J Heuristics (2015) 21:433–456
DOI 10.1007/s10732-015-9284-3

An effective variable selection heuristic in SLS for weighted Max-2-SAT

Shaowei Cai · Zhong Jie · Kaile Su

Received: 4 November 2012 / Revised: 14 October 2014 / Accepted: 19 January 2015 /
Published online: 3 February 2015

Abstract Stochastic local search (SLS) is an appealing method for solving the maximum satisfiability (Max-SAT) problem. This paper proposes a new variable selection heuristic for Max-SAT local search algorithms, which works particularly well for weighted Max-2-SAT instances. Evolving from the recent configuration checking strategy, this new heuristic works in three levels and is called CCTriplex. According to the CCTriplex heuristic, a variable that is both decreasing and configuration changed has the higher priority to be flipped than a decreasing variable, which in turn has the higher priority than a configuration changed variable. The CCTriplex heuristic is used to develop a new SLS algorithm for weighted Max-2-SAT called CCMaxSAT. We evaluate CCMaxSAT on random benchmarks with different densities, and the hand crafted Frb benchmark, as well as weighted Max-2-SAT instances encoded from MaxCut, MaxClique and sports scheduling problems. Compared with the state-of-the-art SLS solver for weighted Max-2-SAT called ITS and the best SLS solver in Max-SAT Evaluation 2012 namely ubcsat-IRoTS, as well as the famous complete solver wMaxSATz, our algorithm CCMaxSAT shows rather good performance on all the benchmarks.

Keywords Local search · Max-2-SAT · Configuration checking

S. Cai (✉)
State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China
e-mail: shaoweicai.cs@gmail.com

Z. Jie
Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China
e-mail: pkutcsj@gmail.com

K. Su
Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
e-mail: k.su@griffith.edu.au

1 Introduction

Given a conjunctive normal form (CNF) formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, the maximum satisfiability (Max-SAT) problem is to find an assignment to the Boolean variables in F to maximize the number of satisfied clauses. Max-SAT is NP-hard even if every clause contains at most two literals (called the Max-2-SAT problem). In the weighted version of Max-SAT, each clause is associated with a positive number as its weight, and the goal is to find an assignment to maximize the total weight of satisfied clauses, or equally, to minimize the total weight of unsatisfied clauses.

Algorithms for Max-SAT can be categorized into two classes: complete algorithms (Li et al. 2007; Heras et al. 2008; Lin et al. 2008; Ansótegui et al. 2013) and stochastic local search (SLS) algorithms. SLS algorithms have exhibited great success in solving SAT and Max-SAT problems. They are especially appealing when the problem instance is large in size, or a reasonably good solution is needed in a short time, or when the knowledge about the problem domain is rather limited (Hoos and Stützle 2004).

A special case of the weighted Max-SAT problem is the weighted Max-2-SAT problem, which has great importance. A lot of realistic problems such as Maximum Cut (Gramm et al. 2003), Maximum Clique (Heras and Bañeres 2010), sports scheduling (Ryuhei and Tomomi 2006), 3D modeling (Staub and Prautzsch 2005), physical design of VLSI circuits (Kastner and Sarrafzadeh 2002), and Internet search (Dimitropoulos et al. 2007) can be transformed into weighted Max-2-SAT problems more naturally than into SAT problems. For solving weighted Max-2-SAT instances, general SLS algorithms for SAT and Max-SAT have weaker performance than those specific for weighted Max-2-SAT (Kochenberger et al. 2005; Palubeckis 2008). This work focuses on designing more efficient SLS algorithms for weighted Max-2-SAT.

The basic schema of SLS algorithms for Max-SAT can be described as follows. First, all variables appearing in the formula are given a random assignment of boolean values. Then, in each subsequent search step, a variable is chosen and flipped. We use *pickVar* to denote the function for choosing the variable to be flipped. The variable selection heuristic in the pickVar function is the essential part of an SLS algorithm for Max-SAT.

SLS algorithms for Max-SAT usually work in two different modes, i.e., the global mode and the focused mode. In the global mode, the algorithms pick a variable to flip among all variables, and usually they prefer to pick a variable whose flip can decrease the number of unsatisfied clauses (thus the global mode is also known as the greedy mode). In the focused mode, the algorithms pick a variable from an unsatisfied clause, usually using randomized strategies and exploiting *diversification properties* of variables such as *age* and *flip count* to pick a variable. Although SLS algorithms for weighted Max-SAT share the basic schema with those for SAT, well-performing SAT solvers do not show good performance on weighted Max-SAT instances. For example, reported in (Smyth et al. 2003), the IRoTS algorithm outperforms extensions of famous SLS algorithms for SAT such as WalkSAT (Selman et al. 1994) and DLM (Wu and Wah 2000) on weighted Max-SAT.

Recently, a strategy called Configuration Checking (CC) was proposed for handling the cycling problem in local search, i.e., revisiting recent candidate solutions (Cai et al. 2011). It has been successfully used in SLS algorithms for SAT (Cai and Su 2012, 2013; Luo et al. 2012, 2013), which show state-of-the-art performance. Specially, the CCASat solver (Cai and Su 2013) won the random track of SAT Challenge 2012. The CC strategy for SAT forbids a variable to be flipped if since the last time it was flipped, none of its neighboring variables has been flipped. However, our experiments show that the direct application of the CC strategy does not result in a well-performing solver for weighted Max-2-SAT.

In this work, we propose a new variable selection heuristic called CCTriplex, which can be regarded as an extension of the CC strategy. A variable is said to be configuration changed if since its last flip, at least one of its neighboring variables has been flipped. According to the CCTriplex heuristic, a variable that is both decreasing and configuration changed has the higher priority to be flipped than a decreasing variable, which in turn has the higher priority than a configuration changed variable. The CCTriplex heuristic is more flexible than the CC strategy and makes a good balance between diversification and intensification, even without clause weighting techniques.

We use the CCTriplex heuristic to develop a new SLS algorithm for weighted Max-SAT, which is named CCMaxSAT. CCMaxSAT switches between the global mode and the focused mode, according to a dynamic probability parameter. This work focuses on using the CCTriplex heuristic to improve the global mode. CCMaxSAT exhibits very good performance on weighted Max-2-SAT instances.

For demonstrating the efficiency of CCMaxSAT, we compare it with two SLS solvers namely ITS (Iterated Tabu Search) (Palubeckis 2008) and ubcsat-IRoTS (Smyth et al. 2003), as well as the famous complete solver wMaxSATz (Li et al. 2009). ITS is the best SLS solver for weighted Max-2-SAT in the literature (to the best of our knowledge), and ubcsat-IRoTS is the best SLS solver in Max-SAT Evaluation 2012, particularly on the random weighted Max-SAT category. The experimental results show that CCMaxSAT significantly outperforms ITS and ubcsat-IRoTS on a large range of random weighted Max-2-SAT instances with different clause-to-variable ratios. On the structured benchmark Frb, where ubcsat-IRoTS fails to find an optimal solution for any instance, the performance of CCMaxSAT is slightly better than that of ITS. On the benchmarks encoded from MaxCut, MaxClique and sports scheduling problems, CCMaxSAT overall performs better than the other two SLS solvers. Additionally, CCMaxSAT finds solutions of better quality (or at least the same quality) than wMaxSATz on all tested instances except for several sparse random instances.

The remainder of the paper is organized as follows. Some necessary background knowledge is provided in the next section. Section 3 presents the CCTriplex heuristic, and Sect. 4 describes the CCMaxSAT algorithm. Section 5 reports the experimental study of CCMaxSAT. This is followed by more discussions about CCMaxSAT as well as related works in Sect. 6. Finally, we summarize our main contributions and give some directions for future work.

2 Definitions and notations

Given a set of Boolean variables $V = \{x_1, \ldots, x_n\}$, a *literal* is either a variable x or its negation $\neg x$, and a *clause* is a disjunction of literals. A CNF formula is a conjunction of clauses, i.e., $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. We use $V(F)$ to denote the set of all variables that appear in the formula F. Two variables are neighbors when they share at least one clause, and $N(x) = \{y|y \ and \ x \ share \ at \ least \ one \ clause\}$ is the set of all neighbors of variable x.

