ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-570

Improving Local Search for Random 3-SAT Using Quantitative Configuration Checking

Chuan Luo[1] and **Kaile Su**[2] and **Shaowei Cai**[2,1]

Abstract. Configuration Checking (CC) was proposed as a new diversification strategy for Stochastic Local Search (SLS) algorithm for solving Minimum Vertex Cover, and has been successfully used for solving the Boolean Satisfiability problems, leading to an SLS algorithm called Swcc. However, the CC strategy for SAT is in the early stage of study, and Swcc cannot compete with the best SLS solvers for SAT in SAT Competition 2011. This paper presents a new strategy called Quantitative Configuration Checking (QCC), which is a quantitative version of the CC strategy for SAT. QCC is based on a new definition of "configuration" and works in a different way from the CC strategy does. Specifically, while previous CC strategies work only in the greedy mode, QCC firstly works in the random mode. We use QCC to improve the Swcc algorithm, resulting in a new SLS algorithm for SAT called Swqcc. Experimental results show that the QCC strategy is more effective than the CC strategy. Furthermore, Swqcc outperforms the best local search SAT solver in SAT Competition 2011 called Sparrow2011 on random 3-SAT instances.

1 INTRODUCTION

The Satisfiability problem (SAT), which has been widely studied in the AI community due to its significant importance in both theory and applications [11], is one of the most important NP-complete problems. Given a propositional formula in conjunctive normal form (CNF) with variables $\{x_1, \cdots, x_n\}$, the SAT problem consists in finding an assignment for the variables so that all clauses are satisfied. Besides interest in instances encoded from industry problems, there is also much interest in random instances. Random SAT instances provide a relatively "unbiased" sample for benchmarking algorithms, and permit algorithms to be tested on statistically significant samples of hard problems. Indeed, random SAT instances have been widely studied and are underlying one of the three categories in the SAT competition [3].

The algorithms used to solve SAT problems can be categorized into two classes: complete algorithms based on the DPLL algorithm and stochastic local search (SLS) algorithms. In this paper, we focus on the latter one. Although SLS solvers are usually incomplete, i.e., they cannot determine with certainty that a given propositional formula is unsatisfiable, they are very efficient in solving satisfiable instances, especially the randomly generated ones.

The main scheme of an SLS algorithm for SAT can be described as follows: In the beginning, the algorithm generates a random assignment of boolean values to the variables appear in the formula. Then the algorithm flips a variable in each search step using a function for selecting the variable to be flipped. The SLS algorithm executes search steps iteratively until it seeks out a satisfiable assignment or timeout. Therefore, the function for selecting the flipping variable is the essential part of an SLS algorithm for SAT.

SLS algorithms for SAT usually work in two different modes, i.e., the greedy mode and the random mode. In the greedy mode, they prefer variables whose flips can decrease the number of unsatisfied clauses; while in the random mode, they tend to better explore the search space and avoid local optima, usually using randomized strategies to pick a variable.

Recently, a diversification strategy called configuration checking (CC), which may help deal with the cycling problem, i.e. returning to a candidate solution that has been visited recently [13], was proposed. This CC strategy was first used to improve a state-of-the-art local search algorithm EWLS [5] for Minimum Vertex Cover (MVC), leading to the much more efficient SLS solver EWCC for MVC [6]. Furthermore, the CC strategy is a general local search strategy and a direct application of CC in SAT has resulted in an SLS algorithm called Swcc [3]. The CC strategy for SAT forbids a variable to flip if all its neighboring variables have not changed their truth values since its last flip. The experimental results in [3] show that Swcc outperforms the best SLS solver in SAT Competition 2009 called TNM on random 3-SAT instances, and also indicate that CC is more effective than the tabu method [9, 10], which is an influential method for dealing with the cycling problem.

