A New Local Search Strategy for SAT

Shaowei Cai[1], Kaile Su[2,3⋆], and Abdul Sattar[2,4]

[1] Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, China
[2] Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
[3] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[4] ATOMIC Project, Queensland Research Lab, NICTA, Australia
shaowei_cai@126.com; {k.su,a.sattar}@griffith.edu.au

Abstract. Recently, a new search strategy called configuration checking (CC) was proposed, for handling the cycling problem of local search. The CC strategy was used to improve the EWLS algorithm, a state-of-the-art local search for Minimum Vertex Cover (MVC). In this paper, we use this strategy to develop a local search algorithm for SAT called CW_{cc} and a local search algorithm for weighted MAX-2-SAT called ANGS_{cc}. The CC strategy takes into account the circumstances of the variables when selecting a variable to flip. Experimental results show that the configuration checking strategy is more efficient than previous strategies for handling the cycling problem. We further improve CW_{cc}; the resulting algorithm SW_{cc} outperforms a state-of-the-art local search SAT solver TNM. ANGS_{cc} is also competitive with a state-of-the-art weighted MAX-2-SAT local search algorithm. Finally, we conduct some further analysis and experiments to compare the CC strategy with two other methods for handling the cycling problem: the tabu mechanism and the promising decreasing variable exploitation strategy.

1 Introduction

Local search algorithms have been successfully used to solve SAT and MAXSAT problems [11]. The local search method is especially appealing when the problem instance is hard and large in size, or a reasonably good solution is needed in a short time, or when the knowledge about the problem domain is rather limited [9]. However, the local search method suffers from the problem of the cycling phenomenon, i.e., returning to a candidate solution that has been visited recently. The cycling phenomenon wastes a local search algorithm much time and prevents it from getting out of local optima. It is impractical to incorporate local search with an additional mechanism to remember all previously visited candidate solutions, which requires exponential space and huge time consumption for checking.

To overcome the cycling problem, some naive methods such as random walk and restarting strategy are incorporated into local search algorithms. Besides, there are two significant previous methods for handling the cycling problem: the *tabu* mechanism and the *promising decreasing variables* (PDV) exploitation strategy [12].

⋆ Corresponding author

Recently, a new search strategy called configuration checking (CC) was proposed to deal with the cycling problem of local search. It has been use to improve EWLS [2], a state-of-the-art local search algorithm for Minimum Vertex Cover (MVC). The resulting algorithm EWCC performs significantly better than EWLS and is currently the best MVC local search algorithm [3].

In this paper, we apply the CC strategy to design local search algorithms for SAT and weighted MAX-2-SAT. The CC strategy remembers the circumstance of a variable when it is flipped, and prevents it from being flipped if its circumstance has not been changed since its last flipping, where the circumstance of a variable refers to truth value of all its neighbors. We note that the CC strategy is novel in SAT local search algorithms.

We develop a local search algorithm for SAT called CW_{cc} (Clause Weighting SAT with Configuration Checking) and a local search algorithm for weighted MAX-2-SAT called ANGS_{cc} (Adaptive Noise Greedy Search with Configuration Checking) by using the CC strategy. To demonstrate the efficiency of the CC strategy, we slightly modify CW_{cc} and ANGS_{cc} and obtain CW_{tabu} and ANGS_{tabu} by using the *tabu* mechanism, and CW_{pdv} by using the *PDV* strategy. The experimental results of these algorithms show that the CC strategy is better than the two previous methods.

Further, we improve CW_{cc}, which results in an algorithm, called SW_{cc}, that outperforms the state-of-the-art local search SAT solver TNM. And ANGS_{cc} is competitive with a state-of-the-art weighed MAX-2-SAT local search algorithm. Nevertheless, the aim of this work is not to design algorithms of best performance, but to demonstrate the power of the configuration checking strategy.

Some further analysis and experiments are conducted and show that the forbidding strength of the CC strategy is between those of the tabu mechanism and the PDV strategy, and appears neither too weak, nor too strong. Thus, the CC strategy is a promising alternative strategy for dealing with the cycling problem.

The remainder of this paper is organized as follows: we provide some necessary background knowledge in the next section. Then we present the configuration checking strategy. After that, we use the configuration checking strategy to develop local search algorithms for SAT and weighted MAX-2-SAT respectively, and carry out some experimental studies to compare the efficiency of the CC strategy with the previous strategies and compare the proposed algorithms with the state-of-the-art algorithms. This is followed by further analysis about the differences between the configuration checking strategy and the previous strategies. Finally we give some conclusions and future directions.

