Local Search for Hard SAT Formulas: The Strength of the Polynomial Law

Sixue Liu
Institute for Interdisciplinary Information Sciences
Tsinghua University

Periklis A. Papakonstantinou
Management Science and Information Systems
Rutgers University

Abstract

Random k-CNF formulas at the anticipated k-SAT phase-transition point are prototypical hard k-SAT instances. We develop a stochastic local search algorithm and study it both theoretically and through a large-scale experimental study. The algorithm comes as a result of a systematic study that contrasts rates at which a certain measure concentration phenomenon occurs. This study yields a new stochastic rule for local search. A strong point of our contribution is the conceptual simplicity of our algorithm. More importantly, the empirical results overwhelmingly indicate that our algorithm outperforms the state-of-the-art. This includes a number of winners and medalist solvers from the recent SAT Competitions.

Introduction

The propositional Satisfiability problem (SAT) is the most well-known NP-complete problem. SAT is of great theoretical and practical importance. Given a propositional formula in conjunctive normal form (CNF), the computational CNF-SAT (or SAT) problem asks to find a boolean assignment to the variables such that all clauses become true. SAT has been extensively scrutinized. Within the last two decades alone this study resulted in numerous, powerful SAT-solvers.

We propose an algorithm that falls within the stochastic local search (SLS) framework. There are two main types of SLS solvers: the greedy search and focused random walk. Greedy search solvers approach the optimal rapidly but get occasionally trapped in local optima, whereas focused random walk has the ability to recover from local optima (Kautz, Sabharwal, and Selman 2009). Most SLS solvers are a combination of these two paradigms (Cai and Su 2012). The art of building such SAT-solvers often times yields rather complicated algorithms. Note that our SLS algorithm has a very simple description. This algorithm is based on the fundamental work of focused random walk (Selman, Kautz, and Cohen 1994), including its theoretical analysis (Schöning 1999), and its follow-ups. Our SAT-solver is shown to empirically outperform the state-of-the-art SAT solvers, including the champions of SAT Competitions from 2012 to 2014 on large-scale hard instances of random formulas.

What is a hard random k-CNF? To specify a distribution over formulas we first limit the number of variables n and clauses m. Then, we sample formulas uniformly. It turns out that the ratio m/n is the important parameter. The first hardness result for random k-CNFs (Mitchell, Selman, and Levesque 1992) shows that the ratio controls the difficulty of Random-SAT in a fashion "easy-hard-easy" as the ratio grows. In (Friedgut, Bourgain, and others 1999) it was shown that there exists a *threshold* r such that if the ratio $r(n)$ for n variables is such that $r > r(n)$ then almost certainly the formula is non-satisfiable (and vice-versa). Following that (Achlioptas and Moore 2002) gave a lower bound on the threshold of random k-SAT. There is also a spade of important works following up these two. There are well-believed conjectures about the precise location of the threshold but no formal proof. In practice, the state-of-the-art SAT solvers are capable of solving random k-CNFs with millions of variables (Cai, Luo, and Su 2014) but the ratio regime where these algorithms work is not the conjectured/anticipated ratio (Mertens, Mézard, and Zecchina 2006).

Our SLS algorithm is a result of a rigorous study regarding the flipping probability of variables during SLS. More precisely, we explore the notion of critical variables. Our analysis also shows that random k-CNF at the threshold are hard for SLS algorithms, which is consistent with previous work (Iwama and Tamaki 2004). In addition to the SLS algorithm we introduce a new technology for speeding up our SAT-solver. This is dubbed as separated-non-caching implementation and is shown to further improve the efficiency of our solver by $\approx +20\%$. The same technology is shown to improve the best known implementation for WalkSAT.

Preliminaries

We introduce notation used throughout the paper and the main algorithmic framework of this work.

Notation A formula F is in *conjunctive normal form* (CNF) if $F = c_1 \bigwedge \cdots \bigwedge c_m$, where each *clause* c_i is a disjunction of *literals*, i.e. variables or its negations from $V = \{v_1, \ldots, v_n\}$. We say that F is a k-CNF formula if each c_i has at most k literals. A *truth assignment* $\alpha : V \to \{0,1\}^n$ is a function from variables to truth values $0/1$ (false/true).