However, the CC strategy is still in its infancy, and Swcc cannot compete with the winner in random satisfiable category of SAT Competition 2011, namely Sparrow2011, which is an improved version of Sparrow [2]. A natural extension of the CC strategy is to consider the quantitative variation of configurations, and prefer to flip those variables whose quantitative variations of configuration are greater. In this paper, we carry out some research towards this direction. Specifically, we propose a new local search strategy called Quantitative Configuration Checking (QCC), which can be seen as a quantization version of CC. Although QCC stems from the configuration checking idea, it is rather different from the CC strategy, in terms of the definitions of configuration and the ways they work. In QCC, a variable's configuration refers to the states of the clauses in which it appears; also, QCC utilizes variation information of configurations to select the flipping variable. Moreover, while previous CC strategies work only in the greedy mode, QCC is used in the random mode, and QCC improves the performance of our algorithm significantly. For more discussions about the differences the CC and QCC strategies, refer to the discussion section. We would also like to note that the recent work [4], done in parallel to ours, combines the CC strategy

[1] Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China.
Email: { chuanluosaber@gmail.com }
[2] Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia.
Email: { k.su@griffith.edu.au; shaowei_cai@126.com }

with an aspiration mechanism, resulting in a new heuristic called CCA. CCA was used to design an SLS algorithm for SAT called Swcca [4], which achieves very good performance on both random 3-SAT and structured SAT instances.

We use QCC to improve the Swcc algorithm, resulting in a new SLS algorithm for SAT called Swqcc. The experimental results show that Swqcc performs consistently significantly better than Swcc, indicating that the QCC strategy is more efficient than the CC strategy. Moreover, to convince the superior performance of Swqcc, we compare Swqcc with Sparrow2011, which is the winner of the random satisfiable category of the SAT Competition 2011. Note that Sparrow2011 makes a breakthrough in solving random 3-SAT instances, and is especially considered to be the best local search solver for random 3-SAT instances. The experimental results show that Swqcc outperforms Sparrow2011 on random 3-SAT instances.

The remainder of the paper is structured as follows. Next section provides some definitions and notions used in this paper. Then we present the CC strategy and the QCC strategy. After that, we use QCC to develop an SLS algorithm for SAT called Swqcc. Experiments demonstrating the effectiveness of QCC and Swqcc are presented next. This is followed by further discussions about QCC. Finally we conclude the paper and give some future work.

2 PRELIMINARIES

Given a set of n boolean variables $V = \{x_1, \cdots, x_n\}$ and the set of corresponding literals $L = \{x_1, \neg x_1, \cdots, x_n, \neg x_n\}$, a *clause* is a disjunction of literals. In k-SAT, each clause contains exactly k different literals. Using clauses and the logical operation AND ($\wedge$), we can construct a CNF formula, i.e., $F = c_1 \wedge \cdots \wedge c_m$, where the number of clauses in the given formula F is denoted m, and $r = m/n$ is its ratio. A formula can be understood as a set of clauses. We use $V(F)$ to denote the set of all variables appear in the formula F. Two different variables are neighbors when they share at least one clause, and $N(x) = \{y \mid y \in V(F), y \; and \; x \; are \; neighbors\}$ is the set of all neighbors of variable x. We also define that $CL(x) = \{c \mid c \; is \; a \; clause \; which \; x \; appears \; in\}$.

A (possibly partial) mapping $\alpha : V(F) \rightarrow \{True, False\}$ is called an *assignment*. If α maps all variables to a Boolean value, it is called *complete*. For local search algorithms for 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. An assignment α satisfies a formula F if α satisfies all clauses in F. Given a CNF formula F, the SAT problem is to find an assignment that make all clauses in F satisfied.

As Swqcc, which uses clause weighting scheme, is a dynamic local search, here we introduce some notations about dynamic local search. In a dynamic local search algorithm for SAT, each clause $c \in F$ is associated with a nonnegative integer $weight(c)$ as its weight. The averaged clause weight over all clauses is denoted by $\overline{w}$. We use $cost(F, \alpha)$ to denote the total weight of unsatisfied clauses under an assignment α. For each variable x in F, we define that $score(x) = cost(F, \alpha) - cost(F, \alpha')$, where α' is obtained from α by flipping the boolean value of x.

3 THE CONFIGURATION CHECKING STRATEGY (CC) FOR SAT

This section introduces the CC strategy for SAT, as QCC is an improved version of CC. The CC strategy remembers each variable's circumstance information, and forbids flipping variables whose circumstance information has not been changed since its last flip.

3.1 The Definition of Configuration Checking

The CC strategy is based on the concept of $configuration$. In the context of SAT, the configuration of a variable refers to the truth values of all its neighboring variables. Formally, we have the following definition.

