Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break

Adrian Balint and Uwe Schöning

Ulm University
Institute of Theoretical Computer Science
89069 Ulm, Germany
{adrian.balint,uwe.schoening}@uni-ulm.de

Abstract. Stochastic local search solvers for SAT made a large progress with the introduction of probability distributions like the ones used by the SAT Competition 2011 winners Sparrow2010 and EagleUp. These solvers though used a relatively complex decision heuristic, where probability distributions played a marginal role.
In this paper we analyze a pure and simple probability distribution based solver probSAT, which is probably one of the simplest SLS solvers ever presented. We analyze different functions for the probability distribution for selecting the next flip variable with respect to the performance of the solver. Further we also analyze the role of *make* and *break* within the definition of these probability distributions and show that the general definition of the *score* improvement by flipping a variable, as *make* minus *break* is questionable. By empirical evaluations we show that the performance of our new algorithm exceeds that of the SAT Competition winners by orders of magnitude.

1 Introduction

The propositional satisfiability problem (SAT) is one of the most studied $\mathcal{NP}$-complete problems in computer science. One reason is the wide range of SAT's practical applications ranging from hardware verification to planning and scheduling. Given a propositional formula in conjunctive normal form (CNF) with variables $\{x_1, \ldots, x_N\}$ the SAT-problem consists in finding an assignment for the variables such that all clauses are satisfied.

Stochastic local search (SLS) solvers operate on complete assignments and try to find a solution by flipping variables according to a given heuristic. Most SLS-solvers are based on the following scheme: Initially, a random assignment is chosen. If the formula is satisfied by the assignment the solution is found. If not, a variable is chosen according to a (possibly probabilistic) variable selection heuristic, which we further call *pickVar*. The heuristics mostly depend on some score, which counts the number of satisfied/unsatisfied clauses, as well as other aspects like the "age" of variables, and others. It was believed that a good

flip heuristic should be designed in a very sophisticated way to obtain a really efficient solver. We show in the following that it is worth to "come back to the roots" since a very elementary and (as we think) elegant design principle for the *pickVar* heuristic just based on probability distributions will do the job extraordinary well.

It is especially popular (and successful) to pick the flip variable from an unsatisfied clause. This is called *focussed* local search in [11, 14]. In each round, the selected variable is flipped and the process starts over again until a solution is eventually found. Depending on the heuristic used in *pickVar* SLS-solvers can be divided into several categories like GSAT, WalkSAT, and dynamic local search (DLS).

Most important for the flip heuristic seems to be the *score* of an assignment, i.e. the number of satisfied clauses. Considering the process of flipping one variable, we get the *relative score change* produced by a candidate variable for flipping as: (*score after flipping* minus *score before flipping*) which is equal to (*make* minus *break*). Here *make* means the number of newly satisfied clauses which come about by flipping the variable, and *break* means the number of clauses which become false by flipping the respective variable. To be more precise we will denote $make(x, \mathbf{a})$ and $break(x, \mathbf{a})$ as functions of the respective flip variable x and the actual assignment $\mathbf{a}$ (before flipping). Notice that in case of focussed flipping mentioned above the value of *make* is always at least 1.

Most of the SLS solvers so far, if not all, follow the strategy that whenever the score improves by flipping a certain variable from an unsatisfied clause, they will indeed flip this variable without referring to probabilistic decisions. Only if no improvement is possible as is the case in local minima, a probabilistic strategy is performed, which is often specified by some decision procedure. The winner of the SAT Competition 2011 category random SAT, Sparrow, mainly follows this strategy but when it comes to a probabilistic strategy it uses a probability distribution function instead of a decision procedure [2]. The probability distribution in Sparrow is defined as an exponential function of the *score*. In this paper we analyze several simple SLS solvers that use only probability distributions within their search.

2 The New Algorithm Paradigm

We propose a new class of solvers here, called probSAT, which base their probability distributions for selecting the next flip variable solely on the *make* and *break* values, but not necessarily on the value of (*make* minus *break*), as it was the case in Sparrow. Our experiments indicate that the influence of *make* should be kept rather weak – it is even reasonable to ignore *make* completely, like in implementations of WalkSAT. The role of make and break in these SLS-type algorithms should be seen in a new light. The new type of algorithm presented here can also be applied for general constraint satisfaction problems and works as follows.

