CCLS: An Efficient Local Search Algorithm for Weighted Maximum Satisfiability

Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su

Abstract—The maximum satisfiability (MAX-SAT) problem, especially the weighted version, has extensive applications. Weighted MAX-SAT instances encoded from real-world applications may be very large, which calls for efficient approximate methods, mainly stochastic local search (SLS) ones. However, few works exist on SLS algorithms for weighted MAX-SAT. In this paper, we propose a new heuristic called CCM for weighted MAX-SAT. The CCM heuristic prefers to select a CCMP variable. By combining CCM with random walk, we design a simple SLS algorithm dubbed CCLS for weighted MAX-SAT. The CCLS algorithm is evaluated against a state-of-the-art SLS solver IRoTS and two state-of-the-art complete solvers namely akmaxsat_ls and New WPM2, on a broad range of weighted MAX-SAT instances. Experimental results illustrate that the quality of solution found by CCLS is much better than that found by IRoTS, akmaxsat_ls and New WPM2 on most industrial, crafted and random instances, indicating the efficiency and the robustness of the CCLS algorithm. Furthermore, CCLS is evaluated in the weighted and unweighted MAX-SAT tracks of incomplete solvers in the Eighth Max-SAT Evaluation (Max-SAT 2013), and wins four tracks in this evaluation, illustrating that the performance of CCLS exceeds the current state-of-the-art performance of SLS algorithms on solving MAX-SAT instances.

Index Terms—Local search, weighted, maximum satisfiability, configuration checking, make

1 Introduction

The Boolean satisfiability (SAT) problem is a prototypical NP-complete problem, and plays a prominent role in many domains of computer science, mathematical logic and artificial intelligence. The maximum satisfiability (MAX-SAT) problem is a generalization of SAT. Given a conjunctive normal form (CNF) formula, the MAX-SAT problem is to find an assignment that maximizes the number of satisfied clauses (equally minimizes the number of unsatisfied clauses). In the weighted MAX-SAT problem, each clause is associated with a positive integer as its weight, and the task is to find an assignment that maximizes the total weight of satisfied clauses (equally minimizes the total weight of unsatisfied clauses).

MAX-SAT, especially its weighted version, is of significant importance in both theory and practice: In theory, both MAX-SAT and weighted MAX-SAT are NP-hard, and it is well known that optimal solutions to MAX-SAT are hard to approximate [1]; Practically, plenty of important realistic problems in a broad range of application areas such as physical design of very-large-scale integration (VLSI) circuits [2], internet search [3], routing [4], bioinformatics [5], scheduling [6] and probabilistic reasoning [7] can be modeled as weighted MAX-SAT. Also, new applications of weighted MAX-SAT have been found to extensive areas such as machine learning [8], automated design debugging [9] and circuit debugging [10].

There are two well-known classes of algorithms for solving MAX-SAT in practice: complete algorithms [11], [12] based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [13], [14] and stochastic local search (SLS) ones evolving out of GSAT [15] and WalkSAT [16]. Although SLS algorithms do not guarantee the optimality of solutions, they are able to return a solution with good quality efficiently. SLS is an efficient method for solving complex problems in science and industry [17]. Especially SLS algorithms are appealing when the instance is large in size, or when a reasonably good solution is needed in a short time, or when the knowledge about the problem domain is rather limited. SLS algorithms have witnessed great success in the SAT problem. However, far fewer works have been done to improve the performance of SLS algorithms for MAX-SAT. Especially, while it is well acknowledged that complete algorithms can solve structured instances efficiently, SLS algorithms are considered to be not as effective on highly structured problem domains [18]. In this paper, we focus on improving SLS for solving structured weighted MAX-SAT instances, including industrial and crafted ones, as well as random weighted ones.

SLS algorithms for MAX-SAT work as follow. In the beginning, an assignment mapping to all variables is generated randomly. Then the algorithm selects a variable to flip iteratively until timeout. Most SLS algorithms for MAX-SAT pick the flipping variable between two modes: the greedy (intensification) mode and the random (diversification) mode. In the greedy mode, they prefer variables whose flips can decrease the number of unsatisfied clauses; while

- *C. Luo and W. Wu are with the Key Laboratory of High Confidence Software Technologies of Ministry of Education, Peking University, Beijing 100871, China. E-mail: {chuanluosaber, william.third.wu}@gmail.com.*
- *S. Cai is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China, and the Queensland Research Laboratory, National ICT Australia, Brisbane 4001, QLD, Australia. E-mail: shaoweicai.cs@gmail.com.*
- *Z. Jie is with the Department of Online Game Business, NetEase, Inc., Hangzhou 310052, Zhejiang, China. E-mail: pkutcsj@gmail.com.*
- *K. Su is with the Institute for Integrated and Intelligent Systems, Griffith University, Brisbane 4111, QLD, Australia. E-mail: k.su@griffith.edu.au.*

Manuscript received 6 Aug. 2013; revised 3 June 2014; accepted 26 June 2014. Date of publication 7 Aug. 2014; date of current version 10 June 2015. Recommended for acceptance by S. Rajasekaran.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2014.2346196

in the random mode, they tend to better explore the search space and avoid local optima, usually using randomized strategies to pick a variable.

An important issue for local search is the cycling problem, i.e., revisiting a candidate solution that has been visited recently [19]. A typical and influential strategy for handling the cycling problem is tabu search [20]. Tabu search has been successfully used in weighted MAX-SAT problem, leading to some algorithms: reactive tabu search (H-RTS) [21], iterated robust tabu search (IRoTS) [1] and so on. Recently, a new diversification strategy called configuration checking (CC) was proposed for the cycling problem. The CC strategy has led to state-of-the-art SLS algorithms for SAT, such as CCASat [22], [23], which won the random SAT track of SAT Challenge 2012. The key notion in existing CC heuristics for SAT is the concept of *configuration changed decreasing* (CCD) variables [22], [23], [24]. However, when applying CCD variables to weighted MAX-SAT, this notion does not lead to good performance. The reason is possibly that the number of CCD variables is rather limited in the search process of SLS for MAX-SAT (Related empirical analyses can be found in Section 4).

In this work, we define a new concept, i.e., the *configuration changed and make positive* (CCMP) variable, which is configuration changed and with positive $make$. Based on CCMP variables, we propose a new heuristic named *configuration checking with make* (CCM), which prefers CCMP variables. Specifically, if CCMP variables exist globally, CCM would pick the one with greatest $score$ to flip; otherwise CCM would pick a variable randomly from a random unsatisfied clause. We combine CCM with a pure random walk heuristic to design an SLS algorithm called *configuration checking local search* (CCLS), which performs very well on weighted MAX-SAT instances. In order to search all CCMP variables more efficiently, we propose a new efficient approach, and also prove that the proposed approach has lower complexity than the naive one.

To evaluate the efficiency and the robustness of CCLS, we compare CCLS against a state-of-the-art SLS solver IRoTS [1] and two state-of-the-art complete solvers namely akmaxsat_ls [25] and New WPM2 [26], on a broad range of weighted MAX-SAT instances including industrial, crafted and random ones. Experimental results show that CCLS significantly outperforms IRoTS and finds much better solutions than akmaxsat_ls and New WPM2 on most difficult instances. Also, CCLS participated in the incomplete solvers track of the Eighth Max-SAT Evaluation (Max-SAT 2013) [27], and won four categories, illustrating that CCLS establishes the latest state of the art in SLS algorithms for MAX-SAT.

The remainder of the paper is structured as follows. In Section 2, we provide some necessary definitions and notations. Then, we review the CC strategy in Section 3. In Section 4, we present the CCM heuristic and use it to design an SLS algorithm called CCLS for weighted MAX-SAT, and also perform theoretical and empirical analyses. After that, experimental evaluations and the analyses of the results in Max-SAT 2013 are illustrated in Section 5. Finally, we conclude the paper and give some future work in Section 6.

2 Preliminaries

Given a set of n Boolean variables $V = \{x_1, x_2, \ldots, x_n\}$ and also the set of literals corresponding to these variables namely $L = \{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n\}$, a *clause* is a disjunction of literals. Using clauses and the logical operation AND ($\wedge$), we can construct a conjunction normal form (CNF) formula, i.e., $F = c_1 \wedge \cdots \wedge c_m$, where the number of clauses in F is denoted by m. A formula can be described as a set of clauses. An exact MAX-k-SAT formula is a CNF formula in which each clause contains precisely k literals. We use $V(F)$ to denote the set of all variables appearing in F. Two different variables are neighbors when they appear in at least one clause simultaneously, and the notation $N(x) = \{y \mid y \in V(F),\ y \text{ and } x \text{ are neighbors}\}$ is the set of all neighbors of variable x. We also use $C(F)$ to denote the set of all clauses appearing in F.

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*. For local search algorithms for MAX-SAT, a candidate solution is a complete assignment. Given a complete assignment α, each clause has two possible *states*: *satisfied* or *unsatisfied*: A clause is satisfied if at least one literal in that clause is true under α; otherwise, it is unsatisfied. We use UC to denote the set of all unsatisfied clauses, and also utilize $V(UC)$ to denote the set of all variables in UC. Additionally, for the weighted MAX-SAT problem, each clause c is associated with its weight $w(c)$, and the weighted MAX-SAT problem consists in seeking out an optimal assignment maximizing the total weight of satisfied clauses. We also define $w_{max} = \max\{w(c) \mid c \in C(F)\}$ and $w_{min} = \min\{w(c) \mid c \in C(F)\}$.