Definition 1 *Given a CNF formula F and the current assignment α to $V(F)$, the $configuration$ of a variable $x \in V(F)$ is a vector $T(x)$ consisting of boolean values of all variables in $N(x)$ under assignment α.*

Given a CNF formula F, the CC 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, meaning the circumstance of x never changes, then it should not be flipped. This strategy serves as a diversification form, which prevents the algorithm from facing a scenario it recently faced. As stated in [3], previous SLS algorithms for SAT never take circumstance information of variables into consideration. They usually select the flipping variable according to the information of variables, such as score [15, 12], break-count [14], and age [8]. However, the CC strategy combines the circumstance information with the traditional heuristics on selecting the flipping variable. This is the essential difference between the CC strategy and the previous works.

3.2 An Implementation of Configuration Checking

In Cai and Su's work on the CC strategy for SAT [3], the CC strategy is implemented with a boolean array $confchange$, whose element refers whether the configuration of a variable is changed or not since its last flip. If $confchange(x)$ is true, it means the configuration of variable x is changed; otherwise on the contrary. During the search procedure, the variables whose $confchange$ values are false are forbidden to be flipped. The $confchange$ array is maintained according to the following rules:

- Rule 1: At the start of local search, all the variables' $confchange$ is initialized as true.
- Rule 2: When flipping variable x , $confchange(x)$ is reset to false, and for each variable $y \in N(x)$, $confchange(y)$ is reset to true.

4 THE QUANTITATIVE CONFIGURATION CHECKING STRATEGY (QCC) FOR SAT

The CC strategy for SAT as presented in the previous section only considers whether the configuration of a variable is changed, and does not care the quantity of times the configuration of a variable is changed. In this section, we introduce a new strategy, called Quantitative Configuration Checking (QCC), by combining the quantity of configurations' variations into the CC strategy.

4.1 Definitions and Notations in QCC

We first give some related definitions and notations in the QCC strategy. Different from the CC strategy, where the configuration of a variable refers to the truth values of all its neighboring variables, in the QCC strategy the configuration of a variable refers to

the states of all clauses it appears in. We give the formal definition of $configuration$ in QCC as follows:

Definition 2 *Given a CNF formula F and an assignment α to $V(F)$, the $configuration$ of a variable $x \in V(F)$ is a vector $configuration(x)$ consisting of the states of all clauses in $CL(x)$ under assignment α.*

For a variable x, a change on any bit of $configuration(x)$ is considered as a change on the whole $configuration(x)$ vector. We use $TCC(x)$ (short for "times of configuration changes") to denote the number of times that $configuration(x)$ has been changed **since x's last flip**. The QCC strategy prefers to pick the variables whose configurations have been changed more times since their last flips.

4.2 An Implementation of Quantitative Configuration Checking

In order to implement the QCC strategy in local search algorithms for SAT, we employ an integer array $ConfVariation$, whose size equals the number of variables in the formula. For a variable x, $ConfVariation(x)$ can be seen as a smoothed version of $TCC(x)$. Inspired by the success of smoothing techniques in clause weighting SLS algorithms for SAT, we use a smoothing mechanism to decrease TCC values periodically. These smoothed TCC values are stored in the $ConfVariation$ array.

Due to the smoothing mechanism, recent changes on configurations contribute more to the $ConfVariation$ values. Therefore, for a variable, the greater its $ConfVariation$ value is, the more its configuration was changed recently. In the QCC strategy, the variables with greater $ConfVariation$ values have more priorities to be flipped. We maintain the $ConfVariation$ array as follow:

- Rule 3: At the start of local search, all the variables' $ConfVariations$ are set to 1.
- Rule 4: When flipping x, $ConfVariation(x)$ is reset to 0; and if flipping x makes some clauses change their states (from satisfied to unsatisfied or from unsatisfied to satisfied), for each variable y appearing in those clauses (except x), $ConfVariation(y)$ is increased by 1.
- Rule 5: When the averaged clause weight $\overline{w}$ is greater than a constant value δ, for each variable x with $ConfVariation(x) > 0$, $ConfVariation(x)$ is smoothed using the formula: $ConfVariation(x) = ConfVariation(x) * \beta + 1$, where $0 \leq \beta \leq 1$.

In Swqcc, we utilize $ConfVariation$ to select the variable to flip. Specifically, in the greedy mode, the algorithm picks the variable with the greatest $score$, preferring the one with the greatest $ConfVariation$ value to break ties. In the random mode, firstly a random unsatisfied clause is selected; then the algorithm picks the variable whose $ConfVariation$ is the greatest in the clause as the flipping variable.