Algorithm 1: ProbSAT

Input : Formula F , $maxTries$, $maxFlips$
Output: satisfying assignment $\mathbf{a}$ or UNKNOWN

```
for i = 1 to maxTries do
    a ← randomly generated assignment
    for j = 1 to maxFlips do
        if (a is model for F) then
            return a
        C_u ← randomly selected unsat clause
        for x in C_u do
            compute f(x, a)
        var ← random variable x according to probability f(x,a) / Σ_{z∈C_u} f(z,a)
        flip(var)
return UNKNOWN;
```

The idea here is that the function f should give a high value to variable x if flipping x seems to be advantageous, and a low value otherwise. Using f the probability distribution for the potential flip variables is calculated. The flip probability for x is proportional to $f(x, \mathbf{a})$. Letting f be a constant function leads in the k-SAT case to the probabilities $(\frac{1}{k}, \ldots, \frac{1}{k})$ morphing the probSAT algorithm to the random walk algorithm that is theoretically analyzed in [12]. In all our experiments with various functions f we made f depend on $break(x, \mathbf{a})$ and possibly on $make(x, \mathbf{a})$, and no other properties of x and $\mathbf{a}$. In the following we analyze experimentally the effect of several functions to be plugged in for f.

2.1 An exponential function

First we considered an exponential decay, 2-parameter function:

$$f(x, \mathbf{a}) = \frac{(c_m)^{make(x,\mathbf{a})}}{(c_b)^{break(x,\mathbf{a})}}$$

The parameters are c_b and c_m. Because of the exponential functions used here (think of $c^x = e^{\frac{1}{T}x}$) this is reminiscence of the way Metropolis-like algorithms (see [14]) select a variable. We call this the *exp-algorithm*. Notice that we separate into the two base constants c_m and c_b which allow us to find out whether there is a different influence of the make and the *break* value – and there is, indeed.

It seems reasonable to try to maximize *make* and to minimize *break*. Therefore, we expect $c_m > 1$ and $c_b > 1$ to be good choices for these parameters. Actually, one might expect that c_m should be identical to c_b such that the above formula simplifies to $c^{make-break} = c^{scorechange}$ for an appropriate parameter c.

To get a picture on how the performance of the solver varies for different values of c_m and c_b, we have done a uniform sampling of $c_b \in [1.0, 4.0]$ and of $c_m \in [0.1, 2.0]$ for this exponential function and of $c_m \in [-1.0, 1.0]$ for the

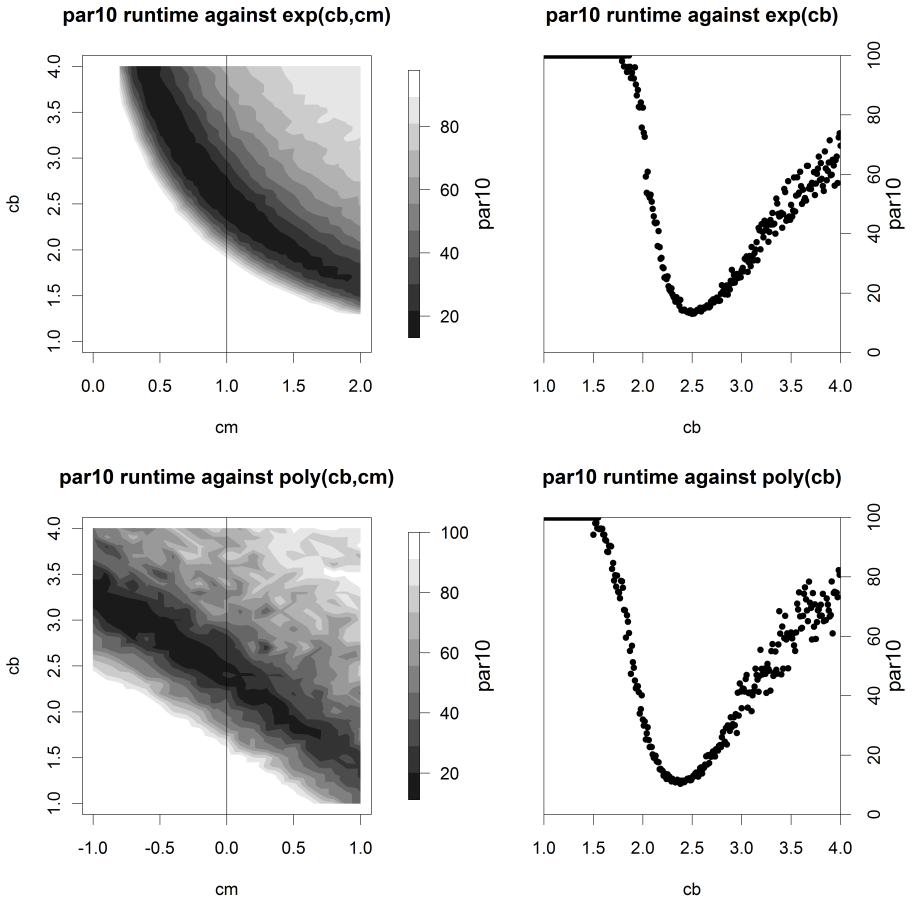

Fig. 1. Parameter space performance plot: The left plots show the performance of different combinations of c_b and c_m for the exponential (upper left corner) and the polynomial (lower left corner) functions. The darker the area the better the runtime of the solver with that parameter settings. The right plots show the performance variation if we ignore the make values (correspond to the cut in the left plots) by setting $c_m = 1$ for the exponential function and $c_m = 0$ for the polynomial function.

polynomial function (see below). We have then ran the solver with the different parameter settings on a set of randomly generated 3-SAT instances with 1000 variables at a clause to variable ratio of 4.26. The cutoff limit was set to 10 seconds. As a performance measure we use *par10*: penalized average runtime, where a timeout of the solver is penalized with 10·(cutoff limit). A parameter setting where the solver is not able to solve anything has a par10 value of 100.