A mapping $\alpha : V(F) \rightarrow \{0, 1\}$ is called an *assignment*. If α maps all variables to a Boolean value, it is called *complete*. In local search algorithms for Max-SAT, a candidate solution is a complete assignment. Given an assignment, a clause is satisfied if it has at least one true literal, and unsatisfied if it has no true literal. In weighted Max-SAT, each clause c has an associated weight $w(c)$, and the goal is to find an optimum assignment that maximizes the total weight of satisfied clauses. A significant special case of the weighted Max-SAT problem is the weighted MAX-2-SAT problem, which restricts each clause in the formula to be of length at most two.

Given a weighted CNF formula (F, w), the *cost* of an assignment α, denoted as $cost(F, \alpha)$, is the total weight of all unsatisfied clauses under α. The *score* of a variable x is defined as $score(x) = cost(F, \alpha) - cost(F, \alpha')$, which indicates the benefit of flipping x, where α' is obtained from α by flipping x. A variable x is *decreasing* if and only if $score(x) > 0$.

3 The CCTriplex variable selection heuristic

In this section, we first introduce the notion of configuration changed variables. Then, based on this notion, we propose a new variable selection heuristic called CCTriplex, which is more flexible than the CC strategy and is shown to be particularly effective for weighted Max-2-SAT.

3.1 Configuration changed variables

Originally introduced in (Cai et al. 2011), configuration checking (CC) is a strategy aiming to handle the cycling problem in local search. The intuition behind this idea is that by reducing cycles on local structures of the candidate solution, we may reduce cycles on the whole candidate solution.

The CC strategy is based on the concept of *configuration*. In the context of SAT, the configuration of a variable refers to truth values of all its neighboring variables (Cai and Su 2011, 2012). The definition of *configuration changed variables* is given as following.

Definition 1 Given a CNF formula F, a variable $x \in V(F)$ is configuration changed if and only if after the last time x was flipped, at least one variable $y \in N(x)$ has been flipped.

To identify configuration changed variables, we employ an array $confChange$, whose element is an indicator for a variable — $confChange(x) = 1$ means x is a configuration changed variable, and $confChange(x) = 0$ on the contrary. During the search procedure, the variables with $confChange(x) = 0$ are forbidden to be flipped in the global mode, which could decrease blind unreasonable greedy search. The $confChange$ array is initialized by setting all $confChange$ values to 1. After that, when flipping a variable x, $confChange(x)$ is reset to 0, and for each variable $y \in N(x)$, $confChange(y)$ is set to 1.

The CC strategy for SAT allows only configuration changed decreasing variables to be flipped in the global mode; if there are not such variables, the algorithm switches to the focused mode (Cai and Su 2011). This strategy is not effective for weighted Max-2-SAT, as will be shown in Sect. 6.1.

3.2 The CCTriplex heuristic

Based on the notion of configuration changed variables, we propose the CCTriplex heuristic, which works in three levels. Before getting into the details of the CCTriplex heuristic, we first introduce three important variable sets.

- $CCD = \{x|score(x) > 0$ and $confChange(x) = 1\}$, consisting of variables that are both decreasing and configuration changed.
- $DNCC = \{x|score(x) > 0$ and $confChange(x) = 0\}$, consisting of variables that are decreasing but not configuration changed.
- $CCND = \{x|score(x) \leq 0$ and $confChange(x) = 1\}$, consisting of variables that are configuration changed but not decreasing.

The CCTriplex heuristic picks a variable to be flipped from one of the three variable sets CCD, $DNCC$, and $CCND$. Obviously, there exist configuration changed variables in each search step, and thus we have $CCD \cup CCND \neq \emptyset$, which guarantees that CCTriplex can always pick a variable successfully.

The CCTriplex heuristic works in three levels. If the CCD set is not empty, CCTriplex selects the variable with the greatest score in the CCD set to flip. Otherwise, if the $DNCC$ set is not empty, CCTriplex selects the variable with the greatest score in the $DNCC$ set. If both CCD and $DNCC$ are empty, then CCTriplex selects the variable with the greatest score in the $CCND$ set. Note that in CCTriplex, all ties are broken randomly.

The CC strategy simply forbids flipping those variables which are not configuration changed (Cai and Su 2011). In our opinion, this is too strict in the sense that there are only a very limited number of candidate flips in each step. Different from the CC strategy used in previous SLS algorithms (Cai and Su 2011, 2012), the CCTriplex heuristic uses the concept of "configuration changed" as a property of variables, and together with the "decreasing" property, it divides the candidate flipping variables into three groups of different priorities. This multilevel heuristic makes a good balance between intensification and diversification during the search. More specifically, CCD variables correspond to the variables whose flips would lead the search in a greedy direction and avoid revisiting recent candidate solutions as well; flipping DNCC variables would lead the search in a pure greedy way; and finally, flipping CCND variables contributes some more diversification.

4 The CCMaxSAT algorithm

We use the CCTriplex heuristic to develop a new SLS algorithm for weighted Max-SAT called CCMaxSAT. As with most SLS algorithms for Max-SAT, CCMaxSAT switches between the global mode and the focused mode. CCTriplex is used as the variable selection heuristic in the global mode. In order to demonstrate the efficacy of the CCTriplex heuristic clearly, we keep the focused mode of CCMaxSAT rather simple.

Algorithm 1: CCMaxSAT

```
Input: CNF-formula $F$, $maxSteps$
Output: An assignment $\alpha^*$ of $F$
1  begin
2      $\alpha \leftarrow$ randomly generated truth assignment;
3      $\alpha^* \leftarrow \alpha$;
4      for $step \leftarrow 1$ to $maxSteps$ do
5          adjust $wp$;
6          with probability $wp$ begin
7              $c \leftarrow$ randomly selected unsatisfied clause;
8              $v \leftarrow$ the variable with the greatest score in $c$, breaking ties randomly;
9          end
10         otherwise begin
11             if $CCD \neq \emptyset$ then
12                 $v \leftarrow x \in CCD$ with the greatest $score$, breaking ties randomly;
13             else if $DNCC \neq \emptyset$ then
14                 $v \leftarrow x \in DNCC$ with the greatest $score$, breaking ties randomly;
15             else
16                 $v \leftarrow x \in CCND$ with the greatest $score$, breaking ties randomly;
17         end
18         $\alpha \leftarrow \alpha$ with $v$ flipped;
19         if $cost(F, \alpha) < cost(F, \alpha^*)$ then $\alpha^* \leftarrow \alpha$;
20     return $\alpha^*$;
21 end
```

The CCMaxSAT algorithm is outlined in Algorithm 1, described as follows. In the beginning, CCMaxSAT generates a complete assignment α randomly, and the best solution α^* is initialized as α. Then, in each step of the following search process, CCMaxSAT selects a variable and flips it, trying to obtain a solution better than α^*. Whenever CCMaxSAT finding a better solution, α^* is updated with the new better solution.

In each step, the algorithm works in either of the two modes, i.e., the global mode and the focused mode. The probability of adopting the focused mode is controlled by a noise parameter wp (walking probability), which is adjusted during the search. For adjusting wp, we adopt the adaptive noise mechanism introduced in (Hoos 2002). In detail, wp is initialized as 0 in the beginning. During the search procedure, each time updating wp, the current objective function value is stored and becomes the basis for measuring improvement. If no improvement in objective function value has been observed over the last $\theta \cdot m$ search steps, where m is the number of clauses of the given instance and $\theta = 1/6$, then $wp := wp + (1 - wp) \cdot \phi$, where $\phi = 0.2$; otherwise, if an improvement in objective function value is observed, then $wp := wp - wp \cdot \phi/2$.

Once the wp parameter is updated, the algorithm works in the focused mode with probability wp, and in the global mode otherwise. The focused mode is rather simple:

the algorithm first picks a random unsatisfied clause c, and then selects the variable with the greatest score in c, breaking ties randomly. In the global mode, CCMaxSAT works according to the CCTriplex heuristic, which has been described in Sect. 3.2.

5 Experimental evaluations

We evaluate CCMaxSAT on weighted Max-2-SAT instances, in comparison with state-of-the-art solvers, including two SLS solvers and a complete solver.

