Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Improving WalkSAT for Random k-Satisfiability Problem with $k > 3$

Shaowei Cai[1,2*] and **Kaile Su**[1] and **Chuan Luo**[2]
[1]Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
[2]Key laboratory of High Confidence Software Technologies, Peking University, Beijing, China
shaoweicai.cs@gmail.com; k.su@griffith.edu.au; chuanluosaber@gmail.com

Abstract

Stochastic local search (SLS) algorithms are well known for their ability to efficiently find models of random instances of the Boolean satisfiablity (SAT) problem. One of the most famous SLS algorithms for SAT is WalkSAT, which is an initial algorithm that has wide influence among modern SLS algorithms. Recently, there has been increasing interest in WalkSAT, due to the discovery of its great power on large random 3-SAT instances. However, the performance of WalkSAT on random k-SAT instances with $k > 3$ lags far behind. Indeed, there have been few works in improving SLS algorithms for such instances. This work takes a large step towards this direction. We propose a novel concept namely $multilevel\ make$. Based on this concept, we design a scoring function called $linear\ make$, which is utilized to break ties in WalkSAT, leading to a new algorithm called WalkSATlm. Our experimental results on random 5-SAT and 7-SAT instances show that WalkSATlm improves WalkSAT by orders of magnitudes. Moreover, WalkSATlm significantly outperforms state-of-the-art SLS solvers on random 5-SAT instances, while competes well on random 7-SAT ones. Additionally, WalkSATlm performs very well on random instances from SAT Challenge 2012, indicating its robustness.

Introduction

The Boolean satisfiability (SAT) problem is a prototypical NP-complete problem, and is an important subject of study in many areas of computer science and artificial intelligence. Given a conjunctive normal form (CNF) formula, the SAT problem is to decide whether there is an assignment to its variables that satisfies the formula. Algorithms for solving SAT can be mainly categorized into two classes: complete algorithms and stochastic local search (SLS) algorithms.

SLS algorithms for SAT perform a local search of the space of truth assignments by starting with a randomly generated assignment, and then repeatedly flipping the truth value of a variable. SLS algorithms are incomplete in the sense that they cannot determine with certainty that a given formula is unsatisfiable. However, they are very efficient in solving satisfiable instances, especially the randomly generated ones. There has been much interest in studying the performance of SLS algorithms on uniform random k-SAT problems at the phase transition region, which have been cited as the hardest group of SAT problems (Kirkpatrick and Selman 1994).

Among SLS algorithms for SAT, WalkSAT (Selman, Kautz, and Cohen 1994) stands out as one of the most influential algorithms, and is still competitive with the state of the art in solving large random 3-SAT instances (Kroc, Sabharwal, and Selman 2010; Balint and Schöning 2012). However, its performance lags far behind on random k-SAT instances with $k > 3$ such as random 5-SAT and 7-SAT instances. This paper aims to improve WalkSAT for random k-SAT instances with $k > 3$.

Most previous works studying SLS algorithms on random instances focus on solving random 3-SAT ones, $e.g.$, (Selman, Kautz, and Cohen 1994; McAllester, Selman, and Kautz 1997; Hoos 2002; Li and Huang 2005; Balint and Fröhlich 2010; Cai and Su 2012; Balint and Schöning 2012), and the random 5-SAT and 7-SAT instances tested in the literature are of rather small size. Whereas SLS algorithms have exhibited great success on random 3-SAT instances, their performance on random k-SAT instances with $k > 3$ has stagnated for a long time. Recently, a few progresses such as Sparrow2011 (Balint and Fröhlich 2010) and CCASat (Cai and Su 2012), have been made in this direction. However, these improvements are limited, and are achieved at the expense of complex heuristics and careful tuning. Actually, neither Sparrow nor CCA heuristic led to better performance than that of previous solvers on random 5-SAT and 7-SAT instances when they were proposed.

This work proposes a very *simple* method that surprisingly improves WalkSAT, rendering it much more efficient than state-of-the-art solvers on random 5-SAT instances. In this sense, this work takes a *large* step towards improving SLS algorithms for random k-SAT instances with $k > 3$. An important notion in our method is the $\tau^{th}\ level\ make$ ($0 < \tau \leq k$), denoted by $make_\tau$. For a variable x, $make_\tau(x)$ measures the number of $(\tau-1)$-true clauses that will become τ-true by flipping x (here a clause is τ-true iff it has exactly τ true literals). More importantly, we combine $make_1$ (also known as the $make$ property) and $make_2$ to design a scoring function named $linear\ make$, abbreviated as $lmake$. Just by replacing the randomized tie-breaking method in WalkSAT with the $lmake$-based one, we obtain a new vari-

*Corresponding author
Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ant of WalkSAT, which is called WalkSATlm (WalkSAT with $linear\ make$). Although tie-breaking methods seem to be a relatively minor concern, the $lmake$-based tie-breaking method plays a critical role in WalkSATlm, as explained in Section "The WalkSATlm Algorithm".

