CCEHC: An Efficient Local Search Algorithm for Weighted Partial Maximum Satisfiability (Extended Abstract)*

Chuan Luo[1], **Shaowei Cai**[2], **Kaile Su**[3], **Wenxuan Huang**[4]
[1]Institute of Computing Technology, Chinese Academy of Sciences, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[3]College of Information Science and Technology, Jinan University, China
[4]Department of Material Science and Engineering, Massachusetts Institute of Technology, USA
chuanluosaber@gmail.com; shaoweicai.cs@gmail.com; k.su@griffith.edu.au; key01027@mit.edu

Abstract

Weighted partial maximum satisfiability (WPMS) is a significant generalization of maximum satisfiability (MAX-SAT), with many important applications. Recently, breakthroughs have been made on stochastic local search (SLS) for weighted MAX-SAT and (unweighted) partial MAX-SAT (PMS). However, the performance of SLS for WPMS lags far behind. In this work, we present a new SLS algorithm named *CCEHC* for WPMS. *CCEHC* is mainly based on a heuristic emphasizing hard clauses, which has three components: a variable selection mechanism focusing on configuration checking based only on hard clauses, a weighting scheme for hard clauses, and a biased random walk component. Experiments show that *CCEHC* significantly outperforms its state-of-the-art SLS competitors. Experiments comparing *CCEHC* with a state-of-the-art complete solver indicate the effectiveness of *CCEHC* on a number of application WPMS instances.

1 Introduction

Given a formula in conjunctive normal form (CNF), the maximum satisfiability (MAX-SAT) problem is to seek out an assignment that maximizes the number of satisfied clauses in the formula. The weighted partial maximum satisfiability (WPMS) problem is a significant generalization of MAX-SAT, with many important applications. The WPMS problem, where clauses are divided into hard ones and soft ones, and each soft clause is associated with a positive integer as its weight, is to seek out an assignment that satisfies all hard clauses and maximizes the total weight of satisfied soft clauses. MAX-SAT and WPMS are typically NP-hard and it is well known that optimal solutions are hard to approximate [Smyth *et al.*, 2003].

There are two popular categories of practical MAX-SAT algorithms: complete algorithms [Lin *et al.*, 2008; Li *et al.*, 2009; Ansótegui *et al.*, 2013a; Ansótegui and Gabàs, 2013; Ansótegui *et al.*, 2013b; Narodytska and Bacchus, 2014] and stochastic local search (SLS) algorithms evolving out of *GSAT* [Selman *et al.*, 1992] and *WalkSAT* [Selman *et al.*, 1994]. Recently, breakthroughs have been achieved on SLS algorithms for solving weighted MAX-SAT and (unweighted) partial MAX-SAT (PMS), resulting in state-of-the-art SLS algorithms namely *CCLS* [Luo *et al.*, 2015b] and *Dist* [Cai *et al.*, 2014] as well as *Dist*'s improvement *DistUP* [Cai *et al.*, 2016]. However, *CCLS*, *Dist* and *DistUP* are not dedicated to solving WPMS specifically, and their performance for WPMS could be further improved. This motivates us to design a more efficient SLS algorithm for WPMS.

In this work, we present a new SLS algorithm named *CCEHC* (*Configuration Checking with Emphasis on Hard Clauses*) for WPMS. *CCEHC* is mainly based on a heuristic emphasizing hard clauses, called EHC. Our main contributions in this paper are summarized as follows.

Firstly, we identify an efficient algorithm framework for solving WPMS. Secondly, we propose a new variable selection mechanism focusing on a new forbidding mechanism of configuration checking. This new configuration checking mechanism emphasizes hard clauses. Finally, by adopting a weighting scheme for hard clauses and adjusting the existing strategy of biased random walk, we integrate the two with our new forbidding mechanism of configuration checking in a subtle way and obtain our new EHC heuristic.

We compare *CCEHC* against *CCLS*, *Dist* and *DistUP* on WPMS benchmarks from the MAX-SAT Evaluation 2014 and four real-world application benchmarks. The experimental results show that *CCEHC* achieves better performance, and establishes a new state-of-the-art performance on SLS algorithms for WPMS. Also, the experiments comparing *CCEHC* with a state-of-the-art complete solver present the effectiveness of *CCEHC* on a number of application WPMS instances. Further, our experimental results indicate that the combination of *CCEHC* and unit propagation initialization [Cai *et al.*, 2016] gives a performance improvement over *CCEHC* on a large number of WPMS instances.