- ITS (Palubeckis 2008), which is an SLS solver specific for weighted Max-2-SAT. ITS significantly outperforms general SLS solvers for Max-SAT such as GWSAT (Selman et al. 1994), GRASP (Festa et al. 2006), adaptNovelty+ (Hoos 2002), SAPS (Hutter et al. 2002), and IRoTS (Smyth et al. 2003) on both random instances and structured instances (Palubeckis 2008). The codes of ITS are download from its author's homepage.[1]
- ubcsat-IRoTS (Smyth et al. 2003), which is the best SLS solver in Max-SAT Evaluation 2012, and performs significantly better than other solvers in the weighted random Max-SAT category.
  The codes of ubcsat-IRoTS are downloaded from its author's homepage.[2]
- wMaxSatz (Li et al. 2009), which is the weighted version of the famous complete MaxSAT algorithm MaxSatz and performs very well in MaxSAT evaluations. We adopt the latest version wMaxSATz2009 from its author's homepage.[3] We also note that there is a more recent version of wMaxSATz namely MaxSatz2013f (version 2013), which is developed parallel to this work. According to the results of MaxSAT Evaluation 2013, the performance of wMaxSATz2009 and MaxSatz2013f are very similar. Specifically, MaxSatz2013f solves 2 more weighted random instances than wMaxSATz2009, while wMaxSATz2009 solves 2 more weighted crafted instances than MaxSatz2013f (there is no industrial weighted category in MaxSAT Evaluation 2013).

Recently, Kroc et al. proposed a strategy for combining DPLL and SLS approaches based on shared memory, and the hybrid solver MiniWalk (Kroc et al. 2009) made a breakthrough in solving MaxSAT instances. However, MiniWalk is designed for unweighted instances and cannot solve weighted instances, and thus is not included in our experiments.

5.1 Benchmarks

We evaluate CCMaxSAT on a broad range of benchmarks, including random instances with different ratios, hard combinatorial instances, and application instances which are encoded from other problems.

[1] http://www.proin.ktu.lt/~gintaras/wmax2sat.html.

[2] http://ubcsat.dtompkins.com/downloads.

[3] http://home.mis.u-picardie.fr/~cli/EnglishPage.html.

For random instances, we consider those generated by the famous makewff generator[4] are the most suitable random weighted Max-SAT benchmarks for evaluating performance of MaxSAT solvers. Note that makewff is a famous weighted Max-SAT generator which has been widely used in Max-SAT evaluations and in the literature (Littman et al. 2001; Simons et al. 2002; Haanpää and Kaski 2005; Janhunen et al. 2006). For random weighted Max-2-SAT instances in this work, each clause weight is an integer chosen uniformly randomly from 1 to 10, as with those used in evaluating IRoTS (Smyth et al. 2003) and ITS (Palubeckis 2008).

For hard combinatorial instances, we adopt the Frb benchmark,[5] which contains hard instances with known optimal values of the objective function. Note that these instances are very difficult to solve by current techniques in spite of their relative small size. They were generated randomly in the phase transition area according to the RB model (Xu et al. 2005). Generally, those phase-transition instances generated by RB have been proven to be hard both theoretically and practically (Xu et al. 2007). The Frb benchmark is extensively used in SAT competitions and Max-SAT evaluations.

For application instances, we adopt three benchmarks which are encoded from MaxCut, MaxClique and sports scheduling problems, respectively. The first benchmark includes all MaxCut instances from MaxSAT Evaluation 2012.[6] The second one consists of instances encoded from the DIMACS MaxClique benchmark.[7] Note that all instances in this benchmark are of the greatest size in their graph families, except for C2000.9, which is well known as the hardest instance in the C family (Grosso et al. 2008; Pullan et al. 2011; Cai et al. 2011, 2013). The last application benchmark consists of instances encoded from the break minimization problem in sports timetabling (Ryuhei and Tomomi 2006), and was downloaded online.[8] In many round-robin tournaments of professional sports, a match is held at the home of one of playing two teams. In such a match, a team playing at home has advantage over its opponent, i.e., a team playing at away. It is considered undesirable that to play consecutive matches held both at home/away for a team. An occurrence of such consecutive matches is called a break in sports timetabling. The break minimization problem in timetabling of a round-robin tournament is to assign home or away to each match so as to minimize the number of breaks (Ryuhei and Tomomi 2006).

5.2 Experiment preliminaries

CCMaxSAT is implemented in C++, and can be downloaded online.[9] CCMaxSAT has a parameter wp, which controls the probability of performing a step that picks a variable from a random unsatisfied clause. However, this parameter is adjusted dynamically during the search, and one does not need to specify it for solving an

4 ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/cotributed/selman/.

5 http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/max-sat-benchmarks.htm.

6 http://maxsat.ia.udl.cat:81/12/benchmarks/index.html.

7 ftp://dimacs.rutgers.edu/pub/challenges.

8 http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/sat/max2sat/.

9 www.shaoweicai.net/MaxSAT.html.

instance. For adjusting wp, we adopt the adaptive noise mechanism introduced in (Hoos 2002), and the parameters for the adaptive mechanism are set to the same values as set in adaptNovelty+ (Hoos 2002) ($\theta = 1/6$ and $\phi = 0.2$, as described in Sect. 4). Note that this parameter setting for the adaptive mechanism is quite robust and we did not find other settings that lead to noticeable better performance. Indeed, this adaptive mechanism is also used in other algorithms such as Reactive SAPS (Hutter et al. 2002), using the same parameter setting as in adaptNovelty+. Both ITS and ubcsat-IRoTS are executed with the default parameter values in their codes.

All experiments are carried out on 2 cores from an I7 CPU with 1.6 GHz and 8 GB memory. For each instance, each SLS solver is performed 20 independent runs with a cutoff time of 15 minutes (900 s), while the complete solver wMaxSATz is performed only one time with the same cutoff time.

For SLS solvers, we report the best solution quality ("best"), i.e., *the minimum unsatisfied weight*, and the averaged solution quality ("average"), i.e., *the mean value of the unsatisfied weights* of all 20 runs returned by each solver. For Frb and sports scheduling benchmarks, we report for each instance the unsatisfied weight of the best solution found by the solvers ("unsatw*"), the success rate ("suc. rate") of finding an "unsatw*" solution, and the averaged runtime ("time") for each instance. The results in bold indicate the best performance.

For the complete solver wMaxSATz, we report the best solution quality ("best") within the cutoff time, as wMaxSATz prints successively the best solution it finds so far. When we report the run time ("time") of wMaxSATz, we refer to the run time for it to find the best solution.

5.3 Experimental results

In the following, we report and discuss the experimental results on each benchmark. The results on all benchmarks illustrate the good performance and robustness of CCMaxSAT.

5.3.1 Results on sparse random benchmark

The first random benchmark we adopt is *wrand*,[10] which contains sparse random weighted Max-2-SAT instances generated by the makewff generator. The clause-to-variable ratios of these *wrand* instances range from 1 to 2. In our experiments, random instances are named in the form of "V(#variables)_C(#clauses)". Note that both #variables and #clauses are measured in thousands. For example, the instance named "V2k_C2.2k" has 2,000 variables and 2,200 clauses.

We first compare the three SLS solvers on this benchmark. Obviously, Table 1 indicates that ubcsat-IRoTS performs worse than CCMaxSAT and ITS on all these instances. So we take a further look at the comparison between CCMaxSAT and ITS. Except for the instances with the smallest clause-to-variable ratios where both algorithms find solutions of the same quality, CCMaxSAT always finds much better solutions than ITS does. On average, in terms of unsatisfied weight, the averaged

[10] http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/sat/max2sat/.