Given a weighted CNF formula F, the cost of an assignment α, denoted as $cost(F, \alpha)$, is the total weight of all unsatisfied clauses under α. In SLS algorithms for weighted MAX-SAT, for a variable x, the property $make(x)$ is defined as the total weight of clauses that would become satisfied if variable x is flipped; $break(x)$ is the total weight of clauses that would become unsatisfied if variable x is flipped; $score(x)$ is the increment in the total weight of satisfied clauses if variable x is flipped, and can be understood as $make(x) - break(x)$. The CCM heuristic utilizes $make$ and $score$ to select the flipping variable.

3 Configuration Checking Review

This section reviews the configuration checking (CC) strategy, which is an important idea in the CCM heuristic. The CC strategy [28] works as a form of diversification, preventing SLS algorithms from encountering a scenario it recently faced. To this end, it forbids flipping any variable whose circumstance information has not been changed since its last flip.

The circumstance information is formally defined as the concept of $configuration$. In the context of SAT (and MAX-SAT), the $configuration$ of a variable x refers to a vector consisting of Boolean values of x's all neighboring variables [24].

The CC strategy forbids a variable x to be flipped if $configuration(x)$ has not changed since x's last flip. Those variables whose configurations have changed after their last flips can be explicitly defined as below.

Definition 1. *A variable x is defined as a* configuration changed *variable if and only if after the last time x was flipped, at least one variable $y \in N(x)$ has been flipped.*

As stated in the literature [24], an implementation of the CC strategy is to employ a Boolean array $confChange$ for variables, where $confChange(x) = 1$ 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 greedy mode.

At the start of local search, for each variable $x \in V(F)$, $confChange(x)$ is initialized as 1; afterwards, when flipping variable x , $confChange(x)$ is reset to 0, and for each variable $y \in N(x)$, $confChange(y)$ is reset to 1.

An important notation in the CC strategy is the *configuration changed decreasing* variable [22], [23], [24], which is formally defined as below.

Definition 2. *A variable x is defined as a* configuration changed decreasing *variable if and only if $score(x) > 0$ and $confChange(x) = 1$.*

In this work, we use *CCDvars* to denote the set of all CCD variables. Previous CC-based SLS algorithms only allow CCD variables to be flipped in the greedy mode [22], [23], [24].

4 The CCM Heuristic and The CCLS Algorithm

In this section, we propose the CCM heuristic, and employ it in designing an SLS algorithm called CCLS for weighted MAX-SAT.

4.1 The CCM Heuristic

Previous CC heuristics for SAT prefer to pick the flipping variable from CCD variables. When applying CCD variables to SLS algorithms for MAX-SAT, this notion loses its power, because there are a very limited number of CCD variables during the search process of SLS for many MAX-SAT instances, as will be shown in Section 4.4. A possible explanation is that for many MAX-SAT instances, the size (or weight) of maximum satisfiable subsets is relatively small compared to the total number (or weight) of clauses, and thus there are not as many decreasing variables ($score(x) > 0$) as in satisfiable instances.

In this work, we propose a new CC heuristic called *configuration checking with make*. Unlike previous CC heuristics which filter candidate variables by combining the $score$ property, CCM filters candidate variables by considering the $make$ property, to enlarge the candidate flipping variable set. This is captured by the concept of the *configuration changed and make positive* variable, which is formally defined as follow.

Definition 3. *A variable x is defined as a* configuration changed and make positive *variable if and only if $make(x) > 0$ and $confChange(x) = 1$.*

In this work, we use *CCMPvars* to denote the set of all CCMP variables during the search. The following propositions state that the *CCMPvars* set is a superset of the *CCDvars* set.

Proposition 1. *For a given variable $x \in V(F)$, if x is a CCD variable, then x is a CCMP variable.*

Proof. At the current search step, for a given variable x, if x is a CCD variable, then $score(x) > 0$ and $confChange(x) = 1$.

Since $score(x) > 0$ and $score(x)$ is defined as $make(x) - break(x)$, we have $make(x) > break(x)$. Recalling that $break(x)$ is the total weight of clauses that would become unsatisfied if variable x is flipped, $break(x)$ is always a nonnegative integer. So, we have $make(x) > break(x) \geq 0$, and thus $make(x) > 0$.

Therefore, variable x is with $make(x) > 0$ and $confChange(x) = 1$. That is, variable x is a CCMP variable. □

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

Proposition 1 and Remark 1 immediately lead to the following proposition.

Proposition 2. *The* CCMPvars *set is a superset of the* CCDvars *set.*

In the CCM heuristic, *CCMPvars* is adopted as the candidate flipping variable set. Thus, in each search step, the number of candidate variables in the CCM heuristic is greater than those in the previous CC heuristics for SAT employing *CCDvars*.

The CCM heuristic switches between the greedy mode and the random mode, depending on whether *CCMPvars* is empty or not. If *CCMPvars* is not empty, CCM works in the greedy mode; otherwise, CCM picks the flipping variable according to the criterion in the random mode. How these two mode work can be described as follows.

The greedy mode. The CCM heuristic does a gradient decreasing walk, i.e., selecting the variable with greatest $score$ in *CCMPvars* as the flipping variable, breaking ties randomly.

The random mode. The CCM heuristic does a random walk, i.e., picking an unsatisfied clause c randomly at first and then selecting a random variable as the flipping variable from c.

4.2 The CCLS Algorithm

In this section, we combine the CCM heuristic with a pure random walk heuristic to design the CCLS algorithm. The pseudo-code of CCLS is outlined in Algorithm 1, as described below.

In the beginning, CCLS generates an assignment α randomly, and the best solution α^* is initialized as α. For each variable x, $confChange(x)$ is initialized as 1.

After the initialization, CCLS executes search steps until the number of search steps exceeds the step limit $maxSteps$. In each search step, with fixed probability p, CCLS applies the random walk heuristic, i.e., selecting an unsatisfied clause c randomly and then picks a variable in c randomly; otherwise, CCLS activates the CCM heuristic between the greedy mode and the random mode, as described in the preceding section, to select the flipping variable. Whenever a better solution is found, α^* is updated accordingly.

After picking the flipping variable, CCLS flips the selected variable. CCLS repeats picking and flipping a variable and updating $confChange$ until exceeding the step limit. Finally, the CCLS algorithm reports the best solution α^* that has been found.

Algorithm 1. CCLS

Input: CNF-formula F, $maxSteps$
Output: An assignment α^* of F

```
1  begin
2    generate a random assignment α, α* ← α;
3    initialize confChange(x) as 1 for each variable x;
4    for step ←1 to maxSteps do
5      if with fixed probability p then
6        c ← a random unsatisfied clause;
7        v ← a random variable in c;
8      else
9        if CCMPvars is not empty then
10         v ← x with the greatest score in CCMPvars,
           breaking ties randomly;
11       else
12         c ← a random unsatisfied clause;
13         v ← a random variable in c;
14     α ← α with v flipped;
15     if cost(F, α) < cost(F, α*) then α* ← α;
16     update confChange;
17   return α*;
18 end
```

We outline the parameter setting of p in the CCLS algorithm in Table 1, and introduce it as follows. For weighted exact MAX-2-SAT instances with $w_{max} - w_{min} < 800$, p is set to 0.37. For weighted exact MAX-3-SAT instances with $w_{max} - w_{min} < 800$, p is set to 0.42. For other instance types, p is set to 0.2.

We would like to note that CCLS has only one parameter (i.e., the fixed probability p), while numerous previous SLS algorithms [1], [22], [23], [24], [29] involve in several parameters.

4.3 An Efficient Approach to Search CCMP Variables

As discussed in preceding sections, the set of CCMP variables is the most important component of CCLS. Thus, an efficient approach to search all CCMP variables is able to accelerate the search efficiency of CCLS.

The naive approach to search all CCMP variables is to visit each variable in the formula and verify whether each variable in the formula conforms to the criterion of a CCMP variable.

Proposition 3. *Given a CNF formula F and an assignment α to $V(F)$, the complexity of the naive approach to search all CCMP variables is $O(|V(F)|)$.*

Proof. According to the description of the naive approach to search all CCMP variables, each variable $x \in V(F)$ should be verified one time. Therefore, the complexity of the naive approach to search all CCMP variables is $O(|V(F)|)$. □

Before introducing the efficient approach to search all CCMP variables, we first prove an important property of CCMP variables—all CCMP variables appear in unsatisfied clauses.

Proposition 4. *For a given variable $x \in V(F)$, if x is a CCMP variable, then x appears in at least one unsatisfied clause.*

TABLE 1
The Parameter Setting of p in the CCLS Algorithm

Instance Type	p in CCLS
Weighted Exact MAX-2-SAT with $w_{max} - w_{min} < 800$	0.37
Weighted Exact MAX-3-SAT with $w_{max} - w_{min} < 800$	0.42
Other Instance Types	0.2

Proof. If at the current search step, for a given variable x, x is a CCMP variable, then $make(x) > 0$ and $confChange(x) = 1$.

Recall that $make(x)$ is defined as the total weight of clauses that would become satisfied if variable x is flipped. Thus, $make(x) > 0$ implies that x appears in at least one unsatisfied clause (otherwise $make = 0$).

Therefore, if x is a CCMP variable, then x appears in at least one unsatisfied clause. □

Based on Proposition 4, we propose an efficient approach to search all CCMP variables, which is to visit each variable $x \in V(UC)$ and verify whether each variable $x \in V(UC)$ conforms to the criterion of a CCMP variable, recalling that $V(UC)$ is the set of all variables in all unsatisfied clauses in the current search step. We would like to note that the maintenance of $V(UC)$ can be accomplished with the maintenance of $confChange$ and $score$, so the complexity of the maintenance of $V(UC)$ can be ignored.