In the remainder of this paper, Section 2 gives the necessary preliminaries of this paper. Section 3 proposes the EHC heuristic and introduces those components in the EHC heuristic. Section 4 presents the CCEHC algorithm. Section 5 reports experiment results on *CCEHC*. Section 6 concludes this paper. For more information about *CCEHC*, please refer to the full journal article version of this paper [Luo *et al.*, 2017].

*This paper is an extended abstract of an article in Artificial Intelligence [Luo *et al.*, 2017].

2 Preliminaries

A formula F in conjunctive normal form (CNF) is a conjunction of clauses, i.e., $F = \bigwedge_i \bigvee_j l_{ij}$, where each l_{ij} is a literal, which is either a Boolean variable or its negation. Given a CNF formula F, $V(F)$ is used to denote the set of all variables in F. Two different variables are neighbors when they appear in at least one clause, and $N(x)$ is used to denote the set of all neighbors of variable x.

A weighted partial CNF formula is such a CNF formula, where all clauses are divided into hard ones and soft ones, and each soft clause c is associated with a positive integer $w(c)$ as its weight. Given a weighted partial CNF formula F, the **weighted partial maximum satisfiability (WPMS)** problem is to find such an assignment which satisfies all hard clauses in F and maximizes the total weight of all satisfied soft clauses in F. A complete assignment is feasible if it satisfies all hard clauses in the formula, and the cost of a feasible assignment α, denoted as $cost(\alpha)$, is the total weight of all unsatisfied soft clauses under α. The optimal feasible assignment is the feasible assignment with the minimum cost.

In SLS algorithms for WPMS, for a variable x, the *hard make score* of x, denoted by $hmake(x)$, is the number (or total weight if using clause weighting scheme) of unsatisfied hard clauses that would become satisfied if x is flipped; the *hard score* of x, denoted by $hscore(x)$, is the increment in the number (or total weight if using clause weighting scheme) of satisfied hard clauses if x is flipped; the *soft make score* of x, denoted by $smake(x)$, is the total weight of unsatisfied soft clauses that would become satisfied if x is flipped; the *soft score* of x, denoted by $sscore(x)$, is the increment in the total weight of satisfied soft clauses if x is flipped. For a variable x, the general *make score* of x, denoted by $make(x)$, can be calculated as $make(x) = A \times hmake(x) + smake(x)$; the general *score* of x, denoted by $score(x)$, can be calculated as $score(x) = A \times hscore(x) + sscore(x)$, where A is a positive integer whose value equals the total weight of all soft clauses plus 1.

3 Heuristic with Emphasis on Hard Clauses

We adopt the framework of *CCLS* in our new algorithm. To improve the performance, we design a heuristic called EHC (*Emphasis on Hard Clauses*), which introduces more differences in treating hard clauses and soft clauses and emphasizes hard clauses. The EHC heuristic is composed of three components: a variable selection mechanism focusing on configuration checking only on hard clauses, a weighting scheme for hard clauses, and a strategy of biased random walk.

3.1 Hard Clauses' States Based Configuration Checking

Inspired by the success of the clause states based configuration checking (CSCC) strategy [Luo *et al.*, 2015a] in solving Boolean satisfiability (SAT), it is natural to adapt this CSCC strategy to solving WPMS. In WPMS, hard clauses are more important, so we propose a new configuration checking strategy based only on the states of hard clauses, named HCSCC (*hard clauses' states based configuration checking*).

Definition 1 *Given a weighted partial CNF formula F and a complete assignment α to $V(F)$, the configuration of a variable $x \in V(F)$ for HCSCC is a vector $configuration(x)$ consisting of the states of all hard clauses where x appears under α.*

For a variable x, a change on any element of $configuration(x)$ is considered as a change on the whole $configuration(x)$ vector. The HCSCC strategy is designed to prevent flipping the variable x whose $configuration(x)$ has not been changed since x's last flip.