Table 1 Comparative results on the wrand benchmark, which consists of sparse random weghted Max-2-SAT instances

Instance	CCMaxSAT		ITS		ubcsat-IRoTS		wMaxSATz
	Best	Average	Best	Average	Best	Average	Best
V2k_C2.2k*	5	5	5	5	10	15.15	5
V2k_C2.4k*	1	1	1	1	3	6.45	1
V2k_C2.6k*	12	12	14	16.25	19	25.9	12
V2k_C2.8k*	7	7	7	8.55	11	16.95	7
V2k_C3k*	53	53.35	59	61.55	65	75.75	53
V2k_C3.1k*	56	58.15	65	68.7	68	78.75	**55**
V2k_C3.2k*	58	59.95	67	70.5	77	85.15	**56**
V2k_C3.3k*	101	104.3	107	109.5	110	119.3	**88**
V2k_C3.4k*	113	115.1	120	126.55	128	141.7	**101**
V2k_C3.5k	**119**	**120.1**	126	130	138	147.2	125
V3k_C3k*	0	0	0	0	2	6.75	0
V3k_C3.2k*	3	3	3	3	10	22.3	3
V3k_C3.4k*	1	1	1	1	6	21.85	1
V3k_C3.6k*	2	2	2	2	7	19.3	2
V3k_C3.8k*	15	15	23	26.6	39	53.75	15
V3k_C4k*	19	19	28	34.05	49	65	19
V3k_C4.2k*	22	22.65	32	35.85	41	65.5	22
V3k_C4.4k*	39	39.85	51	56.55	67	80.5	**38**
V3k_C4.6k*	65	66.55	76	80.55	92	109.6	**62**
V3k_C4.8k	**112**	**113.8**	126	134.25	137	150.15	121
V5k_C5.1k*	0	0	0	0	20	27.9	0
V5k_C5.4k*	1	1	1	1	28	39.8	1
V5k_C5.7k*	1	1	1	1	21	33.45	1
V5k_C6k*	8	8	12	15.65	53	70.3	8
V5k_C6.3k*	6	6	10	14.6	48	71.85	6
V5k_C6.6k*	33	33	57	62.1	94	127.9	33
V5k_C6.9k*	70	71.7	100	105.6	142	168.9	**54**
V5k_C7.2k*	82	83.2	126	135.45	149	200.85	**73**
V5k_C7.5k	**1,902**	**1,903.4**	1,957	1,980.8	2,002	2,045.7	2,231
V5k_C7.8k	**2,109**	**2,110.75**	2,171	2,185.45	2,224	2,246.8	2,343

For each instance, each SLS solver is performed 20 runs, while wMaxSATz is performed one time, with a cutoff time of 900 seconds. The instances where wMaxSATz finds an optimal solution are marked with '*'

quality of solutions returned by CCMaxSAT is about one fifth better than those returned by ITS.

It is also interesting to observe that the complete solver wMaxSATz finds an optimal solution for most of these sparse random instances (with only 4 exceptions). CCMaxSAT also finds optimal solutions for most of such instances. This indicates these random instances with ratios smaller than 2 are easy to solve. Compared to SLS solvers, wMaxSATz finds solutions of better or equivalent quality on instances with small ratios, but worse on 4 instances with big ratios, namely, V2k_C3.5k, V3k_C4.8k, V5k_C7.5k and V5k_C7.8k, where CCMaxSAT finds the best solutions.

5.3.2 Results on dense random benchmark

The *wrand* benchmark contains only random weighted Max-2-SAT instances whose clause-to-variable ratios are smaller than 2. In order to evaluate the performance of CCMaxSAT on denser random instances, we use the makewff generator to generate 45 weighted Max-2-SAT instances with bigger clause-to-variable ratios, whose sizes range from 2,000 to 5,000 variables. For each group of instances with the same size, the clause-to-variable ratio ranges from 2 to 6 in increments of 1, and there are 3 instances for each ratio.

Table 2 presents the comparative performance on these dense random instances. CCMaxSAT dominates on all these random weighted Max-2-SAT instances, and ubcsat-IRoTS cannot rival CCMaxSAT and ITS. Also, it is clear from the table that CCMaxSAT substantially outperforms ITS on the whole benchmark. The best solutions found by CCMaxSAT are better than the ones found by ITS on all instances but one. Furthermore, the performance of CCMaxSAT is always better than that of ITS in terms of averaged solution quality.

Table 2 Comparative results on the dense random weighted Max-2-SAT benchmark

Instance	CCMaxSAT		ITS		Ubcsat-IRoTS		wMaxSATz
	Best	Average	Best	Average	Best	Average	Best
V2k_C4k_1	**1,378**	**1,378**	1,379	1,383.6	1,387	1,391.3	1,522
V2k_C4k_2	**1,468**	**1,468**	1,473	1,476.9	1,477	1,485.85	1,617
V2k_C4k_3	**1,429**	**1,429**	1,433	1,437.8	1,442	1,450.4	1,528
V2k_C6k_1	**2,755**	**2,755**	2,759	2,759.9	2,759	2,767.6	2,982
V2k_C6k_2	2,781	**2,781**	2,781	2,782.8	2,788	2,792.55	3,141
V2k_C6k_3	**2,786**	**2,786**	2,789	2,789.5	2,789	2,793.5	3,125
V2k_C8k_1	**4,403**	**4,403**	4,405	4,408	4,413	4,420.75	4,848
V2k_C8k_2	**4,431**	**4,431**	4,433	4,435.7	4,440	4,447.6	4,887
V2k_C8k_3	4,415	**4,415**	4,415	4,419.5	4,418	4,431.6	4,772
V2k_C10k_1	6,220	**6,220**	6,220	6,228.1	6,224	6,232.35	6,745
V2k_C10k_2	6,073	**6,073**	6,073	6,076.3	6,086	6,098.9	6,622
V2k_C10k_3	6,233	**6,233**	6,233	6,233.4	6,237	6,244.8	6,753
V2k_C12k_1	**8,036**	**8,036.95**	8,037	8,037.8	8,047	8,053.9	8,723
V2k_C12k_2	**7,977**	**7,978.65**	7,979	7,980.5	7,984	7,988.85	8,567
V2k_C12k_3	8,010	8,010	8,010	8,010	8,015	8,022.5	8,546
V3k_C6k_1	**2,068**	**2,068**	2,083	2,090.05	2,104	2,122.45	2,414
V3k_C6k_2	**2,045**	**2,045.75**	2,056	2,065.45	2,075	2,090.05	2,258
V3k_C6k_3	**2,172**	**2,172.65**	2,191	2,201.45	2,206	2,227.5	2,377

Table 2 continued

Instance	CCMaxSAT		ITS		Ubcsat-IRoTS		wMaxSATz
	Best	Average	Best	Average	Best	Average	Best
V3k_C9k_1	**4,107**	**4,107.45**	4,125	4,130.3	4,124	4,144.9	4,585
V3k_C9k_2	**4,179**	**4,179**	4,194	4,203.5	4,208	4,218.85	4,683
V3k_C9k_3	**4,265**	**4,266.15**	4,278	4,287.05	4,287	4,298.75	4,787
V3k_C12k_1	**6,691**	**6,691**	6,694	6,698.4	6,708	6,719.7	7,372
V3k_C12k_2	**6,537**	**6,537**	6,546	6,553.6	6,566	6,573.95	7,025
V3k_C12k_3	**6,685**	**6,685**	6,689	6,706.75	6,729	6,734.45	7,309
V3k_C15k_1	**9,329**	**9,330.9**	9,344	9,353	9,369	9,381.9	10,234
V3k_C15k_2	**9,172**	**9,174.05**	9,179	9,185.05	9,208	9,225.6	9,919
V3k_C15k_3	**9,274**	**9,274**	9,181	9,289.95	9,305	9,319.45	10,086
V3k_C18k_1	**12,166**	**12,167.75**	12,168	12,172.9	12,198	12,220	13,077
V3k_C18k_2	**11,907**	**11,907.15**	11,909	11,924.85	11,946	11,966.5	13,016
V3k_C18k_3	**12,108**	**12,108**	12,114	12,125.2	12,147	12,163.9	13,088
V5k_C10k_1	**3,428**	**3,429**	3,485	3,498.5	3,508	3,551.7	4,034
V5k_C10k_2	**3,398**	**3,398.05**	3,449	3,470.6	3,475	3,499.6	3,924
V5k_C10k_3	**3,435**	**3,436.6**	3,513	3,524.4	3,522	3,564.4	3,859
V5k_C15k_1	**7,036**	**7,036.7**	7,088	7,106	7,111	7,135.3	7,732
V5k_C15k_2	**6,937**	**6,939.3**	6,990	7,012.9	6,993	7,035.8	7,868
V5k_C15k_3	**7,062**	**7,066.55**	7,128	7,141.6	7,148	7,168	7,907
V5k_C20k_1	**11,071**	**11,073.65**	11,122	11,134.8	11,144	11,174.3	12,237
V5k_C20k_2	**10,813**	**10,816.45**	10,865	10,878.8	10,888	10,915.1	12,096
V5k_C20k_3	**11,098**	**11,099.7**	11,148	11,164.45	11,189	11,218.5	12,138
V5k_C25k_1	**15,384**	**15,388**	15,415	15,423.15	15,439	15,483	16,800
V5k_C25k_2	**15,427**	**15,429.05**	15,443	15,491.7	15,560	15,585.9	16,998
V5k_C25k_3	**15,117**	**15,119.4**	15,144	15,157.8	15,205	15,231.5	16,472
V5k_C30k_1	**20,019**	**20,030.15**	20,040	20,070.5	20,124	20,170.3	21,721
V5k_C30k_2	**20,302**	**20,306.75**	20,325	20,352.8	20,384	20,434.7	22,117
V5k_C30k_3	**19,714**	**19,720.4**	19,735	19,755.75	19,796	19,833.2	21,414

For each instance, each SLS solver is performed 20 independent runs, while the complete solver wMaxSATz is performed one time, with a cutoff time of 900 s

Table 2 also shows that SLS algorithms always find better solutions than the complete solver wMaxSATz does. This indicates SLS is a promising approach for solving large-sized weighted random Max-SAT instances, especially those with big clause-to-variable ratios.

