally, the focused random walk is applied for 20.87% of the performed flips and ranges from 12.06% for the 4-SAT instances to 28.47% for the 7-SAT instances.

Table 1. Observations on the flips done in `CCASat` on the k-SAT instances, $k \geq 4$, of the SAT Challenge 2012.

k	S_{CCD}	S_{SD}	S_{FR}
4	83.59	4.35	12.06
5	81.97	1.01	17.02
6	73.68	0.40	25.91
7	71.28	0.25	28.47
Average	**77.63**	**1.50**	**20.87**

According to these observations, it is clear that the aspiration criterion (which corresponds to the flip of SD variables) is rarely applied. Let us recall that this criterion selects a variable to flip with a score greater than some threshold which is the average of the clause weights, $\overline{w}$. We have measured the value of this threshold for the above instances and observed that this value remains around 1 which means that the weights of clauses do not vary significantly during the search. Hence, if there is no CCD variable, it is hard to have an SD one. These elements may explain the low values of S_{SD}.

To obtain a more accurate aspiration phase and to balance the use of the diversification phase, we propose to replace the selection of SD variables by the heuristic used in `Novelty`. Indeed, instead of selecting a SD variable when $X_{CCD} = \emptyset$, we select a variable appearing in a falsified clause selected randomly according to the `Novelty` heuristic. However, the variable scores used in `Novelty` are those obtained by considering the weights of clauses (in the original version of `Novelty`, the score of a variable x_i is equal to the number of falsified clauses which will become satisfied if x_i is flipped minus the number of satisfied clauses which will become falsified if x_i is flipped). Such adaptation of `Novelty` is close to the one used in `gNovelty+` [18].

Accordingly, we modify the heuristic used in `CCASat` to consider the integration of `Novelty`. The resulting algorithm is called `Ncca+`. It works as follows:

1. If the set of configuration changed decreasing variables is not empty ($X_{CC} \neq \emptyset$) then `Ncca+` selects a variable among X_{CC} of the highest score breaking ties in favor of the variable with the highest subscore.
2. Else, `Ncca+` updates the weights of the clauses of F according to PAWS scheme [20]. In PAWS, all clause weights are initialized to 1. Hence, with probability sp (smooth probability) and for each satisfied clause whose weight is greater than 1, PAWS decreases the weight of this clause by 1. Otherwise (with probability $1-sp$), PAWS increases by 1 the weights of all the falsified clauses.
3. With a probability dp (diversification probability), `Ncca+` selects a variable to flip according to `Novelty`(p) heuristic. Otherwise (with a probability $1-dp$), `Ncca+` selects the oldest variable in a falsified clause selected randomly.

Algorithm 1. `Ncca+` Algorithm for k-SAT instances, $k \geq 4$

Input: k-SAT formula F, $maxTries$, $maxSteps$
Output: A Satisfying assignment I, if F is SAT, or "Unknown"
for try =1 to $maxTries$ **do**
  $I \leftarrow$ randomly generated truth assignment
  Initialize clause weights to 1
  Initialize p and dp 0
  for step =1 to $maxSteps$ **do**
    if I satisfies F **then**
      return I
    end if
    $x_i \leftarrow$ `Pick_Var`(p, dp)
    Update `Novelty` probability parameters p and dp
    $I \leftarrow I$ with x_i flipped
  end for
end for
return "Unknown"

We give the general scheme of `Ncca+` in Algorithm 1. It starts by generating randomly a complete truth assignment I and while I does not satisfy the input formula F or a maximum number of flips $maxSteps$ is not reached, `Ncca+` selects a variable to flip following the heuristic detailed before.The values of the probabilities p and dp (used in `Pick_Var` function) are dynamically updated [11,13] (it is based on two parameters Φ and Θ to control the change of values of p and dp). The selected variable is flipped. All these steps (the inner loop of Algorithm 1) are repeated up to $maxTries$ (the outer loop).

4. Experimental Evaluation

This section is dedicated to the experimental evaluation of `Ncca+`. The evaluation is done on 480 random k-SAT instances (all satisfiable) from the SAT Challenge held in 2012[3]. The values of k are ranging from 4 to 7 with 120 instances per k value. Each set is also divided into 10 subsets of 12 instances with different sizes (regarding the number of the variables and the clauses). Table 2 gives the characteristics of these instances. Balint et al. [1] detail the generation and selection of these instances. We have selected these instances because they have different sizes and difficulties. The instances of the SAT Competition 2011, particularly the 5-SAT ones, are treatable rather easily by the current SLS SAT solvers. Thus, we do not include them in these experiments.