Our experiments show that WalkSATlm outperforms WalkSAT by orders of magnitudes on random 5-SAT and 7-SAT instances, and is able to solve 5-SAT instances more than twice larger than those WalkSAT can handle. Furthermore, WalkSATlm dramatically outperforms state-of-the-art SLS solvers, including the winners from recent SAT competitions, on random 5-SAT instances, and competes well on random 7-SAT ones. In addition, WalkSATlm performs very well on the random k-SAT ($k > 3$) benchmark of SAT Challenge 2012, indicating its robustness. As a by-product, our results provide evidences against the conjecture that the $make$ property is of no use in focused random walk algorithms such as WalkSAT (Balint and Schöning 2012).

In the next section, we give necessary definitions and notations. Then we present a review on WalkSAT. After that, we propose the $make_\tau$ concept and the $lmake$ function, and describe the WalkSATlm algorithm. This is followed by experiments evaluating WalkSATlm. Finally we summarize our main results and give some future directions.

Definitions and Notations

Given a set of n Boolean $variables$ $\{x_1, x_2, ..., x_n\}$, a $literal$ is either a variable x or its negation $\neg x$, and a $clause$ is a disjunction of literals. A conjunctive normal form (CNF) formula $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ is a conjunction of clauses. The Boolean satisfiability (SAT) problem consists in testing whether all clauses in a given CNF formula F can be satisfied by some consistent assignment of truth values to variables. For a CNF formula F, we use $V(F)$ to denote the set of all variables that appear in F, and $r = m/n$ to denote its ($clause$-to-$variable$) $ratio$. Two variables are neighbors if and only if they share at least one clause. The neighbourhood of a variable x is $N(x) = \{y | y$ occurs in at least one clause with $x\}$.

The most well-known generation model for random SAT problems is the random k-SAT model (Achlioptas 2009). A random k-SAT formula with n variables and m clauses is a CNF formula where the clauses are chosen uniformly, independently and without replacement among all $2^k\binom{n}{k}$ non-trivial clauses of length k, $i.e.$, clauses with k distinct, non-complementary literals.

For a CNF formula F, an $assignment$ α is a mapping from $V(F)$ to $\{0, 1\}$, and it is $complete$ if it maps all variables to a Boolean value. If a literal evaluates to true under the given assignment, we say it is a $true\ literal$. Otherwise, we say it is a $false\ literal$. A clause is $satisfied$ if it has at least one true literal under the given assignment, and $unsatisfied$ otherwise.

Usually, SLS algorithms for SAT select a variable to flip according to $scoring\ functions$. A $scoring\ function$ can be a simple property or any mathematical expression with one or more properties. For a variable x, the properties $make(x)$ and $break(x)$ are the number of clauses that would become satisfied and unsatisfied by flipping x respectively, and $score(x)$ is defined as $make(x) - break(x)$, measuring the increment in the number of satisfied clauses by flipping x. $Focused\ random\ walk$ is a special SLS approach for SAT, which always selects the variable to flip from a (random) unsatisfied clause (Papadimitriou 1991), and WalkSAT is one of the most well-known focused random walk algorithms.

WalkSAT Review

In this section, we introduce the WalkSAT algorithm, which serves as the basis of our algorithm in this work. WalkSAT is one of the most influential SLS algorithms for SAT, not only because its algorithmic framework has been widely used (either directly or indirectly), but also it is still competitive with the state of the art in solving random 3-SAT instances.

Originally introduced in (Selman, Kautz, and Cohen 1994), WalkSAT applies the following variable selection scheme in each step. First, an unsatisfied clause C is selected randomly. If there exist variables with a $break$ value of 0 in clause C, $i.e.$, if C can be satisfied without breaking another clause, one of such variables is flipped (so-called *zero-damage* step). If no such variable exists, then with a certain probability p (the noise parameter), one of the variables from C is randomly selected; in the remaining cases, one of the variables with the minimum $break$ value from C is selected. Note that in WalkSAT, all ties are broken randomly. For details about WalkSAT, please refer to (Hoos and Stützle 2004) and (Kautz, Sabharwal, and Selman 2009).