In the case of 3-SAT a very good choice of the parameters is $c_b > 1$ (as expected) and $c_m < 1$ (totally unexpected), for example, $c_b = 3.6$ and $c_m = 0.5$

(see Figure 1 left upper diagram and the survey in Table 1) with a small variation depending on the considered set of benchmarks. In the interval $c_m \in [0.3, 1.8]$ the optimal choice of parameters can be described by the hyperbola-like function $(c_b - 1.3) \cdot c_m = 1.1$. Almost optimal results were also obtained if c_m is set to 1 (and c_b to 2.5), see Figure 1, both upper diagrams. In other words, the value of *make* is not taken into account in this case.

As mentioned, it turns out that the influence of make is rather weak, therefore it is reasonable, and still leads to very good algorithms – also because the implementation is simpler and has less overhead – if we ignore the make-value completely and consider the one-parameter function:

$$f(x, \mathbf{a}) \;=\; (c_b)^{-break(x,\mathbf{a})}$$

We call this the *break-only-exp-algorithm*.

2.2 A polynomial function

Our experiments showed that the exponential decay in probability with growing *break*-value might be too strong in the case of 3-SAT. The above formulas have an exponential decay in probability comparing different (say) *break*-values. The relative decay is the same when we compare $break = 0$ with $break = 1$, and when we compare, say, $break = 5$ with $break = 6$. A "smoother" function for high values would be a polynomial decay function. This led us to consider the following, 2-parameter function ($\epsilon = 1$ in all experiments):

$$f(x, \mathbf{a}) \;=\; \frac{(make(x, \mathbf{a}))^{c_m}}{(\epsilon + break(x, \mathbf{a}))^{c_b}}$$

We call this the *poly-algorithm*. The best parameters in case of 3-SAT turned out to be $c_m = -0.8$ (notice the minus sign!) and $c_b = 3.1$ (See Figure 1, lower part). In the interval $c_m \in [-1.0, 1.0]$ the optimal choice of parameters can be described by the linear function $c_b + 0.9c_m = 2.3$. Without harm one can set $c_m = 0$, and then take $c_b = 2.3$, and thus ignore the make-value completely.

Ignoring the make-value (i.e. setting $c_m = 0$) brings us to the function

$$f(x, \mathbf{a}) \;=\; (\epsilon + break(x, \mathbf{a}))^{-c_b}$$

We call this the *break-only-poly-algorithm*.

2.3 Some Remarks

As mentioned above, in both cases, the exp- and the poly-algorithm, it was a good choice to ignore the make-value completely (by setting $c_m = 1$ in the exp-algorithm, or by setting $c_m = 0$ in the poly-algorithm). This corresponds to the vertical lines in Figure 1, left diagrams. But nevertheless, the optimal choice in both cases, was to set $c_m = 0.5$ and $c_b = 3.6$ in the case of the exp-algorithm (and similarly for the poly-algorithm.) We have $\frac{0.5^{make}}{3.6^{break}} \approx 3.6^{-(break + make/2)}$. This

can be interpreted as follows: instead of the usual $scorechange = make - break$ a better score measure is $-(break + make/2)$.

The value of c_b determines the greediness of the algorithm. We concentrate on c_b in this discussion since it seems to be the more important parameter. The higher the value of c_b, the more greedy is the algorithm. A low value of c_b (in the extreme, $c_b = 1$ in the exp-algorithm) morphs the algorithm to a random walk algorithm with flip probabilities $(\frac{1}{k}, \ldots \frac{1}{k})$ like the one considered in [12]. Examining Figure 2, almost a phase-transition can be observed. If c_b falls under some critical value, like 2.0, the expected run time increases tremendously. Turning towards the other side of the scale, increasing the value of c_b, i.e. making the algorithm more greedy, also degrades the performance but not with such an abrupt rise of the running time as in the other case.

3 Experimental Analysis of the functions

To determine the performance of our probability distribution based solver we have designed a wide variety of experiments. In the first part of our experiments we try to determine good settings for the parameters c_b and c_m by means of automatic configuration procedures. In the second part we will compare our solver to other state-of-the-art solvers.

3.1 The Benchmark Formulae

All random instances used in our settings are uniform random k-SAT problems generated with different clause to variable ratios, which we denote with α. The class of random 3-SAT problems is the best studied class of random problems and because of this reason we have four different sets of 3-SAT instances.