5.3.3 Results on Frb benchmark

Comparative results on the Frb benchmark are shown in Table 3. The ubcsat-IRoTS solver fails to find an optimal solution for any instance in this benchmark, and thus its results are not reported in the table. For these instances, CCMaxSAT and ITS usually

Table 3 Comparative performance results on the Frb benchmark

Instance	unsatw*	CCMaxSAT		ITS		wMaxSATz
		Suc. rate (%)	Time	Suc. rate (%)	Time	Best
frb40-19-1	720	100	**26.50**	100	49.10	737
frb40-19-2	720	**100**	217.10	35	765.95	735
frb40-19-3	720	100	28.10	100	**18.25**	732
frb40-19-4	720	**100**	99.75	70	494.55	732
frb40-19-5	720	**100**	312.60	25	786.85	733
frb45-21-1	900	95	206.40	**100**	149.60	915
frb45-21-2	900	95	339.95	95	**203.65**	917
frb45-21-3	900	**60**	597.50	20	781.15	918
frb45-21-4	900	100	116.50	100	**112.45**	915
frb45-21-5	900	55	575.75	**65**	538.55	917
frb50-23-1	1,100	**35**	694.30	5	894.60	1,118
frb50-23-2	1,100	5	866.40	**10**	849.25	1,117
frb50-23-3	1,101	100	**65.75**	100	290.75	1,118
frb50-23-4	1,100	100	235.30	100	**148.65**	1,115
frb50-23-5	1,100	50	557.85	**80**	404.55	1,119
frb53-24-1	1,220	**85**	429.45	70	423	1,240
frb53-24-2	1,219	0	900	**5**	894.35	1,239
frb53-24-3	1,219	**30**	767.70	25	769.30	1,238
frb53-24-4	1,219	0	900	**10**	870.10	1,238
frb53-24-5	1,219	5	886.65	**25**	770.40	1,240
frb56-25-1	1,345	**35**	695.60	10	875.10	1,494
frb56-25-2	1,345	**40**	729.90	25	812.30	1,368
frb56-25-3	1,345	**70**	588.40	65	537.90	1,365
frb56-25-4	1,344	15	821.40	**20**	796.80	1,361
frb56-25-5	1,344	30	842	30	**745.25**	1,367
frb59-26-1	1,476	**30**	813.35	15	811.80	1,494
frb59-26-2	1,476	20	821.20	**25**	763.80	1,496
frb59-26-3	1,475	**5**	895	0	900	1,503
frb59-26-4	1,476	**35**	789.35	5	861.40	1,498
frb59-26-5	1,475	25	816	**65**	506.25	1,494

For each instance, each SLS solver is performed 20 independent runs, while the complete solver wMaxSATz is performed one time, with a cutoff time of 900 s. The results of ubcsat-IRoTS are not reported in this table, as it fails to find an optimal solution for any of these instances, which indicates it is essentially worse than the other two SLS solvers on this benchmark

have the same best solution quality; moreover, the gap between their averaged solution qualities never exceeds 1. Therefore, for this benchmark, we do not report the best solution quality and the averaged solution quality; instead, we report the success rate of reaching the best solution, and the averaged runtime. When comparing the two SLS solvers, we adopt a measuring method similar to the one used in SAT competitions: A

Table 4 Comparative results on the MaxCut benchmark

Instance	CCMaxSAT		ITS		ubcsat-IRoTS		wMaxSATz
	Best	Average	Best	Average	Best	Average	Best
t3g3-5555.spn*	11,006,100	11,006,100	11,006,100	11,006,100	1,1006,100	11,006,100	1100610
t4g3-6666.spn*	2,275,606	2,275,606	2,275,606	2,275,606	2,275,606	2,275,606	2275606
t5g3-7777.spn*	4,241,951	4,241,951	4,241,951	4241,951	4,241,951	4,241,951	4241951
t6g3-8888.spn*	7,844,119	7,844,119	7,844,119	7,844,119	7,844,119	7,844,119	7844119
t7g3-9999.spn	11,954,769 (**90,%**)	**11,954,786**	Runtime error	Runtime error	11,954,769 (5,%)	11,960,597	12,961,793

For each instance, each SLS solver is performed 20 runs, while wMaxSATz is performed one time, with a cutoff time of 900 s. The instances where wMaxSATz finds an optimal solution are marked with '*'

solver is said to perform better than the other one if it achieves a better success rate, or has a smaller value of the averaged runtime when the two solvers have the same success rate.

As can be seen from Table 3, CCMaxSAT and ITS are competitive and complementary on the Frb benchmark, as they dominate on different instances. For example, on the two largest sized group, CCMaxSAT dominates on 6 instances, and ITS dominates on the other 4 instances. However, the overall performance of CCMaxSAT is better than ITS: the averaged success rate of CCMaxSAT is 54 %, compared to 46.67 % for ITS. Table 3 also shows that SLS solvers find much better solutions than wMaxSATz does on this structured benchmark.

5.3.4 Results on MaxCut benchmark

Comparative results on the MaxCut benchmark are shown in Table 4. Among the 5 weighted Max-2-SAT instances encoded from the MaxCut problem, 4 of them are so easy that all solvers can find an optimal solution within just a few seconds. For the largest-sized instance `t7g3-9999.spn`, CCMaxSAT obviously has the best performance. ITS has runtime failure for this instance, and the exact solver wMaxSATz finds a much worse solution than CCMaxSAT and ubcsat-IRoTS do. Although the best solutions found by CCMaxSAT and ubcsat-IRoTS are of the same quality (with the unsatisfied weight 11954769), CCMaxSAT finds such a solution in 18 out of 20 trails, while ubcsat-IRoTS does so only in one trail. This indicates the superiority of CCMaxSAT on these instances encoded from the MaxCut problem.

5.3.5 Results on MaxClique benchmark

Table 5 presents the experimental results on the instances encoded from the MaxClique problem. As can be seen from Table 5, CCMaxSAT finds better solutions than other solvers on three instances. For the remaining instances, CCMaxSAT also has the best performance (shared with ITS), except for MANN_a81, where IRoTS finds better solutions. However, the performance of IRoTS is much worse than that of CCMaxSAT on all instances but MANN_a81.

Table 5 Comparative results on the DIMACS MaxClique benchmark

Instance	CCMaxSAT		ITS		ubcsat-IRoTS		wMaxSATz
	Best	Average	Best	Average	Best	Average	Best
brock800_4	**774**	**778.5**	779	779	780	780.95	781
C2000.9	1,922	**1,922.3**	1,922	1,922.4	1,935	1,938.19	1,944
DSJC100.5	985	985	985	985	986	986.57	987
gen400_0.9_75	325	325	325	325	325	325.86	363
hamming10-4	984	984	984	984	988	989.33	992
keller6	**3,302**	**3,303.2**	3,304	3,306	3,318	3,320.1	3,330
MANN_a81	2,240	2,240	2,237	2,237	**2,228**	**2,230.1**	2,241
phat1500-3	1,406	1,406	1,406	1,406	1,413	1,416.81	1,416

For each instance, each SLS solver is performed 20 runs, while wMaxSATz is performed one time, with a cutoff time of 900 s

Table 6 Comparative results on the sports scheduling benchmark

Instance	unsatw*	CCMaxSAT		ITS		ubcsat-IRoTS		wMaxSATz
		Suc. rate (%)	Time	Suc. rate (%)	Time	Suc. rate (%)	Time	Time
break_16_120_224	16	100	<0.01	100	<0.01	100	<0.01	0.1
break_18_153_288	20	100	<0.01	100	<0.01	100	<0.01	4.6
break_20_190_360	24	100	<0.01	100	0.1	100	<0.01	37
break_22_231_440	29	100	<0.01	100	0.3	100	<0.01	680

For each instance, each SLS solver is performed 20 runs, while wMaxSATz is performed one time, with a cutoff time of 900 s. The optimality of unsatw* has been proved by wMaxSATz for all these instances

5.3.6 Results on sports scheduling benchmark

The experimental results on the minimum break problem in sports scheduling are reported in Table 6. These instances have been used to test branch and cut algorithms for Max-2-SAT (Ryuhei and Tomomi 2006). However, they turn out to be too easy for SLS solvers, especially for CCMaxSAT and IRoTS, both of which find an optimal solution in less than 0.01 second for all these instances. Unfortunately, we could not access the generator of this benchmark and thus could not test our solver on larger instances. Nevertheless, the results show that SLS solvers can find an optimal solution much faster than the exact solver wMaxSATz on these sports scheduling instances.