2 Preliminaries

Given a Conjunctive Normal Form (CNF) formula $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ on a set of variables $\{x_1, x_2, ..., x_n\}$, the satisfiability problem (SAT) consists in testing whether all clauses in F can be satisfied by some consistent assignment of truth values to variables. A clause c_i is a disjunction of literals, where a literal l_j is either a variable x_j or its negation $\overline{x_j}$. We say a literal l occurs in a clause, if this clause contains l. However, when we say a variable x occurs in a clause, we mean that this clause contains either x or $\overline{x}$. By $V(F)$ we denote the set of all variables of F. $N(x) = \{y | y \in V(F)$ and y

occurs in at least one clause with $x\}$ is the neighborhood of a variable x. We use UC and $V(UC)$ to denote the set of unsatisfied clauses under the current assignment and the set of variables occur in at least one unsatisfied clause.

3 The Configuration Checking Strategy for SAT

In this section, we present the configuration checking (CC) strategy for handling the cycling problem of local search, using SAT as example domain. In the context of SAT solving, the CC strategy remembers each variable's circumstance information and prevents a variable from being flipped if its circumstance has not been changed since its last flipping. *The intuition behind this idea is that we can reduce cycles on the whole candidate solution by reducing local structure cycles.*

3.1 Definition of Configuration Checking

The CC strategy is based on the concept $configuration$, which denotes a variable's circumstance. Here the configuration of a variable refers to the truth value of all its neighbors. The formal definitions are as follows:

Definition 1. *Given a CNF formula F and s the current assignment to $V(F)$, the $configuration$ of a variable $x \in V(F)$ is a vector C_x consisting of truth value of all variables in $N(x)$ under s (i.e., $C_x = s|_{N(x)}$, which is the assignment restricted to $N(x)$).*

Given a CNF formula F, the $configuration\ checking$ strategy can be described as follows: when selecting a variable to flip, for a variable $x \in V(F)$, if the configuration of x has not changed since x's last flip, which means the circumstance of x never changes, then it should not be flipped. This strategy is reasonable in terms of avoiding cycles; otherwise, the algorithm is led to a scenario it has recently faced, which is likely to cause a cycle.

The configuration checking heuristic takes into account the variables' circumstance when selecting a variable to flip. It appears reasonable and helpful to incorporate such a circumstance-concerning strategy to the traditional variable-based heuristics, as the best decision on a variable should come from not only its evaluation, but also its circumstance, such as the state of the community it belongs to.

3.2 An Implementation of Configuration Checking

In order to implement the configuration checking strategy in local search algorithms for SAT (and MAXSAT), we employ an array $confChange$, whose element is an indicator for a variable — $confChange[x] = 1$ means the configuration of x has changed since x's last flipping; and $confChange[x] = 0$ on the contrary. *During the search procedure, the variables with $confChange[x] = 0$ are forbidden to be flipped.* We maintain the $confChange$ array as follows:

– Rule 1: In the beginning, for each variable x, $confChange[x]$ is initialized as 1.

– Rule 2: When flipping x, $confChange[x]$ is reset to 0, and for each $y \in N(x)$, $confChange[y]$ is set to 1.

The CC strategy is general and simple, and it can be used directly to constraint satisfactory problems (CSP) and other combinatorial problems.

We also note that to make a balance between the accuracy of the CC strategy and the complexity per step, we adopt the above approximate implementation of CC, which does not achieve full checking. Recall that the spirit of the CC strategy is that a variable x for which the configuration is not changed after its last flip is forbidden to be flipped. Consider x is flipped ($confChange[x]$ is set to 0); after that, a variable $y \in N(x)$ is flipped ($confChange[x]$ is set to 1) and then afterwards y is flipped again (a flip of y's another neighbor may make $confChange[y] = 1$ and thus y can be flipped again). Suppose other neighbors of x do not change their truth values. In this case, the configuration for x is considered changed ($confChange[x] = 1$) by the implementation in this paper, but it is not really changed since the truth value of all x's neighbors are the same as the last time x is flipped.

A naive accurate implementation of the CC strategy is to store the configuration for a variable x (store state of its neighbors) when it is flipped, and check the configuration when needed, say, when considering selecting x as the flip variable. Nevertheless, as is usual in local search algorithms, there is a tradeoff between the accuracy of heuristics and the complexity per step. It is rather time-consuming to execute the CC strategy in this naive accurate way. This accurate implementation needs $O(\Delta(V(F)))$ for both storing and checking the configuration for a variable, where $\Delta(V(F)) = max\{|N(x)||x \in V(F)\}$. Therefore, the worst time complexity per step for the CC strategy is $O(\Delta(V(F))|V(F)|) + O(\Delta(V(F))) = O(\Delta(V(F))|V(F)|)$ where (check the configuration for all variables and store the configuration for the flipped variable). While for the approximate implementation in this work, both checking and updating the $confChange$ indicator of a variable need only 1 operation, and thus the worst time complexity per step for the CC strategy is only $O(|V(F)|) + 1 + O(\Delta(V(F))) = O(|V(F)|)$ (check the $confChange$ indicators for all variables, reset the $confChange$ indicator for the flipped variable and update the $confChange$ indicators for the neighbors of the flipped variable).