Proposition 5. *Given a CNF formula F and an assignment α to $V(F)$, the complexity of the proposed efficient approach to search all CCMP variables is $O(|V(UC)|)$.*

Proof. According to the description of the proposed efficient approach to search all CCMP variables, each variable $x \in V(UC)$ should be verified one time. Therefore, the complexity of the proposed efficient approach to search all CCMP variables is $O(|V(UC)|)$. □

As it is apparent that $V(UC)$ is a subset of $V(F)$, thus the complexity of the proposed efficient approach is less than that of the naive approach.

In the implementation of CCLS, we adopt this proposed efficient approach to search CCMP variables.

4.4 Differences between the CCM Heuristic and Previous CC Based Heuristics

Compared to the CCM heuristic which filters candidate variables by CCMP variables, previous CC based heuristics [22], [23], [24] focus on CCD variables, and use the set of CCD variables as the candidate variable set. We have theoretically proved that the set of CCMP variables is a superset of the set of CCD variables (Proposition 2). Now we conduct experiments to figure out the significant gap between the averaged number of CCMP variables and that of CCD variables for many MAX-SAT instances.

Table 2 reports the averaged numbers of CCMP variables and CCD variables in each step of CCLS on the hardest structured MAX-SAT instances whose empirical hardness is demonstrated in both Tables 5 and 6 in Section 5. CCLS is preformed 10 runs for each instance with a cutoff time of 1,000 seconds. From Table 2, there are significantly more CCMP variables than CCD variables for a number of

TABLE 2
Averaged Numbers of CCMP Variables and CCD Variables in Each Step of CCLS on the Hardest Structured Instances

Instance	CCLS		
	avg #CCMP	avg #CCD	fraction
mul_8_3.wcnf	399.26	362.89	1.10
mul_8_9.wcnf	567.49	524.87	1.08
mul_8_11.wcnf	1127.25	1062.3	1.06
mul_8_13.wcnf	955.79	894.31	1.07
mul_8_14.wcnf	1031.59	974.71	1.06
sbox_8.wcnf	2006.87	1563.35	1.28
ram_k4_n18.ra1.wcnf	74.22	2.34	31.72
ram_k4_n19.ra1.wcnf	104.78	2.37	44.21
ram_k4_n20.ra1.wcnf	136.67	2.33	58.66
t7g3-9999.spn.wcnf	268.11	1.30	206.24
frb59-26-1.wcnf	1481.91	5.03	294.61
frb59-26-2.wcnf	1482.07	5.03	294.65
frb59-26-3.wcnf	1482.04	5.03	294.64
frb59-26-4.wcnf	1482.00	5.03	294.63
frb59-26-5.wcnf	1482.07	5.03	294.65

The legends 'avg #CCMP' and 'avg #CCD' mean the averaged numbers of CCMP variables and CCD variables in each step of CCLS, respectively. The legend 'fraction' means the fraction of the averaged number of CCMP variables in each step of CCLS over the averaged number of CCD variables in each step of CCLS.

relatively difficult instances (i.e., all five instances in the ram_k4_n* class, the t7g3-9999.spn.wcnf instance and all 5 instances in the frb59-26-* class).

An important issue in SLS algorithms for MAX-SAT is the balance between intensification and diversification. Selecting CCMP variables and CCD variables contributes to intensification, while performing random walk contributes to diversification. Fig. 1 clearly presents the relationship between intensification and diversification as well as the contributions of CCMP variables, CCD variables and random walk in SLS algorithms. As the number of CCD variables is rather limited, previous CC based heuristics [22], [23], [24], which filter candidate variable set by CCD variables, would bias too much towards to diversification, and thus is not reasonable.

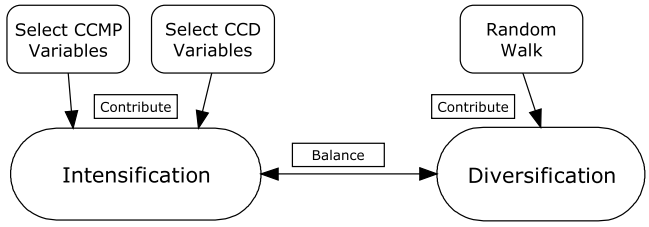


Fig. 1. The relationship between intensification and diversification as well as the contributions of CCMP variables, CCD variables and random walk in SLS algorithms.

To show the superiority of the CCM heuristic over previous CC based heuristics which concentrate on CCD variables, we modify CCLS to prefer CCD variables instead of CCMP ones (lines 9-10 in Algorithm 1), resulting in an alternative algorithm called CCLSdeg1. We conduct a comparison between CCLS and CCLSdeg1 with the optimal setting ($p = 0.2$, according to some preliminary experiments) on the hardest structured instances, 10 runs for each instance with a cutoff time of 1,000 seconds. The results are summarized in Table 3, which shows that CCLS finds better solutions than CCLSdeg1 on 10 (out of 15) instances. For other instances but one, the two algorithms find the solutions with the same quality.

Also, Table 4 reports the frequencies of intensification and diversification steps in CCLS and CCLSdeg1 on these hardest structured instances. According to Table 4, the frequency of intensification steps and diversification steps in CCLS is approximately 80 and 20 percent, which is a desirable proportion for SLS algorithms, as suggested in the literature [30]. Moreover, the famous 80/20 rule has been applied in many fields, such as collection management [31], library loans [32] and software engineering [33]. Our experiments also justify this implication about the 80/20 rule in SLS algorithms.

From Tables 2 and 3, for instances where the number of CCMP variables is significantly more than that of CCD variables, CCLS performs much better than CCLSdeg1, indicating that the CCM heuristic is more effective than previous CC heuristics [22], [23], [24] which focus on CCD variables.

TABLE 3
Comparison among CCLS and Its Degenerating Versions on the Hardest Structured Instances

Instance	CCLS		CCLSdeg1		CCLSdeg2	
	sol.	avg sol.	sol.	avg sol.	sol.	avg sol.
mul_8_3.wcnf	36	36	36	36	36	37.1
mul_8_9.wcnf	42	42	42	42	42	42.3
mul_8_11.wcnf	64	65.2	64	**64.2**	64	70.6
mul_8_13.wcnf	60	60	60	60	60	60.7
mul_8_14.wcnf	56	56	56	56	56	62.1
sbox_8.wcnf	414	417.3	443	449.3	**333**	**359.2**
ram_k4_n18.ra1.wcnf	**159**	**301**	318	351.1	2,920	4212.5
ram_k4_n19.ra1.wcnf	**881**	**1082.2**	931	1090.5	4,616	6388.9
ram_k4_n20.ra1.wcnf	**2409**	**2784.7**	2483	2831.9	7,950	10114.4
t7g3-9999.spn.wcnf	**1,20,29,063**	**1,21,12,122**	1,23,38,213	1,24,63,872	1,41,81,675	15836007.6
frb59-26-1.wcnf	**1476**	**1476.4**	1479	1479.9	1,481	1482.2
frb59-26-2.wcnf	**1476**	**1476.6**	1478	1479.8	1,480	1481.7
frb59-26-3.wcnf	**1476**	**1476.8**	1479	1480.2	1,481	1482.7
frb59-26-4.wcnf	**1476**	**1476.7**	1479	1,480	1,480	1481.5
frb59-26-5.wcnf	**1475**	**1475.7**	1479	1480.1	1,481	1482.9

The legend 'sol.' means the minimum unsatisfied weight among all runs within the cutoff time. The legend 'avg sol.' means the averaged value of the unsatisfied weights of all runs within the cutoff time.

TABLE 4
The Frequencies of Intensification and Diversification Steps in CCLS and CCLSdeg1

Instance	CCLS		CCLSdeg1	
	inten.	diver.	inten.	diver.
mul_8_3.wcnf	79.98%	20.02%	64.32%	35.68%
mul_8_9.wcnf	79.98%	20.02%	65.48%	34.52%
mul_8_11.wcnf	79.98%	20.02%	67.98%	32.02%
mul_8_13.wcnf	79.98%	20.02%	67.91%	32.09%
mul_8_14.wcnf	79.98%	20.02%	66.67%	33.33%
sbox_8.wcnf	79.97%	20.03%	61.49%	38.51%
ram_k4_n18.ra1.wcnf	79.98%	20.02%	65.13%	34.87%
ram_k4_n19.ra1.wcnf	79.98%	20.02%	65.27%	34.73%
ram_k4_n20.ra1.wcnf	79.98%	20.02%	65.06%	34.94%
t7g3-9999.spn.wcnf	79.98%	20.02%	61.89%	38.11%
frb59-26-1.wcnf	79.98%	20.02%	71.93%	28.07%
frb59-26-2.wcnf	79.98%	20.02%	71.94%	28.06%
frb59-26-3.wcnf	79.98%	20.02%	71.91%	28.09%
frb59-26-4.wcnf	79.98%	20.02%	71.92%	28.08%
frb59-26-5.wcnf	79.98%	20.02%	71.94%	28.06%

The legends 'inten.' and 'diver.' mean the frequencies of intensification and diversification steps during the search, respectively.

The reason is that, compared to previous CC heuristics, the CCM heuristic stricks a better balance between intensification and diversification, as shown in Table 4.