5 LOCAL SEARCH USING QUANTITATIVE CONFIGURATION CHECKING

We use the QCC strategy to improve the Swcc algorithm, whose pseudo-code can be found in [3], resulting in a new SLS algorithm for SAT, which is called Swqcc (Smoothed Weighting with Quantitative Configuration Checking).

5.1 Smoothing Clause Weighting Scheme

Swqcc uses a smoothed clause weighting scheme, as its original version Swcc does. It is well acknowledged that clause weighting scheme especially those with smoothing mechanisms, can significantly improve the performance of SLS algorithms for SAT [11]. These smoothed clause weighting schemes have been used in some state-of-the-art local search algorithms, such as Sparrow2011 and EagleUP [7] which are the best SLS solvers in SAT Competition 2011.

In the Swqcc algorithm, every clause is combined with a weight which is a nonnegative integer, and we use $weight(c)$ to denote clause c's weight. At the start of the algorithm, all the weights is set to 1. When the algorithm gets stuck in local optima, the weights of unsatisfied clauses are increased by 1. Moreover, Swqcc uses a smoothing mechanism to periodically smooth clause weights. The smoothing mechanism in Swqcc is simple:

- Rule 6: When the averaged clause weight $\overline{w}$ is greater than a threshold value δ, all clause weights are smoothed as $weight(c_i) := \lfloor \gamma \cdot weight(c_i) \rfloor + \lfloor (1-\gamma)\overline{w} \rfloor$, where $0 < \gamma < 1$.

5.2 The Swqcc Algorithm

As mentioned in the introduction section, the most important part of an SLS algorithm for SAT is the function for selecting the flipping variable. On the basis of the selecting function, the SLS algorithms for SAT can be divided into three main classes: GSAT, WalkSAT and Dynamic Local Search. State-of-the-art SLS solvers mix these classes in their selecting function, such as the three winners of the random satisfiable category in the SAT Competition 2011, namely Sparrow2011, Sattime2011 and EagleUP [7]. Swqcc also combines heuristics from these classes in its selecting function. We outline the algorithm Swqcc in Algorithm 1, as described below.

In the beginning, the candidate assignment α is initialized randomly. All the clause weights are initialized as 1, and the $scores$ for all variables are computed accordingly. Also, $ConfVariation(x)$ is initialized as 1 for each variable x. Then the algorithm puts the variables with $score(x) > 0$ into a candidate variable set named G. G is the candidate variable set, which is maintained during the search process(line 15 and 19), consisting of "good" variables.

After the initialization, the algorithm executes a loop until the number of search steps reaches $maxSteps$ or a satisfiable assignment is found. Swqcc switches between the greedy mode and the random mode to select a variable to be flipped. Which mode the algorithm chooses to select the flipping variable depends on G is empty or not. If G is not empty, the algorithm works in greedy mode and chooses the flipping variable from G; otherwise, the algorithm picks the flipping variable in the random mode. How these two mode work can be described as follows.

The greedy mode: The algorithm chooses a variable $x \in G$ with the greatest $score$ as the flipping variable, breaking ties by preferring the one with the greatest $ConfVariation(x)$ (line 10).

The random mode: All unsatisfied clauses' $weights$ are increased by 1. If $average\ weight\ \overline{w}$ exceeds δ, all the variables' configurations and clauses' weights are smoothed, referring to Rule 5 and 6. After that, the algorithm randomly picks an unsatisfied clause c, and chooses the variable x whose $ConfVariation(x)$ is the greatest in c as the flipping variable, breaking ties by preferring the least recently flipped one (line 12 to 17).

After picking the flipping variable, the algorithm flips the chosen variable and performs some updating work, such as updating $scores$

and $ConfVariations$ (line 18). Swqcc repeats picking and flipping a variable until it finds a satisfiable assignment α or reaches the step limit. If the algorithm finds a satisfiable assignment, it outputs the satisfiable assignment; otherwise it outputs $Unknown$.