4 Improving Local Search for SAT by Configuration Checking

We develop a simple local search algorithm for SAT, called CW$_{cc}$ (Clause Weighting with Configuration Checking), which utilizes a clause weighting scheme that updates clause weights when stuck in local optima, and combine the CC strategy into the algorithm. The experimental results demonstrate that the configuration checking strategy significantly improves the performance of CW$_{cc}$. We also improve it to achieve the state-of-the-art performance.

4.1 The CW$_{cc}$ Algorithm

The complete procedure of CW$_{cc}$ is shown in Algorithm 1. From the procedure, we can see that, besides the CC strategy, CW$_{cc}$ adopts the clause weighting technique, which

has been widely used in local search algorithms for SAT [14, 16, 10, 19]. In CW_{cc}, each clause c is associated to a positive integer number $w(c)$ as its weight. For a variable x, we use $\Delta w(x)$ to denote the change in the total weight of all satisfied clauses caused by flipping x. When CW_{cc} gets stuck in local optima, for each unsatisfied clause c, $w(c)$ is increased by 1, and then CW_{cc} takes a random step to continue to search from another starting point. We maintain the set D (in line 8) dynamically during the search procedure, rather than scanning $V(F)$ each step.

Algorithm 1: CW_{cc}

```
1  CW_cc(F, maxSteps)
   Input: CNF-formula F, maxSteps
   Output: A satisfying truth assignment s of F, if found
2  begin
3      s ← randomly generated truth assignment;
4      initialize all clause weights as 1 and compute Δw(x) for each variable x;
5      initialize confChange[x] as 1 for each variable x;
6      for step ← 1 to maxSteps do
7          if s satisfies F then return s ;
8          if D = {x|Δw(x) > 0 and confChange[x] = 1} ≠ ∅ then
9              v ← x ∈ D such that Δw(x) is the largest, breaking ties in favor of the least
               recently flipped variable;
10         else
11             w(c_i) ← w(c_i) + 1 for each unsatisfied clause c_i;
12             c ← randomly selected unsatisfied clause;
13             v ← the least recently flipped variable in c;
14         flip v, update confChange array according to Rule SAT_2;
15     return "Solution not found";
16 end
```

4.2 Experimental Evaluation of CW_{cc}

We demonstrate the effectiveness of the CC strategy on SAT local search algorithms by comparing the performance of CW_{cc} with its alternative version CW_{tabu}. CW_{tabu} works in the same way as CW_{cc}, except for one subtle modification: CW_{tabu} does not utilize the CC strategy; instead, it utilizes the tabu mechanism to keep track of the recently flipped variables in the tabu list and prevents them from being flipped. The length of the tabu list is called *tabu tenure (tt)*. we run CW_{tabu} with $tt = 1$ and 3.

The experiments are carried out using 20 random phase-transition 3-SAT problems and several classes of structured problems. The random 3-SAT instances are from the SAT 2009 competition benchmark[5], including the 10 largest instances in the "satisfiable random" category (3SAT-560vars) and 10 instances in the "unknown random" category

[5] http://www.satcompetition.org/

(3SAT-2000vars). The structured problems include AIS, the satisfiable instances in QG and SSA classes, and the largest (and the hardest) instance in Logistics and Blockworld classes, all available in SATLIB[6].

CW_{cc} and CW_{tabu} are implemented in C++ and compiled by g++ with the '-O2' option. The experiments are run on a 3 GHz Intel Core 2 Duo CPU E8400 and 4GB RAM under Linux. We say an instance is solved by a solver if the solver finds a solution satisfying all clauses of the instance. The portion of solved instances represents the success rate of the solver for this class. We run each solver within 1000 seconds for each instance in each class to get a success rate for this class, and repeat the execution 100 times to get the final averaged success rate ("suc"). We also report the total run time in seconds for each solver to run 100 times for each class ("total time").

Instance	CW_{tabu}(tt=1)		CW_{tabu}(tt=3)		CW_{cc}	
	suc	total time	suc	total time	suc	total time
3SAT-560vars(10 instances)	100%	4012	100%	3760	100%	**928**
3SAT-2000vars(10 instances)	76.9%	169631	82%	136864	**98.3%**	**38381**
satQG(10 instances)	86.4%	126518	88.9%	115689	**97.3%**	**62027**
ais12	100%	6.5	100%	7	100%	**5.5**
bw_large.d	100%	1400	100%	**811**	100%	1217

Table 1. Comparative results of CW_{cc} with CW_{tabu}

The 4 SSA instances and logistics.d are solved by all the three solvers in 100% success rate with the average run time less than 0.01 second, so they are not reported. As can be seen from Table 1, CW_{cc} outperforms the two CW_{tabu} algorithms on all instances, except for a little worse than CW_{tabu} with $tt = 3$ on bw_large.d. CW_{cc} achieves a significant improvement over CW_{tabu} on the random 3-SAT instances and the satQG instances. Specially, the run time of CW_{tabu} on the random 3-SAT instances are about 4 times that of CW_{cc}.