We consider proper *complete* truth assignments and *partial* ones (for α a partial function). In the k-SAT problem we are given a k-CNF F and we wish to decide if F is satisfiable, i.e. whether there is α, where $\alpha(F) = 1$ (i.e. every c_i is satisfied). Given F, v, α, the *break value of* v $\mathsf{break}(v)$ is the number of clauses to be falsified in F if v is flipped in α.

The distribution of *uniform random k-CNF formulas* is defined for a given m, n: a uniform random k-CNF is constructed by choosing uniformly at random m clauses from $2^k\binom{n}{k}$ clauses. The *ratio* for this statistical k-CNF model is $r := m/n$.

Algorithmic Framework Each *Focused Random Walk* algorithm from this family of algorithms (Algorithm 1) performs a memoryless walk in the assignment space as follows: given the current truth assignment, choose an unsatisfied clause c and then use the algorithmic rule $\mathsf{pickVar}$ to choose a variable. By flipping this variable we construct the truth assignment for the next step.

The rule $\mathsf{pickVar}$ is the main object of study in this paper and there is also extensive previous work (see Introduction). The classic SLS algorithm is called WalkSAT/SKC and flips a variable v if there is one with $\mathsf{break}(v) = 0$, otherwise with probability p chooses a variable uniform randomly to flip and with the remaining probability flips a variable among those with minimal break values. We will provide theoretical and extensive experiments justifying these (at a first glance not-so-natural) choices. Then, we systematically show that there is a better alternative.

An alternative that makes fine-grained decisions is probSAT: $\mathsf{pickVar}$ chooses a variable v according to a probability distribution determined by what in this work we will call *break valuation* function $f : \mathbb{Z}^{\geq 0} \to \mathbb{R}^{\geq 0}$. This function scores break values thus inducing for fixed c a probability distribution $p(v) := \frac{f(\mathsf{break}(v))}{\sum_{v \in c} f(\mathsf{break}(v))}$. Now, every variable in a clause is associated with a distinct probability according to its break value. Variables with low break value have more chance to be flipped. It was empirically verified (Balint and Schöning 2012) that this improves dramatically over WalkSAT/SKC with the best known break valuation be the polynomial $f(x) = (\varepsilon + x)^{-\mathsf{cb}}$ for 3-SAT and the exponential $f(x) = \mathsf{cb}^{-x}$ for k-SAT ($k \geq 4$), for a constant $\mathsf{cb} > 1$. However, these complicated function forms are extremely sensitive to the parameters. For instance, certain parameterizations of the exponential function do not work for 3-SAT because the decay of probability is too strong, which means too greedy. On the other hand, the polynomial function seems that it does not fit long-clause-SAT (Balint and Schöning 2012).

Our Study We identify as a performance bottleneck the form of the break valuation f. We study $\mathsf{pickVar}$ restricted[1] as follows: (i) given only $\mathsf{break}(v_1), \dots, \mathsf{break}(v_k)$ for the k variables in c, it then (ii) determines a probability distribution p according to f, and (iii) samples according to p a variable to flip. Therefore, we consider $\mathsf{pickVar}$ parameterized by f. Our systematic study overwhelmingly favors the choice of a certain (not too large not too small) break valuation.

[1]Some restriction is needed in a systematic study. Arbitrary, $\mathsf{pickVar}$ includes one that given F first solves the whole instance.

Algorithm 1: Focused Random Walk Framework

```
  Input: CNF-formula $F$, $maxSteps$
  Output: Satisfying assignment $\alpha$ of $F$, or Unsatisfiable
1 begin
2     $\alpha \leftarrow$ random generated assignment;
3     for $step \leftarrow 1$ to $maxSteps$ do
4         if $\alpha$ satisfies $F$ then return $\alpha$;
5         $c \leftarrow$ an unsatisfied clause chosen randomly;
6         $v \leftarrow \boxed{\mathsf{pickVar}(c;\ F, \alpha)}$;
7         $\alpha \leftarrow \alpha$ with $v$ flipped;
8     return Unsatisfiable
```

Search Parameter: # of Critical Variables

Critical variables played key-role in theoretical works in SAT algorithms (Paturi et al. 2005; Hertli 2014). Here, we revisit critical variables in the context of random formulas (unlike in previous work) and use them to theoretically study the magnitude of break values (see next section). Furthermore, critical variables will be used to justify our choice for the break valuation function.