1. 3sat1k[15]: 10^3 variables at $\alpha = 4.26$ (500 instances)
2. 3sat10k[15]: 10^4 variables at $\alpha = 4.2$ (500 instances)
3. 3satComp[16]: all large 3-SAT instances from the SAT Competition 2011 category random with variables range $2 \cdot 10^3 \ldots 5 \cdot 10^4$ at $\alpha = 4.2$ (100 instances)
4. 3satExtreme: $10^5 \ldots 5 \cdot 10^5$ variables at $\alpha = 4.2$ (180 instances)

The 5-SAT and 7-SAT problems used in our experiments come from [15]: 5sat500 (500 variables at $\alpha = 20$) and 7sat90 (90 variables at $\alpha = 85$). The 3sat1k, 3sat10k, 5sat500 and 7sat90 instance classes are divided into two equal sized classes called train and test. The train set is used to determine good parameters for c_b and c_m and the second class is used to report the performance. Further we also include the set of satisfiable random and crafted instances from the SAT Competition 2011.

	3sat1k	3sat10k	5sat500	7sat90
$exp(c_b, c_m)$	3.6 0.5	3.97 0.3	3.1 1.3	3.2 1.4
$poly(c_b, c_m)$	3.1 -0.8	2.86 -0.81	-	-
$exp(c_b)$	2.50	2.33	3.6	4.4
$poly(c_b)$	2.38	2.16	-	-

Table 1. Parameter setting for c_b and c_m: Each cell represents a good setting for c_b and c_m dependent on the function used by the solver. Parameters values around these values have similar good performance.

3.2 Good parameter setting for c_b and c_m

The problem that every solver designer is confronted with is the determination of good parameters for its solvers. We have avoided to accomplish this task by manual tuning but instead have used an automatic procedure.

As our search space is relatively small, we have opted to use a modified version of the iterated F-race [5] configurator, which we have implemented in Java. The idea of F-race is relatively simple: good configurations should be evaluated more often than poor ones which should be dropped as soon as possible. F-race uses a family Friedman test to check if there is a significant performance difference between solver configurations. The test is conducted every time the solvers have been run on an instance. If the test is positive poor configurations are dropped, and only the good ones are further evaluated. The configurator ends when the number of solvers left in the race is less than 2 times the number of parameters or if there are no more instances to evaluate on.

To determine good values for c_b and c_m we have run our modified version of F-race on the training sets 3sat1k, 3sat10k, 5sat500 and 7sat90. The cutoff time for the solvers were set to 10 seconds for 3sat1k and to 100 seconds for the rest. The best values returned by this procedure are listed in Table 1. Values for the class of 3sat1k problems were also included, because the preliminary analysis of the parameter search space was done on this class. The best parameter of the break-only-exp-algorithm for k-SAT can be roughly described by the formula $c_b = k^{0.8}$.

For the 3sat10k instance set the parameter space performance plots in Figure 2 looks similar to that of 3sat1k (Figure 1), though the area with good configurations is narrower, which can be explained by the short cutoff limit of 100 seconds used for this class (SLS solvers from the SAT Competition 2011 had an average runtime of 180 seconds on this type of instances).

In case of 5sat500 and 7sat90 we have opted to analyze only the exponential function because the polynomial function, other than in the 3SAT case, exhibited poor performance on these sets. Figure 3 shows the parameter space performance plot for the 5sat500 and 7sat90 sets. When comparing these plots with those for 3-SAT, the area with good configurations is much larger. For the 7-SAT instances the promising area seems to take almost half of the parameter space. The performance curve of the break-exp-only algorithm is also wider than that of 3-SAT and in the case of 7-SAT no clear curve is recognizable.