Recently, there has been increasing interest in WalkSAT due to the discovery of its great power on large random 3-SAT instances. In 2004, Aurell $et\ al.$ observed that WalkSAT was far more powerful than had been appreciated (Aurell, Gordon, and Kirkpatrick 2004). Further empirical studies showed that WalkSAT (with $p = 0.567$) scales linearly in n for random 3-SAT instances with a $ratio$ up to 4.2 (Seitz, Alava, and Orponen 2005; Kroc, Sabharwal, and Selman 2010). The more recent study in (Balint and Schöning 2012) illustrated WalkSAT has extremely good performance on random 3-SAT instances ($r = 4.2$) with 5×10^5 variables.

However, compared to the good performance on random 3-SAT instances, WalkSAT performs relatively poorly on random k-SAT problems with $k > 3$, and cannot rival state-of-the-art SLS solvers. This sharp contrast motivates our work towards improving WalkSAT for random k-SAT problems with $k > 3$.

Multilevel Make

In this section, we propose the concept of $\tau^{th}\ level\ make$, denoted by $make_\tau$. Based on this concept, we design a scoring function called $linear\ make$, which incorporates two $make_\tau$ properties of different levels.

As the concept of $make_\tau$ concerns the number of true literals in clauses, we give the following definition.

Definition 1 Given a CNF formula F and α the current assignment to $V(F)$, a clause is τ-true if and only if it contains exactly τ true literals under assignment α.

Apart from 0-true clauses which are actually unsatisfied clauses, another class of clauses of special interest is the 1-true clauses. These clauses are the most unstable satisfied clauses, as they can become unsatisfied by flipping only one variable. In this sense, 1-true clauses lie on the boundary between satisfied and unsatisfied clauses.

With the definition of τ-true clauses, we give the concept of $make_\tau(x)$.

Definition 2 For a variable x, its τ^{th} $level$ $make$, denoted by $make_\tau(x)$, is the number of $(\tau-1)$-true clauses that would become τ-true by flipping x.

Note that one may define τ^{th} $level$ $break$ and τ^{th} $level$ $score$ in a similar way, but they are beyond the scope of this paper.

Apparently, $make_1$ is the commonly used $make$ property, which measures the number of unsatisfied clauses that would become 1-true clauses by flipping a variable. In this sense, the $make_\tau$ concept can be regarded as the generalization of the $make$ property. Another important $make_\tau$ property is $make_2$, as 1-true clauses are the most unstable satisfied clauses, and this property measures the number of 1-true clauses that would become 2-true by flipping a variable. Based on these two important $make_\tau$ properties, we design a scoring function named $linear$ $make$ ($lmake$), as it is a linear combination of $make_1$ and $make_2$. Specifically, for a variable x, its $lmake$ value is

$$lmake(x) = w_1 \cdot make_1(x) + w_2 \cdot make_2(x).$$

While $make_1$ encourages transformations from unsatisfied clauses to satisfied ones, $make_2$ encourages those from 1-true clauses, which can be $broken$ by one flip, to 2-true ones. Therefore, flipping a variable with a greater $make_2$ value leads to smaller $break$ values of its neighbors, which is beneficial in the latter search. This is especially the case for those algorithms preferring to flip variables with the minimum $break$, such as WalkSAT. The balance between the two objects of $lmake$ is controlled by the two weighting factors w_1 and w_2.

The WalkSATlm Algorithm

One distinguishing feature of WalkSAT is that it utilizes the $break$ property rather than $score$. The latter combines $break$ and $make$, and is used in most SLS algorithms for SAT. It is believed the $make$ property is of no use in WalkSAT. However, we significantly improve WalkSAT by the $lmake$ function. The resulting algorithm is called WalkSATlm, whose pseudo codes are shown in Algorithm 1.

WalkSATlm differs from WalkSAT only in the tie-breaking method (of choosing a variable from those with the equally minimum $break$ value). In detail, while WalkSAT breaks ties randomly, WalkSATlm does so by preferring the variable with the greatest $lmake$ value (further ties are broken randomly). It is reasonable to break ties using the $make_\tau$ properties, which capture complementary information to the $break$ property.

It might seem that the tie-breaking method is a relatively minor concern. In effect, however, it has an essential impact on the WalkSATlm algorithm. This is because when the algorithm selects a variable with the minimum $break$ value to flip, there is usually more than one such variable.