5.3.7 Performance on Max-3-SAT and SAT instances

In this work, CCMaxSAT is tuned to perform well on weighted Max-2-SAT instances. However, our experiments show that CCMaxSAT also exhibits good performance on weighted Max-3-SAT instances. We have conducted an experimental study comparing CCMaxSAT with ubcsat-IRoTS on random weighted Max-3-SAT instances with 5000 variables, whose clause-to-variable ratios range from 1 to 6. Our experimental results show that CCMaxSAT is very competitive with IRoTS on these weighted Max-3-SAT instances. We believe by adjusting CCMaxSAT carefully, we can improve it on general Max-SAT instances, but this is beyond the scope of this paper, and we leave it for future work.

On the other hand, the performance of CCMaxSAT is obviously worse than state-of-the-art SLS SAT solvers on SAT instances. Our experiments on random SAT instances from SAT Competition 2011 show that CCMaxSAT performs significantly worse than Swcc (Cai and Su 2011), as well as the winners from the random satisfiable category of SAT Competition 2011.

6 Discussions and related work

In this section, we provide more insights about CCMaxSAT through experimental analysis, and discuss related works. Specifically, we explore the effectiveness of the CCTriplex heuristic and whether integrating a pure random walk can improve CCMaxSAT; we also investigate the frequencies of each type of search steps in CCMaxSAT. Then, we discuss the differences among CCMaxSAT, ITS and IRoTS algorithms. Finally, we discuss related works which share similar ideas with CCMaxSAT.

6.1 Effectiveness of the CCTriplex heuristic

We demonstrate the effectiveness of the CCTriplex heuristic by comparing CCMaxSAT with its alternative algorithm CCMaxSAT$_0$. The CCMaxSAT$_0$ algorithm applies the CC strategy directly, just as the SAT local search algorithm Swcc (Cai and Su 2011) does. The pickVar function in CCMaxSAT$_0$ is outlined in Algorithm 2.

Algorithm 2: pickVar-function in CCMaxSAT$_0$

```
1  adjust $wp$;
2  with probability $wp$ begin
3      $c \leftarrow$ randomly selected unsatisfied clause;
4      return $v \leftarrow$ the variable with the greatest score in $c$, breaking ties randomly;
5  end
6  otherwise begin
7      if $CCD \neq \emptyset$ then
8          return $v \leftarrow x \in CCD$ with the greatest $score$, breaking ties randomly;
9      else
10         return $v \leftarrow$ a random variable in a random unsatisfied clause;
11 end
```

We compare CCMaxSAT and CCMaxSAT$_0$ on some selected instances, including the largest sized instances from each random benchmark, the two largest sized

Table 7 Comparative performance results of CCMaxSAT and CCMaxSAT_0 on random instances

Instance	CCMaxSAT		CCMaxSAT_0	
	Best	Average	Best	Average
V5k_C5100	0	0	0	0
V5k_C5400	1	1	1	1
V5k_C5700	1	1	1	1
V5k_C6000	8	8	8	8
V5k_C6300	6	6	6	6
V5k_C6600	**33**	**33**	39	41.5
V5k_C6900	**70**	**71.7**	78	84.25
V5k_C7200	**82**	**83.2**	94	103.5
V5k_C7500	**1,902**	**1,903.4**	1,956	1,966.9
V5k_C7800	**2,109**	**2,110.75**	2,156	2,167.1
V5k_C10k_1	**3,428**	**3,429**	3,467	3,480.6
V5k_C10k_2	**3,398**	**3,398.05**	3,435	3,450.35
V5k_C10k_3	**3,435**	**3,436.6**	3,475	3,490.6
V5k_C15k_1	**7,036**	**7,036.7**	7,058	7,069.45
V5k_C15k_2	**6,937**	**6,939.3**	6,970	6,978.9
V5k_C15k_3	**7,062**	**7,066.55**	7,101	7,107.65
V5k_C20k_1	**11,071**	**11,073.65**	11,113	11,120.65
V5k_C20k_2	**10,813**	**10,816.45**	10,839	10,851.85
V5k_C20k_3	**11,098**	**11,099.7**	11,126	11,135.8
V5k_C25k_1	**15,384**	**15,388**	15,416	15,429.05
V5k_C25k_2	**15,427**	**15,429.05**	15,454	15,471.3
V5k_C25k_3	**15,117**	**15,119.4**	15,153	15,159.4
V5k_C30k_1	**20,019**	**20,030.15**	20,072	20,094.9
V5k_C30k_2	**20,302**	**20,306.75**	20,338	20,360.55
V5k_C30k_3	**19,714**	**19,720.4**	19,752	19,764.55

Each algorithm is performed 20 independent runs on each instance with a cutoff time of 900 s

groups from the Frb benchmark, as well as the hardest instances from the MaxCut and MaxClique benchmarks.

As can be seen from Table 7, CCMaxSAT_0 performs substantially worse than CCMaxSAT on all the selected random instances. The solutions that CCMaxSAT finds are significantly better than those found by CCMaxSAT_0, except for the 5 sparse random instances where both solvers find solutions of the same quality.

As for the Frb benchmark, CCMaxSAT_0 fails to find a solution whose total unsatisfied weight is unsatw* for any of these instances (Table 8). Comparatively, CCMaxSAT successfully finds an unsatw* solution for all Frb instances, indicating its essential superiority over CCMaxSAT_0 on these hard combinatorial instances.

Table 8 Comparative performance results of CCMaxSAT and CCMaxSAT$_0$ on the Frb benchmark

Instance	unsatw*	CCMaxSAT			CCMaxSAT$_0$		
		Suc. rate (%)	Best	Average	Suc. rate (%)	Best	Average
frb56-25-1	1,345	**35**	**1,345**	**1,345.65**	0	1,347	1,347.7
frb56-25-2	1,345	**40**	**1,345**	**1,345.6**	0	1,347	1,347.9
frb56-25-3	1,345	**65**	**1,345**	**1,345.35**	0	1,347	1,347.5
frb56-25-4	1,344	**15**	**1,344**	**1,345.25**	0	1,347	1,347.6
frb56-25-5	1,344	**30**	**1,344**	**1,345.05**	0	1,347	1,347.5
frb59-26-1	1,476	**30**	**1,476**	**1,476.7**	0	1,478	1,478.8
frb59-26-2	1,476	**20**	**1,476**	**1,476.8**	0	1,478	1,478.9
frb59-26-3	1,475	**5**	**1,475**	**1,476.7**	0	1,478	1,478.9
frb59-26-4	1,476	**35**	**1,476**	**1,476.65**	0	1,478	1,478.85
frb59-26-5	1,475	**25**	**1,475**	**1,476.35**	0	1,477	1,478.7

Each algorithm is performed 20 independent runs on each instance with a cutoff time of 900 s

Table 9 Comparative performance results of CCMaxSAT and CCMaxSAT$_0$ on MaxCut and MaxClique instances

Instance	CCMaxSAT		CCMaxSAT$_0$	
	Best	Average	Best	Average
t6g3-8888.spn	7,844,119	7,844,119	7,844,119	7,844,119
t7g3-9999.spn	**11,954,769**	**11,954,786**	11,970,576	11,985,764
brock800_4	**774**	**778.5**	779	779
C2000.9	**1,922**	**1,922.3**	1,930	1,935.21
keller6	**3,302**	**3,303.2**	3,304	3,304
MANN_a81	2,240	2,240	**2,238**	**2,239**

Each algorithm is performed 20 independent runs on each instance with a cutoff time of 900 s

The comparative results of CCMaxSAT and CCMaxSAT$_0$ on MaxCut and MaxClique instances are presented in Table 9. The results show that CCMaxSAT$_0$ performs worse than CCMaxSAT on all the selected instances, except for MANN_a81.