To implement HCSCC more efficiently, we employ a Boolean array $hardConf$ whose size equals the number of the variables in the formula. The array $hardConf$ is maintained during the search. Initially, for each variable x, $hardConf(x)$ is set to 1. Whenever a variable x is flipped, $hardConf(x)$ is set to 0. Then each hard clause c, where x appears, is checked whether c's state is changed (from satisfied to unsatisfied or vice versa). If c's state is indeed changed, for each variable y ($y \neq x$) in c, $hardConf(y)$ is set to 1.

Thus, in the implementation of our HCSCC strategy, a variable x's configuration has been changed since x's last flip if $hardConf(x) = 1$. We define the notion of HCSCCD (*hard clauses' states based configuration changed decreasing*) variables as follows: a variable $x \in V(F)$ is HCSCCD if $hardConf(x) = 1$ and $hscore(x) > 0$. The notation *HCSCCDvars* is used to denote the set of all HCSCCD variables during the search.

An important heuristic in *CCLS* is CCM (*configuration checking with make*), which prefers to select the CCMP (*configuration changed and make positive*) variable with the highest score [Luo *et al.*, 2015b]. A variable x is CCMP if $make(x) > 0$ and, since x's last flip, at least one of x's neighboring variables has been flipped [Luo *et al.*, 2015b]. The notation *CCMPvars* denotes the set of all CCMP variables during the search. The relationship between the HCSCCD variable and the CCMP variable is presented as follows.

Lemma 1 *For a given variable x, if x is a HCSCCD variable, then x is a CCMP variable.*

3.2 Weighting Scheme for Hard Clauses

Clause weighting schemes have been used prominently and successfully in SLS algorithms for solving SAT. This motivates us to further extend the *CCLS* algorithm framework with an effective clause weighting scheme.

To put higher priority on hard clauses than soft clauses in clause weighting, it is natural to adopt a clause weighting scheme that only works for hard clauses. We utilize the one in *Dist* [Cai *et al.*, 2014], which only adds weights to hard clauses. The weighting scheme is similar to PAWS [Thornton *et al.*, 2004] and works as follows.

- In the beginning of the SLS algorithm, for each hard clause c, the weight of c (i.e., $w(c)$) is set to 1.
- During the search, when the hard clause weighting scheme is activated, with probability sp (sp is a real number and $0 \leq sp \leq 1$), for each satisfied hard clause c with $w(c) > 1$, $w(c)$ is decreased by 1; otherwise (with probability $1 - sp$), for each unsatisfied hard clause c, $w(c)$ is increased by 1.

Algorithm 1: The *CCEHC* Algorithm

```
   Input: Weighted partial CNF formula F, maxSteps
1  generate a random assignment α, α* ← α;
2  for step ←1 to maxSteps do
3      if time limit is exceeded or all hard and soft clauses are
         satisfied then break;
4      if with fixed probability p then
5          if there exists any unsatisfied hard clause then c ← a
             random unsatisfied hard clause;
6          else c ← a random unsatisfied soft clause;
7          v ← a variable x with greatest sscore(x) in c,
             breaking ties randomly;
8      else
9          if HCSCCDvars is not empty then
10             v ← a variable randomly selected from
                 HCSCCDvars;
11         else
12             Update hard clause weights;
13             if CCMPvars is not empty then
14                 v ← x with the greatest score(x) in
                     CCMPvars, breaking ties randomly;
15             else
16                 c ← a random unsatisfied clause;
17                 v ← a random variable in c;
18     α ← α with v flipped;
19     if cost(α) < cost(α*) then α* ← α;
20 if α* satisfies all hard clauses then return α*;
21 else return "No feasible assignment is found";
```

3.3 The Biased Random Walk Component

An important component of the *CCLS* algorithm is the random walk for the diversification mode. However, standard random walk may not be suitable for WPMS. Since hard clauses are forced to be satisfied in feasible solutions of the WPMS problem, it is reasonable for us to employ a biased random walk component that prefers selecting a hard clause with a higher priority to choosing a soft clause. The biased random walk strategy is suggested by the literature [Jiang *et al.*, 1995] and described as follows.