`Ncca+` is implemented in C/C++ and compiled with `g++` with the compilation flags `-static -O3`. The experiments are made on a cluster of servers equipped with Intel Xeon 2.4 Ghz processors and 24 GB of RAM and running under a GNU/Linux operating system. The smooth probability sp of the PAWS scheme is $sp = 0.75$ for k-SAT with $k \in \{4,5\}$ and $sp = 0.92$ for k-SAT with $k \in \{6,7\}$ [4].

[3] `baldur.iti.kit.edu/SAT-Challenge-2012/downloads.html`

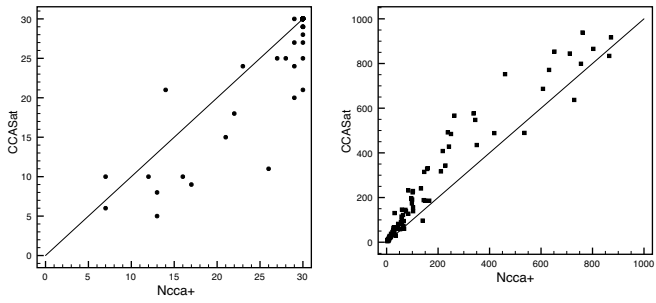
Table 2. The minimum (*min*.) and the maximum (*max*.) number of variables (n) and of clauses (m) of the k-SAT instances, $k \geq 4$, of the SAT Challenge 2012.

	4-SAT		5-SAT		6-SAT		7-SAT	
	n	m	n	m	n	m	n	m
max.	10000	90000	1600	32000	400	16000	200	17000
min.	800	7945	300	6335	200	8674	100	8779

4.1. Ncca+ vs. CCASat

Since Ncca+ is based on CCASat, we have first performed a detailed comparison of the two solvers. For this purpose, each solver is launched 30 times on each instance. Each launch terminates when a solution is found or the cutoff time of 1000 seconds is reached[4]. For each instance, we calculate the number of successful runs and the average runtime to reach a solution. If a run failed to solve an instance, a penalized time of 1000 seconds is used to compute the average time. We use two types of graphics to compare the two solvers: the first one compares the number of successful runs and consequently the solver robustness. The second one compares the average runtimes. We give and discuss these two graphs for each $k = 4 \cdots 7$ (Fig. 1 to 4) where each point in the graphs correspond to one instance. However, some plotted points may overlap if the results of the two solvers are equal or close. Finally, the parameter values of the adaptive noise mechanism are $\Phi = 10$ and $\Theta = 5$. These values are those used in competitive SLS solvers, such as TNM [13] and Sattime [14].

Random 4-SAT (Fig. 1) For these instances, we find the role of Novelty in the improvement of the robustness and the speed of CCASat. Indeed, Ncca+ enhances the robustness of the last solver on 22 instances. Also, the right graphic of Fig. 1 indicates clearly the reduction of the running time of CCASat to reach a solution. This observation is confirmed by the comparison of the average runtimes of the two solvers over the 120 4-SAT instances: 129 seconds for Ncca+ and 179 seconds for CCASat. For the instances with 3800 to 10000 variables, this time is almost divided by 2.

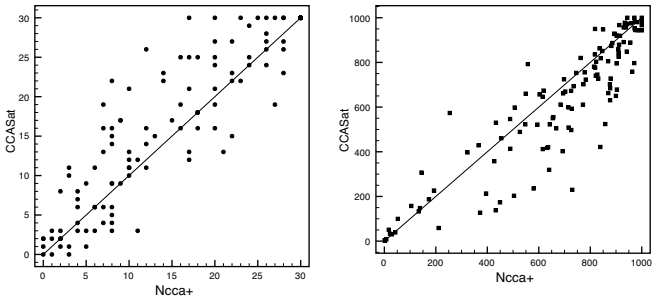


(a) Number of successful runs (b) Solving time

Figure 1. Ncca+ vs. CCASat on random 4-SAT instances.

[4]The used cutoff time during the SAT Challenge was 900 seconds under Intel Xeon 2.83 Ghz processors.

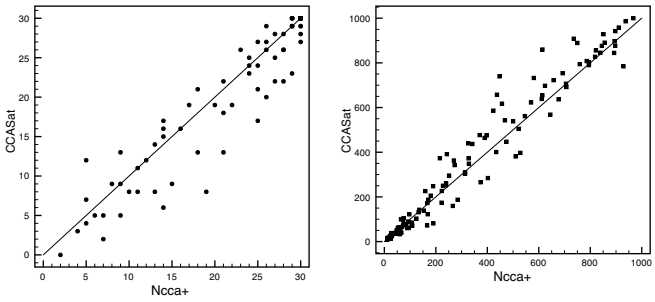
Random 5-SAT (Fig. 2) CCASat seems to be better particularly regarding the robustness criterion. It has a better success rate on 55 instances while Ncca+ is better on 40 instances. However, the average number of successful runs are very close: 15.87 successful runs for CCASat and 16.70 for Ncca+. Also, Ncca+ solved all the instances, at least once, while CCASat failed to solve 3 instances. Concerning the average runtimes over all the instances, the values are 650 and 620 seconds for Ncca+ and CCASat respectively. Hence, the last solver is 5% faster than the first one.



