# Improving Configuration Checking for Satisfiable Random $k$-SAT Instances

**André Abramé** and **Djamal Habet** and **Donia Toumi**
Aix-Marseille University – LSIS UMR CNRS 7296 – France
{Andre.Abrame,Djamal.Habet,Donia.Toumi}@lsis.org

**Abstract**

Local search methods based on Configuration Checking (CC) have shown their efficiency in solving random $k$-SAT instances. The main idea behind CC is to consider the neighborhood of variables in the SAT formula to avoid cycling problem in local search. In this paper, we propose to improve CC strategy thanks to the powerful local search algorithm which is `Novelty` with adaptive noise setting. We also introduce a new and simple criterion to refine the selection of the variables to flip. We name the resulting solver `Ncca+` and show its effectiveness when treating satisfiable random $k$-SAT instances. Besides, `Ncca+` is the winner of the bronze medal of the random SAT track of the SAT Competition 2013.

## Introduction

Considering a propositional formula $\mathcal{F}$ in Conjunctive Normal Form (CNF) on a set of Boolean variables $\{x_1, \ldots, x_n\}$, the satisfiability problem (SAT) consists in testing whether all clauses in $\mathcal{F}$ can be satisfied by an assignment of truth values to the variables of $\mathcal{F}$. SAT is the first known (Cook 1971) and one of the most studied NP-complete problems. It has many applications like graph coloring, circuit designing or planning, since such problems can be encoded into CNF formulas in a natural way.

The incomplete methods dedicated to solve SAT are mainly based on Stochastic Local Search (SLS). Starting by a randomly generated truth assignment of the variables of $\mathcal{F}$, a 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 variable selected according to a given heuristic (see examples in (Hoos 2002; Li and Huang 2005; McAllester, Selman, and Kautz 1997; Selman, Kautz, and Cohen 1994)). Besides, a local search algorithm for SAT must struck a balance between two phases which are intensification and diversification. The intensification is a greedy step where local search aims at improving strictly the current assignment regarding the number of satisfied clauses. The diversification is designed to explore well the search space and prevent the search from being stuck in local minima. In dynamic local search methods (Morris 1993), a positive weight is associated to each clause. Accordingly, a variable is generally evaluated regarding its score which measures the variation of the sum of the weights of the falsified clauses, if this variable is flipped. If this variation is positive then the corresponding variable is said to be *decreasing*.

Recently, Configuration Checking (CC) strategy appears as a promising dynamic local search approach to solve SAT (Cai, Luo, and Su 2012; Cai and Su 2012). The first purpose of CC is to prevent cycling in local search by considering neighbors of variables when looking for the one to flip. The neighbors of a variable $x_j$ are the variables appearing in the same clauses as $x_j$. In a CC-based solver, $x_j$ is a candidate for flipping only if one of the variables in its neighborhood was flipped since the last flip of $x_j$. This strategy is applied in a greedy step to select a decreasing variable to flip. This step is coupled with a diversification process which consists generally in fostering the least flipped variables which appear in a randomly selected falsified clause.

In this paper, we propose to improve the use of configuration checking strategy on random $k$-SAT instances. When $k \geq 4$, this improvement is done thanks to `Novelty` with adaptive noise setting. `Novelty` is a powerful SLS algorithm issued from the `Walksat` family (Hoos 2002; McAllester, Selman, and Kautz 1997; Selman, Kautz, and Cohen 1994). Our motivation is also to enhance the efficiency of diversification and intensification phases in CC-based solvers. When $k = 3$, we introduce a simple criterion based on the number of occurrences of variables in falsified clauses to refine the selection of the variable to flip. The result of our work is a new SLS algorithm which we name `Ncca+`. Our solver is implemented on the basis of the powerful CC solver `CCASat` (Cai, Luo, and Su 2012) 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 such as `probSAT2013` (Balint and Schöning 2013), `Sparrow2011` (Balint and Fröhlich 2010), `CScoreSAT2013` (Cai, Luo, and Su 2013) and `Sattime2013` (Li and Li 2013). This empirical study confirms the efficiency of our solver. `Ncca+` also participated in the SAT competition 2013 (Habet, Toumi, and Abramé

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

2013) and won the bronze medal of the random SAT track[2].