We have performed an experiment study for WalkSATlm to figure out how frequently the tie-breaking mechanism is performed in those steps where a variable with the minimum $break$ value is selected, including the zero-damaged steps. The experiment is performed with random 5-SAT and 7-SAT instances from SAT Competition 2011. Our experimental results based on 100 runs show that, the tie-breaking mechanism is performed in about 40% of such steps for 5-SAT instances with 2000 variables, and 32% for 7-SAT instances with 200 variables. Therefore, the tie-breaking mechanism plays a substantial role in the WalkSATlm algorithm.

Algorithm 1: WalkSATlm (F)

```
   Input: A CNF formula F
1  Parameters Noise parameter p ∈ [0, 1]
   Output: A satisfying assignment for F, or FAIL
2  begin
3      for i ← 1 to MAX − TRIES do
4          σ ← a randomly generated truth assignment for F;
5          for j ← 1 to MAX − FLIPS do
6              if σ satisfies F then return σ;
7              C ← an unsatisfied clause chosen at random;
8              if ∃ variable x ∈ C with break(x) = 0 then
9                  v ← x, breaking ties by preferring the one
                   with the greatest lmake value;
10             else
11                 With probability p:
12                     v ← a variable in C chosen at random;
13                 With probability 1 − p:
14                     v ← a variable in C with the minimum
                       break, breaking ties by preferring the one
                       with the greatest lmake value;
15             Flip v in σ;
16     return FAIL;
17 end
```

Experimental Evaluation

We carry out extensive experiments to test WalkSATlm on random k-SAT instances with $k > 3$. For the purpose of comparison, we run the tests with WalkSAT, and also state-of-the-art SLS solvers including the winners from SAT Competition 2011 and SAT Challenge 2012.

Benchmarks

To evaluate WalkSATlm, we set up three benchmarks.

5-SAT benchmark: which includes all large random 5-SAT instances from SAT Competition 2011 ($r = 20$, $750 \leq n \leq 2000$, 10 instances each size). Moreover, to extend the benchmark, we generate 500 random 5-SAT instances of larger sizes ($r = 20$, n ranges from 2500 to 4500 in increments of 500, 100 instances each size).

7-SAT benchmark: which includes large random 7-SAT instances from SAT Competition 2011 ($r = 85$, $150 \leq n \leq$

250, 10 instances each size). Those with 300 and 400 variables are beyond the solving ability of all the solvers and are thus not reported. To extend the benchmark, we generate 500 random 7-SAT instances ($r = 85$, n ranges from 180 to 280 in increments of 20, 100 instances each size).

SAT Challenge 2012 benchmark: which consists of all the 480 random k-SAT instances with $k > 3$ from SAT Challenge 2012. There are 120 instances for each k-SAT ($k = 4, 5, 6, 7$), which vary in both size and $ratio$.

Experimental Preliminaries

Implementation: WalkSATlm is implemented in C++, complied by g++ with the '-O3' option. The parameter setting for WalkSATlm is reported in Table 1, which is based on some preliminary experiments. We believe through more careful tuning, better settings can be found, and thus the performance of WalkSATlm can be further improved.

	w_1	w_2	p
4-SAT	3	1	$1.5 - 0.1r$
5-SAT	3	2	$1.19 - 0.04r$
6-SAT	4	3	$1.45 - 0.03r$
7-SAT	5	4	$0.972 - 0.01r$

Table 1: Parameter setting for WalkSATlm

Competitors: We compare WalkSATlm with WalkSAT, as well as five state-of-the-art SLS solvers, including Sparrow2011 (Balint and Fröhlich 2010), sattime2011 (Li and Li 2012), EagleUP (Gableske and Heule 2011), CCASat (Cai and Su 2012), and probSAT (Balint and Schöning 2012). The first three are the winners in the random satisfiable category of SAT Competition 2011, and CCASat is the winner in the random track of SAT Challenge 2012, while probSAT is one of the best SLS solvers in the recent literature.

For WalkSAT, we adopt the latest version (v50) from its author's website[1], and the noise parameter p is set to 0.25 for 5-SAT and 0.10 for 7-SAT. This setting is optimized by the iterated F-race configurator (Birattari et al. 2010), as reported in (Balint and Schöning 2012). We also tested WalkSAT with the same setting to p as in WalkSATlm, which leads to worse performance and is thus not adopted. Since we could not find any setting for 4-SAT and 6-SAT problems, and also WalkSAT performs significantly worse than other solvers on random 5-SAT and 7-SAT instances, we do not test it on SAT Challenge 2012 benchmark.