(a) Number of successful runs (b) Solving time

Figure 2. Ncca+ vs. CCASat on random 5-SAT instances.

Random 6-SAT (Fig. 3) Ncca+ seems to be better. Indeed, CCASat is more robust on 21 instances while Ncca+ outperforms CCASat on 33 instances. Also, over all the runs, Ncca+ successfully solved all the instances while CCASat failed to solve 1 instance. The average runtimes are 344 seconds for Ncca+ against 362 seconds for CCASat. Ncca+ solves better the instances with 200 to 300 variables, while CCASat is better on the instances with 320 and 340 variables.

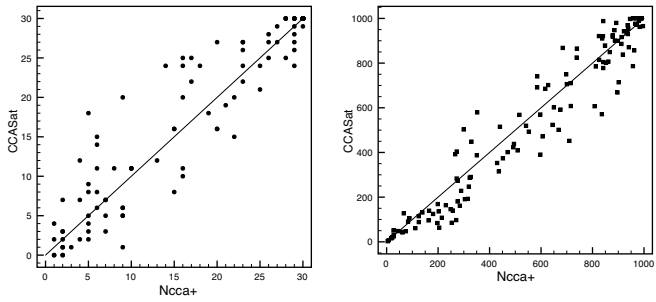


(a) Number of successful runs (b) Solving time

Figure 3. Ncca+ vs. CCASat on random 6-SAT instances.

Random 7-SAT (Fig. 4) For these instances, the results are mixed. Indeed, CCASat is more robust on 43 instances and Ncca+ on 38 instances. The average runtimes are 519 seconds for CCASat and 541 seconds for Ncca+. However, Ncca+ does better than

CCASat by solving all the instances at least one time. CCASat failed to solve 4 instances. The average number of successful runs of the two solvers over all the instances are very close: 18.57 for Ncca+ and 19.08 for CCASat.



(a) Number of successful runs (b) Solving time

Figure 4. Ncca+ vs. CCASat on random 7-SAT instances.

Over all these first results, Ncca+ seems generally better than CCASat regarding the number of solved instances. Concerning the robustness, the improvements provided by Ncca+ are clearly visible for the 4-SAT and 6-SAT instances. The same observation remains true concerning the running times. For the 5-SAT and 7-SAT instances, the results are more mixed. Nevertheless, Ncca+ solved all the instances while CCASat never reached a solution for some of them.

4.2. Ncca+ vs. State-of-the-Art SLS Solvers

In this section, we compare Ncca+ to other powerful SLS solvers including TNM [13], Sparrow2011 [2], CScoreSAT2013 [5], Sattime2013 [15], ProbSAT2013 [3] and CCASat. The solvers labelled by 2013 are taken from the SAT Competition 2013[5]. Sparrow2011 and ProbSAT2013 are the winners of the gold medal of the random SAT track of the SAT competitions 2011 and 2013 respectively. CCASat was the winner of this same category during the SAT Challenge 2012. CScoreSAT2013 is a powerful solver based on configuration checking. Sattime regularly won medals during SAT competitions of the same track. The comparison to TNM and Sattime is also motivated by the fact that these two solvers use Novelty and the set of promising decreasing variables [12] which is a subset of configuration changed decreasing variables [6].

All the solvers are run on the random instances of the SAT Challenge 2012. The cutoff time is 3600 seconds to observe the behavior of the solvers with a higher execution time than the one used during this challenge. We run each solver one time for each instance. Table 3 details the results for each k-SAT problem. It appears that Ncca+ improves CCASat for each k value, except for $k = 7$ for which CCASat solves one instance more.