When biased random walk is called, if there exist unsatisfied hard clauses, the algorithm chooses an unsatisfied hard clause randomly; otherwise, an unsatisfied soft clause is selected randomly. Then the algorithm picks a variable in the chosen clause. In this work, this is accomplished by selecting the variable x with the greatest $sscore(x)$ in the chosen clause, inspired by the literature [Cai *et al.*, 2014].

4 The *CCEHC* Algorithm

Based on the above ideas, this section presents the *CCEHC* (Algorithm 1) in detail.

Initially, *CCEHC* generates an assignment α randomly. Then it performs a loop until one of the termination criteria is met. During the search, whenever a better solution is found, the best solution α^* is updated accordingly.

In each search step, *CCEHC* selects a variable to be flipped. With probability p, *CCEHC* calls the biased random walk component (lines 5–7): if there exists any unsatisfied hard clause, an unsatisfied hard clause is selected randomly; otherwise, an unsatisfied soft clause is picked randomly; then *CCEHC* selects a variable x with greatest $sscore(x)$ in the chosen hard or soft clause as the variable to be flipped. With probability $1-p$, *CCEHC* first checks whether the *HCSCCDvars* set is empty or not; if the *HCSCCDvars* set is not empty, *CCEHC* picks a variable randomly selected from *HCSCCDvars* (with bias towards the ones with the best $hscore$), inspired by the literature [Cai *et al.*, 2014] (line 10 in Algorithm 1). Otherwise, *CCEHC* activates the hard clause weighting scheme (line 12), and then selects a variable according to the CCM heuristic [Luo *et al.*, 2015b] (lines 13–17). After the variable to be flipped is selected, the *CCEHC* algorithm flips the selected variable and then starts the next search step (line 18 in Algorithm 1).

Finally, when the search terminates, if α^* is feasible, *CCEHC* reports α^* as the solution; otherwise, *CCEHC* outputs "No feasible assignment is found".

Table 1: The parameter settings reported by *SMAC* for *CCEHC*, *Dist* and *CCLS*.

Instance Type	*CCEHC*		*Dist*			*CCLS*
	p	sp	wp	sp	t	p
Random	0.2	0.0001	0.1	0.001	15	0.535
Crafted	0.177	0.003	0.1	0.001	15	0.204
Industrial/Application	0.279	0.085	0.038	0.002	6	0.2

5 Experimental Evaluations

We evaluate *CCEHC* on random, crafted and industrial WPMS benchmarks from MAX-SAT Evaluation 2014 as well as four real-world application benchmarks, including computational protein design[1] (CPD) [Allouche *et al.*, 2012; 2014], advanced encryption standard[2] (AES) [Gwynne and Kullmann, 2011], the pedigree problem[3] [Sánchez *et al.*, 2008] and cluster expansion[4] (CE) [Huang *et al.*, 2016].

CCEHC is implemented in C++ and compiled by g++ with '-O2'. All experiments are performed on a cluster of workstations with Intel Xeon E7-8830 2.13 GHz CPU, 24MB L3 cache and 1.0TB RAM under the operating system CentOS.

Each solver performs one run on each instance. For each solver on each benchmark, we report the number of instances where the solver finds the best solution among all competing solvers in the related experiment, denoted by '#win.', and the averaged time of doing so on such winning instances, denoted by 'time' (the unit is CPU second). The cutoff time of each run is set to 300 CPU seconds.

5.1 Comparing *CCEHC* with SLS Competitors

Our *CCEHC* algorithm is compared against three state-of-the-art SLS algorithms, namely *CCLS* [Luo *et al.*, 2015b],

[1] http://genoweb.toulouse.inra.fr/~degivry/evalgm/CFN/ProteinDesign/

[2] In the directory 'aes/', http://maxsat.udl.cat/14/benchmarks/pms_industrial.tgz

[3] http://genoweb.toulouse.inra.fr/~degivry/evalgm/CFN/Pedigree/

[4] http://lcs.ios.ac.cn/~caisw/Resource/cluster-expansion.zip

Table 2: Comparison of *CCEHC* with SLS solvers including *DistUP*, *Dist* and *CCLS*.