Definition 1 (critical variable)**.** *Given a satisfiable formula F, a variable x is* critical *if only one of $F|_{x=0}$ and $F|_{x=1}$ is satisfiable.*

Our experimental study (see related section) is conducted in the empirically *suggested threshold value* for the ratio m/n. For a k-CNF we denote the value by r_k. For 3,4,5,6,7-SAT we define this value to be $r_3 = 4.267$, $r_4 = 9.931$, $r_5 = 21.117$, $r_6 = 43.37$, $r_7 = 87.79$ respectively (Mertens, Mézard, and Zecchina 2006).

Lemma 1. *Given a uniform random k-CNF F with n variables and ratio r_k, there are at least $\rho_k n$* critical variables*s, where $\rho_3 = 0.82$, $\rho_4 = 0.92$, $\rho_5 = 0.96$, $\rho_6 = 0.98$, $\rho_7 = 0.99$, with probability $1 - 1/2^{\Omega(n)}$.*

Proof. Let F be a uniform random k-CNF with n variables and X the number of satisfying assignments. By Markov's inequality we have that $\Pr[X \geq c] \leq \mathbb{E}[X]/c$. We know that $\mathbb{E}[X] = 2^n \Pr[\alpha$ satisfies $F]$ where α is a random assignment. The event that α satisfies each individual clause is independent, thus $\Pr[\alpha$ satisfies $F] = \Pr[\alpha$ satisfies a random clause$]^m = (1 - 2^{-k})^m = (1 - 2^{-k})^{r_k n}$, then $\mathbb{E}[X] = (2(1 - 2^{-k})^{r_k})^n$. Finally, we have:

$$\Pr[X \geq c] \leq (2(1 - 2^{-k})^{r_k})^n / c$$

For $r_7 = 87.79$ we choose $c = 1.005^n$; note that any $c = \left((2(1 - 2^{-k})^{r_k}) + \varepsilon\right)^n$, $\varepsilon > 0$ asymptotically works. Then,

$$\Pr[X \geq 1.005^n] \leq 0.9997^n = 1/2^{\Omega(n)}$$

So the number of variables that are not critical variables is at most $\log_2 1.005^n \leq 0.0072n$, which means that more than a 0.99 fraction of variables are critical. Using the same argument for random 3-CNF to 6-CNF at suggested threshold ratios we obtain that the fractions for the critical variables are 0.82, 0.92, 0.96, and 0.98. □

That is, almost every step of the focused random walk will be dealing with critical variables.

Search Parameter: Break Value Magnitude

Our goal is to find the break valuation function that optimizes the performance of pickVar as defined in the Preliminaries. Now, we study the magnitude of break values assumed in random formulas, and in particular the role of low and zero break values.

Additional Notation Given a formula F, a truth assignment α, and a clause c, we say that $x \in c$ is a *0-break variable* if $\mathsf{break}(x) = 0$ and c is unsatisfied under α. Finally, we define the notion of critical clause (Paturi et al. 2005). Let x be a critical variable. A *critical clause* for x and an satisfying truth assignment α is a clause that its only true literal under α is x or $\bar{x}$. It is straightforward to see that there is at least one critical clause for each critical variable and any satisfying assignment.

Why biased towards low break values? Works that showed great empirical success determine break valuations that favor low break values and in particular 0-break values. Let us now theoretically justify this issue. By Lemma 1 we know that in a random formula the vast number of variables are critical. The expected number of occurrences of a variable in a uniform random k-CNF is $r_k k$. Same for the expected number of clauses containing a variable x. Thus, with probability at least $\frac{\mathsf{break}(x)}{r_k k}$, flipping a critical variable x will falsify a critical clause. Importantly, this cannot be satisfied unless x is flipped over again. This explains why previous SAT-solvers prejudiced towards lower break values in the so-called "intensification" component of the algorithm. A process that always flips variables with minimal break values gets trapped in local optima and to that end the use of probabilistic choices serves in what is commonly termed as "diversification".

0-break variables A 0-break variable is least likely to be a critical variable because flipping it does not falsify any clause. So, it is not a critical variable unless all the critical clauses are satisfied by other variables under the current assignment. Thus, it is safe to flip it.

Due to the special role of 0-break values in previous research we also study their role empirically. Since we have not concluded yet on the break valuation, we first fix a simple setting to study in isolation. Specifically, we design a generalized version of WalkSAT, outlined in Algorithm 2.