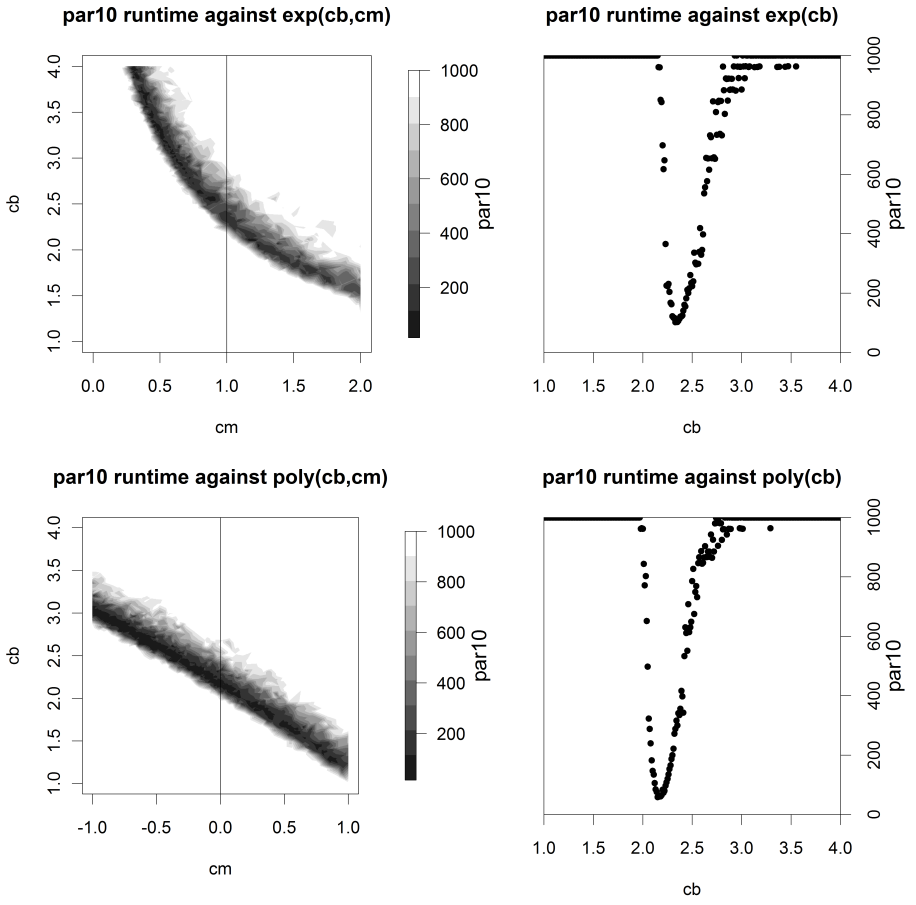

Fig. 2. Parameter space performance plot: The runtime of the solver using different function and for varying c_b and c_m on a the 3sat10k instances set.

4 Evaluations

In the second part of our experiments we compare the performance of our solvers to that of the SAT Competition 2011 winners and also to WalkSAT SKC. An additional comparison to a survey propagation algorithm will show how far our probSAT local search solver can get.

4.1 Soft- and Hardware

The solvers were run on a part of the bwGrid clusters [4] (Intel Harpertown quad-core CPUs with 2.83 GHz and 8 GByte RAM). The operating system was Scientific Linux. All experiments were conducted with EDACC, a platform that distributes solver execution on clusters [1].

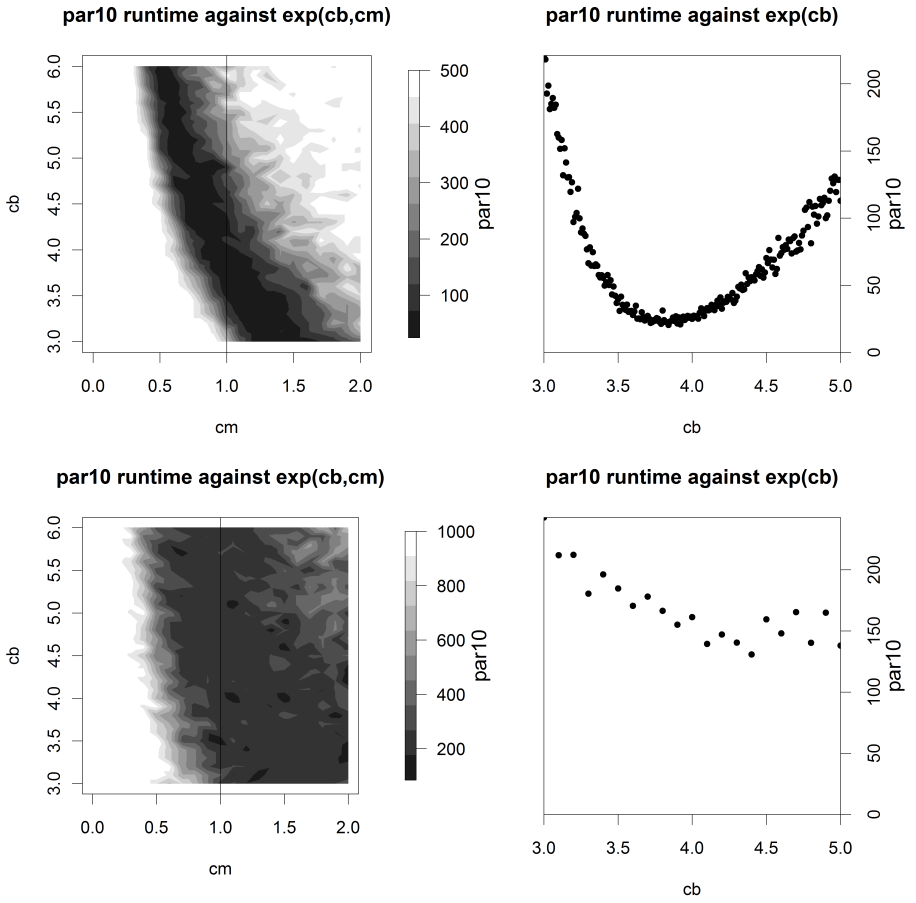

Fig. 3. Parameter space performance plot: The runtime of the exp-solvers with different functions and varying c_b and c_m on a the 5sat500 instances at the top and on the 7sat90 instances bottom.