4.5 Effectiveness of Pure Random Walk

We also compare CCLS with another alternative degenerating algorithm CCLSdeg2, which works without the pure random walk heuristic (i.e., setting p to 0). The benchmarks and experimental settings are the same as those used in the evaluation of the CCLSdeg1 algorithm. As can be seen from Table 3, CCLS performs much better than CCLSdeg2 on all instances but one, indicating that the pure random walk heuristic makes contribution to the diversification on CCLS and thus improves the performance of CCLS.

4.6 Differences between CCLS and IRoTS

In this section, we discuss the differences between CCLS and a state-of-the-art SLS solver for MAX-SAT called IRoTS [1]. IRoTS [1] is an iterated local search (ILS) algorithm, which alternates between local searches and so-called perturbation phases. The so-called perturbation phases are designed to take the search away from the local optimum reached by the subsidiary local search procedure. IRoTS diversifies the search by the tabu mechanism [20], and uses a robust tabu search for both the subsidiary local search and perturbation phases.

There are significant differences between CCLS and IRoTS. First, compared to the ILS paradigm utilized by IRoTS, CCLS adopts the two-mode SLS paradigm which switches between intensification and diversification according to a fixed probability p. Second, CCLS relies on the CCM heuristic and thus diversifies the search by the CC strategy, while IRoTS diversifies the search by the tabu mechanism.

5 Experimental Evaluation

In this section, we first introduce the benchmarks, the competitors and the experimental preliminaries about our experiments. Then CCLS is compared with its state-of-the-art competitors on a broad range of benchmarks. Finally, we report the performance and results of CCLS in Max-SAT 2013. We note that CCLS wins these four tracks of incomplete solvers in Max-SAT 2013.

5.1 The Benchmarks

We evaluate CCLS on a broad range of benchmarks including industrial, crafted, and random instances. There are three benchmarks adopted in the experiments.

The first one contains all instances from the industrial weighted category in the Sixth Max-SAT Evaluation (Max-SAT 2011)[1] and the crafted weighted category in the Seventh Max-SAT Evaluation (Max-SAT 2012)[2] (except for frb instances, which are included in the second benchmark). We would like to note that those instances from the industrial weighted category in Max-SAT 2011 are all translated from components of the advanced encryption standard (AES) and small-scale variants. More importantly, AES has been widely used in a variety of applications, such as secure communication systems, high-performance database servers, digital video/audio recorders, RFID tags, and smart cards [34], and also has been lately utilized for the bitstream security mechanisms in the field-programmable gate arrays (FPGAs) for increasing the reliability of the FPGA-based designs [35], [36]. There is no non-partial industrial weighted instance in Max-SAT 2012 and the instances from crafted weighted category in Max-SAT 2011 are identical to those from crafted weighted category in Max-SAT 2012.

The second benchmark is the famous crafted benchmark frb.[3] The frb instances are generated in the phase transition area according to Model RB [37], and are very difficult to solve by current techniques in spite of their relatively small size. Also, these instances generated in the phase transition have proved to be difficult in both theory and practice [38].

For the third benchmark, we use the makewff generator adopted in the literature [39] with minor modifications to generate 80 random weighted MAX-2-SAT instances and 80 random weighted MAX-3-SAT instances ($n = 4{,}000, 5{,}000$; clause-to-variable ratio $r = 2, 4, 6, 8$; each clause weight is an integer chosen randomly from 1 to 10). There are 10 instances for each ratio. Because the original makewff generator[4] can only generate unweighted random MAX-SAT instances, we just modify the generator to let it generates the clause weights. We do not adopt random instances from MAX-SAT evaluations as they are too small and most of them are easier for modern SLS solvers to find the optimal solutions.

5.2 The Competitors

We evaluate CCLS on a broad range of weighted MAX-SAT instances, compared with a state-of-the-art SLS solver called IRoTS [1] and two state-of-the-art complete solvers namely akmaxsat_ls [25] and New WPM2 [26].

1. http://maxsat.ia.udl.cat:81/11/benchs/wms_industrial.tgz
2. http://www.maxsat.udl.cat/12/benchs/wms_crafted.tgz
3. http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/max-sat-benchmarks.htm
4. http://www.cs.wustl.edu/~zhang/projects/backboneGuidedSearch/bgwalksat/bg-dyna-walksat.tgz

TABLE 5
Results on the First Benchmark

Instance	CCLS		IRoTS		akmaxsat_ls			New WPM2	
	sol. avg sol.	time	sol. avg sol.	time	ls. sol. avg ls. sol.	sol. avg sol.	time #proved	sol.	time #proved
mul_8_11.wcnf	**64**	57.5	448	<1.0	389	Not Improving	Time Out	97,97,87,520	Time Out
	65.2		465.4		427.5	Not Improving	0		0
mul_8_13.wcnf	**60**	2.3	426	<1.0	408	Not Improving	Time Out	196	Time Out
	60		465.8		431.2	Not Improving	0		0
mul_8_14.wcnf	**56**	23.2	317	61.9	394	Not Improving	Time Out	93,35,69,280	Time Out
	56		425.6		413	Not Improving	0		0
mul_8_3.wcnf	**36**	<1.0	72	391.3	72	Not Improving	Time Out	32,95,98,720	Time Out
	36		90.3		89.2	Not Improving	0		0
mul_8_9.wcnf	**42**	<1.0	156	255.3	148	Not Improving	Time Out	51,18,60,480	Time Out
	42		172.4		158.4	Not Improving	0		0
sbox_8.wcnf	**414**	595.8	672	33.2	N/A	N/A	Time Out	664	Time Out
	417.3		686		N/A	N/A	0		0
ram_k4_n18.ra1.wcnf	**159**	507.4	275	345.3	664	Not Improving	Time Out	3,06,40,81	Time Out
	301		379.4		826.7	Not Improving	0		0
ram_k4_n19.ra1.wcnf	**881**	439.6	1,142	388.1	1763	Not Improving	Time Out	38,83,075	Time Out
	1082.2		1278.6		2078.3	Not Improving	0		0
ram_k4_n20.ra1.wcnf	**2409**	429.3	3046	453.8	4327	Not Improving	Time Out	48,44,949	Time Out
	2784.7		3337.5		4613.8	Not Improving	0		0
t7g3-9999.spn.wcnf	1,20,29,063	420.2	11954769	286.6	1,23,11,531	11,95,47,69	Time Out	1,60,55,421	Time Out
	12112122.9		**1,19,56,603**		12446037.4	12145614.2	0		0

To save space, we do not report the 79 relatively simple instances. For the 79 instances not reported in this table, akmaxsat_ls proves optimum for 69 instances and New WPM2 proves optimum for 9 instances; also, CCLS and IRoTS very quickly find better or same-quality solutions compared to akmaxsat_ls and New WPM2. As the local search component of akmaxsat_ls fails to report a solution on the instance sbox_8.wcnf within the cutoff time (1,000 seconds), we report 'N/A' for the quality of solution of akmaxsat_ls on sbox_8.wcnf.

TABLE 6
Results on the Second Benchmark

Instance Class	#var	CCLS		IRoTS		akmaxsat_ls			New WPM2	
		sol. avg sol.	time	sol. avg sol.	time	ls. sol. avg ls. sol.	sol. avg sol.	time #proved	sol.	time #proved
frb25-13	325	300	<1.0	300	134.17	302	Not Improving	Time Out	305.4	Time Out
		300		300.66		302.42	302.2	0		0
frb30-15	450	**420**	<1.0	421.4	294.57	423	Not Improving	Time Out	429	Time Out
		420		421.94		423.7	Not Improving	0		0
frb35-17	595	**560**	5.99	563	259.55	564.6	Not Improving	Time Out	569	Time Out
		560		563.44		565.22	Not Improving	0		0
frb40-19	760	**720**	62.7	724.2	288.48	725.6	Not Improving	Time Out	734.4	Time Out
		720		724.86		726.72	726.7	0		0
frb45-21	945	**900**	192.57	905.6	267.19	907	Not Improving	Time Out	917.6	Time Out
		900		906.62		908.16	Not Improving	0		0
frb50-23	1,150	**1100.2**	154.98	1107.6	272.1	1108.6	Not Improving	Time Out	1123.2	Time Out
		1100.54		1108.14		1109.64	Not Improving	0		0
frb53-24	1,272	**1219.2**	304.68	1227.2	328.86	1228.4	Not Improving	Time Out	1242.2	Time Out
		1219.68		1228.18		1229.82	Not Improving	0		0
frb56-25	1,400	**1344.4**	424.52	1352.8	298.74	1,354	Not Improving	Time Out	1369.2	Time Out
		1344.96		1353.92		1355.36	Not Improving	0		0
frb59-26	1,534	**1475.8**	267.31	1484.6	325.55	1485.4	Not Improving	Time Out	1499.8	Time Out
		1476.38		1485.64		1487.26	Not Improving	0		0

Each class presented in this table contains 5 instances. The legend '#var' means the number of variables of each instance in this instance calss. To save space, we do not report the 14 relatively simple instances (3 instance classes). For the 14 instances not reported in this table, akmaxsat_ls proves optimum for all these 14 instances and New WPM2 proves optimum for 9 instances; also, CCLS and IRoTS very quickly find better or same-quality solutions compared to akmaxsat_ls and New WPM2.