We also replace the CC strategy in CW_{cc} with the PDV strategy, resulting in another alternative version of CW_{cc}, called CW_{pdv}. CW_{pdv} follows the same overall procedure as CW_{cc}, except for one subtle modification: it does not utilize the CC strategy; instead, it uses the PDV strategy. In detail, it maintains an array $promising$ where $promising[x] = 1$ means x is a promising decreasing variable; and $promising[x] = 0$ on the contrary. Then, the D set in Algorithm 1 (line 8) is defined as $D = \{x | \Delta w(x) > 0 \text{ and } promising[x] = 1\}$ in CW_{pdv}. Our experiments show that CW_{pdv} performs much worse than CW_{cc} and CW_{tabu} on these instances, so we do not report the results of CW_{pdv} in Table 1.

In addition, we run G^2WSat[7] (the representative of local search SAT solvers with the PDV strategy) on the 10 3SAT-560vars instances, and the total run time for running all instances 100 times is 2509s, compared to 928s for CW_{cc}.

[6] http://www.satlib.org

[7] downloaded from http://www.satcompetition.org/

4.3 SW_{cc}: An Improved Version of CW_{cc}

We compare the CC strategy with the tabu mechanism and the PDV strategy in a simple algorithmic framework. In order to focus on the comparison, we keep CW_{cc} rather simple. To convince that the CC strategy for SAT is genuinely useful, we further improve CW_{cc}, making it achieve state-of-the-art performance, especially on random instances.

We introduce a smoothing mechanism into the clause weighting scheme in CW_{cc}, resulting in the algorithm called SW_{cc} (Smoothed Weighting with Configuration Checking). Smoothing mechanisms have been used to improve clause weighting local search methods for SAT [20, 16, 10, 19]. The smoothing mechanism in SW_{cc} is simple: when the averaged clause weight (over all clauses) $\overline{w}$ is bigger than a threshold value γ, all the clause weights are smoothed as $w(c_i) := \lfloor \rho \cdot w(c_i) \rfloor + \lfloor (1-\rho)\overline{w} \rfloor$. In our experiments, $\gamma := 300$ and $\rho := 0.3$. The two parameters are set preliminarily according to a few experiments. We believe if we set these parameters more carefully, further improvement could be made.

We compare SW_{cc} with a state-of-the-art local search algorithm TNM (Two Noise Mechanisms), which won a GOLD Medal in the "Satisfiable Random" category of the SAT 2009 competition[8]. The experiments are carried out on some random phase-transition instances from the SAT 2009 competition. The cutoff time of each run is still 1000 seconds. We report the success rate, the total run-time in seconds and the averaged step number over all 1000 runs.

Instance	SW_{cc}			TNM		
	suc	total time	ave step	suc	total time	ave step
3SAT-560vars(10 instances)	100%	**500**	**1109239**	100%	1450	3097895
3SAT-2000vars(10 instances)	100%	**1565**	**2566081**	100%	1623	3272796
3SAT-4000vars(10 instances)	**100%**	**23808**	**27442994**	94.9%	72170	134020195

Table 2. Comparative results of SW_{cc} with TNM on random 3SAT instances

As shown in Table 2, SW_{cc} overall performs better than TNM on the random phase-transition 3-SAT instances and significantly outperforms TNM on the random 3-SAT instances with 560 and 4000 variables. Specially, on the hardest instance (for both algorithms) in the 3SAT-4000vars group, which is much harder than the other instances, SW_{cc} achieves a success rate 100% while this number is 49% for TNM. Also, SW_{cc} is always better than TNM in terms of step performance, which is an implementation-independent measure of algorithms.

In order to convince the power of the CC strategy in SW_{cc}, we modify SW_{cc} slightly by replacing the CC strategy with the tabu mechanism and the PDV strategy, which results in SW_{tabu}(with $tt = 1$ and $tt = 3$) and SW_{pdv}. Our experiments show that SW_{tabu} and SW_{pdv} perform essentially worse than SW_{cc}. For example, both SW_{tabu} and SW_{pdv} can not find a solution for the hardest instance in the 3SAT-4000vars group in 50 runs. Therefore, we conclude that the CC strategy plays an important role in SW_{cc}.

[8] downloaded from http://www.satcompetition.org/

5 Improving Local Search for Weighted MAX-2-SAT by Configuration Checking

We develop a simple local search algorithm for weighted MAX-2-SAT which adopts an adaptive noise parameter for controlling the probability of performing a random step. We then combine the CC strategy into the algorithm, and get an improved algorithm called ANGS$_{cc}$ (Adaptive Noise Greedy Search with Configuration Checking). The experimental results demonstrate that the configuration checking strategy significantly improves the algorithm and makes it achieve state-of-the-art performance.