We conduct a thorough empirical study supporting further our previous theoretical observations. If lower break value deserves higher probability to be flipped, then p_0 should

Algorithm 2: pickVar for Generalized WalkSAT

```
Input: An unsatisfied clause $c$, CNF formula $F$,
       complete assignment $\alpha$
Output: A variable $v \in c$
begin
    if $\exists$ variable $x \in c$ with break($x$) = 0 then
        With probability $p_0$:
            $v \leftarrow$ a variable in $c$ chosen at random;
        With probability $1 - p_0$:
            $v \leftarrow x$;
    else
        With probability $p_1$:
            $v \leftarrow$ a variable in $c$ chosen at random;
        With probability $1 - p_1$:
            $v \leftarrow$ a variable in $c$ with minimum break;
    return $v$
```

Solvers	W0	W1	W2	W3	W4	W5
p_0	0	0.1	0.2	0.3	0.4	0.492
p_1	0.567	0.556	0.542	0.527	0.510	0.492

Table 1: Parameters of generalized WalkSAT, smaller p_0 values indicate smaller preference to 0-break variables

be less than p_1. In fact, W0 ($p_0 = 0$) and W5 ($p_0 = p_1$) are two prototypes of WalkSAT (McAllester, Selman, and Kautz 1997). We report the best p_1 corresponding to each p_0 in Table 1, p_0 ranges from 0 to p_1, as well as their performances depicted in Figure 1. The "original" marker represents the original caching implementation with XOR technology, while the "separated-non-caching" marker is under our new separated-non-caching implementation introduced in section "Separated-non-caching Technology".

As a result, W0 (choose 0-break variable if there exists one) dominates the rest even when using our novel implementation. Thus, flipping 0-break variables should be given priority in similar random walk algorithms.

Putting Everything Together: Which Break Valuation?

Considering the significance of 0-break variables, we propose the new algorithm polyLS outlined in Algorithm 3. Each step chooses 0-break variables same as WalkSAT/SKC, whereas if there is no 0-break variable it uses an inverse polynomial valuation (see previous section for justification) of the form: $f(x) = 1/g(x)$.

One of our main contributions is the choice of a very simple break valuation that significantly outperforms the state-of-the-art on hard SAT instances. In this section we theoretically justify why polynomial valuation functions outperform the exponential ones. **New notation:** $g_p(x)$ and $g_e(x)$ denote a polynomial and an exponential function respectively. The break valuation function will be $f(x) := \frac{1}{g_p(x)}$ or $\frac{1}{g_e(x)}$.

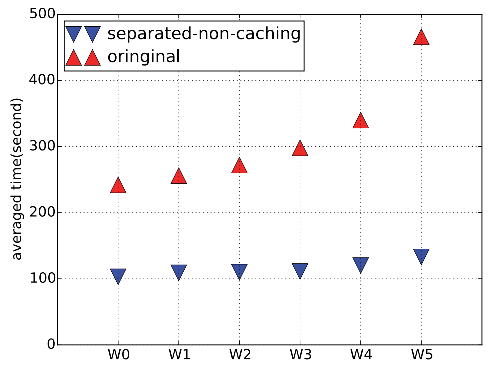

Figure 1: Performances under different parameters for random 3-SAT with ratio 4.267 from SAT Challenge 2012. W0 is the best known implementation for WalkSAT/SKC

Algorithm 3: The pickVar Function of polyLS

```
Input: An unsatisfied clause $c$, CNF formula $F$,
       complete assignment $\alpha$
Output: A variable $v \in c$
1 begin
2     if $\exists$ variable $x \in c$ with break($x$) = 0 then
3         $v \leftarrow x$;
4     else
5         foreach $v \in c$ do
6             choose $v$ and break the loop with
              probability $p = \frac{f(\text{break}(v))}{\sum_{v \in c} f(\text{break}(v))}$;
7     return $v$
```

Polynomial vs Exponential Valuations

The behavior of algorithms at the suggested threshold values is still a big open theoretical question for more than 20 years now. Our theoretical analysis is conditioned on commonly used assumptions.

In what follows we denote a clause (disjunction of literals) as a set. Let us consider the following (natural) example of a formula.

$$\{x_0, x_1, x_2\}, \{x_0, x_1, \bar{x}_2\}, \{x_0, \bar{x}_1, x_2\}, \{x_0, \bar{x}_1, \bar{x}_2\}, \quad (1)$$
$$\{\bar{x}_0, x_3, x_4\}, \{\bar{x}_0, x_3, \bar{x}_4\}, \{\bar{x}_0, \bar{x}_3, x_4\}, \quad (2)$$
$$\dots$$
$$, \{\bar{x}_0, x_{2l-1}, x_{2l}\}, \{\bar{x}_0, x_{2l-1}, \bar{x}_{2l}\}, \{\bar{x}_0, \bar{x}_{2l-1}, x_{2l}\} \quad (l)$$

Each line includes 3 clauses except the first line, there are $2l + 1$ variables and $3l + 1$ clauses in total.