The IRoTS solver is the best SLS solver for weighted MAX-SAT instances in Max-SAT 2012 and wins several tracks of incomplete solvers in that evaluation. Reported in the literature [1], IRoTS outperforms many effective SLS algorithms such as GLS [40] and ILS-YI [41] on weighted MAX-SAT instances. Also, reported in the literature [40], GLS outperforms MaxWalkSAT [42] and DLM [43] on weighted MAX-SAT instances. Therefore, IRoTS represents the state of the art in SLS for weighted MAX-SAT instances. We adopt the best implemented version which is integrated in the UBCSAT framework [44] and is available online.[5]

5. http://ubcsat.dtompkins.com/downloads/ubcsat-1-1-0.tar.gz?attredirects=0&d=1

TABLE 7
Results on Random Weighted MAX-2-SAT Instances in the Third Benchmark

Instance Class	CCLS		IRoTS		akmaxsat_ls			New WPM2	
	sol. avg sol.	time	sol. avg sol.	time	ls. sol. avg ls. sol.	sol. avg sol.	time #proved	sol.	time #proved
k2-v4000-r2	**2790.2**	466.32	2803.6	465.5	2912.1	Not Improving	Time Out	9835.1	Time Out
	2799.07		2815.82		2950.16	Not Improving	0		0
k2-v4000-r4	**8881.6**	548.97	8887.9	518.67	9032.4	Not Improving	Time Out	21071.1	Time Out
	8888.49		8900.22		9092.39	Not Improving	0		0
k2-v4000-r6	**16007.2**	509.27	16025.3	517.64	16236.3	Not Improving	Time Out	31970.3	Time Out
	16019.05		16040.07		16304.12	Not Improving	0		0
k2-v4000-r8	**23677.7**	553.53	23697.1	534.53	23942.5	Not Improving	Time Out	43073.2	Time Out
	23695.4		23716.83		24024.62	Not Improving	0		0
k2-v5000-r2	**3463.4**	427.85	3486.4	537.82	3652.2	Not Improving	Time Out	12498.5	Time Out
	3476.79		3502.05		3711.86	Not Improving	0		0
k2-v5000-r4	**11079.7**	452.98	11100.6	512.64	11390.7	Not Improving	Time Out	26567.6	Time Out
	11090.49		11113.6		11481.51	Not Improving	0		0
k2-v5000-r6	**19959.2**	610.69	19983.9	582.36	20445.7	Not Improving	Time Out	40481.1	Time Out
	19976.59		20010.82		20533.89	Not Improving	0		0
k2-v5000-r8	**29719.9**	507.21	29766.6	547.85	30325.5	Not Improving	Time Out	54251.8	Time Out
	29749.38		29797.57		30466.29	Not Improving	0		0

Each class contains 10 instances.

TABLE 8
Results on Random Weighted MAX-3-SAT Instances in the Third Benchmark

Instance Class	CCLS		IRoTS		akmaxsat_ls			New WPM2	
	sol. avg sol.	time	sol. avg sol.	time	ls. sol. avg ls. sol.	sol. avg sol.	time #proved	sol.	time #proved
k3-v4000-r2	**57.8**	364.29	174.1	510.26	248.8	Not Improving	Time Out	981.5	Time Out
	59.6		191.17		277.72	Not Improving	0		0
k3-v4000-r4	**1482.2**	490.06	1651.6	490.87	1847.8	Not Improving	Time Out	87729.1	Time Out
	1511.42		1677.16		1922.96	Not Improving	0		0
k3-v4000-r6	**3777.2**	538.84	3929.4	533.95	4244.2	Not Improving	Time Out	131860	Time Out
	3851.29		3981.15		4323.48	Not Improving	0		0
k3-v4000-r8	**6578.7**	594.84	6737.5	498.43	7059.4	Not Improving	Time Out	175844.8	Time Out
	6672.87		6,796		7194.07	Not Improving	0		0
k3-v5000-r2	**81.6**	439.77	234.7	519.94	356	Not Improving	Time Out	1320.2	Time Out
	84.1		264.97		391.13	Not Improving	0		0
k3-v5000-r4	**1886.5**	545.78	2138.4	513.52	2586.3	Not Improving	Time Out	109929.1	Time Out
	1930.74		2175.2		2661.56	Not Improving	0		0
k3-v5000-r6	**4748.9**	538.03	5,003	496.23	5691	Not Improving	Time Out	164972.4	Time Out
	4829.19		5056.24		5809	Not Improving	0		0
k3-v5000-r8	**8211.7**	642.33	8387.3	550.22	9371.8	Not Improving	Time Out	219873.9	Time Out
	8324.93		8482.44		9519.38	Not Improving	0		0

Each class contains 10 instances.

The akmaxsat_ls solver is the best complete solver for weighted MAX-SAT instances in Max-SAT 2012 and wins several tracks of complete solvers in that evaluation. Reported in the literature [25], akmaxsat performs better than previously best complete solvers such as IUT_BMB_Maxsatz (IUT_BCMB_WMaxsatz), WMaxSatz [45] and IncMaxSatz (IncWMaxSatz) [12] on randomly generated instances with a high clauses-to-variables ratio. The akmaxsat_ls solver equips akmaxsat with a local search solver by first running the local search solver to get a good initialized solution. Therefore, the akmaxsat_ls solver as an improved version of the akmaxsat solver is able to be seen as the state of the art of complete solvers for weighted MAX-SAT instances. In this experimental evaluation, we first download the latest version (Version 1.1) of the akmaxsat solver online,[6] and then build the adopting competing version of akmaxsat_ls based on the downloaded version of akmaxsat following its author's instructions. From our experimental results (Tables 5, 6, 7, and 8), on several relatively difficult structured instances (the first and second benchmarks), the complete component akmaxsat improves the quality of solution initialized by the local search component of akmaxsat_ls, while on the random instances (the third benchmark), akmaxsat does not improve the quality of solution initialized by the local search component.

New WPM2 shows state-of-the-art performance on solving industrial and crafted partial instances. Reported in the

6. http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/kuegel/akmaxsat_1.1.tgz

literature [26], New WPM2 performs better than WPM1 [46], original WPM2 [47], bincd2 [48], ilp [49], pwbo2.1 [50], ShinMaxSat and QMaxSAT [51] on industrial and crafted partial instances. The binary of New WPM2 is kindly provided by its author.

5.3 Experimental Preliminaries

CCLS is implemented in C++, and compiled by g++ with the '−O3' option. For IRoTS, akmaxsat_ls and New WPM2, we use the settings for the parameter settings introduced in literature [1], [25], [46], respectively.

Experiments are carried out on a machine with Intel Core i7 2.7 GHz CPU and 15.6 GB memory under Linux. Each SLS solver as well as the complete solver akmaxsat_ls performs 10 runs with a cutoff time of 1,000 seconds for each instance, while the other complete solver New WPM2 performs one run with the same cutoff time for each instance as it is deterministic.

For SLS solvers namely CCLS and IRoTs on each instance in the first benchmark, as well as each instance class in the second and the third benchmarks, we report the best quality of solution ('sol.') within the cutoff time, i.e., the minimum unsatisfied weight among all runs (for the second and third benchmark, this metric is averaged over all instances in each instance class); we report the averaged quality of solution ('avg sol.') within the cutoff time, i.e., the averaged value of the unsatisfied weights of all runs; we also report the averaged time ('time') for seeking out the best quality of solution among all runs.

For complete solvers namely akmaxsat_ls and New WPM2 on each instance in the first benchmark, as well as each instance class in the second and the third benchmarks, we report the best quality of solution ('sol.') within the cutoff time (for the second and third benchmark, this metric is averaged over all instances in each instance class); we report the averaged time ('time') for seeking out the best quality of solution among all runs not including proving time (if a complete solver fails to prove optimum in any run, we report 'Time Out' for the complete solver); we report the number of runs ('#proved') where the related complete solver proves optimum (for the second and third benchmark, this metric is averaged over all instances in each instance class). For akmaxsat_ls, we also report the averaged quality of solution ('avg sol.') within the cutoff time.

If akmaxsat_ls fails to prove optimum in any run, we report the solution found by akmaxsat_ls and its local search component separately—the notations 'sol.' and 'ls. sol.' mean the best quality solution found by akmaxsat_ls and its local search component, respectively (for the second and third benchmark, these two metrics are averaged over all instances in each instance class), and the notations 'avg sol.' and 'avg ls. sol.' mean the averaged averaged quality of solution found by akmaxsat_ls and its local search component, respectively. Further, if akmaxsat_ls fails to prove optimum in any run and also akmaxsat (the complete component of akmaxsat_ls) fails to improve the best quality of solution initialized by the local search component of akmaxsat_ls, we report 'Not Improving' for the notation 'sol.' of akmaxsat_ls. Also, if akmaxsat_ls fails to prove optimum in any run and also akmaxsat fails to improve the averaged quality of solution initialized by the local search component of akmaxsat_ls, we report 'Not Improving' for the notation 'avg sol.' of akmaxsat_ls.

When comparing several solvers, we first evaluate the quality of solution found by these solvers. If a solver reports a better (average) quality of solution than all its competitors, the solver performs better than others. Also, if there is more than one solver which reports the best (average) quality of solution among all these solvers, the solver with the less averaged time performs better than others. The results in the **bold** font indicate the best performance for an instance (or an instance class).

5.4 Experimental Results

Discussions on results of the first benchmark. Table 5 shows experimental results of CCLS and IRoTS, akmaxsat_ls and New WPM2 on the industrial and crafted instances. There are 89 instances in the first benchmark. To save space, we do not report the 79 relatively simple instances where CCLS and IRoTS find solutions of the same quality very quickly, which are better or the same compared to those found by akmaxsat_ls and New WPM2. Also we note that akmaxsat_ls proves optimum for 69 instances, and New WPM2 proves optimum for nine instances. This indicates that CCLS finds *optimal* solutions for at least 69 instances.