The binaries of Sparrow2011, sattime2011 and EagleUP are downloaded from the webpage of SAT competitions[2], and the binary of CCASat is downloaded online[3], while that of probSAT is provided by its author.

Hard- and Soft-ware: All the experiments are carried out on the EDACC v4.1 platform (Balint et al. 2010), using 2 cores of Intel(R) Core(TM) i7-2640M 2.8 GHz CPU and 7.8 GByte RAM, under the Ubuntu Linux Operation System.

[1]http://www.cs.rochester.edu/~kautz/walksat/Walksat_v50.zip

[2]http://www.cril.univ-artois.fr/SAT11/solvers/SAT2011-static-binaries.tar.gz

[3]http://www.shaoweicai.net/research.html

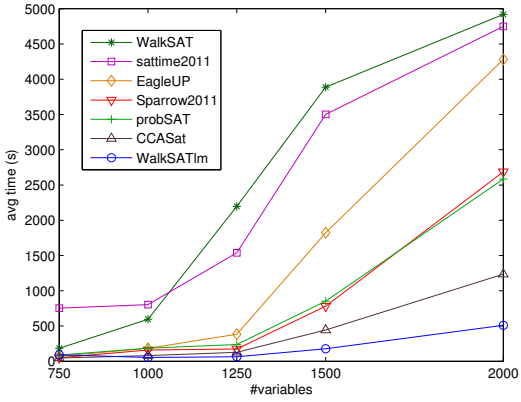

Figure 1: Averaged run time comparison on large random 5-SAT instances from SAT Competition 2011.

Evaluating Methodology: For 5-SAT and 7-SAT benchmarks, each solver is performed 10 times on each instance from SAT Competition 2011, and one time on each randomly generated instance, with a cutoff time of 5000 seconds (the same as in SAT competitions). In this way, each solver is performed 100 times for each instance class. For SAT Challenge 2012 benchmark, each solver is performed one time on each instance, with a cutoff time of 1000 seconds (close to that in SAT Challenge 2012, $i.e.$, 900 seconds). Each run terminates upon either finding a satisfying assignment or reaching the given cutoff time. As for performance metrics, we report the success rate ('suc') and the averaged run time ('time') for each instance class. The best results for an instance class are highlighted in **bold** font. If a solver has no successful run for an instance class, the corresponding 'time' is marked with "-".

Experimental Results

In this subsection, we present and discuss the comparative results of WalkSATlm and its competitors. For the first two benchmarks, we first discuss the comparison between WalkSATlm and WalkSAT, to demonstrate the effectiveness of the $lmake$-based tie-breaking method, and then we compare WalkSATlm against state-of-the-art SLS solvers.

WalkSATlm vs. WalkSAT on 5-SAT Benchmark As demonstrated in Table 2, WalkSATlm shows a dramatic improvement over WalkSAT. The performance of WalkSAT dramatically decreases with n when $n \geq 1250$, and it becomes futile for instances with $n \geq 2000$. The behavior of WalkSATlm, on the other hand, is quite different: 3000 variable instances are quite manageable, and even 4500 variable instances can be solved with 27% success rate. In respect of averaged run time, WalkSATlm is one to two orders of magnitudes faster than WalkSAT. Actually, as is clear from Figure 1, WalkSAT cannot rival state-of-the-art SLS solvers on random 5-SAT instances; on the other hand, just by replacing the tie-breaking mechanism, the improved algorithm WalkSATlm significantly outperforms them.

WalkSATlm vs. State-of-the-art Solvers on 5-SAT Benchmark Now we turn our attention to the comparison

	Sparrow2011		sattime2011		EagleUP		probSAT		CCASat		WalkSAT		WalkSATlm	
	suc	time	suc	time	suc	time	suc	time	suc	time	suc	time	suc	time
`5sat750`	100%	51	99%	754	100%	72	100%	88	100%	**47**	100%	184	100%	92
`5sat1000`	100%	159	97%	804	100%	184	100%	185	100%	81	98%	596	100%	**55**
`5sat1250`	100%	174	91%	1538	100%	384	100%	237	100%	128	71%	2197	100%	**64**
`5sat1500`	99%	781	52%	3501	88%	1823	98%	853	100%	443	36%	3888	100%	**177**
`5sat2000`	72%	2688	10%	4749	25%	4281	71%	2585	93%	1236	3%	4919	**100%**	**511**
5sat2500	72%	2925	4%	4939	7%	4789	66%	2901	88%	1762	0%	-	**100%**	**635**
5sat3000	31%	4130	0%	-	1%	4990	40%	3866	64%	3203	0%	-	**100%**	**1701**
5sat3500	8%	4747	0%	-	0%	-	6%	4888	35%	4290	0%	-	**78%**	**2171**
5sat4000	4%	4879	0%	-	0%	-	3%	4941	10%	4787	0%	-	**56%**	**3343**
5sat4500	0%	-	0%	-	0%	-	0%	-	0%	-	0%	-	**27%**	**4380**