The paper is organized as follows. Firstly, we give the needed definitions, notations and background. Particularly, we recall the working scheme of Novelty and configuration checking based solvers. Secondly, we describe our contribution by detailing our algorithm Ncca+. Thirdly, we detail the experimental evaluation of Ncca+ by comparing it to powerful SLS solvers. Eventually, we conclude.

## Preliminaries

### Definitions and Notations

An instance $\mathcal{F}$ of the satisfiability problem (SAT) is defined by the couple $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ such that $\mathcal{X} = \{x_1 \cdots x_n\}$ is a set of $n$ Boolean variables (their values belong to the set $\{true,\ false\}$) and $\mathcal{C} = \{c_1 \cdots c_m\}$ is a set of $m$ clauses. A clause $c_i \in \mathcal{C}$ is a finite disjunction of literals and a literal is either a variable ($x_i$) or its negation ($\neg x_i$). Also, a clause can be represented by the set of its literals and $var(c_i)$ is the set of variables appearing in $c_i$. The size of a clause $c_i$ is the number of its literals and it is denoted by $|c_i| = |var(c_i)|$. If the size of each clause in $\mathcal{C}$ is equal to $k$ ($\forall c_i \in \mathcal{C},\ |c_i| = k$) then the instance is said $k-$SAT.

A truth assignment $\mathcal{I}$ of the variables of $\mathcal{F}$ is a set of literals such as $\forall\{l_1,\ l_2\} \subseteq \mathcal{I}, var(l_1) \neq var(l_2)$. $\mathcal{I}$ is partial if $|I| < |\mathcal{X}|$ and complete if $|I| = |\mathcal{X}|$ (all the variables are fixed). A variable that appears positively (resp. negatively) in $\mathcal{I}$ is said to be fixed to $true$ (resp. $false$). A clause $c_j \in \mathcal{C}$ is satisfied by $\mathcal{I}$ iff it contains at least one satisfied literal, otherwise $c_j$ is falsified by $\mathcal{I}$ (if $\mathcal{I}$ is complete).

Two variables occurring in a same clause (at least once) are neighbors. The set of neighbors of $x_i$ is denoted by $\mathcal{N}(x_i)$. Furthermore, for a subset $\mathcal{X}' \subset \mathcal{X}$ and an assignment $\mathcal{I}$, $\mathcal{I}[\mathcal{X}']$ is the projection of $\mathcal{I}$ on the variables of $\mathcal{X}'$. A model of $\mathcal{F}$ is an assignment that satisfies all clauses of $\mathcal{F}$. Finally, the satisfiability problem (SAT) consists in deciding if $\mathcal{F}$ has a model. If this is the case then $\mathcal{F}$ is satisfiable, otherwise $\mathcal{F}$ is unsatisfiable.

### (Dynamic) Stochastic Local Search for SAT

For a given CNF formula $\mathcal{F}$, a basic Stochastic Local Search (SLS) algorithm for SAT starts by randomly generating a complete assignment $\mathcal{I}$ 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 $\mathcal{C}$ and the quality of an assignment $\mathcal{I}$ is the sum of the weights of the clauses falsified under $\mathcal{I}$, which we denote by $\mathcal{Q}(\mathcal{I})$. The score of a variable $x_i$ is defined by $score(x_i) = \mathcal{Q}(\mathcal{I}) - \mathcal{Q}(\mathcal{I}_{x_i})$ where $\mathcal{I}_{x_i}$ is the complete assignment obtained by flipping $x_i$ in $\mathcal{I}$. 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 issued from Walksat (McAllester, Selman, and Kautz 1997; Selman, Kautz, and Cohen 1994). Novelty$(p)$ selects randomly a falsified 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+$(p, wp)$ (Hoos 1999) and Novelty++$(p, dp)$ (Li and Huang 2005) to escape from such regions of the search space:

- With probability $wp$, Novelty+$(p, wp)$ picks randomly a variable from $c_j$ and with probability $1-wp$ it does like Novelty$(p)$.
- With 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 version of these algorithms, the values of $p$, $wp$ and $dp$ are adjusted depending on the evolution of the solving process. This adaptive noise setting is first introduced in (Hoos 2002) and applied in other works such as (Li and Huang 2005).

### Configuration Checking for SAT

Configuration Checking (CC) is initially introduced in (Cai, Su, and Sattar 2011) to deal with the minimum vertex cover problem. It aims at avoiding cycling during local search solving (i.e. revisiting the already visited assignments too early). In the SAT context, CC considers during the search the relations between variables expressed by the neighbor relation defined in the section above. Indeed, CC defines the configuration $C(x_i)$ of a variable $x_i$ by a subset of $\mathcal{I}$ which is restricted to the variables of $\mathcal{N}(x_i)$. In other words, we have $C(x_i) = \mathcal{I}[\mathcal{N}(x_i)]$. If a variable in $C(x_i)$ has been flipped since the last flip of $x_i$ then $C(x_i)$ is said *changed*.

CC-based solver attempts to flip decreasing variables (with positive scores) having their configurations changed (Cai and Su 2012). These variables are said *configuration changed decreasing* and $\mathcal{X}_{CC}$ denotes the set of such variables. Hence, CC forbids the flip of a variable $x_i$ if its configuration $C(x_i)$ has not changed since the last flip of $x_i$.

Nevertheless, this restriction may be too strong and lead to an empty $\mathcal{X}_{CC}$ set. In such a situation, an aspiration criterion is introduced by selecting a *significant decreasing variable* verifying $score(x_i) > g$ such as $g > 0$. The value of $g$ is equal to the average of clause weights in $\mathcal{C}$ denoted by $\overline{w}$ (Cai and Su 2012). The set of such variables is denoted by $\mathcal{X}^d$.

The existing powerful solvers based on CC (Cai, Luo, and Su 2012; Cai and Su 2012) first try to select a variable to flip belonging to $\mathcal{X}_{CC}$. If $\mathcal{X}_{CC} = \emptyset$ then they select a variable from $\mathcal{X}^d$. In the other cases, they apply a simple diversification strategy which selects a variable appearing in a falsified clause selected randomly.

[2] www.satcompetition.org/2013/

## Ncca+: a Solver for Random $k$-SAT Problem

This section describes our contribution which improves configuration checking for SAT when dealing with random $k$-SAT instances. We distinguish the 3-SAT case from the other cases. Firstly, we give the heuristic used to select the variable to flip, in each case. Secondly, we describe completely our solver which we named Ncca+.

### Variable Selection in Ncca+ for $k \geq 4$

When dealing with $k$-SAT instances such as $k \geq 4$, the heuristic used in Ncca+ to select a variable to flip acts as follows:

1. If the set of configuration changed decreasing variables is not empty ($\mathcal{X}_{CC} \neq \emptyset$) then Ncca+ selects a variable among $\mathcal{X}_{CC}$ of the highest score breaking ties in favor of the oldest one.
2. Else, Ncca+ updates the weights of the clauses of $\mathcal{F}$ according to PAWS scheme (Thornton et al. 2004). 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. Once the weights are updated and 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. This case is close to Novelty++ and the values of $p$ and $dp$ are dynamically adjusted (Hoos 2002).

Ncca+ replaces the aspiration criterion used in CC-based solvers such as CCASat (Cai, Luo, and Su 2012) and Swcca (Cai and Su 2012) by Novelty heuristic.