[5] Available from www.satcompetition.org/2013/

Table 3. Detailed results on the 7 solvers. For each $k = 3 \cdots 7$, we give the number of solved instances by the solvers. A number in bold indicates that the solver outperforms its challengers. The numbers between brackets are the average times to find a solution for the instances in the k-SAT sets. If a solver fails to reach a solution then its runtime is penalized by 3600 seconds.

k	Ncca+	CCASat	Prob-SAT2013	CScore-SAT2013	Sattime-2013	Sparrow-2011	TNM
4	**120**	119	119	118	61	93	84
	(220)	(246)	(102)	(253)	(2250)	(1125)	(1320)
5	96	94	84	**99**	68	76	41
	(1380)	(1357)	(1763)	**(1227)**	(2221)	(1883)	(2819)
6	**113**	108	101	**113**	110	87	105
	(599)	(755)	(1215)	**(490)**	(863)	(1688)	(1253)
7	103	**104**	81	96	97	86	95
	(1090)	**(1154)**	(1833)	(1192)	(1183)	(1503)	(1366)

For $k = 4$, Ncca+ remains better than the other solvers. For $k = 5$, CScoreSAT2013 solves 3 instances more than Ncca+ and 5 instances more than CCASat. For $k = 6$, Ncca+ and CScoreSAT2013 solve the highest number of instances (113). For $k = 7$, CCASat solves only one instance more than Ncca+ which seems to be faster. Ncca+ is clearly better than Sattime and TNM. These two last solvers also integrate the Novelty++ algorithm and they have better performances than this algorithm. Regarding all the SAT instances, Ncca+ is the better solver by solving 432 instances, the second and the third best ones are CScoreSAT2013 and CCASat which solve 426 and 425 instances respectively. We would like to note that the current scheme of Ncca+ is close to AdaptG2Wsat2009++ which alternates between greedy search thanks to the promising decreasing variables and diversification thanks to Novelty++ with adaptive noise setting [12]. We have not included the results of AdaptG2Wsat2009++ because it is outperformed by the recent solvers of the same authors, such as Sattime and TNM.

5. Conclusion

In this paper, we have presented a competitive and robust algorithm Ncca+ dedicated to random k-SAT instances with long clauses. It combines the configuration checking (CC) strategy and a heuristic similar to Novelty with adaptive noise setting. Ncca+ improved the intensification and the diversification phases used in CC-based solvers. The empirical evaluation accomplished on random instances of the SAT Challenge 2012 confirmed our purpose and the bronze medal obtained in the SAT Competition 2013 consolidated this evaluation [6].

[6] The detailed results of the SAT Competition 2013 are available from http://satcompetition.org/2013

References

[1] Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Sat challenge 2012 random sat track: Description of benchmark generation. In: Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, pp. 72–73 (2012)

[2] Balint, A., Fröhlich, A.: Improving stochastic local search for sat with a new probability distribution. In: Proceedings of SAT'10, pp. 10–15 (2010)

[3] Balint, A., Schöning, U.: Probsat. In: Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions, p. 70 (2013)

[4] Cai, S., Luo, C., Su, K.: Ccasat: Solver description. In: Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, pp. 13–14 (2012)

[5] Cai, S., Luo, C., Su, K.: Cscore2013. In: Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions, pp. 18–19 (2013)

[6] Cai, S., Su, K.: Configuration checking with aspiration in local search for sat. In: Proceedings of AAAI-2012, pp. 434–440 (2012)

[7] Cai, S., Su, K.: Local search for boolean satisfiability with configuration checking and subscore. Artif. Intell. **204**, 75–98 (2013)

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

[9] Habet, D., Toumi, D., Abramé, A.: Ncca+: Configuration checking and novelty+ like heuristic. In: Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions, p. 62 (2013)

[10] Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for sat. In: Proceedings of AAAI '99/IAAI '99, pp. 661–666 (1999)

[11] Hoos, H.H.: An adaptive noise mechanism for walksat. In: Proceedings of AAAI-2002, pp. 655–660 (2002)

[12] Li, C.M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. In: Proceedings of SAT'05, pp. 158–172 (2005)

[13] Li, C.M., Huang, W.Q.: Switching between two adaptive noise mechanisms in local search for sat. In: SAT 2009 competitive events booklet, p. 57 (2009)

[14] LI, C.M., LI, Y.: Satisfying versus falsifying in local search for satisfiability. In: Proceedings of SAT-2012, pp. 477–478. Springer (2012)

[15] Li, C.M., Li, Y.: Description of sattime2013. In: Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions, pp. 77–78 (2013)

[16] McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of AAAI-1997, pp. 321–326 (1997)

[17] Morris, P.: The breakout method for escaping from local minima. In: Proceedings of AAAI'93, pp. 40–45. AAAI Press (1993)

[18] Pham, D.N., Thornton, J., Gretton, C., Sattar, A.: Combining adaptive and dynamic local search for satisfiability. JSAT **4**(2-4), 149–172 (2008)

[19] Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of AAAI-94, pp. 337–343 (1994)

[20] Thornton, J., Pham, D.N., Bain, S., Ferreira, V.: Additive versus multiplicative clause weighting for sat. In: Proceedings of AAAI'04, pp. 191–196. AAAI Press (2004)