Lemma 2. *Let F 3-CNF containing all $3l + 1$ clauses from (1) to (l). Then, $g_p(x)$ is strictly better than $g_e(x)$ in terms of the running time of Algorithm 3, with probability $1 - 1/2^{\Omega(l)}$ for sufficiently large l.*

Proof. The focused random walk starts with a random complete assignment, x_0 is assigned to 1 with probability $1/2$. Then, all clauses are satisfied with probability $1 - 1/4^{l-1}$. The other case is more interesting: x_0 is assigned to 0 with probability $1/2$, then all the clauses from (2) to (l) are satisfied, while there is one and only one clause in (1) that will be unsatisfied. Without loss of generality let this clause be $c = \{x_0, x_1, x_2\}$. There are 3 variables x_0, x_1, and x_2 in c. We have $\text{break}(x_1) = \text{break}(x_2) = 1$ because there is one and only one satisfied clause in (1) such that the only true literal of this clause corresponds to x_1 or x_2. Let us now consider the break value of x_0: every three clauses from (2) to (l) provide break value 1 for x_0 with probability $3/4$, there are $l - 1$ triples, then the expectation of $\text{break}(x_0)$ is $3(l - 1)/4$. By applying Chernoff bound we know the probability of $\text{break}(x_0) \geq l/2$ is $1 - 1/2^{\Omega(l)}$.

Let the break values for c be $(b, 1, 1)$, where $b \geq l/2$. If we do not choose x_0 to flip, there always exists an unsatisfied clause in (1). The only way to satisfy all the clauses is to flip x_0. The probability of flipping x_0 is $\frac{\frac{1}{g(b)}}{\frac{1}{g(b)} + 2\frac{1}{g(1)}} \approx \frac{1}{g(b)}$ (omit the constant) for sufficiently large b, thus the expected steps needed to flip x_0 is $g(b)$. *Regardless of which polynomial or exponential function* (i.e. which constants we choose to parameterize it) we choose, there is always sufficiently large l for which $g_p(b) < g_e(b)$, since $b \geq l/2$. □

Theoretical Analysis for Random Formulas

In a uniform random 3-CNF, the situation is not exactly as in our example above. The next, and most important, step is to understand what happens in a uniform random k-CNF. After all, our SLS algorithm is about random formulas. Our main theoretical result relies on the following lemma and two empirical observations.

Lemma 3. *Let a random k-CNF with sufficiently large number of variables n at ratio r_k and fix an arbitrary variable x in it. Then, with probability $1 - o(1)$ the break value of x does not change within $o(n)$ steps.*

The (easy) proof of this lemma argues by considering the average number of occurrences of each variable. Therefore, we can study how different laws behave within $o(n)$ steps.

Observation 1. *(Balint et al. 2014) During the focused random walk for uniform random k-CNF ($k = 3, 4, 5, 6, 7$) with ratio r_k, about 90% break values encountered for variables within the chosen unsatisfied clauses are less than 5, while a constant fraction of break values can reach up to $r_k k/2$.*

Our result Assuming Observation 1, given a uniform random k-CNF F at ratio r_k, there exists a polynomial $g_p(x)$ such that for any exponential $g_e(x)$, using $\frac{1}{g_p(x)}$ as the break valuation function under our framework makes in expectation a smaller number of steps to solve F than when using $\frac{1}{g_e(x)}$.

To show the contribution of a constant fraction of variables with high break values (Observation 1) to the expectation of overall steps, let us consider an unsatisfied clause $\{x_1, \dots, x_k\}$ in F, while x_1 is a critical variable assigned in a wrong way leading to unsatisfiable. Suppose there exists an x_i, $2 \leq i \leq k$, with small break value, while $\text{break}(x_1) = b$

	3-SAT	4-SAT	5-SAT	6-SAT	7-SAT
κ	2	4	5	7	7
β	-0.08	0.06	0.03	0.08	0.35

Table 2: Parameter settings of polyLS

is relatively high. Then, the probability of flipping x_1 is approximately $\frac{1}{g(b)}$. Note that the upper bound of b concentrates around $r_k k/2$ which is a constant, thus by choosing a sufficiently large s and considering s steps of focused random walk, b remains the same and x_1 is flipped within s steps with high probability (Lemma 3). Based on this we can then compare the expected number of steps required to flip x_1 under polynomial/exponential law.