It is clear from Table 5 that the performance of CCLS is much better than that of IRoTS. In detail, CCLS finds significantly better solutions than IRoTS on nine instances, but finds a slightly worse solution on only one instance. Now we turn to the comparison among CCLS, akmaxsat_ls and New WPM2, the solutions found by CCLS is dramatically better than that found by akmaxsat_ls and New WPM2 on these difficult instances. On the other hand, we also notice the gap between the quality of solution found by IRoTS and that found by akmaxsat_ls is not significant. Therefore, CCLS really improves the state-of-the-art performance of SLS solvers on solving industrial and crafted instances.

Discussions on results of the second benchmark. We also compare CCLS and its competitors on the well-known frb benchmark, which contains 59 instances (12 instance classes). We do not report 14 relatively easy instances (three instance classes) where CCLS and IRoTS quickly find *optimal* solutions. Among these 14 instances, akmaxsat_ls proves optimum for all these 14 instances, and New WPM2 proves optimum for nine instances.

For the remaining 45 instances, Table 6 shows that SLS solvers (CCLS and IRoTS) find significantly better solutions than complete solvers (akmaxsat_ls and New WPM2). This suggests that it is better to use SLS heuristics to approximately solve some difficult MAX-SAT instances which can not be solved by complete solvers. When comparing the two SLS solvers (CCLS and IRoTS) on frb instances, CCLS significantly outperforms IRoTS. Totally, on the instances with $\#var \geq 325$, the solutions found by CCLS are always better than those found by IRoTS, akmaxsat_ls and New WPM2, and the gap becomes larger with the size of the instances increase. Specially, on the largest instance class frb59-26 with $\#var = 1{,}534$, the averaged best quality found by CCLS is 1,475.8, while these figures are 1,484.6 and 1,485.4 for IRoTS and akmaxsat_ls respectively, and the quality found by New WPM2 is 1,499.8. Overall, CCLS

TABLE 9
Results of CCLS+Dec, CCLS and IRoTS on the Spinglass Instance Class

Instance	CCLS+Dec sol. avg sol. time	CCLS sol. avg sol. time	IRoTS sol. avg sol. time
t6g3-8888	78,44,119	78,44,119	78,44,119
.spn.wcnf	7844119.0	7844119.0	7844119.0
	<1.0	4.9	33.3
t7g3-9999	1,19,54,769	1,20,29,063	1,19,54,769
.spn.wcnf	**11954769.0**	12112122.9	11956603.0
	17.2	420.2	286.6

To save space, we do not report three relatively simple instance where all these three SLS solvers quickly find optimal solutions.

shows superiority over other state-of-the-art solvers on frb instances.

Discussions on results of the third benchmark. Experimental results of CCLS and its competitors on random weighted MAX-2-SAT instances and random weighted MAX-3-SAT instances are demonstrated in Tables 7 and 8, respectively. We note that two complete solvers namely akmaxsat_ls and New WPM2 do not prove optimum for any of these instances. According to both Tables 7 and 8, it is apparent that CCLS and IRoTS finds better solutions than akmaxsat_ls and New WPM2. This is not surprising as it is well known that SLS methods are more efficient than complete ones in finding good solutions for random instances. Now we compare the two SLS solvers namely CCLS and IRoTS. It is clear that CCLS obviously outperforms IRoTS on all instance classes, including both random weighted MAX-2-SAT instances and random weighted MAX-3-SAT ones.

Improving CCLS on the spinglass instances. From Table 5, CCLS performs worse than IRoTS on the difficult spinglass instance t7g3-9999.spn.wcnf, which is the most difficult instance in the spinglass instance class. According to Tables 3 and 4, on this instance, CCLS performs better than CCLSdeg1 and also the frequency of intensification steps in CCLS (79.98 percent) is greater than that in CCLSdeg1 (61.89 percent). Thus, we naturally suppose that solving spinglass instances needs more intensification.

TABLE 10
Results of CCLS and Its Competitors on Random Weighted MAX-SAT Track of Incomplete Solvers in Max-SAT 2013

Instance Class	#inst	CCLS		iraNovelty++		optimax-it		SAT4Jms-ext-i		SAT4Jms-int-i	
		#opt	time	#opt	time	#opt	time	#opt	time	#opt	time
wmax2sat/100v	40	**40**	2.18	39	18.51	0	0.00	0	0.00	0	0.00
wmax2sat/120v	40	40	**2.99**	40	20.24	0	0.00	0	0.00	0	0.00
wmax2sat/140v	40	**40**	3.50	37	35.65	0	0.00	0	0.00	0	0.00
wmax3sat/hi	40	40	**2.05**	40	2.86	0	0.00	0	0.00	0	0.00
Total #opt	160	**160**		156		0		0		0	

TABLE 11
Results of CCLS and Its Competitors on Crafted Weighted MAX-SAT Track of Incomplete Solvers in Max-SAT 2013

Instance Class	#inst	CCLS		iraNovelty++		optimax-it		SAT4Jms-ext-i		SAT4Jms-int-i	
		#opt	time	#opt	time	#opt	time	#opt	time	#opt	time
frb	34	**34**	10.25	10	49.44	26	39.59	4	1.90	4	2.08
ramsey	15	**14**	25.87	8	44.16	0	0.00	3	11.32	3	12.36
wmaxcut/dimacs-mod	62	62	**1.22**	62	2.20	4	0.64	2	0.34	2	0.32
wmaxcut/spinglass	5	**5**	35.30	3	13.84	0	0.00	1	275.61	0	0.00
Total #opt	116	**115**		83		30		10		9	

TABLE 12
Results of CCLS and Its Competitors on Random Unweighted MAX-SAT Track of Incomplete Solvers in Max-SAT 2013

Instance Class	#inst	CCLS		iraNovelty++		optimax-it		SAT4Jms-ext-i		SAT4Jms-int-i	
		#opt	time	#opt	time	#opt	time	#opt	time	#opt	time
highgirth/3sat	50	**49**	27.32	42	92.90	0	0.00	0	0.00	0	0.00
highgirth/4sat	32	28	16.62	**31**	25.01	0	0.00	0	0.00	0	0.00
max2sat/120v	50	**50**	2.30	35	25.01	0	0.00	0	0.00	0	0.00
max2sat/140v	50	**50**	2.79	22	45.95	0	0.00	0	0.00	0	0.00
max3sat/70v	50	50	**1.72**	50	4.71	0	0.00	0	0.00	0	0.00
max3sat/80v	50	50	**1.86**	50	3.72	0	0.00	0	0.00	0	0.00
min2sat/160v	48	**48**	2.90	46	12.24	0	0.00	0	0.00	0	0.00
min2sat/200v	48	**48**	3.65	41	37.27	0	0.00	0	0.00	0	0.00
Total #opt	378	**373**		317		0		0		0	

TABLE 13
Results of CCLS and Its Competitors on Crafted Unweighted MAX-SAT Track of Incomplete Solvers in Max-SAT 2013

Instance Class	#inst	CCLS		iraNovelty++		optimax-it		SAT4Jms-ext-i		SAT4Jms-int-i	
		#opt	time	#opt	time	#opt	time	#opt	time	#opt	time
d/bipartite/maxcut-140-630-0.7	50	**50**	2.92	31	65.31	0	0.00	0	0.00	0	0.00
d/bipartite/maxcut-140-630-0.8	50	**50**	2.73	33	60.37	0	0.00	0	0.00	0	0.00
maxcut/dimacs-mod	62	**62**	4.17	61	3.24	8	5.05	2	0.49	2	0.45
maxcut/spinglass	5	**5**	2.37	4	34.84	1	0.78	0	0.00	0	0.00
Total #opt	167	**167**		129		9		2		2	

TABLE 14
Results of CCLS and Its Competitors on Industrial Unweighted MAX-SAT Track of Incomplete Solvers in Max-SAT 2013

Instance Class	#inst	optimax-it		CCLS		SAT4Jms-int-i		SAT4Jms-ext-i		iraNovelty++	
		#opt	time	#opt	time	#opt	time	#opt	time	#opt	time
ial/circuit-debugging-problems	3	**3**	31.01	0	0.00	1	24.04	1	24.81	0	0.00
sean-safarpour	52	**42**	34.59	9	4.73	5	58.46	4	61.35	0	0.00
Total #opt	55	**45**		9		6		5		0	

To support our arguments, we add a new heuristic, which focuses on picking decreasing variable ($score(x) > 0$), to enhance intensification in CCLS. We modify CCLS to obtain an alternative version called CCLS+Dec as follows: in each step, CCLS+Dec firstly checks whether decreasing variables exist. If no decreasing variable exists, CCLS+Dec directly performs the original procedures of CCLS. If decreasing variables exist, with the fixed probability $dp = 0.2$, CCLS+Dec selects the decreasing variable x with the greatest $score$ to flip; in the remaining case, CCLS+Dec executes the original procedure of CCLS. Table 9 reports the experimental results of CCLS+Dec, CCLS and IRoTS on the spinglass class (each solver performs 10 runs on each instance with a cutoff time of 1,000 seconds). Seen from Table 9, it is clear that CCLS+Dec performs best, which justifies our conjecture and also indicates that CCLS is able to combine with other heuristics in an effective way.