4.2 The competitors

The WalkSAT SKC solver is implemented within our own code basis. We use our own implementation and not the original code provided by Henry Kautz, because our implementation is approximately 1.35 times faster. We have used version 1.4 of the survey propagation solver provided by Zecchina[1], which was changed to be DIMACS conform. For all other solvers we have used the binaries from the SAT Competition 2011[2].

[1] `http://users.ictp.it/~zecchina/SP/`

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

Parameter settings for competitors Sparrow is highly tuned on our target set of instances and incorporates optimal settings for each set within its code. WalkSAT has only one single parameter, the walk probability wp. In case of 3-SAT we took the optimal values for $wp = 0.567$ computed in [7]. Because we could not find any settings for 5-SAT and 7-SAT problems we have run our modified version of F-race to find good settings. For 5sat500 the configurator reported $wp = 0.25$ and for 7sat90 $wp = 0.1$. The survey propagation solver was evaluated with the default settings reported in [17].

4.3 Results

We have evaluated our solvers and the competitors on the test set of the instance sets 3sat1k, 3sat10k, 5sat500 and 7sat90 (note that the training set was used only for finding good parameters for the solvers). The parameter setting for c_b and c_m are those from Table 1 (in case of 3-SAT we have always used the parameters for 3sat10k). The results of the evaluations are listed in Table 2.

	3sat10k	3satComp	3satExtreme	5sat500	7sat90
$exp(c_b, c_m)$	46.6 (998)	93.84 **(500)**	-	12.49 (10^3)	201.68 (974)
$poly(c_b, c_m)$	46.65 (996)	76.81 **(500)**	-	-	-
$exp(c_b)$	53.02 (997)	126.59 **(500)**	-	**7.84** (10^3)	134.06 (984)
$poly(c_b)$	**22.80 (1000)**	**54.37 (500)**	1121.34 **(180)**	-	-
Sparrow2011	199.78 (973)	498.05 (498)	47419 (10)	9.52 (10^3)	**14.94** (10^3)
WalkSAT	61.74 (995)	172.21 (499)	1751.77 (178)	14.71 (10^3)	69.34 (994)
sp 1.4	3146.17 (116)	18515.79 (63)	**599.01 (180)**	5856 (6)	6000 (0)

Table 2. Evaluation results: Each cell represents the par10 runtime and the number of successful runs for the solvers on the given instance set. Results are highlighted if the solver succeeded in solving all instances within the cutoff time, or if it has the best par10 runtime. Cutoff times are 600 seconds for 3sat1k, 5sat500 and 7sat90 and 5000 seconds for the rest.

On the 3-SAT insatances, the polynomial function yields the overall best performance. On the 3-SAT competition set all of our solver variants exhibited the most stable performance, being able to solve all problems within cutoff time. The survey propagation solver has problems with the 3sat10k and the 3satComp problems (probably because of the relatively small number of variables). The good performance of the break-only-poly-solver remains surprisingly good even on the 3satExtreme set where the number of variables reaches $5 \cdot 10^5$ (ten times larger than that from the SAT Competition 2011). From the class of SLS solvers it exhibits the best performance on this set and is only approx. 2 times slower than survey propagation. Note that a value of $c_b = 2.165$ for the break-only-poly solver further improved the runtime of the solver by approximately 30% on the 3satExtreme set.

On the 5-SAT instances the exponential break-only-exp solver yields the best performance being able to beat even Sparrow, which was the best solver for

5-SAT within the SAT Competition 2011. On the 7-SAT instances though the performance of our solvers is relatively poor. We observed a very strong variance of the run times on this set and it was relatively hard for the configurator to cope with such high variances.

Overall the performance of our simple probability based solvers reaches state-of-the-art performance and can even get into problem size regions where only survey propagation could catch ground.

Scaling behavior with N The survey propagation algorithm scales linearly with N on formulas generated near the threshold ratio. The same seems to hold for WalkSAT with optimal noise as the results in [7] shows. The 3satExtreme instance set contains very large instances with varying $N \in \{10^5 \dots 5 \cdot 10^5\}$. To analyze the scaling behavior of our probSAT solver in the break-only-poly variant we have computed for each run the number of flips per variable performed by the solver until a solution was found. The number of flips per variable remains constant at about $2 \cdot 10^3$ independent of N. The same holds for WalkSAT, though WalkSAT seems to have a slight larger variance of the run times.