CCMaxSAT$_0$ is implemented on the codes of CCMaxSAT, and the only difference between CCMaxSAT and CCMaxSAT$_0$ is that CCMaxSAT employs the CCTriplex heuristic in the global mode, while CCMaxSAT$_0$ utilizes the heuristic based on the pure CC strategy. Hence we attribute the good performance of CCMaxSAT mainly to the CCTriplex heuristic.

6.2 Integrating random walk into CCMaxSAT

An important property of local search algorithms is probabilistically approximately complete (PAC). If a local search algorithm is PAC, then by running it long enough, the probability of missing an existing solution can be made arbitrarily small. A way

Table 10 Comparative performance results of CCMaxSAT and CCMaxSAT$_{rw}$

Instance	CCMaxSAT		CCMaxSAT$_{rw}$	
	Best	Average	Best	Average
frb56-25-1	1,345	1,345.65	**1,344**	**1,345.53**
frb56-25-2	1,345	1,345.6	1,345	**1,345.3**
frb56-25-3	1,345	1,345.35	1,345	**1,345.3**
frb56-25-4	1,344	**1,345.25**	1,344	1,345.35
frb56-25-5	1,344	1,345.05	1,344	1,345.05
t6g3-8888.spn	7,844,119	7,844,119	7,844,119	7,844,119
t7g3-9999.spn	11,954,769	**11,954,786**	11,954,769	11,960,858.2
brock800_4	774	778.5	774	778.5
C2000.9	**1,922**	**1,922.3**	1,923	1,923
DSJC100.5	985	985	985	985
gen400_0.9_75	325	325	325	325
hamming10-4	984	984	984	984
keller6	3,302	3,303.2	3,302	3,303.2
MANN_a81	2,240	2,240	2,240	2,240
phat1500-3	1,406	1,406	1,406	1,406

Each algorithm is performed 20 independent runs on each instance with a cutoff time of 900 s

of making local search algorithms PAC is to extend them with random walk in such a way, that for each local search step, with a fixed probability a random walk step is performed Hoos (1999).

To see how a random walk step may improve CCMaxSAT, we modify CCMaxSAT to make it perform a random walk (flipping a random variable in a random unsatisfied clause) with a fixed probability (0.01) in each step. This variant is called CCMaxSAT$_{rw}$. We carry out experiments to compare CCMaxSAT and CCMaxSAT$_{rw}$ on structured instances, and the results are summarized in Table 10.

As shown in Table 10, CCMaxSAT and CCMaxSAT$_{rw}$ have similar performance on the instances. Specifically, CCMaxSAT$_{rw}$ performs a little better than CCMaxSAT on `frb` instances, while its performance degrades on a MaxCut instance `t7g3-9999.spn` and a MaxClique instance `C2000.9`. On the other instances, they have the same performance in terms of solution quality. This indicates that a pure random walk step has limited impact on the performance of CCMaxSAT. However, enhancing CCMaxSAT with a pure random walk might still be a good choice, which makes it PAC.

6.3 Frequencies of different types of search steps

In order to better understand the run-time behaviour of CCMaxSAT, we investigate the frequencies of each type of search steps (random/CCD/DNCC/CCND) on the benchmarks. For our experimental study, we choose two representative instances from the

Table 11 Frequencies of each type of search steps for CCMaxSAT on different types of instances. Each result is based on 20 independent runs with a cutoff time of 900 seconds

Instance	Random (%)	CCD (%)	DNCC (%)	CCND (%)
V3k_C4k	16.87	39.03	15.99	28.11
V3k_C4.6k	14.26	31.34	24.58	29.82
V3k_C15k_1	17.89	42.10	18.99	21.02
V3k_C18k_1	17.80	42.03	18.82	21.34
frb50-23-1	17.32	61.60	0	21.08
frb53-24-1	17.17	61.14	0	21.69
t6g3-8888.spn	15.47	36.54	0	47.99
t7g3-9999.spn	14.87	30.98	0	54.14
brock800_4	24.25	61.62	0	14.81
C2000.9	14.57	62.06	0	23.42
keller6	16.89	60.78	0	23.11
MANN_a81	32.89	50.17	0	17.23

random and Frb benchmark, as well as two hardest instances in the MaxCut benchmark and four instances from different graph families in the MaxClique benchmark.

The results of this experimental study are reported in Table 11, from which we have the following observations:

- The proportion of random steps is quite stable (varies from 14 to 17 %) on random instances, as well as Frb and MaxCut instances;
- CCD and CCND steps are the two most often executed steps. Particularly, more than half steps are CCD steps for Frb and MaxClique instances.
- A particular feature of CCMaxSAT's behaviour is that it never executes DNCC steps when solving structured instances, which means all decreasing variables are configuration changed for these instances.

6.4 More comments on CCMaxSAT, ITS and IRoTS

Both ITS and IRoTS algorithms are Iterated Local Search (ILS) algorithms and alternate between two phases: local search and so-called solution perturbation. The latter phase takes the search away from the local optimum reached by the local search phase. Also, both ITS and IRoTS algorithms utilize the tabu method (Glover 1989) to diversify the search. However, they are rather different in the perturbation phase. While ITS makes use of large neighborhood steps (i.e., flipping several variables in one step) to perturb the local optima, IRoTS employs the same tabu search procedure in both local search and perturbation phases and adopts a larger tabu tenure for the perturbation phase.

CCMaxSAT differs significantly from ITS and IRoTS in two aspects. First, instead of adopting the ILS scheme, CCMaxSAT switches between the global mode and the focused mode according to an adaptive noise parameter. Secondly and more importantly, CCMaxSAT diversifies the search by (mainly) the CC strategy, while both ITS and IRoTS algorithms do so by the tabu method.

6.5 Related work

In this section, we present some related works. In particular, we discuss the relationship between CCD variables and promising decreasing variables, and discuss the adaptive noise mechanism and the greedy component in WalkSAT.

6.5.1 CCD variables versus promising decreasing variables

In the following, we discuss the relationship between CCD variables and promising decreasing variables (Li and Huang 2005). The concept of promising decreasing variables has been widely used to improve the global mode of SLS algorithms for SAT. Particularly, all awarded SLS solvers in SAT competitions 2007, 2009 and 2011 switch between the greedy and focused modes depending on the existence or not of promising decreasing variables.

First, we would like to recall some concepts:

- A variable x is *decreasing* iff $score(x) > 0$, and *increasing* iff $score(x) < 0$.
- A *configuration changed decreasing* (CCD) variable is a decreasing variable that $confChange(x) = 1$.
- Li and Huang (2005) Let x be a variable which is not decreasing. If it becomes decreasing after another variable y is flipped, then we say that x is a *promising decreasing variable* after y is flipped. For a promising decreasing variable x, it remains promising as long as it is decreasing after one or more other flips.

For the relationship between CCD variables and promising decreasing variables, we have the following conclusions.

Proposition 1 *For a given variable x, if x is a promising decreasing variable, then x is a CCD variable.*

Proof The proof is given by induction.

(a) *Becoming a promising decreasing variable*. If x becomes a promising decreasing variable after flipping another variable y, then we conclude $y \in N(x)$. Otherwise, y is independent of x and flipping y does nothing to $score(x)$. Since $y \in N(x)$, along with flipping y, $confChange(x)$ would be set to 1. As x is a decreasing variable and $confChange(x) = 1$, x is a CCD variable by definition.

(b) *Remaining a promising decreasing variable*. For a promising decreasing variable x, if x remains promising decreasing, then x has not been flipped after the last time it became a promising decreasing variable. Otherwise, because x is decreasing, i.e., $score(x) > 0$, flipping x would make $score(x) < 0$ (flipping x would make $score(x)$ be its opposite number). But this means x is no longer a decreasing variable, and thus not a promising decreasing variable. Recalling that only flipping x can set $confChange(x)$ to 0, we conclude that $confChange(x)$ remains 1. Thus, x remains a CCD variable. □

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

Proof For a variable x to be CCD, it suffices that one of its neighboring variable is flipped and $score(x) > 0$. To be promising, one or several neighboring variables should be flipped to make its score positive. When an increasing variable is flipped, it is CCD as soon as one of its neighboring variables is flipped and its score remains positive. However, it cannot be promising until its neighboring variables are flipped to make its score non-positive and then positive. □

To sum up, our analysis shows that promising decreasing variables are a subset of CCD variables. In some sense, CCD variables and promising decreasing variables may be two extremities, and there may be an intermediate notion more effective to be investigated in the future.