5.1 The ANGS$_{cc}$ Algorithm

As is usual, ANGS$_{cc}$ treats weighted MAX-2-SAT as a minimization problem, and the objective function is

$$f(s) = \sum_{c_i \text{ is not satisfied under s}} w(c_i)$$

where $w(c_i)$ is the weight of clause c_i and s is an assignment to $V(F)$. For a variable x, we use $\Delta w(x)$ to denote the change in the total weight of all satisfied clauses caused by flipping x.

ANGS$_{cc}$ performs either a greedy step or a random step at each step. The probability of performing a random step is controlled by a noise parameter wp (walking probability), which is adjusted during the search. For adjusting wp, we adopt the adaptive noise mechanism introduced in [8]. In detail, in the beginning wp is initialized as 0. During the search procedure, each time updating wp, the current objective function value is stored and becomes the basis for measuring improvement. If no improvement in objective function value has been observed over the last $\theta \cdot m$ search steps, where m is the number of clauses of the given instance and $\theta = 1/6$, then $wp := wp + (1 - wp) \cdot \phi$, where $\phi = 0.2$; otherwise, if an improvement in objective function value is observed, then $wp := wp - wp \cdot \phi/2$. The complete procedure of ANGS$_{cc}$ is shown in Algorithm 2.

5.2 Empirical Study of ANGS$_{cc}$

We demonstrate the effectiveness of the CC strategy on weighted MAX-2-SAT by comparing the performance of ANGS$_{cc}$ with its alternative version ANGS$_{tabu}$. ANGS$_{tabu}$ works in the same way with ANGS$_{cc}$, except for one subtle modification: ANGS$_{tabu}$ does not utilize the CC strategy; instead, it uses the tabu mechanism to keep track of the last flipped variable and prevents it from being re-flipped in the next step. We also compare the performance of ANGS$_{cc}$ with a state-of-the-art local search algorithm for weighted MAX-2-SAT called ITS (Iterated Tabu Search) [15], which significantly outperforms general MAXSAT local search algorithms such as GWSAT [17], Adaptive Novelty+ [8], SAPS [10], and IRoTS [18] on the BHOSLIB benchmarks and some random instances generated by the authors [15]. ITS is known as one of the best local

Algorithm 2: ANGS$_{cc}$

```
1  ANGS_cc(F, maxSteps)
   Input: weighted MAX-2-SAT instance F, maxSteps
   Output: A truth assignment s* of F
2  begin
3      s ← randomly generated truth assignment;
4      s* ← s;
5      compute Δw(x) for each variable x;
6      initialize confChange[x] as 1 for each variable x;
7      wp ← 0;
8      for step ← 1 to maxSteps do
9          if f(s) < f(s*) then s* ← s;
10         adjust wp;
11         with probability wp begin
12             c ← randomly selected unsatisfied clause;
13             v ← randomly selected variable in c;
14         end
15         otherwise begin
16             v ← x ∈ V(UC) and confChange[x] = 1 such that Δw(x) is the largest,
               breaking ties randomly;
17         end
18         flip v, update confChange array according to Rule 2;
19     return s*;
20 end
```

search algorithms for weighted MAX-2-SAT, especially for the BHOSLIB benchmarks [7].

The experiments are carried out using the BHOSLIB benchmarks, which are hard random instances with known optimal values of the objective function[9]. The BHOSLIB (Benchmarks with Hidden Optimum Solutions) benchmarks were generated randomly in the phase transition area according to the RB model [21]. Generally, those phase-transition instances generated by RB have been proven to be hard both theoretically [24] and practically [23, 22].