Results on the SAT Competition 2011 random set We have compiled an adaptive version of our probSAT solver and of WalkSAT, that first checks the size of the clauses (i.e. k) and then sets the parameters accordingly (like Sparrow2011 does). We have ran this solvers on the complete satisfiable instances set from the SAT Competition 2011 random category along with all other competition winning solvers from this category: Sparrow2011, sattime2011 and EagleUp. Cutoff time was set to 5000 seconds. We report only the results on the large set, as the medium set was completely solved by all solvers and the solvers had a median runtime under one second. As can be seen from the results of the cactus plot in Figure 4, the adaptive version of probSAT would have been able to win the competition. Interestingly is to see that the adaptive version of WalkSAT would have ranked third.

Results on the SAT Competition 2011 satisfiable crafted set We have also ran the different solvers on the satisfiable instances from the crafted set of SAT Competition 2011 (with a cutoff time of 5000 seconds). The results are listed in Table 3. We have also inculded the results of the best three complete solvers from the crafted category. The probSAT solver and the WalkSAT solver performed best in their 7-SAT break-only configuration solving 81 respectively 101 instances. The performance of WalkSAT could not be improved by changing the walk probability. The probSAT solver though exhibited better performance with $cb = 7$ and a switch to the polynomial break-only scheme, being then able to solve 93 instances. With such a high cb value (very greedy) the probability of getting stuck in local minima is very high. By adding a static restart strategy after $2 \cdot 10^4$ flips per variable the probSAT solver was then able to solve 99 instances (as listed in the table).

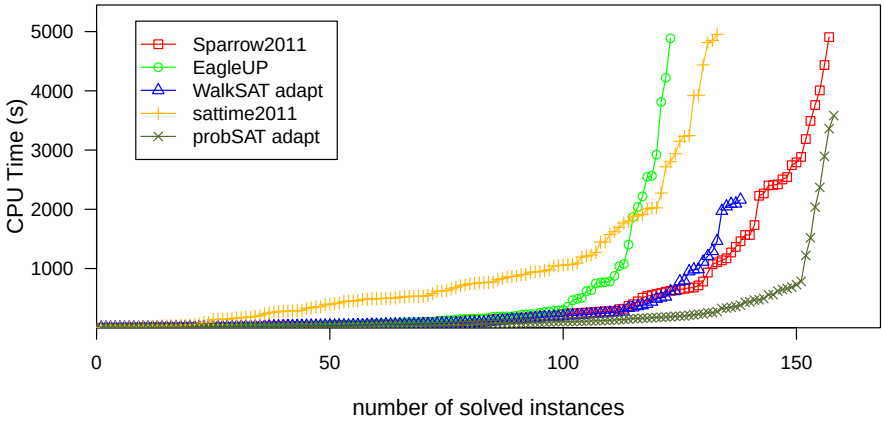

Fig. 4. Results on the "large" set of the SAT Competition 2011 random instances.

The high greediness level needed for WalkSAT and probSAT to solve the crafted instances indicates that this instances might be more similar to the 7-SAT instances (generally to higher k-SAT). A confirmation of this conjecture is that Sparrow with fixed parameters for 7-SAT instances could solve 103 instances vs. 104 in the default setting. We suppose that improving SLS solvers for random instances with large clause length would also yield improvements for non random instances.

To check weather the performance of SLS solvers can be improved by preprocessing the instances first, we have run the preprocessor of lingeling [3], which incorporates all main preprocessing techniques, to simplify the instances. The results unluckily show the contrary of what would have been expected (see Table 3). None of the SLS solvers could benefit from the preprocessing step, solving equal or less instances.

	sattime	Sparrow	WalkSAT	probSAT	MPhaseSAT (complete)	clasp (complete)	SArTagnan (complete)
Crafted	**107**	104	101	99	93	81	46
Crafted pre.	86	97	**101**	95	98	80	48

Table 3. Results on the crafted satisfiable instances: Each cell reports the number of solved instances within the cutoff time (5000 seconds). The first line shows the results on the original instances and the second on the preprocessed instances.

5 Comparison with WalkSAT

In principle, WalkSAT [10] also uses a certain pattern of probabilities for flipping one of the variables within a non-satisfied clause. But the probability distribution

does not depend on a single continuous function f as in our algorithms described above, but uses some non-continuous if-then-else decisions as described in [10].