Table 2: Experimental results on the 5-SAT benchmark based on 100 runs for each instance class, with a cutoff time of 5000 seconds. Instances from SAT Competition 2011 are indicated in `typewriter` font in the table.

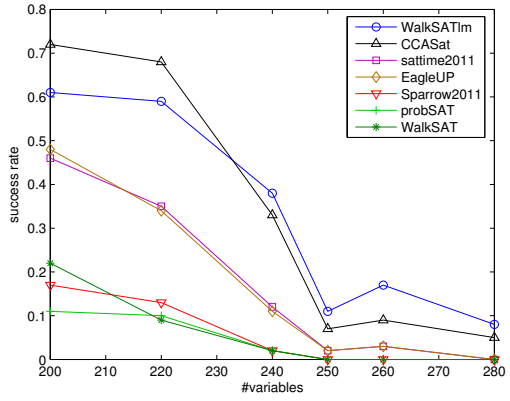

Figure 2: Success rate comparison on large random 7-SAT instances with $n \geq 200$.

between WalkSATlm and state-of-the-art SLS solvers on the random 5-SAT benchmark. Table 2 shows that WalkSATlm is the only solver that solves all instances from SAT Competition 2011 with 100% success rate. The comparative results with respect to averaged run time on these instances are summarized in Figure 1, which indicates that WalkSATlm is much faster than other solvers.

The superiority of WalkSATlm is more significant on the larger instances, whose sizes range from 2500 to 4500 variables. As we can see from Table 2, the success rates of all competitors descend steeply with the size of the instance, and decline to 10% or less on the 5sat4000 instance class. Comparatively, WalkSATlm consistently solves all the instances with up to 3000 variables, and succeeds in more than half runs on the 5sat4000 instances. In addition, it is the only solver that is able to solve some of the 4500 variable instances. To sum up, WalkSATlm significantly improves the state of the art in solving random 5-SAT instances.

WalkSATlm vs. WalkSAT on 7-SAT Benchmark Table 3 shows that WalkSATlm performs dramatically better than WalkSAT on random 7-SAT instances, and their performance gap rises rapidly with the size of the instance. For example, the ratio of the success rate of WalkSATlm to that of WalkSAT is approximately 3:1 (61%:22%) on the 7sat200 instances, and goes up to 19:1 (38%:2%) on the 7sat240 instances. Furthermore, while WalkSAT fails to solve any instance with $n \geq 250$, WalkSAT can handle 7-SAT instances with up to 280 variables. Also, the averaged run time of WalkSATlm is much less than that of WalkSAT. Note that the failed runs would have taken even longer time if they had not been stopped by a cutoff, and thus the run time gap between WalkSATlm and WalkSAT would be much larger as WalkSAT has much more failed runs.

WalkSATlm vs. State-of-the-art Solvers on 7-SAT Benchmark An obvious observation from Table 3 is that CCASat and WalkSATlm perform much better than other solvers, and thus we focus on the comparison between them. As demonstrated in Table 3 and Figure 2, the two solvers are competitive and in some sense complementary to each other on these random 7-SAT instances. For the instances with $n \leq 220$, the overall averaged success rate of CCASat is 85%, a little higher than that of WalkSATlm, *i.e.*, 78.75%. On the other hand, WalkSATlm shows superior performance on the instances with $n > 220$. On these larger instances, WalkSATlm achieves an overall averaged success rate of 18.5%, compared to 13.5% for CCASat, which suggests that WalkSATlm is promising on large 7-SAT instances.

Results on SAT Challenge 2012 Benchmark We report in Table 4 the number of solved instances and the averaged run time for each solver on the benchmark of all random k-SAT ($k > 3$) instances from SAT Challenge 2012. WalkSATlm is competitive with CCASat, and both of them significantly outperform the remaining solvers. WalkSATlm solves 357 instances, only two less than CCASat does, indicating WalkSATlm is very competitive with CCASat on this benchmark, which is also illustrated in Figure 3. Considering the random nature of the behavior of SLS algorithms, the performance of the two solvers is indistinguishable.