Algorithm 1: Swqcc

```
1  Swqcc(F, maxSteps)
   Input: CNF-formula F, maxSteps
   Output: A satisfying truth assignment α of F or Unknown
2  begin
3      initialize a random assignment α;
4      initialize all weight(c) as 1 and compute socre(x) for each
       variable x;
5      initialize ConfVariation(x) as 1 for each variable x;
6      put variables with score(x) > 0 into the G set;
7      for step ← 1 to maxSteps do
8          if α satisfies F then return α;
9          if G is not empty then
10             v ← x with the greatest score(x) in G, breaking
               ties by preferring the one with the greatest
               ConfVariation(x);
11         else
12             increase all unsatisfied clauses' weights by 1;
13             if w̄ > δ then
14                 smooth ConfVariations by Rule 5 and
                   smooth clause weights by Rule 6;
15                 G ← G ∪ {y | score(y) > 0 &
                   ConfVariation(y) > 0};
16             c ← randomly choose an unsatisfied clause;
17             v ← the greatest ConfVariation(x) variable in
               clause c, breaking tie by choosing least recently
               flipped variable;
18         flip v; update ConfVariations by Rule 4 and scores;
19         G ← (G− {y | score(y) ≤ 0}) ∪
           {y | score(y) > 0 & y ∈ N(x)};
20     if α satisfies F then
21         return α;
22     else
23         return Unknown;
24 end
```

6 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of Swqcc on random 3-SAT instances. The experiments are divided into two parts. In the first part, we compare Swqcc with Swcc and Sparrow2011 on the large random 3-SAT instances from the SAT Competition 2011. In the second part, we compare Swqcc with Swcc and Sparrow2011 on some huge random 3-SAT instances according to the fixed clause length model. Finally, we conclude the results of the comparisons.

6.1 Software and Hardware

The algorithm Swqcc is implemented in C++. We set δ to 300, β to 0.3 and γ to 0.3, based on preliminary manual tuning. The code of Swcc, which is also implemented in C++, is provided by its author. We re-implement the data structure of Swcc by using arrays to replace double lists for literals and clauses, leading to a speedup of about 26%, and we use this improved version of Swcc in our experiments. We build our Swqcc solver on the top of this implementation. The code of Sparrow2011 we test for comparison is the one submitted to SAT Competition 2011 [1].

All experiments are run on 4 cores from Intel(R) Xeon(R) E7520 CPU(16 cores) with 1.87GHz and 15.7GB memory under Linux.

6.2 The Benchmarks

In our experiment, we set up two benchmarks. The first benchmark contains all large random 3-SAT instances of the SAT Competition 2011 ($2500 \leq number\ of\ variables \leq 50000$) [1]. The second benchmark is a set of formulas created according to the fixed clause length model (no tautologies, no duplicate clauses, no duplicate literals in a clause, $52000 \leq number\ of\ variables \leq 60000$).

For the first benchmark, there are 10 classes of instances and each class has 10 instances. For the second benchmark, there are 5 classes of instances and each class has 100 instances, and the size ranges from 52000 to 60000 variables in increments of 2000. All instances share the same ratio of 4.2 and are satisfiable, so they can be used to evaluate the efficiency of SLS algorithms.

6.3 Evaluating Criterion and Result Reporting Pattern

In this experiment, we adopt the evaluating criterion like the one SAT competitions use. By comparing the executing time, the number of searching steps and the success rate, we can evaluate the solvers in a direct way. Each run terminates upon finding a satisfiable assignment or exceeding the time limit which is set to 1000 seconds for the first benchmark and 1500 seconds for the second benchmark.

For the first benchmark, all solvers run 100 times on each instance and thus 1000 times for each class. For the second benchmark, all solvers run 5 times on each instance and thus 500 times for each class. We say a run is successful if the solver finds a satisfiable assignment. For each solver on each instance class, the averaged run time, averaged steps and success rate (the number of successful runs divided by the number of total runs) are reported.

6.4 Results

In this section, we discuss the performance of Swqcc, Swcc and Sparrow2011 in the experiments.

Discussion on the first benchmark: Table 1 presents the performance of Swqcc, Swcc and Sparrow2011 on large random 3-SAT instances from the SAT Competition 2011. It is clear that Swqcc significantly outperforms Swcc. Both the averaged time and the number of averaged steps of Swqcc are less than Swcc's. The success rate of Swqcc is more than Swcc's on each instance class. Moreover the difference is more significant on those instances with at least 35000 variables.