Benchmark	#inst.	*CCEHC*		*DistUP*		*Dist*		*CCLS*		*VBS*	
		#win.	time	#win.	time	#win.	time	#win.	time	#win.	time
Random	280	**279**	0.10	**279**	**0.04**	**279**	0.06	271	7.93	279	<0.01
Crafted	310	**282**	45.03	194	6.83	192	5.37	146	7.75	302	42.26
Industrial	410	**226**	124.43	154	96.53	90	95.50	35	39.59	389	125.15
CPD	20	**11**	94.85	2	174.88	2	31.73	3	162.02	15	93.70
AES	7	**5**	11.08	1	0.00	2	150.74	**5**	**2.31**	7	47.97
Pedigree	20	**11**	94.85	2	174.88	2	31.73	3	162.02	15	93.70
CE	6	**6**	0.05	**6**	**<0.01**	**6**	0.01	3	<0.01	6	<0.01

Table 3: Comparison of *CCEHC* and the complete solver *WPM-2014*. As a reference, we report the number of instances where *Eva* proves optimality within the cutoff time.

Benchmark	#inst.	#prov. by *Eva*	*CCEHC*		*WPM-2014*		*VBS*	
			#win.	time	#win.	time	#win.	time
Random	280	1	**279**	0.10	0	0	279	0.10
Crafted	310	141	**247**	29.76	170	16.82	310	25.16
Industrial	410	355	59	37.74	**396**	14.08	410	16.02
CPD	20	0	**14**	130.34	6	42.01	20	103.84
AES	7	1	**6**	17.57	1	96.03	7	28.78
Pedigree	20	17	14	65.48	**15**	28.09	20	64.65
CE	6	0	**6**	0.05	0	0	6	0.05

Table 4: Experimental results of *CCEHC* and *CCEHC+UP* on all testing WPMS benchmarks.

Benchmark	#inst.	*CCEHC*		*CCEHC+UP*		*VBS*	
		#win.	time	#win.	time	#win.	time
Random	280	**279**	0.10	**279**	**0.06**	279	0.06
Crafted	310	249	32.10	**261**	37.55	301	46.42
Industrial	410	153	119.38	**269**	78.41	361	98.95
CPD	20	10	132.28	**11**	137.79	15	150.99
AES	7	5	11.08	**6**	24.23	6	22.62
Pedigree	20	**16**	81.27	15	43.19	19	65.75
CE	6	**6**	0.05	**6**	**<0.01**	6	<0.01

Dist [Cai *et al.*, 2014] and *DistUP* [Cai *et al.*, 2016]. We also report the results of the Virtual Best Solver (*VBS*), i.e., the perfect selector – on each instance, the solution of *VBS* is the best one of the solutions reported by all competing solvers included in this experiment on solving this instance. We tune parameters of these solves using *SMAC* [Hutter *et al.*, 2011], and the resulting parameter settings are presented in Table 1.

The results comparing *CCEHC* with other SLS solvers (Table 2) shows that *CCEHC* outperforms its SLS competitors.

5.2 Comparing *CCEHC* with Complete Solver

We compare *CCEHC* with a state-of-the-art complete solver *WPM-2014* [Ansótegui *et al.*, 2013a]. We adopt the version of *WPM-2014* that uses the outputting format of incomplete solvers (i.e., printing the better-quality solution immediately once the solver finds one), which won the industrial WPMS category in the incomplete solver track of the MAX-SAT Evaluation 2014. We also report the results for *Eva* [Narodytska and Bacchus, 2014], which won the industrial WPMS category in the complete solver track. As *Eva* only finds one feasible solution finally when it proves optimality, the results for *Eva* are just reported to indicate the performance of the current state-of-the-art complete solver on these benchmarks.

The results (Table 3) show that, although *CCEHC* performs worse than *WPM-2014* on the Industrial benchmark and the pedigree benchmark, it is much better on random, crafted benchmarks and three real-world application benchmarks. The results of *VBS* present that *CCEHC* could be complementary to *WPM-2014* on the Crafted, Industrial, CPD, AES and pedigree benchmarks.

5.3 Initializing *CCEHC* by Unit Propagation

Inspired by the success of *DistUP*, which equips *Dist* with an unit propagation initialization [Cai *et al.*, 2016], we combine *CCEHC* with the unit propagation initialization, and empirically evaluate the resulting hybrid solver on all testing benchmarks. By replacing *Dist* with *CCEHC* in the *DistUP* solver, we obtain a new solver namely *CCEHC+UP*. The comparative results of *CCEHC+UP* and *CCEHC* are reported in Table 4. *CCEHC+UP* performs better than *CCEHC* on all testing benchmarks but one (the pedigree benchmark).