Instance	#vars	#clauses	f^*	ANGS$_{tabu}$ suc	ANGS$_{cc}$ suc	ITS suc
frb40-19-1	760	42173	720	100%	100%	100%
frb40-19-2	760	42127	720	70%	**80%**	30%
frb40-19-3	760	41927	720	90%	**100%**	**100%**
frb40-19-4	760	42485	720	40%	**100%**	70%
frb40-19-5	760	42499	720	0	**90%**	20%
frb45-21-1	945	60311	900	30%	80%	**90%**
frb45-21-2	945	59716	900	40%	80%	**100%**
frb45-21-3	945	59330	900	10%	**20%**	**20%**
frb45-21-4	945	59652	900	40%	**100%**	**100%**
frb45-21-5	945	59691	900	20%	**80%**	60%
frb50-23-1	1150	81422	1100	0	**40%**	10%
frb50-23-2	1150	82201	1100	10%	10%	10%
frb50-23-3	1150	82360	1101	80%	**100%**	90%
frb50-23-4	1150	81608	1100	30%	80%	**100%**
frb50-23-5	1150	81385	1100	10%	30%	**70%**
frb53-24-1	1272	95711	1220	20%	60%	**100%**
frb53-24-2	1272	95773	1220	80%	80%	**100%**
frb53-24-3	1272	95611	1219	0	10%	**20%**
frb53-24-4	1272	95792	1220	20%	**30%**	**30%**
frb53-24-5	1272	95710	1219	0	**10%**	**10%**
frb56-25-1	1400	111328	1345	0	20%	**40%**
frb56-25-2	1400	111081	1345	10%	20%	**40%**
frb56-25-3	1400	111012	1344	0	**10%**	0
frb56-25-4	1400	111682	1344	0	0	**10%**
frb56-25-5	1400	111235	1344	10%	10%	**20%**
frb59-26-1	1534	128308	1476	**20%**	10%	**20%**
frb59-26-2	1534	127956	1476	0	**30%**	10%
frb59-26-3	1534	127911	1476	0	10%	**30%**
frb59-26-4	1534	128799	1476	0	**10%**	**10%**
frb59-26-5	1534	127763	1475	0	**40%**	**40%**
Averaged success rate				24.3%	48%	48.3%

Table 3. Comparative results of ANGS$_{cc}$ with ANGS$_{tabu}$ and ITS, where f^* is the best value of the objective function provided by the three algorithms

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

Both ANGS$_{cc}$ and ANGS$_{tabu}$ are implemented in C++, and the code of ITS is download from http://www.soften.ktu.lt/~gintaras/wmax2sat.html. Since ITS is coded for running on Windows, we compile all these three solvers in Microsoft Visual Studio 2003, and all experiments in this section are run on a 1.83 GHz Intel Core 2 Duo CPU T5600 and 2GB RAM under Windows XP. For each instance, each algorithm performs 10 independent runs with the cutoff time 600 seconds. For each algorithm on each instance, we report the success rate of finding a best solution returned by the three algorithms. The results in **bold** is the best performance for an instance.

Comparative results on BHOSLIB benchmarks are shown in Table 3. The two groups of small instances ($frb30$, $frb35$) are not reported, as all the solvers can find an optimal solution in 100% success rate within a few minutes. As can be seen from Table 3, ANGS$_{cc}$ outperforms ANGS$_{tabu}$ on almost all instances. Furthermore, there are 10 instances for which ANGS$_{tabu}$ never reaches the best solution that ANGS$_{cc}$ finds. The averaged success rate over all instances of ANGS$_{cc}$ is twice that of ANGS$_{tabu}$.

Table 3 also suggests that ANGS$_{cc}$ and ITS are competitive and complementary as they dominates on different instances — each has its own territory. Also their averaged success rate are nearly the same.

6 Further Analysis and Comparison

In this section, we review the tabu mechanism and the promising decreasing variable exploitation strategy, and discuss the differences between the CC strategy and them in the context of SAT. For simplicity, in the following discussions, when talking about the CC strategy, we refer to the implementation in this work.

6.1 Tabu vs. Configuration Checking

The *tabu* mechanism [5, 6] has been widely used in local search algorithms [13, 1, 18, 4]. To prevent the local search to immediately return to a previously visited candidate solution and to avoid cycling, the tabu mechanism forbids reversing the recent changes, where the forbidding strength is controlled by a parameter called *tabu tenure*. Usually, the *tabu tenure* is set to be 1 consistently, to avoid manually adjusting this parameter.

Proposition 1. *For a given variable x, if x is forbidden to flip by the tabu mechanism ($tabu\ tenure = 1$), then $confChange[x] = 0$.*

Proof: If at the current local search step, x is forbidden to flip by the tabu mechanism with $tabu\ tenure = 1$, then x is the variable that just be flipped at last step ($confChange[x]$ would be set to be 0); since there is no flips between last step and the current step, $confChange[x]$ would be still 0 when selecting the variable to flip at the current step. ∎

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

According to Proposition 1 and Remark 1, we conclude that the forbidding strength of the CC strategy is stronger than that of the tabu mechanism ($tabu\ tenure = 1$).

6.2 Promising Decreasing Variable vs. Configuration Checking

Recently, a deterministic exploitation strategy based on *promising decreasing variables* (PDV) was introduced in the G^2WSat algorithm [12]. This strategy can in some way handle the cycling problem of local search algorithms for SAT, and has been used in most winning local search solvers in the recent SAT competitions.

The following definitions are taken from [12]: A variable is said *decreasing* if flipping it would decrease the number of unsatisfied clauses. Let x and y be variables, $x \neq y$, x is not decreasing. If it becomes decreasing after y is flipped, then we say that x is a *promising decreasing variable* after y is flipped. Let x be a promising decreasing variable after some variable is flipped. If x is always decreasing after one or more other moves, it is always promising. We use $score(x)$ to denote the difference of the number of satisfied clauses by flipping x. Then a variable x is *decreasing* iff $score(x) > 0$.