Let for simplicity $g_e(b) := \alpha^b$ and $g_p(b) := b^\beta$, and $\lambda := \alpha^b/b^\beta$. There exists a λ_0, such that if $\lambda > \lambda_0$, the expectation of steps to solve F using $g_p(x)$ is smaller than that using $g_e(x)$, because there exist a constant fraction of such variables to be flip. If $\lambda \le \lambda_0$, we have $\alpha \le (\lambda_0 b^\beta)^{1/b} = (1+\varepsilon)^\beta$. Let b be large enough thus ε is a small positive number. Now, let us consider another unsatisfied clause c' with break values $(1, 2, \ldots, 2)$. Instead of this one we could have considered any not-all-equal break values, whereas c' represents the most common case according to Observation 1. The first variable with break value 1 should be flipped with highest probability. However, if we want the probability of flipping the first variable using $g_e(x)$ to be as high as that using $g_p(x)$ (this is essential because otherwise the capacity of intensification is lost), then these two probabilities have to be equal, i.e. $2^\beta/(2^\beta + k - 1) = \alpha/(\alpha + k - 1)$. This implies $\alpha = 2^\beta$, which contradicts that $\alpha \le (1+\varepsilon)^\beta$.

In order to balance the intensification and diversification, the curve of the break valuation function has to change slowly.

Our Local Search and Implementation

We propose a SAT-solver for hard random instances. This consists of (i) the algorithm and (ii) a new implementation technology we introduce. Roughly speaking, the latter is responsible for ≈20% of the speedup.

$$g(x) = (((x-1)^{\frac{\kappa}{2}} + 2)^2 + \beta)^{-1},\ \kappa \in \mathbb{Z}^+, -1 < \beta < 1$$

Here is why we chose to fit this form: (i) parameter κ decides the decay rate of the function; (ii) parameter β affects mostly the first few values of g. Although, these small x's are few the role of β is significant. Recall that in probSAT ≈90% of the flipped variables' break values are less than 4, whereas for 7-SAT is 5 (Balint et al. 2014).

We fit the parameters of polyLS following a two-step tuning mechanism using the random k-SAT benchmark from SAT Challenge 2012: first we try to fix κ by setting $\beta = 0$, then we perform validation experiments to find the best β corresponding to κ. The best parameters reported by our method are listed on Table 2.

Separated-non-caching Technology

Implementation has a notable effect on the performance of SLS. Current state-of-the-art: probSAT uses caching scheme with XOR technology (Balint et al. 2014), while WalkSAT in UBCSAT framework (Tompkins and Hoos 2005) and the latest version of WalkSATlm (Cai, Luo, and Su 2014) use non-caching implementation. The latter one also separates positive and negative literals for each variable. We take a step further building on top of the previous very influential works. We propose *separated-non-caching*, which is even more efficient. The term "separated" indicates that we have separated the non-caching process from the break value calculation, detailed can be found in the technical report (Liu 2015). We stress out that polyLS can make use of the separated-non-caching implementation to calculate break values, while probSAT cannot.

3-SAT	4-SAT	5-SAT	6-SAT	7-SAT
9.8%	10.7%	7.9%	6.5%	4.3%

Table 3: Percentage of 0-break variables among all flipping variables

Implementations	3-SAT	4-SAT
Caching without XOR	3.98	2.46
XOR-caching	4.99	2.95
Non-caching	4.84	2.78
Separated-non-caching	**5.30**	**3.07**

Table 4: Average 10^6 flips per second

Table 3 shows that for 3,4-SAT, about 10% of the flipping of polyLS only executes simple calculations to find 0-break variable, which boost efficiency. These strongly suggest that the separated-non-caching technology executes more flips within the given time, outperforming the rest on 3-SAT and 4-SAT. Table 4 shows our new technology gives a considerable speed up over the previous implementations, actually this also leads to the best known implementation of WalkSAT/SKC (W0, showed in Figure 1).

Empirical Results

We conducted large-scale experiments to evaluate polyLS on uniform random k-SAT instances sampled at the suggested threshold ratios.