Swqcc also outperforms Sparrow2011 on success rate, averaged time and averaged steps for each class. On the classes with more than 35000 variables, Swqcc's success rates are obviously more than those of Sparrow2011. Specially, for the k3-v50000 class which is the most difficult class in the benchmark, the success rate of Swqcc is about 30% more than that of Sparrow2011. Additionally, for three instances in the k3-v5000 class, Swqcc succeeds to find satisfiable assignments in 74, 68 and 59 runs respectively, while Sparrow2011 only succeeds in 22, 14 and 16 runs separately. The gaps on success rate and averaged time can be seen clearly in Figure 1.

Table 1 Comparing Swqcc with Swcc and Sparrow2011 on the large random 3-SAT instances in SAT Competition 2011

Instance Class	Swcc			Swqcc			Sparrow2011		
	succ rate	avg time	avg steps	succ rate	avg time	avg steps	succ rate	avg time	avg steps
k3-v2500	99.6%	32.9	40,043,150	99.8%	21.8	24,886,573	99.1%	39.2	40,365,308
k3-v5000	99.5%	65.1	57,738,399	100.0%	21.9	20,595,572	100.0%	25.9	24,068,133
k3-v10000	99.2%	96.9	63,612,896	100.0%	39.2	27,627,351	99.6%	54.1	42,469,154
k3-v15000	97.8%	156.9	77,438,850	99.9%	69.4	39,087,773	99.9%	79.3	56,856,478
k3-v20000	96.3%	246.9	98,828,157	99.9%	120.7	54,318,782	99.1%	138.5	89,391,880
k3-v25000	94.1%	339.7	115,923,056	99.4%	199.8	75,328,586	96.9%	233.8	139,605,914
k3-v30000	90.6%	396.0	118,562,251	99.1%	228.5	73,637,941	94.6%	284.1	156,394,900
k3-v35000	75.7%	568.9	151,716,114	97.2%	340.2	96,665,351	85.1%	465.4	235,298,103
k3-v40000	84.1%	467.7	111,274,783	96.4%	290.4	71,742,619	85.8%	407.4	192,945,177
k3-v50000	46.9%	796.3	152,118,922	86.8%	483.1	92,103,323	58.5%	675.5	283,724,973

Table 2 Comparing Swqcc with Swcc and Sparrow2011 on the huge random 3-SAT instances according to the fixed clause length model

Instance Class	Swcc			Swqcc			Sparrow2011		
	succ rate	avg time	avg steps	succ rate	avg time	avg steps	succ rate	avg time	avg steps
k3-v52000	66.2%	1028.6	179,795,225	98.6%	411.9	73,811,429	84.6%	723.5	296,249,010
k3-v54000	60.6%	1071.2	185,385,515	98.4%	423.8	72,286,463	81.0%	792.8	317,965,296
k3-v56000	41.4%	1254.5	205,058,562	98.0%	488.6	79,500,530	78.6%	851.5	338,833,676
k3-v58000	31.4%	1316.7	211,233,037	97.2%	544.3	84,458,959	79.6%	819.0	315,199,569
k3-v60000	23.4%	1390.5	213,633,129	95.2%	606.9	90,070,893	71.0%	944.9	359,898,174

Discussion on the second benchmark: We also compare these solvers on the huge random 3-SAT instances. Seen from Table 2 and also Figure 2, Swqcc significantly outperforms Sparrow2011, which in turn significantly outperforms Swcc on these huge random 3-SAT instances. Firstly we look at the success rates. On each class of the benchmark, the success rate of Swqcc is about 20% more than that of Sparrow2011. Specially, on the most difficult class k3-v60000, the success rate of Swqcc is 95.2%, while that of Swcc and Sparrow2011 is only 23.4% and 71.0%.

Swqcc also outperforms the other two solvers in terms of averaged time and averaged steps. There is only one class on which the averaged time of Swqcc exceeds 600 seconds (which is 607 seconds exactly), while the averaged time of Sparrow2011 varies from 723.5 to 944.9 seconds and that of Swcc varies from 1028 to 1390 seconds. The averaged steps of Swqcc are dramatically less than those of Swcc and Sparrow on these huge instances.

Summarization: The experiments show that Swqcc consistently outperforms Sparrow2011 in solving random 3-SAT instances, while Sparrow2011 in turn outperforms Swcc. We believe the better performance of Swqcc is mainly attributed to the QCC strategy, since Swqcc is derived from Swcc by replacing the CC strategy with QCC.