Proposition 2. *For a given variable x, if x is a promising decreasing variable, then $confChange[x] = 1$.*

Proof: The proof is given by deduction.
(a) *Becoming a promising decreasing variable*. If x becomes a promising decreasing variable after flipping another variable y, then we conclude $y \in N(x)$. Otherwise y is independent with x and flipping y does nothing to $score(x)$. Since $y \in N(x)$, along with flipping y, $confChange[x]$ would be set to be 1.
(b) *Remaining a promising decreasing variable*. For a promising decreasing variable x, if x remains promising decreasing, then we conclude that x has not been flipped after the last time it became a promising decreasing variable. Otherwise, because x is decreasing, i.e., $score(x) > 0$, flipping x would make $score(x) < 0$ (flipping x would make $score(x)$ to be its opposite number); this means x is no longer a decreasing variable, and thus not a promising decreasing variable. Since x has not been flipped after it becoming a promising decreasing variable, recalling only flipping x would set $confChange[x]$ to 0, so $confChange[x]$ remains 1. ∎

Remark 2. The reverse of Proposition 2 is not necessarily true.

To see this, consider a variable x with $score(x) < 0$, x is flipped (for example, in a random step). Then $score(x) > 0$, i.e., x becomes decreasing. However, x does not become a promising decreasing variable, because x becomes decreasing by flipping itself, rather than another variable. Afterwards, a variable $y \in N(x)$ is flipped ($confChange[x]$ would be set to 1), which may keep $score(x) > 0$. This still does not make x become a promising decreasing variable. Actually, in order to make x a promising decreasing variable, there are two necessary phases: (1) making $score(x) < 0$ again; (2) x becomes decreasing by flipping one of its neighbors.

According to Proposition 2 and Remark 2, we conclude that the forbidding strength of the CC strategy is not so strong as that of the PDV strategy.

6.3 Experimental Study for Comparing Forbidding Strength

As stated above, the forbidding strength of the CC strategy sits between tabu and PDV. An experimental study for comparing the forbidding strengths of these three strategies

is conducted as follows. We use the following notations: $B = \{x|\Delta w(x) > 0$ and x is forbidden to flip$\}$, and $S = \{x|\Delta w(x) > 0\}$.

We run CW_{cc}, CW_{tabu} and CW_{pdv} on each instance 10 times within $maxSteps = 10^8$. For each run of these algorithms, the ratio $\frac{|B|}{|S|}$ (which corresponds to the forbidding strength) averaged over all steps are computed, and the mean values of $\frac{|B|}{|S|}$ over all runs on all instances for each instance class are reported in Table 4. We do not perform experiments on the ais12 instance because CW_{cc} finds a solution in a few steps, and the computed statistical data can not reflect its forbidding strength.

Table 4 shows that the mean values of $\frac{|B|}{|S|}$ of CW_{cc} are exactly between those of CW_{tabu} ($tt <= 3$) and CW_{pdv}, which indicates the forbidding strength of CW_{tabu} is weaker than that of CW_{cc}, which in turn weaker than CW_{pdv}. This observation is consistent with our formal analysis on the forbidding strength of the three strategies. From the viewpoint of striking a balance between exploration and exploitation in local search, CC seems better than PDV or tabu.

Instance	CW_{tabu}(tt=1) $\frac{\lvert B\rvert}{\lvert S\rvert}$	CW_{tabu}(tt=3) $\frac{\lvert B\rvert}{\lvert S\rvert}$	CW_{cc} $\frac{\lvert B\rvert}{\lvert S\rvert}$	CW_{pdv} $\frac{\lvert B\rvert}{\lvert S\rvert}$
3SAT-560vars(10 formulas)	11.28%	28%	57%	85.7%
3SAT-2000vars(10 formulas)	7.78%	28.34%	81.92%	93.26%
satQG(10 formulas)	9.42%	23.3%	26.4%	76.87%
bw_large.d	2.89%	11.42%	81.39%	91.79%

Table 4. Forbidding strength comparison of CW_{cc} with CW_{tabu} and CW_{pdv}

Based on the analysis above, it is clear that the CC strategy provides an interesting alternative to the tabu mechanism and the PDV strategy for handling the cycling problem in SAT local search algorithms. Also, we provide a framework for handing the cycling problem, in which forbidding strength naturally corresponds to 'diversification' of local search. We believe that studying different strategies in a same framework is helpful for developing new strategies that can handle the cycling problem better.

7 Conclusions and Future Work

The configuration checking (CC) strategy was recently proposed to deal with the cycling problem of local search [3]. By reducing local structure cycles, the CC strategy can handle the cycling problem well. We have utilized the CC strategy to design two algorithms CW_{cc} and ANGS_{cc} for SAT and weighted MAX-2-SAT respectively. Furthermore, we have improved CW_{cc} by a clause weight smoothing mechanism, resulting in SW_{cc}. The CC strategy in the context of SAT solving remembers each variable's configuration (truth value of all its neighbors), and prevents a variable from being flipped if its configuration has not been changed since its last flip.