The Benchmarks and Competitors

In the following we use SC to denote SAT Competition, and RSC for its random SAT track. We adopt 3 benchmark classes from SC 2013, 2014, and random generated classes named "large", details are in the result Table 5 and Table 6. We compare polyLS with state-of-the-art SAT solvers:

- Dimetheus: 1st place in RSC 2014.
- BalancedZ (Li and Huang 2005): 2nd place in RSC 2014.
- DCCASat (Luo et al. 2014): 3rd place in RSC 2014.
- probSAT (Balint and Schöning 2012): 1st place in RSC 2013, based on WalkSAT framework.
- CScoreSAT (Cai and Su 2013) is specialized in k-SAT($k > 3$).

Instance Class	BalancedZ	DCCASat	Dimetheus	probSAT	WalkSATlm	polyLS
	suc	suc	suc	suc	suc	suc
	par10	par10	par10	par10	par10	par10
3SAT-SC13	29.4%	29.2%	33.8%	28.6%	27.0%	**36.6%**
	35481	35711	33432	35834	36565	**32002**
3SAT-SC14	30.3%	24.3%	25.7%	18.0%	26.0%	**36.0%**
	35178	37916	37322	41022	37525	**32132**
3SAT-large	20.0%	25.0%	10.0%	10.0%	5.0%	**35.0%**
	40158	37584	45065	45122	47542	**32955**

Table 5: The comparative results of polyLS and their state-of-the-art competitors on the random 3-SAT threshold benchmark, 50 instances form SAT Competition 2013 and 30 instance for 2014, 20 random generated instances in 3SAT-large with 15000 variables. The best performance is achieved by polyLS, which dominates others especially in large instances. Note that probSAT and WalkSATlm are both based on focused random work

Instance Class	BalancedZ	DCCASat	Dimetheus	probSAT	WalkSATlm	CScoreSAT	polyLS
	suc	suc	suc	suc	suc	suc	suc
	par10	par10	par10	par10	par10	par10	par10
4SAT-SC13	23.8%	23.0%	28.8%	27.4%	15.8%	24.2%	**30.4%**
	38371	38667	35892	36436	42174	38158	**35015**
4SAT-SC14	9.7%	15.7%	**20.0%**	16.7%	5.3%	8.0%	**20.0%**
	45194	42416	**40109**	41782	47335	46281	40162
4SAT-large	10.0%	20.0%	25.0%	20.0%	10.0%	15.0%	**30.0%**
	45047	40191	37858	40130	45036	42652	**35263**
5SAT-SC13	19.6%	18.8%	17.6%	21.6%	15.6%	18.6%	**28.6%**
	40356	40792	41357	39394	42280	40810	**35993**
5SAT-SC14	45.0%	48.0%	49.7%	44.7%	32.7%	42.3%	**53.3%**
	27627	26551	26018	28132	34031	29253	**23926**
5SAT-large	15.0%	35.0%	10.0%	25.0%	30.0%	30.0%	**50.0%**
	42617	32642	45333	37638	35424	35396	**26012**
6SAT-SC13	29.8%	32.8%	31.0%	28.8%	29.2%	18.4%	**33.4%**
	35706	33992	34872	36167	35994	40986	**33604**
6SAT-SC14	44.3%	48.7%	49.0%	42.7%	40.3%	49.3%	**50.0%**
	28025	25920	25937	28864	29967	25590	**25246**
6SAT-large	0.0%	20.0%	10.0%	5.0%	10.0%	15.0%	**25.0%**
	50000	40309	45032	47549	45222	42582	**37662**
7SAT-SC13	45.2%	45.4%	45.4%	44.6%	41.2%	41.6%	**50.2%**
	27777	27623	27601	28020	29481	29481	**25252**
7SAT-SC14	**43.3%**	**43.3%**	42.0%	43.0%	**43.3%**	42.3%	**43.3%**
	28400	28444	29153	28670	28517	28954	28484
7SAT-large	10.0%	35.0%	15.0%	0.0%	5.0%	40.0%	**55.0%**
	45137	33201	42631	50000	47523	30853	**23780**

Table 6: Comparison on random k-SAT with $k > 3$, 50 instances form SAT Competition 2013 and 30 instance for 2014, 20 random generated instances in kSAT-large with 15000, 3000, 600, 300, 170 variables for 3,4,5,6,7SAT respectively. polyLS outperformed other state-of-the-art solvers on all the instance classes except 4SAT-SC14 and 7SAT-SC14 (but also reached the same successful rates as the best), which is much more obvious as the instance size increases. WalkSATlm and CScoreSAT are designed especially towards long clause SAT, but their performances are relatively poor

- WalkSATlm (Cai, Su, and Luo 2013): novel non-caching for 3-SAT, improved WalkSAT for others.