7 DISCUSSIONS

In this section, we further investigate the QCC strategy. Specifically, we discuss the differences between CC and QCC, and study the effectiveness of the smoothing mechanism in QCC.

7.1 Differences between CC and QCC

Configuration checking is a general local search idea, and the CC and QCC strategies are both evolved from this idea. However, these two strategies have three significant differences, as described below:

- In CC, a variable's configuration refers to the truth vales of its neighboring variables, while in QCC, a variable's configuration refers to the states of the clauses in which it appears.
  We have conducted some experiments to compare Swqcc with its alternative version where the configuration is defined as in CC. The experiments show that this alternative strategy of QCC degrades the performance of Swqcc. For example, over 100 runs on the k3-v50000 group, it only achieves a success rate of 67%.
- The CC strategy only considers whether a variable's configuration has been changed; in contrast, the QCC strategy takes into account the quantitative variation of a variable's configuration.
- The CC strategy is only used as a condition for candidate variables to be flipped in the greedy mode, while QCC is used to break ties in the greedy mode and serves as the priority selecting criterion in the random mode. This is the first time the CC idea is used in random mode.
  We have conducted some experiments to compare Swqcc with its alternative version which directly picks the least recently flipped variable from the selected unsatisfied clause in random mode. The experiments show that this alternative version of Swqcc performs significantly worse than Swqcc. For example, over 100 runs on the k3-v50000 group, it only achieves a success rate of 27%.

7.2 Discussion on the Smoothing Mechanism in the Quantitative Configuration Checking

To study the effectiveness of the smoothing mechanism in QCC (Rule 5), we run the alternative version of Swqcc without Rule 5 on random 3-SAT instances from the SAT Competition 2011. This alternative version of Swqcc performs worse, especially on large instances. For example, over 100 runs on the k3-v50000 group, it only achieves a success rate of 53%, compared to that of Swqcc is 86.8%.

An alternative smoothing mechanism in the QCC strategy to smooth $ConfVariations$ based on the threshold of the mean value of all $ConfVariations$. However, it is time consuming as we should additionally maintain the mean value of $ConfVariations$, which may be updated in each step.

Figure 1 Success rate and averaged time of solvers on the large random 3-SAT instances in SAT Competition 2011

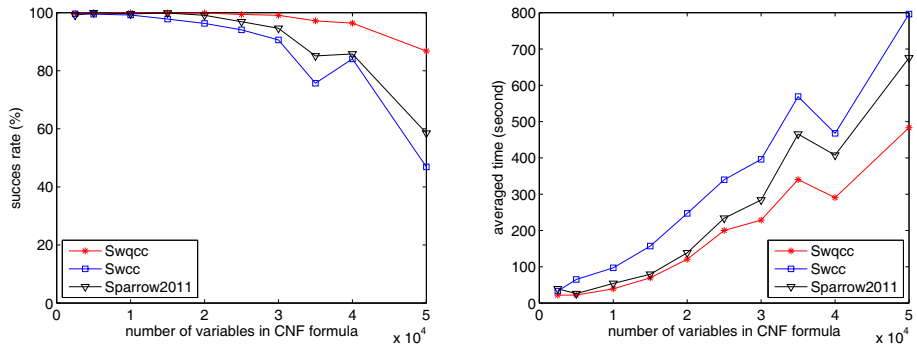
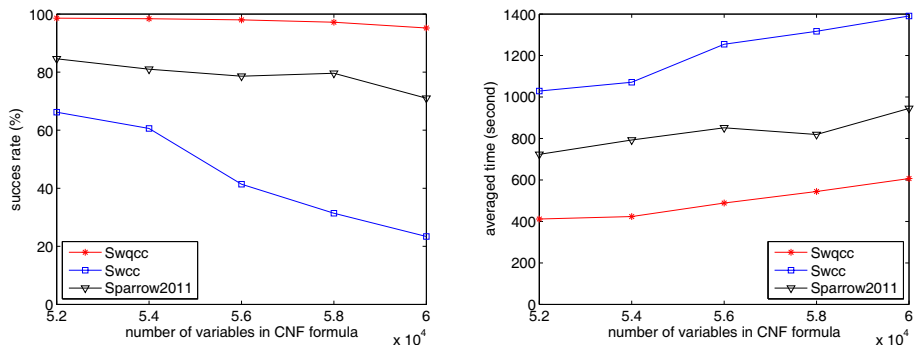