Summarization. The experiments show that CCLS consistently outperforms the state-of-the-art SLS solver IRoTS on weighted MAX-SAT instances including the industrial, crafted and random ones. Moreover, CCLS finds better solutions than two state-of-the-art complete solvers akmaxsat_ls and WPM2 on relatively difficult weighted MAX-SAT instances. For easy instances which are not reported in the tables, CCLS always finds better or the same-quality solutions than IRoTS and the two complete solvers. The detailed experimental results on all the three benchmarks can be downloaded online [52].

5.5 Evaluation Results in the Eighth Max-SAT Evaluation (Max-SAT 2013)

To demonstrate the state-of-the-art performance of the CCLS solver and to compare CCLS against other most recent efficient solvers, we submit CCLS to the weighted and unweighted MAX-SAT tracks of incomplete solvers in the Eighth Max-SAT Evaluation (Max-SAT 2013) [27], as the objective of the evaluation is assessing the state of the art in the field of MAX-SAT solvers.[7]

In this section, we report and analyze the experimental results of the random weighted, crafted weighted, random unweighted, crafted unweighted and industrial unweighted tracks of incomplete solvers in Max-SAT 2013, and the experimental results (Tables 10, 11, 12, 13, and 14) in this section are all taken from the homepage of Max-SAT 2013[8] [27]. For the version of the CCLS solver submitted to Max-SAT 2013, the setting of parameter p for weighted instances is identical to that reported in Table 1, and the setting of parameter p for unweighted instances is set to 0.1. The binary of CCLS submitted to Max-SAT 2013 can be downloaded from the homepage of Max-SAT 2013[9] [27]. The iraNovelty++ [29], optimax-it, SAT4Jms-ext-i [53] and SAT4Jms-int-i [53] solvers are the competitors in the tracks which we report in Max-SAT 2013. The iraNovelty++ solver is a recent developed SLS solver, and the optimax-it, SAT4Jms-ext-i and SAT4Jms-int-i solvers are complete solvers but use the outputting method which is adopted by incomplete solvers.

For each solver on each instance class, we report #opt as the number of instances whose best quality of solution given among the competing solvers is found by this solver in this instance class, and the averaged time among all runs of this solver ('time') on this instance class. For each solver on each benchmark, we also report the total #opt, which means the sum of #opt of each instance class in this benchmark.

The rules at MAX-SAT competitions establish that the winner is the solver which finds the greatest #opt, breaking ties by selecting the solver with the least averaged time.

7. http://maxsat.ia.udl.cat:81/13/introduction/index.html
8. http://maxsat.ia.udl.cat:81/13/results-incomplete/index.html
9. http://maxsat.ia.udl.cat:81/13/solvers/ccls_

Discussions on results of the random weighted track. Table 10 reports the experimental results of random weighted track of incomplete solvers, which show that CCLS is the best solver for the benchmark. Apparently, optimax-it, SAT4Jms-ext-i, and SAT4Jms-int-i performs essentially worse than CCLS and iraNovelty++. For example, the best solution for each instance is always given by CCLS or iraNovelty++. Moreover, CCLS performs better than iraNovelty++, in terms of both the total #opt and the averaged time of each instance class.

Discussions on results of the crafted weighted track. The results of crafted weighted track of incomplete solvers are illustrated in Table 11, and show that CCLS stands out as the best solver on this benchmark. Among 116 total instances in this benchmark, CCLS finds optimal solutions for 115 of them, while this figure is only 83, 30, 10 and 9 for iraNovelty++, optimax-it, SAT4Jms-ext-i and SAT4Jms-int-i respectively. Particularly, on the relatively difficult instance classes (frb, ramsey and wmaxcut/spinglass), CCLS performs much better than its competitors.

Discussions on results of the random unweighted track. The results of random unweighted track of incomplete solvers are presented in Table 12. It is clear that CCLS and iraNovelty++ are the best two solvers and significantly outperform the other three solvers on the whole benchmark. So we focus on the comparison between CCLS and iraNovelty++. Seen from Table 12, CCLS outperforms iraNovelty++ on all the instance classes but one namely highgirth/4sat, for which CCLS still competes well with iraNovelty++. On the overall performance of this track, the total #opt of CCLS is 373, while that of iraNovelty++ is 317 and those of optimax-it, SAT4Jms-ext-i and SAT4Jms-int-i are all 0.

Discussions on results of the crafted unweighted track. Table 13 illustrates the results of crafted unweighted track of incomplete solvers, and demonstrates that CCLS significantly outperforms its competitors on these instances. On the total performance, the total #opt of CCLS is 167, which is equal to the number of instances in this benchmark, while those of the iraNovelty++, optimax-it, SAT4Jms-ext-i and SAT4Jms-int-i solvers are 129, 9, 2 and 2, respectively. Among the four competitors, iraNovelty++ has the closest performance with CCLS. However, CCLS also outperforms iraNovelty++ on each instance class in this benchmark.

Discussions on results of the industrial unweighted track. Table 14 reports the results of industrial unweighted track of incomplete solvers. From Table 14, although the complete solver optimax-it performs best, CCLS shows better performance than the SLS solver iraNovelty++ as well as other complete solvers SAT4Jms-int-i and SAT4Jms-ext-i. As CCLS is designed for weighted MAX-SAT instances, CCLS does not strike a good balance between intensification and diversification for solving industrial unweighted Max-SAT instances. To support our arguments, we use the CCLS (the version submitted to Max-SAT 2013) to do empirical analysis to show the frequencies of intensification and diversification steps on these industrial unweighted MAX-SAT instances. The results present that the frequencies of intensification and diversification steps are 90 and 10 percent, which deviates far from the 80/20 rule. This should be the reason why CCLS is not effective on industrial unweighted MAX-SAT instances. Nevertheless, CCLS achieves good performance on random unweighted and crafted unweighted MAX-SAT instances in Max-SAT 2013 (Tables 12 and 13), indicating the robustness of CCLS.

Summarization. The above results in Max-SAT 2013 indicates the efficiency and the effectiveness of CCLS. Also, the results on unweighted MAX-SAT instances indicate the robustness of CCLS.

6 Conclusions and Future Work

We proposed a heuristic named configuration checking with make based on CC. Previous CC heuristics used in SLS for SAT filter candidate variables by *score* (i.e., CCD variables), and are thus not suitable for solving MAX-SAT, as the number of CCD variables in the search process of SLS is too few for most MAX-SAT instances. Unlike previous CC heuristics, the CCM heuristic filters candidate variables by the *make* property namely CCMP variables. In theory, we prove that the *CCMPvars* set is a superset of the *CCDvars* set; in practice, our experimental analysis demonstrates that there are significantly more CCMP variables than CCD variables in the search process of CCLS for many difficult instances.

We utilized the CCM heuristic to design an SLS algorithm called CCLS. The experimental results illustrate that CCLS significantly outperforms the state-of-the-art SLS solver IRoTS and finds better solutions than two state-of-the-art complete solvers namely akmaxsat_ls and New WPM2 on a broad range of weighted MAX-SAT instances, including industrial, crafted and random ones. Furthermore, CCLS is evaluated in the weighted and unweighted MAX-SAT tracks of incomplete solvers in Max-SAT 2013 and wins four tracks in the evaluation, indicating CCLS establishes the latest state of the art in SLS algorithms for MAX-SAT.

CCLS is simple yet surprisingly efficient. In this sense, CCLS can serve as a good algorithmic framework, and more efficient algorithms can be proposed and implemented on top of it.

For future work, we would like to adjust the CCM heuristic by replacing the neighboring variable based *configuration* which is currently adopted in this work with the clause state based *configuration* [54], [55], and test the combination of these two configuration checking strategies [56] to further improve the algorithm, especially for industrial unweighted MAX-SAT instances. Also we would like to test CCLS on partial MAX-SAT instances.

Acknowledgments

This work was supported by China National 973 Projects 2010CB328103 and 2014CB340301, ARC Grant FT0991785, and NSFC 61370072. Shaowei Cai is the corresponding author for this paper. The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] K. Smyth, H. H. Hoos, and T. Stützle, "Iterated robust tabu search for MAX-SAT," in *Proc. 16th Can. Conf. Adv. Artif. Intell.*, 2003, pp. 129–144.

[2] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: Use and theory for increasing predictability and avoiding coupling," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 21, no. 7, pp. 777–790, Jul. 2002.

[3] X. A. Dimitropoulos, D. V. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. C. Claffy, and G. F. Riley,"AS relationships: Inference and validation," *Comput. Commun. Rev.*, vol. 37, no. 1, pp. 29–40, 2007.

[4] H. Xu, R. A. Rutenbar, and K. A. Sakallah, "sub-SAT: A formulation for relaxed Boolean satisfiability with applications in routing," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 22, no. 6, pp. 814–820, Jun. 2003.

[5] D. M. Strickland, E. R. Barnes, and J. S. Sokol, "Optimal protein structure alignment using maximum cliques," *Oper. Res.*, vol. 53, no. 3, pp. 389–402, 2005.

[6] M. Vasquez and J.-K. Hao, "A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite," *Comp. Opt. Appl.*, vol. 20, no. 2, pp. 137–157, 2001.

[7] J. D. Park, "Using weighted MAX-SAT engines to solve MPE," in *Proc. 18th Nat. Conf. Artif. Intell.*, 2002, pp. 682–687.

[8] Q. Yang, K. Wu, and Y. Jiang, "Learning action models from plan examples using weighted MAX-SAT," *Artif. Intell.*, vol. 171, nos. 2/3, pp. 107–143, 2007.

[9] Y. Chen, S. Safarpour, J. Marques-Silva, and A. G. Veneris, "Automated design debugging with maximum satisfiability," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 29, no. 11, pp. 1804–1817, Nov. 2010.