6 Conclusions

In this work, we design a heuristic with emphasis on hard clauses, and develop a new SLS algorithm named *CCEHC* for solving WPMS. We evaluate *CCEHC* on random, crafted, industrial and real-world application instances. Experiments show that *CCEHC* significantly outperforms state-of-the-art SLS algorithms namely *CCLS*, *Dist* and *DistUP* on these WPMS benchmarks. Experiments comparing *CCEHC* with a state-of-the-art complete solver *WPM-2014* show the effectiveness of *CCEHC* on random, crafted instances and many WPMS instances based on real-world applications. Also, we conduct empirical evaluations to study the combination of *CCEHC* and the unit propagation initialization.

Acknowledgements

This work is partially supported by the National Key Research and Development Program of China under Grant 2016YFB0200803 and Grant 2016YFC1401700, partially supported by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing under Grant 2016A06, partially supported by the National Natural Science Foundation of China under Grant 61502464, Grant 61572234, Grant 61472369 and Grant 61370072, and partially supported by the Australian Research Council under Grant DP150101618.

References

[Allouche *et al.*, 2012] David Allouche, Seydou Traoré, Isabelle André, Simon de Givry, George Katsirelos, Sophie Barbe, and Thomas Schiex. Computational protein design as a cost function network optimization problem. In *Proceedings of CP 2012*, pages 840–849, 2012.

[Allouche *et al.*, 2014] David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steven David Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.

[Ansótegui and Gabàs, 2013] Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial MaxSAT with ILP. In *Proceedings of CPAIOR 2013*, pages 403–409, 2013.

[Ansótegui *et al.*, 2013a] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (weighted) partial MaxSAT. In *Proceedings of CP 2013*, pages 117–132, 2013.

[Ansótegui *et al.*, 2013b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[Cai *et al.*, 2014] Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial MaxSAT. In *Proceedings of AAAI 2014*, pages 2623–2629, 2014.

[Cai *et al.*, 2016] Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016.

[Gwynne and Kullmann, 2011] Matthew Gwynne and Oliver Kullmann. Towards a better understanding of SAT translations. In *The Twelfth International Workshop on Logic and Computational Complexity*, 2011.

[Huang *et al.*, 2016] Wenxuan Huang, Daniil A. Kitchaev, Stephen Dacek, Ziqin Rong, Alexander Urban, Shan Cao, Chuan Luo, and Gerbrand Ceder. Finding and proving the exact ground state of a generalized Ising model by convex optimization and MAX-SAT. *Physical Review B*, 94:134424, 2016.

[Hutter *et al.*, 2011] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION 2011*, pages 507–523, 2011.

[Jiang *et al.*, 1995] Yueyun Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *First International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

[Li *et al.*, 2009] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *Proceedings of SAT 2009*, pages 467–480, 2009.

[Lin *et al.*, 2008] Han Lin, Kaile Su, and Chu Min Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *Proceedings of AAAI 2008*, pages 351–356, 2008.

[Luo *et al.*, 2015a] Chuan Luo, Shaowei Cai, Kaile Su, and Wei Wu. Clause states based configuration checking in local search for satisfiability. *IEEE Transactions on Cybernetics*, 45(5):1014–1027, 2015.

[Luo *et al.*, 2015b] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.

[Luo *et al.*, 2017] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017.

[Narodytska and Bacchus, 2014] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of AAAI 2014*, pages 2717–2723, 2014.

[Sánchez *et al.*, 2008] Martí Sánchez, Simon de Givry, and Thomas Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2):130–154, 2008.

[Selman *et al.*, 1992] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of AAAI 1992*, pages 440–446, 1992.

[Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of AAAI 1994*, pages 337–343, 1994.

[Smyth *et al.*, 2003] Kevin Smyth, Holger H. Hoos, and Thomas Stützle. Iterated robust tabu search for MAX-SAT. In *Proceedings of Canadian Conference on AI 2003*, pages 129–144, 2003.

[Thornton *et al.*, 2004] John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proc of AAAI 2004*, pages 191–196, 2004.