Our purpose is to improve both intensification and diversification phases. Indeed, the aspiration above is based on the variables appearing in falsified clauses with a score greater than some threshold. We have measured the rate of the application of this (aspiration) criterion on 420 $k$-SAT ($k \geq 4$) instances used in the experimental evaluation section and we have found that it is applied only around $3\%$ (at average) of the flips done by CCASat. In Ncca+, the number of applications of Novelty($p$) is dynamic and depends on the value of the probability $wp$ which follows the evolution of the search.

In the initial version of Novelty, as for the Waksat variants, the score of a variable is calculated without considering the weights of clauses. Indeed, the score of a variable $x_i$ is based on the number of falsified clauses which will become satisfied if $x_i$ is flipped and the number of satisfied clauses which will become falsified if $x_i$ is flipped. In Ncca+, the score of a variable follows the definition used dynamic SLS algorithms.

### Variable Selection in Ncca+ for $k = 3$

For the 3-SAT instances, Ncca+ uses a heuristic close to the one used in CCASat and Swcca. The heuristic of Ncca+ works as follows:

1. If the set of configuration changed decreasing variables is not empty ($\mathcal{X}_{CC} \neq \emptyset$) then Ncca+ selects a variable among $\mathcal{X}_{CC}$ of the highest score breaking ties in favor of the **variable appearing the most in the falsified clauses.**
2. Else, if $\mathcal{X}^d \neq \emptyset$ then it selects a significant decreasing variable breaking ties in the favor of the oldest variable.
3. Else, the weights of the clauses are updated following the Swcca scheme: the weights of the falsified clauses are increased by 1. If the weight average $\overline{w}$ exceeds a threshold $\gamma$ then all the clause weights are smoothed as $w(c_i) = \rho \times w(c_i) + (1 - \rho) \times \overline{w}$.
4. Once the weights are updated, Ncca+ selects the oldest variable in a falsified clause selected randomly.

The difference between Ncca+ and CCASat/Swcca is that the breaking ties in Ncca+ is in favor of the variable occurring the most in the falsified clauses while it is in favor of the oldest variable in the two other solvers. We now motivate this criterion. Variable scores are calculated according to the weights of clauses and two variables with the same score may satisfy a different number of clauses, if they are flipped. Thus, the progress of the clause weights can lead to a situation where some falsified clauses have very high scores which can hide the real impact of flipping the variables of these clauses regarding the objective which is satisfying all the clauses of the initial formula. Indeed, it could be better to flip a variable which satisfies more clauses than another with the same score but satisfying less clauses. Also, we have measured the rate of flips done by selecting a variable appearing in $\mathcal{X}_{CC}$. This rate ranges generally from $70\%$ to $80\%$ which means that the variables of this set are often flipped and the age of the variables (the iteration number of their flip) is less differentiating in this case, particularly for the 3-SAT instances.

### Ncca+ Algorithm

We give the general scheme of Ncca+ in Algorithm 1. It starts by generating randomly a truth assignment $\mathcal{I}$ and while $\mathcal{I}$ does not satisfy the input formula $\mathcal{F}$ or a maximum number of flips $maxSteps$ is not reached, Ncca+ selects a variable to flip following the heuristics detailed before. For the $k-$SAT instances such as $k \geq 4$, the values of the probabilities $p$ and $wp$ (used in Pick_Var_Large() function) are dynamically updated as in (Hoos 2002; Li and Huang 2009) (it is based on two parameters $\Phi$ and $\Theta$ to control the change of values of $p$ and $dp$). All these steps (the inner loop of Algorithm 1) are repeated up to $maxTries$ (the outer loop).

We have implemented Ncca+ on the basis of CCASat source code[3]. CCASat distinguishes also the treatment of the input $k$-SAT instance according to the value of $k$. Particularly, when $k \geq 4$ CCASat considers other information to choose the variable to flip. This information is related to the difference between the ages of variables candidate to flip and to a subscore function which considers the number of literals which are satisfied in the clauses. We keep these details in the implementation of Ncca+.