Note that as with WalkSAT, the performance of WalkSATlm critically depends upon the setting of the noise parameter p, and will very likely be improved with better settings. Specially, we believe the performance of WalkSATlm on random k-SAT instances with various ratios would be significantly improved if we set p in a more careful way, such as the step-linear model used for WalkSAT (Kroc, Sabharwal, and Selman 2010). However, this kind of empirical study requires a significant computational power, and is not carried out in this work. Nonetheless, our experiments do

	Sparrow2011		Sattime2011		EagleUP		probSAT		CCASat		WalkSAT		WalkSATlm	
	suc	time	suc	time	suc	time	suc	time	suc	time	suc	time	suc	time
7sat150	100%	642	100%	512	98%	445	88%	1579	100%	**232**	98%	917	100%	532
7sat180	88%	2137	96%	1066	98%	1019	78%	2239	**100%**	**644**	79%	2121	95%	1090
7sat200	17%	4562	46%	3601	48%	3625	11%	4757	**72%**	**2312**	22%	4387	61%	3090
7sat220	13%	4706	35%	4100	34%	4061	10%	4723	**68%**	**2789**	9%	4770	59%	3079
7sat240	2%	4921	12%	4689	11%	4782	2%	4951	33%	4008	2%	4846	**38%**	**3990**
7sat250	0%	-	2%	4965	2%	4976	0%	-	7%	4881	0%	-	**11%**	**4791**
7sat260	0%	-	3%	4932	3%	4985	0%	-	9%	4786	0%	-	**17%**	**4567**
7sat280	0%	-	0%	-	0%	-	0%	-	5%	4855	0%	-	**8%**	**4799**

Table 3: Experimental results on the random 7-SAT benchmark based on 100 runs for each instance class, with a cutoff time of 5000 seconds. Instances from SAT Competition 2011 are indicated in `typewriter` font in the table.

	Sparrow2011	sattime2011	EagleUP	probSAT	CCASat	WalkSATlm
#solved	268	229	251	298	**359**	357
time	585	663	615	511	**389**	396

Table 4: Results on the SAT Challenge 2012 benchmark consisting of all random k-SAT instances with $k > 3$.

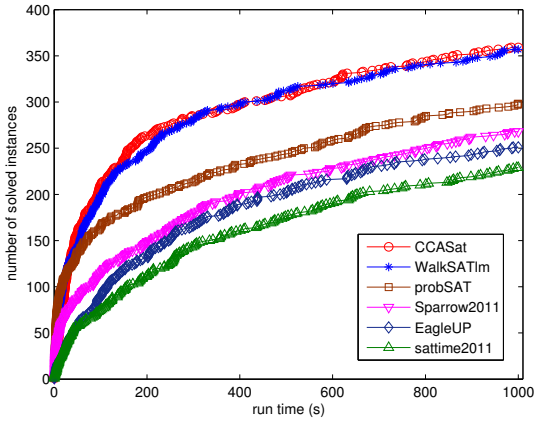

Figure 3: Comparison of run time distributions on SAT Challenge 2012 benchmark consisting of all random k-SAT instances with $k > 3$.

show promising results of WalkSATlm.

Discussions

Generally, we can define a *linear make function* as a linear combination of $make_\tau$ properties, *i.e.*, $f(x) = \sum_{\tau=1}^{k} w_\tau * make_\tau(x)$, where k is the maximum clause length of the formula, and w_τ is the weighting factor of $make_\tau$. Clearly, $lmake$ is one of such *linear make functions*. Thus it is interesting to see whether other linear make functions can also improve WalkSAT for random k-SAT problems with $k > 3$.

In what follows, we present three alternative functions, with a brief performance report for the resulting algorithms.

1. $f_1(x) = make_1(x)$: fails to solve any 5sat2000 or 7sat200 instance.
2. $f_2(x) = make_2(x)$: fails to solve any 5sat2000, but succeeds in 55 out of 100 runs on 7sat200 instances.
3. $f_3(x) = w_1 * make_1(x) + w_2 * make_2(x) + w_3 * make_3(x)$: when w_3 is set much smaller than w_1 and w_2, for example, $(w_1, w_2, w_3) = (18, 12, 1)$ for 5-SAT and $(w_1, w_2, w_3) = (10, 8, 1)$ for 7-SAT, the alternative algorithm performs similarly to WalkSATlm in terms of step performance but requires more time due to the computation of $make_3$. We did not find any setting leading to better performance than that of WalkSATlm.