[10] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah, "Improved design debugging using maximum satisfiability," in *Proc. Formal Methods Comput. Aided Des.*, 2007, pp. 13–19.

[11] C. M. Li, F. Manyà, and J. Planes, "New inference rules for Max-SAT," *J. Artif. Intell. Res.*, vol. 30, pp. 321–359, 2007.

[12] H. Lin, K. Su, and C. M. Li, "Within-problem learning for efficient lower bound computation in Max-SAT solving," in *Proc. 23rd AAAI Conf. Artif. Intell.*, 2008, pp. 351–356.

[13] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[14] M. Davis, G. Logemann, and D. W. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[15] B. Selman, H. J. Levesque, and D. G. Mitchell, "A new method for solving hard satisfiability problems," in *Proc. 10th Nat. Conf. Artif. Intell.*, 1992, pp. 440–446.

[16] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *Proc. 12th Nat. Conf. Artif. Intell.*, 1994, pp. 337–343.

[17] T. V. Luong, N. Melab, and E.-G. Talbi, "GPU computing for parallel local search metaheuristic algorithms," *IEEE Trans. Comput.*, vol. 62, no. 1, pp. 173–185, Jan. 2013.

[18] L. Kroc, A. Sabharwal, C. P. Gomes, and B. Selman, "Integrating systematic and local search paradigms: A new strategy for MaxSAT," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, 2009, pp. 544–551.

[19] W. Michiels, E. H. L. Aarts, and J. H. M. Korst, *Theoretical Aspects of Local Search*. New York, NY, USA: Springer-Verlag, 2007.

[20] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[21] R. Battiti and M. Protasi, "Reactive search, a history-sensitive heuristic for MAX-SAT," *ACM J. Exp. Algorithmics*, vol. 2, p. 2, 1997.

[22] S. Cai and K. Su, "Configuration checking with aspiration in local search for SAT," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 434–440.

[23] S. Cai and K. Su, "Local search for Boolean satisfiability with configuration checking and subscore," *Artif. Intell.*, vol. 204, pp. 75–98, 2013.

[24] S. Cai and K. Su, "Local search with configuration checking for SAT," in *Proc. IEEE 23rd Int. Conf. Tools Artif. Intell.*, 2011, pp. 59–66.

[25] A. Kügel, "Improved exact solver for the weighted Max-SAT problem," in *Proc. Pragmatics SAT*, 2010, pp. 15–27.

[26] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving WPM2 for (weighted) partial MaxSAT," in *Proc. 19th Int. Conf. Constraint Program.*, 2013, pp. 117–132.

[27] The homepage of eighth Max-SAT evaluation (Max-SAT 2013). [Online]. Available: http://maxsat.ia.udl.cat:81/13/introduction/index.html

[28] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artif. Intell.*, vol. 175, nos. 9/10, pp. 1672–1696, 2011.

[29] A. Abramé and D. Habet, "Inference rules in local search for Max-SAT," in *Proc. IEEE 24th Int. Conf. Tools Artif. Intell.*, 2012, pp. 207–214.

[30] C. M. Li, C. Huang, and R. Xu, "Balance between intensification and diversification: Two sides of the same coin," in *Proc. SAT Competition 2013: Solver Benchmark Descriptions*, 2013, pp. 10–11.

[31] W. A. Britten, "A use statistic for collection management: The 80/20 rule revisited," *Library Acquisitions: Practice Theory*, vol. 14, no. 2, pp. 183–189, 1990.

[32] Q. L. Burrell, "The 80/20 rule: Library lore or statistical law?" *J. Doc.*, vol. 41, no. 1, pp. 24–39, 1985.

[33] M. Iqbal and M. Rizwan, "Application of 80/20 rule in software engineering waterfall model," in *Proc. Int. Conf. Inf. Commun. Technol.*, 2009, pp. 223–228.

[34] B. Liu and B. M. Baas, "Parallel AES encryption engines for many-core processor arrays," *IEEE Trans. Comput.*, vol. 62, no. 3, pp. 536–547, Mar. 2013.

[35] S. Trimberger, "Security in SRAM FPGAs," *IEEE Des. Test Comput.*, vol. 24, no. 6, p. 581, Nov. 2007.

[36] M. M. Kermani and A. Reyhani-Masoleh, "Concurrent structure-independent fault detection schemes for the advanced encryption standard," *IEEE Trans. Comput.*, vol. 59, no. 5, pp. 608–622, May 2010.

[37] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, "A simple model to generate hard satisfiable instances," in *Proc. 19th Int. Joint Conf. Artif. Intell.*, 2005, pp. 337–342.

[38] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, "Random constraint satisfaction: Easy generation of hard (satisfiable) instances," *Artif. Intell.*, vol. 171, nos. 8/9, pp. 514–534, 2007.

[39] W. Zhang, A. Rangan, and M. Looks, "Backbone guided local search for maximum satisfiability," in *Proc. 18th Int. Joint Conf. Artif. Intell.*, 2003, pp. 1179–1186.

[40] P. Mills and E. P. K. Tsang, "Guided local search for solving SAT and weighted MAX-SAT problems," *J. Autom. Reasoning*, vol. 24, nos. 1/2, pp. 205–223, 2000.

[41] M. Yagiura and T. Ibaraki, "Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation," *J. Heuristics*, vol. 7, no. 5, pp. 423–442, 2001.

[42] Y. Jiang, H. Kautz, and B. Selman, "Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT," presented at the 1st Int. Joint Workshop Artif. Intell. Oper. Res., Timberline, OR, USA, 1995.

[43] B. W. Wah and Y. Shang, "Discrete lagrangian-based search for solving MAX-SAT problems," in *Proc. Int. Joint Conf. Artif. Intell.*, 1997, pp. 378–383.

[44] D. A. D. Tompkins and H. H. Hoos, "UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT," in *Proc. 7th Int. Conf. Theory Appl. Satisfiability Testing*, 2004, pp. 37–46.

[45] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes, "Exploiting cycle structures in Max-SAT," in *Proc. 12th Int. Conf. Theory Appl. Satisfiability Testing*, 2009, pp. 467–480.

[46] C. Ansótegui, M. L. Bonet, and J. Levy, "SAT-based MaxSAT algorithms," *Artif. Intell.*, vol. 196, pp. 77–105, 2013.

[47] C. Ansótegui, M. L. Bonet, and J. Levy, "A new algorithm for weighted partial MaxSAT," in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 3–8.

[48] F. Heras, A. Morgado, and J. Marques-Silva, "Core-guided binary search algorithms for maximum satisfiability," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, pp. 36–41.

[49] C. Ansótegui and J. Gabàs, "Solving (weighted) partial MaxSAT with ILP," in *Proc. Int. Conf. Integr. Artif. Intell. Oper. Res. Techn. Constraint Program.*, 2013, pp. 403–409.

[50] R. Martins, V. M. Manquinho, and I. Lynce, "Exploiting cardinality encodings in parallel maximum satisfiability," in *Proc. IEEE 23rd Int. Conf. Tools Artif. Intell.*, 2011, pp. 313–320.

[51] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver," *J. Satisfiability, Boolean Model. Comput.*, vol. 8, nos. 1/2, pp. 95–100, 2012.

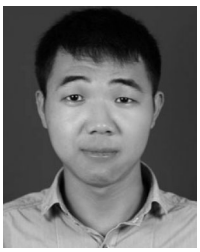
[52] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su. Detailed experimental results of CCLS on weighted MAX-SAT instances. (2014). [Online]. Available: http://shaoweicai.net/Paper/CCLS_experimental_results.pdf

[53] D. L. Berre and A. Parrain, "The Sat4j library, release 2.2," *J. Satisfiability, Boolean Model. Comput.*, vol. 7, nos. 2/3, pp. 59–64, 2010.

[54] C. Luo, K. Su, and S. Cai, "Improving local search for random 3-SAT using quantitative configuration checking," in *Proc. 20th Eur. Conf. Artif. Intell.*, 2012, pp. 570–575.
[55] C. Luo, S. Cai, W. Wu, and K. Su, "Focused random walk with configuration checking and break minimum for satisfiability," in *Proc. 19th Int. Conf. Constraint Program.*, 2013, pp. 481–496.
[56] C. Luo, S. Cai, W. Wu, and K. Su, "Double configuration checking in stochastic local search for satisfiability," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 2703–2709.

Chuan Luo received the BEng degree in computer science from the South China University of Technology, Guangzhou, China, in 2011. He is currently working toward the PhD degree in computer science at Peking University, Beijing, China. His research interests include heuristic search and optimization.

Shaowei Cai received the BEng degree in computer science from the South China University of Technology, Guangzhou, China, in 2008, and the PhD degree in computer science from Peking University, Beijing, China, in 2012. His research interests include optimization, heuristics, and randomized algorithms.



Wei Wu received the BEng degree in computer science from Central South University, Changsha, China, in 2011. He is currently working toward the MSc degree in computer science at Peking University, Beijing, China. His research interests include local search and data mining.

Zhong Jie received the BSc and MSc degrees both in computer science from Peking University, Beijing, China, in 2010 and 2013, respectively. He is currently a research and development engineer at NetEase, Inc., Hangzhou, China. His research interests include local search and experimental algorithms for hard problems.

Kaile Su received the PhD degree in mathematics from Nanjing University, Nanjing, China, in 1995. He is currently an associate professor with the Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia. His research interests include modal logic, model checking multi-agent systems, and experimental algorithms for hard problems.