[3] Available from shaoweicai.net/research.html

**Algorithm 1:** Ncca+ Solver

**Input:** $k$-SAT formula $\mathcal{F}$, $maxTries$, $maxSteps$.
**Output:** A Satisfying assignment $\mathcal{I}$, if $\mathcal{F}$ is SAT, or "Unknown"

```
for try =1 to maxTries do
    I ← randomly generated truth assignment;
    Initialize clause weights to 1;
    Initialize p and wp 0;
    for step =1 to maxSteps do
        if I satisfies F then return I;
        if k = 3 then
            x_i ← Pick_Var_3-SAT();
        end
        if k ≥ 4 then
            x_i ← Pick_Var_Large();
            Update Novelty noise parameters p and wp;
        end
        I ← I with x_i flipped
    end
end
return "Unknown";
```

## Experimental Evaluation

This section is dedicated to the experimental evaluation of Ncca+. The evaluation is done on 600 random $k$-SAT instances (all satisfiable), which are issued from the SAT Challenge held in 2012[4]. The values of $k$ are ranging from 3 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 1 gives the characteristics of these instances and more details about their generation and selection can be found in (Balint et al. 2012b).

| | 3-SAT | | 4-SAT | | 5-SAT | | 6-SAT | | 7-SAT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\|\mathcal{V}\|$ | $\|\mathcal{C}\|$ | $\|\mathcal{V}\|$ | $\|\mathcal{C}\|$ | $\|\mathcal{V}\|$ | $\|\mathcal{C}\|$ | $\|\mathcal{V}\|$ | $\|\mathcal{C}\|$ | $\|\mathcal{V}\|$ | $\|\mathcal{C}\|$ |
| $max.$ | 40000 | 168000 | 10000 | 90000 | 1600 | 32000 | 400 | 16000 | 200 | 17000 |
| $min.$ | 2000 | 8543 | 800 | 7945 | 300 | 6335 | 200 | 8674 | 100 | 8779 |

Table 1: Characteristics of SAT Challenge 2012 instances.

Ncca+ is implemented in C/C++ and compiled with g++ with the compilation flags `-static -O3`. The experiments are made on blades of servers equipped with Intel Xeon 2.4 Ghz processors and 24 GB of RAM and running under a GNU/Linux operating system. The parameters of clause weighting for 3-SAT instances are $\gamma = 300$ and $\rho = 200 + (|\mathcal{V}| + 250)/500$. 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 $k \in \{6, 7\}$ (Cai, Luo, and Su 2012).

### 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 (fixed to 1000 seconds) is reached. 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 $k = 3 \cdots 7$ (Fig. 1 to 5) 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 very close on several instances. Eventually, the parameter values of the adaptive noise mechanism are $\Phi = 10$ and $\Theta = 5$.

In each figure from 1 to 5, the left graphic compares Ncca+ and CCASat regarding the number of successful runs per instance. The right one compares them regarding the average runtime on the 30 runs.

**Random 3-SAT (Fig. 1)** Ncca+ has generally a better behavior than CCASat. Indeed, it is more robust and needs less time to reach a solution. When examining the results in details and by considering the success rates, Ncca+ is better on 50 instances while CCASat is better on 16 instances. Also, Ncca+ failed to solve only 2 instances while CCASat failed on 4 instances. Over all the 3-SAT instances, the average runtimes of Ncca+ and CCASat are 416 and 483 seconds respectively. Accordingly, it seems that considering the number of occurrences of the variables in falsified clauses gives a significant improvement both in terms of execution time and robustness. Indeed, Ncca+ increased the robustness of CCASat over 41% of the instances. Also, Ncca+ is 14% faster than CCASat.