In Table 3 we compare the flipping probabilities in WalkSAT (using the noise value 0.57) with the break-only-poly-algorithm (with $c_b = 2.3$) and the break-only-exp-algorithm (with $c_b = 2.5$) using a few examples of *break*-values that might occur within a 3-CNF clause. Even though the probabilities look very similar, we think that the small differences renders our approach to be more robust in case of 3-SAT and 5-SAT.

breaks			WalkSAT			break-only-poly			break-only-exp		
0	0	0	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33
0	0	1	0.5	0.5	0	0.46	0.46	0.08	0.42	0.42	0.16
0	1	1	1.0	0	0	0.72	0.14	0.14	0.56	0.22	0.22
0	1	2	1.0	0	0	0.79	0.15	0.06	0.64	0.26	0.1
0	2	2	1.0	0	0	0.88	0.06	0.06	0.76	0.12	0.12
1	1	1	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33
1	1	2	0.4	0.4	0.19	0.42	0.42	0.16	0.42	0.42	0.16
1	2	2	0.62	0.19	0.19	0.56	0.22	0.22	0.56	0.22	0.22
1	2	3	0.62	0.19	0.19	0.63	0.24	0.13	0.64	0.26	0.1

Table 4. Probability comparison of WalkSAT and probSAT: The first columns show some typical *break*-value combinations that occur within a clause in a 3-SAT formula during the search. For the different solvers considered here the probabilities for the each of the 3 variables to be flipped are listed.

6 Summary and Future Work

We introduced a simple algorithmic design principle for a SLS solver which does its job without heuristics and "tricks". It just relies on the concept of probability distribution and focused search. It is though flexible enough to allow plugging in various functions f which guide the search.

Using this concept we were able to discover a non-symmetry regarding the importance of the *break* and *make*-values: the *break*-value is the more important one; one can even do without the *make*-value completely.

We have systematically used an automatic configurator to find the best parameters and to visualize the mutual dependency and impact of the parameters.

Furthermore, we observe a large variation regarding the running times even on the same input formula. Therefore the issue of introducing an optimally chosen restart point arises. Some initial experiments show that performing restarts, even after a relatively short period of flips (e.g. $20N$) does give favorable results on hard instances. It seems that the probability distribution of the number of flips until a solution is found, shows some strong heavy tail behavior (cf. [9],[13]).

Plugging in the *age* property into the distribution function and analyze how strong its influence should be is also of interest.

Finally, a theoretical analysis of the Markov chain convergence and speed of convergence underlying this algorithm would be most desirable, extending the results in [12].

Acknowledgments We would like to thank the BWGrid [4] project for providing the computational resources. This project was funded by the Deutsche Forschungsgemeinschaft (DFG) under the number SCHO 302/9-1. We thank Daniel Diepold and Simon Gerber for implementing the F-race configurator and providing different analysis tools within the EDACC framework. We would also like to thank Andreas Fröhlich for fruitful discussions on this topic.

References

1. Balint, A. et al: EDACC - An advanced Platform for the Experiment Design, Administration and Analysis of Empirical Algorithms In: *Proceedings of LION5*, pages 586–599.
2. Balint, A., Fröhlich, A.: Improving stochastic local search for SAT with a new probability distribution. *Proceedings of SAT 2010*, pages 10–15, 2010.
3. Biere, Armin: Lingeling and Friends at the SAT Competition 2011. Tehnical report 11/1, FMV Reports Series, http://fmv.jku.at/papers/Biere-FMV-TR-11-1.pdf
4. bwGRiD (http://www.bw-grid.de), member of the German D-Grid initiative, funded by the Ministry for Education and Research (Bundesministerium für Bildung und Forschung) and the Ministry for Science, Research and Arts Baden-Württemberg (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg)
5. M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle (2010). F-race and iterated F-race: An overview. Empirical Methods for the Analysis of Optimization Algorithms, pp. 311-336, Springer, Berlin, Germany.
6. Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In: *Proceedings of AAAI 2002*, 635–660 (2002)
7. Kroc, L., Sabharwal, A., Selman, B.: An empirical study of optimal noise and runtime distribution in local search. In: *Proceedings of SAT 2010*, pages 346-351, 2010.
8. Li, C.M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. *SAT 2005*. LNCS, vol. 3569, 158–172. Springer, Heidelberg (2005)
9. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. *Information Proc. Letters* 47 (1993) 173–180.
10. McAllester, D., Selman, B., Kautz, H.: Evidence for invariant in local search. In: *Proceedings of AAAI-97*, pages 321-326, 1997
11. Papadimitriou, C.H.: On selecting a satisfying truth assignment. *Proceedings FOCS 1991*, IEEE, 163–169.
12. Schöning, U.: A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: *Proceedings FOCS 1999*, IEEE, 410–414.
13. Schöning, U.: Principles of stochastic local search. *Unconventional Computation 2007*. LNCS, Vol. 4618, 178–187.
14. Seitz, S., Alava, M., Orponen, P.: Focused local search for random 3-satisfiability. arXiv:cond-mat/051707v1 (2005)
15. Tompkins, D.A.D, Balint, A., Hoos, H.H: Captain Jack: New variable selection heuristics in local search for SAT. *Proceedings of SAT 2011*, pages 302–316, 2011.
16. The SAT Competition Homepage: `http://www.satcompetition.org`
17. Survey propagation: an algorithm for satisfiability. A. Braunstein, M. Mezard, R. Zecchina, Random Structures and Algorithms 27, 201-226 (2005)