6.5.2 Adaptive noise and WalkSAT

The adaptive noise mechanism used in our algorithm is the one proposed by Hoos $et al$. in the adaptNovelty+ algorithm (Hoos 2002). This significant adaptive noise mechanism has also been successfully used in other SLS algorithms for SAT, such as adaptG^{2}WSAT and adaptadaptG^{2}WSAT+p (Li et al. 2007).

In the focused mode, CCMaxSAT picks a variable from a random unsatisfied clause c. Instead of picking a random variable, it selects the variable with the greatest score in c, which is greedy to some extend. This mixed random step in some sense resembles the “greedy” component of the WalkSAT algorithm (Selman et al. 1994), i.e., picking the variable with the minimum $break$ value from a random unsatisfied clause.

7 Conclusions and future work

Inspired by the configuration checking (CC) strategy for SAT local search algorithms, we proposed a new variable selection heuristic called CCTriplex for Max-SAT local search algorithms. CCTriplex is a three-level heuristic based on the notion of configuration changed variables. Compared to the CC heuristic, CCTriplex is more flexible and can make a better balance between intensification and diversification.

We utilized the CCTriplex heuristic to develop a new SLS algorithm for weighted Max-SAT called CCMaxSAT, which exhibits very good performance in solving weighted Max-2-SAT instances. We compared CCMaxSAT against two state-of-the-art SLS solvers namely ITS and ubcsat-IRoTS and the famous complete solver wMaxSATz. Experimental results show that CCMaxSAT significantly outperforms ITS and and ubcsat-IRoTS on random instances and the structured benchmark Frb. Also, CCMaxSAT show better performance on application instances encoded from several problems. Additionally, the solutions that CCMaxSAT finds are always better or as good as those found by the exact algorithm wMaxSATz on all the tested instances, except for a few sparse random instances.

To some extent, CCMaxSAT seems too greedy as a local search procedure. For example, it picks the variable with the greatest score from an unsatisfied clause even when it gets stuck in local optima. We believe by introducing more diversification may further improve the performance of the algorithm. However, this requires adjusting not only the focused mode, but also the global mode, in order to make them work well as a whole. We also plan to improve the CCMaxSAT algorithm for general (unweighted and weighted) Max-SAT problems.

Acknowledgments We would like to thank the anonymous reviewers for their valuable comments, which helped to improve the quality of this article. This work is supported by China National 973 project 2014CB340301, ARC Future Fellowship FT0991785, and National Natural Science Foundation of China 61370072 and 61472369.

References

Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)

Cai, S., Su, K.: Local search with configuration checking for SAT. In: Proceeding of the ICTAI-11, pp. 59–66 (2011)

Cai, S., Su, K.: Configuration checking with aspiration in local search for SAT. In: Proceedings of the AAAI-12, pp. 434–440 (2012)

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

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)

Cai, S., Su, K., Luo, C., Sattar, A.: NuMVC: an efficient local search algorithm for minimum vertex cover. J. Artif. Intell. Res. **46**, 687–716 (2013)

Dimitropoulos, X., Krioukov, D., Fomenkov, M., Huffaker, B., Hyun, Y., Claffy, K., Riley, G.: As relationships: inference and validation. Comput. Commun. Rev. **37**(1), 29–40 (2007)

Festa, P., Pardalos, P., Pitsoulis, L., Resende, M.: GRASP with path relinking for the weighted Max-SAT problem. ACM J. Exp. Algorithmics (11) (2006)

Glover, F.: Tabu search—part I. INFORMS J. Comput. **1**(3), 190–206 (1989)

Gramm, J., Hirsch, E., Niedermeier, R., Rossmanith, P.: Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. Discret. Appl. Math. **130**(2), 139–155 (2003)

Grosso, A., Locatelli, M., Pullan, W.: Simple ingredients leading to very efficient heuristics for the maximum clique problem. J. Heuristics **14**(6), 587–612 (2008)

Haanpää, H., Kaski, P.: The near resolvable 2-(13, 4, 3) designs and thirteen-player whist tournaments. Des. Codes. Cryptogr. **35**(3), 271–285 (2005)

Heras, F., Bañeres, D.: The impact of Max-SAT resolution-based preprocessors on local search solvers. J. Satisf. Boolean Model. Comput. **7**, 89–126 (2010)

Heras, F., Larrosa, J., Oliveras, A.: MiniMax-SAT: an efficient weighted max-sat solver. J. Artif. Intell. Res. (JAIR) **31**, 1–32 (2008)

Hoos, H.H.: On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. In: Proceedings of the AAAI-99, pp. 661–666 (1999)

Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In: Proceedings of the AAAI-02, pp. 655–660 (2002)

Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Elsevier, Amsterdam (2004)

Hutter, F., Tompkins, D.A.D., Hoos, H.H.: Scaling and probabilistic smoothing: efficient dynamic local search for SAT. In: Proceedings of the CP-02, pp. 233–248 (2002)

Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding partiality and disjunctions in stable model semantics. ACM Trans. Comput. Log. **7**(1), 1–37 (2006)

Kastner, R., Bozorgzadeh, E., Sarrafzadeh, M.: Pattern routing: use and theory for increasing predictability and avoiding coupling. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 233–248 (2002)

Kochenberger, G., Glover, F., Alidaee, B., Lewis, K.: Using the unconstrained quadratic program to model and solve Max-2-SAT problems. Int. J. Oper. Res. **1**, 89–100 (2005)

Kroc, L., Sabharwal, A., Gomes, C., Selman, B.: Integrating systematic and local search paradigms: a new strategy for MaxSAT. In: Proceedings of the IJCAI-09, pp. 544–551 (2009)

Li, C., Huang, W.: Diversification and determinism in local search for satisfiability. In: Proceedings of the SAT-05, pp. 158–172 (2005)

Li, C., Manyà, F., Planes, J.: New inference rules for Max-SAT. J. Artif. Intell. Res. (JAIR) **30**, 321–359 (2007)

Li, C., Manyà, F., Mohamedou, N., Planes, J.: Exploiting cycle structures in Max-SAT. In: Proceedings of the SAT-09, pp. 467–480 (2009)

Li, C., Wei, W., Zhang, H.: Combining adaptive noise and look-ahead in local search for SAT. In: Proceedings of the SAT-07, pp. 121–133 (2007)

Lin, H., Su, K., Li, C.: Within-problem learning for efficient lower bound computation in max-sat solving. In: Proceedings of the AAAI-08, pp. 351–356 (2008)

Littman, M.L., Majercik, S.M., Pitassi, T.: Stochastic boolean satisfiability. J. Autom. Reason. **27**(3), 251–296 (2001)

Luo, C., Su, K., Cai, S.: Improving local search for random 3-SAT using quantitative configuration checking. In: Proceedings of the ECAI-12, pp. 570–575 (2012)

Luo, C., Cai, S., Wu, W., Su, K.: Focused random walk with configuration checking and break minimum for satisfiability. In: Proceedings of the CP-13, pp. 481–496 (2013)

Palubeckis, G.: Solving the weighted Max-2-SAT problem with iterated tabu search. J. Inf. Technol. Control **37**, 275–284 (2008)

Pullan, W., Mascia, F., Brunato, M.: Cooperating local search for the maximum clique problem. J. Heuristics **17**(2), 181–199 (2011)

Ryuhei, M., Tomomi, M.: Semidefinite programming based approaches to the break minimization problem. Comput. OR **33**, 1975–1982 (2006)

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

Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. **138**(1–2), 181–234 (2002)

Smyth, K., Hoos, H.H., Stützle, T.: Iterated robust tabu search for MAX-SAT. In: Proceedings of the Canadian Conference on AI, pp. 129–144 (2003)

Staub, R., Prautzsch, H.: Creating optimized cutout sheets for paper models from meshes. In: Ninth SIAM Conference on Geometric Design and Computing (2005)

Wu, Z., Wah, B.W.: An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In: Proceedings of the AAAI-00, pp. 310–315 (2000)

Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: A simple model to generate hard satisfiable instances. In: Proceedings of the IJCAI-05, pp. 337–342 (2005)

Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random constraint satisfaction: easy generation of hard (satisfiable) instances. Artif. Intell. **171**(8–9), 514–534 (2007)