Figure 2 Success rate and averaged time of solvers on the huge random 3-SAT instances according to the fixed clause length model



8 CONCLUSION AND FUTURE WORK

We proposed a new strategy called Quantitative Configuration Checking (QCC) for local search SAT algorithms. The QCC strategy is considered as a diversification strategy. Although QCC can be seen as a quantitative version of the CC strategy, there are significant differences between the two strategies. The QCC strategy considers the quantitative variation of a variable's configuration when choosing the variable to flip, where the definition of $configuration$ of a variable is based on the states of the clauses it appears in.

We utilized the QCC strategy to derive a new SLS algorithm called Swqcc from the Swcc algorithm, and validated its effectiveness through experiments on random 3-SAT instances. By comparing Swqcc with Swcc, we showed that the QCC strategy is more effective than the CC strategy. Furthermore, our experiments showed that Swqcc outperforms the best SLS solver called Sparrow2011 in SAT Competition on random 3-SAT instances, especially those large ones.

For future work, we plan to improve Swqcc by combining other ideas such as the aspiration idea [4] used in Swcca or the probability distribution idea [2] used in Sparrow2011. We would also like to improve the QCC strategy by defining configuration in a more effective way, and test our algorithm on the random k-SAT for $k > 3$.

9 ACKNOWLEDGEMENTS

This work is partially supported by National Basic Research Program (973 Program) 2010CB328103, Australian Research Council Discovery Project DP120102489, and Australian Research Council Future Fellowship FT0991785. We would like to thank the anonymous referees for their helpful comments.

REFERENCES

[1] The SAT competition homepage. http://www.satcompetition.org.
[2] Adrian Balint and Andreas Fröhlich, 'Improving stochastic local search for SAT with a new probability distribution', in *Proc. of SAT-10*, pp. 10–15, (2010).
[3] Shaowei Cai and Kaile Su, 'Local search with configuration checking for SAT', in *Proc. of ICTAI-11*, pp. 59–66, (2011).
[4] Shaowei Cai and Kaile Su, 'Configuration checking with aspiration in local search for SAT', in *Proc. of AAAI-12*, p. To appear, (2012).
[5] Shaowei Cai, Kaile Su, and Qingliang Chen, 'EWLS: A new local search for minimum vertex cover', in *Proc. of AAAI-10*, pp. 45–50, (2010).
[6] Shaowei Cai, Kaile Su, and Abdul Sattar, 'Local search with edge weighting and configuration checking heuristics for minimum vertex cover', *Artif. Intell.*, **175**(9-10), 1672–1696, (2011).
[7] Oliver Gableske and Marijn Heule, 'EagleUP: Solving random 3-SAT using SLS with unit propagation', in *Proc. of SAT-11*, pp. 367–368, (2011).
[8] Ian P. Gent and Toby Walsh, 'Towards an understanding of hill-climbing procedures for SAT', in *Proc. of AAAI-93*, pp. 28–33, (1993).
[9] Fred Glover, 'Tabu search – part i', *ORSA Journal on Computing*, **1**(3), 190–206, (1989).
[10] Fred Glover, 'Tabu search – part ii', *ORSA Journal on Computing*, **2**(1), 4–32, (1990).
[11] Henry A. Kautz, Ashish Sabharwal, and Bart Selman, 'Incomplete algorithms', in *Handbook of Satisfiability*, 185–203, IOS Press, (2009).
[12] Chu Min Li and Wen Qi Huang, 'Diversification and determinism in local search for satisfiability', in *Proc. of SAT-05*, pp. 158–172, (2005).
[13] Wil Michiels, Emile H. L. Aarts, and Jan H. M. Korst, *Theoretical aspects of local search*, Springer, 2007.
[14] Bart Selman, Henry A. Kautz, and Bram Cohen, 'Noise strategies for improving local search', in *Proc. of AAAI-94*, pp. 337–343, (1994).
[15] Bart Selman, Hector J. Levesque, and David G. Mitchell, 'A new method for solving hard satisfiability problems', in *Proc. of AAAI-92*, pp. 440–446, (1992).