All the source codes we used can be downloaded from http://www.satcompetition.org/edacc/.

Evaluation Methodology

The cutoff time is set to 5000 seconds as same as in SAT Competition 2013 and 2014. Each run terminates finding a satisfying within the cutoff time is a successful run. We run each solver 10 times for each instance from SAT Competition 2013 and 2014. Thus, 500 runs for each class in SC13 and 300 runs for SC14. For large class instances we run each instance only once. We report: (i) *suc* the ratio of successful runs over total runs, (ii) *par10*, the penalized average run time (an unsuccessful run is penalized as 10 times cutoff time). The result in **bold** is the best performance for a class.

All the experiments are carried out on our machine with Intel Core Xeon E5-2650 2.60GHz CPU and 32GB memory under Linux.

Experimental Results

All details about the experiments are listed on Table 5 and Table 6, which shows that polyLS outperforms the state-of-the-art SAT solvers overwhelmingly in both successful rate and penalized average time, especially on large scale instances.

Conclusions

We have shown through a large-scale empirical study and rigorous argumentation that the use of an inverse polynomial law in SLS is very beneficial for solving hard random CNF formulas. Our empirical study showed that our elegant algorithm combined with the separated-non-caching technology clearly outperforms many state-of-the-art SAT-solvers.

Acknowledgments

Many thanks to the anonymous reviewers for the helpful comments and to Shaowei Cai for insightful discussions. This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003.

References

Achlioptas, D., and Moore, C. 2002. The asymptotic order of the random k-sat threshold. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 779–788. IEEE.

Balint, A., and Schöning, U. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In *Theory and Applications of Satisfiability Testing–SAT 2012*. Springer. 16–29.

Balint, A.; Biere, A.; Fröhlich, A.; and Schöning, U. 2014. Improving implementation of sls solvers for sat and new heuristics for k-sat with long clauses. In *Theory and Applications of Satisfiability Testing–SAT 2014*. Springer. 302–316.

Cai, S., and Su, K. 2012. Configuration checking with aspiration in local search for sat. In *AAAI*.

Cai, S., and Su, K. 2013. Comprehensive score: Towards efficient local search for sat with long clauses. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 489–495. AAAI Press.

Cai, S.; Luo, C.; and Su, K. 2014. Improving walksat by effective tie-breaking and efficient implementation. *The Computer Journal* bxu135.

Cai, S.; Su, K.; and Luo, C. 2013. Improving walksat for random k-satisfiability problem with $k > 3$. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Friedgut, E.; Bourgain, J.; et al. 1999. Sharp thresholds of graph properties, and the k-sat problem. *Journal of the American mathematical Society* 12(4):1017–1054.

Hertli, T. 2014. 3-sat faster and simpler—unique-sat bounds for ppsz hold in general. *SIAM Journal on Computing* 43(2):718–729.

Iwama, K., and Tamaki, S. 2004. Improved upper bounds for 3-sat. In *SODA*, volume 4, 328–328.

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

Li, C. M., and Huang, W. Q. 2005. Diversification and determinism in local search for satisfiability. In *Theory and Applications of Satisfiability Testing*, 158–172. Springer.

Liu, S. 2015. An efficient implementation for walksat. *arXiv preprint arXiv:1510.07217*.

Luo, C.; Cai, S.; Wu, W.; and Su, K. 2014. Double configuration checking in stochastic local search for satisfiability. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for invariants in local search. In *AAAI/IAAI*, 321–326.

Mertens, S.; Mézard, M.; and Zecchina, R. 2006. Threshold values of random k-sat from the cavity method. *Random Structures & Algorithms* 28(3):340–373.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of sat problems. In *AAAI*, volume 92, 459–465.

Paturi, R.; Pudlák, P.; Saks, M. E.; and Zane, F. 2005. An improved exponential-time algorithm for k-sat. *Journal of the ACM (JACM)* 52(3):337–364.

Schöning, U. 1999. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, 410–414. IEEE.

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

Tompkins, D. A., and Hoos, H. H. 2005. Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat. In *Theory and Applications of Satisfiability Testing*, 306–320. Springer.