We emphasize that in order to demonstrate the effectiveness of the CC strategy clearly, we keep the algorithms simple, and the programming skill is rather naive. Even

so, the power of the CC strategy makes them achieve state-of-the-art performance. The experiments on random 3-SAT instances indicate that SW_{cc} is better than TNM, the best solver on random instances in SAT competition 2009. And ANGS_{cc} is competitive with the state-of-the-art local search solver for weighted MAX-2-SAT.

Also, we have conducted some further analysis and experiments to compare the CC strategy with two other significant methods for handling the cycling problem: the tabu mechanism and the promising decreasing variable exploitation strategy. The forbidding strength of the CC strategy stands in the middle of the tabu mechanism and the PDV strategy, and is neither too weak, nor too strong. The effectiveness of the CC strategy is clearly shown by the fact that CW_{cc} significantly outperforms CW_{tabu} and CW_{pdv}, and ANGS_{cc} significantly outperforms ANGS_{tabu}. Thus, the CC strategy is a promising alternative strategy for handling the cycling problem.

As for future work, we would like to apply the configuration checking idea to other combinatorial search algorithms. In particular, given the success of SW_{cc} in this work, we believe that it may further improve the state of the art in SAT solving if we combine the proposed CC strategy for SAT to some stronger SAT local search solver. A more general direction is to exploit the circumstance information of solution components to design more efficient algorithms for combinatorial problem.

References

1. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica 29(4), 610–637 (2001)
2. Cai, S., Su, K., Chen, Q.: Ewls: A new local search for minimum vertex cover. In: Proc. of AAAI-10. pp. 45–50 (2010)
3. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. 175(9-10), 1672–1696 (2011)
4. Gaspero, L.D., Schaerf, A.: A composite-neighborhood tabu search approach to the traveling tournament problem. J. Heuristics 13(2), 189–207 (2007)
5. Glover, F.: Tabu search – part i. ORSA Journal on Computing 1(3), 190–206 (1989)
6. Glover, F.: Tabu search – part ii. ORSA Journal on Computing 2(1), 4–32 (1990)
7. Heras, F., Bañeres, D.: The impact of max-sat resolution-based preprocessors on local search solvers. Journal on Satisfiability, Boolean Modeling and Computation 7, 89–126 (2010)
8. Hoos, H.H.: An adaptive noise mechanism for walksat. In: Proc. of AAAI-02. pp. 655–660 (2002)
9. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Elsevier / Morgan Kaufmann (2004)
10. Hutter, F., Tompkins, D.A.D., Hoos, H.H.: Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In: Proc. of CP-02. pp. 233–248 (2002)
11. Kautz, H.A., Sabharwal, A., Selman, B.: Incomplete algorithms. In: Handbook of Satisfiability, pp. 185–203 (2009)
12. Li, C.M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. In: Proc. of SAT-05. pp. 158–172 (2005)
13. Mazure, B., Sais, L., Grégoire, É.: Tabu search for sat. In: Proc. of AAAI-97. pp. 281–285 (1997)
14. Morris, P.: The breakout method for escaping from local minima. In: Proc. of AAAI-93. pp. 40–45 (1993)

15. Palubeckis, G.: Solving the weighted max-2-sat problem with iterated tabu search. Journal of Information Technology and Control 37, 275–284 (2008)
16. Schuurmans, D., Southey, F.: Local search characteristics of incomplete sat procedures. Artif. Intell. 132(2), 121–150 (2001)
17. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proc. of AAAI-94. pp. 337–343 (1994)
18. Smyth, K., Hoos, H.H., Stützle, T.: Iterated robust tabu search for max-sat. In: Proc. of Canadian Conference on AI. pp. 129–144 (2003)
19. Thornton, J., Pham, D.N., Bain, S., Jr., V.F.: Additive versus multiplicative clause weighting for sat. In: Proc. of AAAI-04. pp. 191–196 (2004)
20. Wu, Z., Wah, B.W.: An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In: AAAI/IAAI. pp. 310–315 (2000)
21. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: A simple model to generate hard satisfiable instances. In: Proc. of IJCAI-05. pp. 337–342 (2005)
22. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random constraint satisfaction: Easy generation of hard (satisfiable) instances. Artif. Intell. 171(8-9), 514–534 (2007)
23. Xu, K., Li, W.: Exact phase transitions in random constraint satisfaction problems. J. Artif. Intell. Res. (JAIR) 12, 93–103 (2000)
24. Xu, K., Li, W.: Many hard examples in exact phase transitions. Theoretical Computer Science 355, 291–302 (2006)