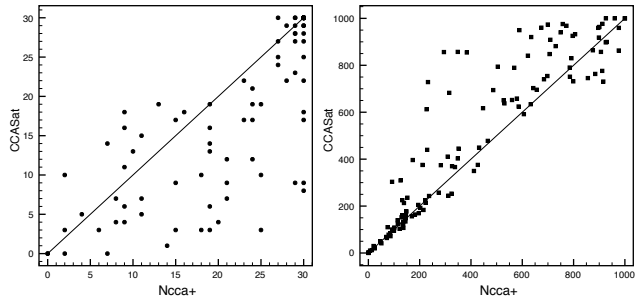


Figure 1: Ncca+ vs. CCASat on random 3-SAT instances.

**Random 4-SAT (Fig. 2)** 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. 2 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.

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

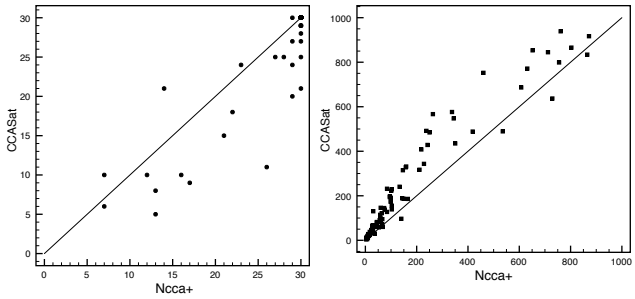


Figure 2: Ncca+ vs. CCASat on random 4-SAT instances.

**Random 5-SAT (Fig. 3)** 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 success rates 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.

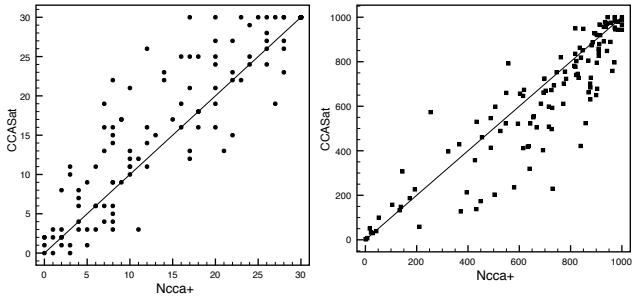


Figure 3: Ncca+ vs. CCASat on random 5-SAT instances.

**Random 6-SAT (Fig. 4)** 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.

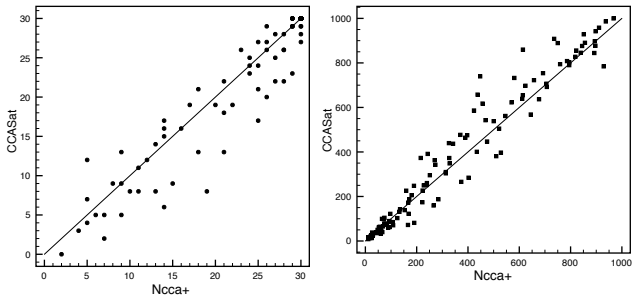


Figure 4: Ncca+ vs. CCASat on random 6-SAT instances.

**Random 7-SAT (Fig. 5)** For these instances, the results are mixed. Indeed, CCASat is more robust on 43 instances and Ncca+ on 38 instances. The average runtimes are 541 seconds for the first solver and 519 seconds for the second one. However, Ncca+ does better than CCASat by solving all the instances at least one time. CCASat failed to solve 4 instances. The average success rates of the two solvers over all the instances are very close: 18.57 for Ncca+ and 19.08 for CCASat.

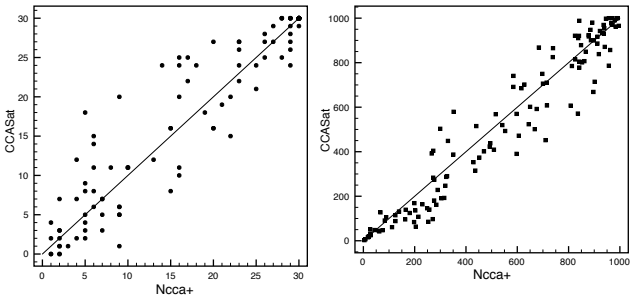


Figure 5: Ncca+ vs. CCASat on random 7-SAT instances.