These experimental results indicate that among linear make functions, $lmake$ is perhaps the best choice for improving WalkSAT. Although there might be more effective (non-linear) functions combining $make_\tau$ properties, we choose linear functions for our study, not only because of their effectiveness but also because they are so simple that we can tune the parameters easily.

Conclusions and Future Work

We proposed a new concept for SAT SLS algorithms, *i.e.*, *multilevel make*, which is a generalization of *make*. Using this concept, we designed a scoring function *linear make*, which is used to break ties in WalkSAT, leading to the WalkSATlm algorithm. The method is simple, yet surprisingly effective.

The experiments show that WalkSATlm outperforms WalkSAT by one to two orders of magnitudes on random 5-SAT and 7-SAT instances. Furthermore, WalkSATlm significantly outperforms state-of-the-art SLS solvers on random 5-SAT instances, and is very competitive with the state of the art (*i.e.*, CCASat) on random 7-SAT instances. Also, the experiments on the random benchmark of SAT Challenge 2012 demonstrate the robustness of WalkSATlm.

Given the great success and its simplicity, we believe the idea of *multilevel make*, and possibly *multilevel break* and *multilevel score*, will lead to more fruitful works. For future work, we plan to apply our method to other SLS algorithms, especially those focused random walk ones.

Acknowledgement

This work is supported by 973 Program 2010CB328103, ARC Grant FT0991785, National Natural Science Foundation of China 61073033 and 60903054, and Fundamental Research Funds for the Central Universities of China 21612414. We would like to thank the anonymous referees for their helpful comments.

References

Achlioptas, D. 2009. Random satisfiability. In *Handbook of Satisfiability*. IOS Press. 245–270.

Aurell, E.; Gordon, U.; and Kirkpatrick, S. 2004. Comparing beliefs, surveys, and random walks. In *Proc. of NIPS-04*, 49–56.

Balint, A., and Fröhlich, A. 2010. Improving stochastic local search for SAT with a new probability distribution. In *Proc. of SAT-10*, 10–15.

Balint, A., and Schöning, U. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In *Proc. of SAT-12*, 16–29.

Balint, A.; Gall, D.; Kapler, G.; and Retz, R. 2010. Experiment design and administration for computer clusters for SAT-solvers (edacc). *JSAT* 7(2-3):77–82.

Birattari, M.; Yuan, Z.; Balaprakash, P.; and Stützle, T. 2010. F-race and iterated F-race: An overview. In *Empirical Methods for the Analysis of Optimization Algorithms*. Springer. 311–336.

Cai, S., and Su, K. 2012. Configuration checking with aspiration in local search for SAT. In *Proc. of AAAI-12*, 334–340.

Gableske, O., and Heule, M. 2011. EagleUP: Solving random 3-SAT using SLS with unit propagation. In *Proc. of SAT-11*, 367–368.

Hoos, H. H., and Stützle, T. 2004. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

Hoos, H. H. 2002. An adaptive noise mechanism for WalkSAT. In *Proc. of AAAI-02*, 655–660.

Kautz, H. A.; Sabharwal, A.; and Selman, B. 2009. Incomplete algorithms. In *Handbook of Satisfiability*. IOS Press. 185–203.

Kirkpatrick, S., and Selman, B. 1994. Critical behavior in the satisfiability of random boolean formulae. *Science* 264:1297–1301.

Kroc, L.; Sabharwal, A.; and Selman, B. 2010. An empirical study of optimal noise and runtime distributions in local search. In *Proc. of SAT-10*, 346–351.

Li, C. M., and Huang, W. Q. 2005. Diversification and determinism in local search for satisfiability. In *Proc. of SAT-05*, 158–172.

Li, C. M., and Li, Y. 2012. Satisfying versus falsifying in local search for satisfiability - (poster presentation). In *Proc. of SAT-12*, 477–478.

McAllester, D. A.; Selman, B.; and Kautz, H. A. 1997. Evidence for invariants in local search. In *Proc. of AAAI-97*, 321–326.

Papadimitriou, C. H. 1991. On selecting a satisfying truth assignment. In *Proc. of FOCS-91*, 163–169.

Seitz, S.; Alava, M.; and Orponen, P. 2005. Focused local search for random 3-satisfiability. *J. Stat. Mech.* P06006.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proc. of AAAI-94*, 337–343.