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 3-SAT, 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.

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

Now, we compare Ncca+ to powerful SLS solvers including TNM (Li and Huang 2009), Sparrow2011 (Balint and Fröhlich 2010), CScoreSAT2013 (Cai, Luo, and Su 2013), Sattime2013 (Li and Li 2013), ProbSAT2013 (Balint and Schöning 2013) and CCASat. The solvers labelled by 2013 are issued 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 use the set of promising decreasing variables which is a subset of configuration changed decreasing variables (Li and Huang 2005; Cai and Su 2012).

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

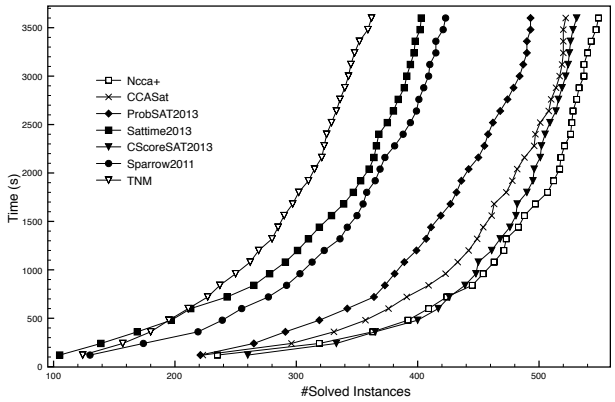


Figure 6: Comparison between Ncca+ and competitive SLS solvers on the instances of SAT Challenge 2012

All the solvers are run on the random instances of SAT Challenge 2012. The cutoff time is 3600 seconds. Fig. 6 shows the results obtained by these solvers by giving the evolution of the number of solved instances during this cutoff time. Ncca+ solved more instances than all the other competitors. Indeed, it solved 22 instances more than CCASat, 14 instances more than CScoreSAT2013 and 52 instances more than probSAT2013.

Table 2 details the results for each $k$-SAT problem ($k = 3 \cdots 7$). It appears that Ncca+ improves CCASat for each $k$ value, except for $k = 7$ for which CCASat solves one instance more.

| $k$ | Ncca+ | CCASat | Prob-SAT2013 | CScore-SAT2013 | Sattime-2013 | Sparrow-2011 | TNM |
|---|---|---|---|---|---|---|---|
| 3 | **113** | 98 | 108 | 105 | 67 | 81 | 57 |
| | **(657)** | (1103) | (759) | (936) | (2094) | (1647) | (2757) |
| 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) |
| Total | **545** | 523 | 493 | 531 | 402 | 423 | 362 |
| | **(789)** | (923) | (1134) | (820) | (1722) | (1589) | (1903) |

Table 2: Detailed results of the 7 solvers which are compared in Fig. 6. For each $k$, we give the number of solved instances and the average times to find a solution (the numbers between brackets). If a solver fails to reach a solution then its runtime is penalized by 3600 seconds.

For $k = 3$, the simple idea of considering the number of occurrences of variables in falsified clauses provides a significant improvement of the performances of CCASat and Ncca+ outperforms the rest of the solvers. 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 which use and outperform Novelty++. Regarding all the $k$-SAT instances such as $k \geq 4$, 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 specify that the current scheme of Ncca+ (for $k$-SAT instances, $k > 3$) is close to AdaptG2Wsat2009++ which alternates between greedy search thanks to the promising decreasing variables and diversification thanks to Novelty++ with adaptive noise setting (Li and Huang 2005). 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.

Ncca+ participated in the SAT Competition 2013 (Habet, Toumi, and Abramé 2013) and won the bronze medal of the random SAT track (Ncca+ used in Fig. 6 is the submitted version to this competition)[6]. During this competition, 280 random SAT instances were used and the cutoff time was about 5000 seconds. Among the 280 instances, a significant part is unsatisfiable and no filtering is applied to delete the UNSAT instances (Balint et al. 2012a). The virtual best solver solves only 135 instances. Ncca+ solved 91 instances, while probSAT2013, the winner of the gold medal, solved 99 instances and Sattime2013, the winner of the silver medal, solved 92 instances. During this competition, 5 solvers based on configuration checking competed and Ncca+ is the best one. The bronze medal obtained in this competition and the results detailed in this section are very encouraging and proves the relevance and the efficiency of our solver.

## Conclusion

In this paper, we have presented a competitive and robust solver Ncca+ dedicated to random $k$-SAT instances. This solver combines the configuration checking (CC) strategy and a heuristic similar to Novelty++ with adaptive noise setting. It also integrates a new criterion based on the number of occurrences of the variables in falsified clauses to improve the variable selection in CC strategy when dealing with the 3-SAT instances. Accordingly, Ncca+ improved the intensification and the diversification phases used in CC-based solvers. The empirical evaluation accomplished on random instances issued from the SAT Challenge 2012 confirmed our purpose and the bronze medal obtained in the SAT Competition 2013 consolidated this evaluation.

In the future, we will extend the experimental evaluation of Ncca+ on crafted instances and improve the articulation between configuration checking mechanism and Walksat like solvers.

[6]Results of SAT Competition 2013 are available from www.satcompetition.org/2013/results.shtml

## References


Balint, A., and Fröhlich, A. 2010. Improving stochastic local search for sat with a new probability distribution. In *Proceedings of the 13th international conference on Theory and Applications of Satisfiability Testing*, SAT'10, 10–15. Berlin, Heidelberg: Springer-Verlag.

Balint, A., and Schöning, U. 2013. Probsat. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, 70.

Balint, A.; Belov, A.; Heule, M. J.; and Järvisalo, M. 2012a. Generating the uniform random benchmarks for sat competition 2013. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, 97–98.

Balint, A.; Belov, A.; Järvisalo, M.; and Sinz, C. 2012b. Sat challenge 2012 random sat track: Description of benchmark generation. In *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, 72–73.

Cai, S., and Su, K. 2012. Configuration checking with aspiration in local search for sat. In *Tewenty-sixth national conference on Artificial intelligence (AAAI-2012)*, 434–440.

Cai, S.; Luo, C.; and Su, K. 2012. Ccasat: Solver description. In *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, 13–14.

Cai, S.; Luo, C.; and Su, K. 2013. Cscore2013. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, 18–19.

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.

Cook, S. A. 1971. The Complexity of Theorem Proving Procedures. In *Proceeding of the Third Annual ACM Symp. on Theory of Computing*, 151–158.

Habet, D.; Toumi, D.; and Abramé, A. 2013. Ncca+: Configuration checking and novelty+ like heuristic. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, 62.

Hoos, H. H. 1999. On the run-time behaviour of stochastic local search algorithms for sat. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, 661–666. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Hoos, H. H. 2002. An adaptive noise mechanism for walksat. In *Eighteenth national conference on Artificial intelligence (AAAI-2002)*, 655–660.

Li, C. M., and Huang, W. Q. 2005. Diversification and determinism in local search for satisfiability. In *Proceedings of the 8th international conference on Theory and Applications of Satisfiability Testing*, SAT'05, 158–172.

Li, C. M., and Huang, W. Q. 2009. Switching between two adaptive noise mechanisms in local search for sat. In *SAT 2009 competitive events booklet*, 57.

Li, C. M., and Li, Y. 2013. Description of sattime2013. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, 77–78.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for invariants in local search. In *Proceedings of AAAI-1997*, 321–326.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, 40–45. AAAI Press.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-94)*, 337–343.

Thornton, J.; Pham, D. N.; Bain, S.; and Ferreira, V. 2004. Additive versus multiplicative clause weighting for sat. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, 191–196. AAAI Press.