Backbone Guided Local Search for Maximum Satisfiability*

Weixiong Zhang, Ananda Rangan and Moshe Looks
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130, USA
Email: zhang@cse.wustl.edu

Abstract

Maximum satisfiability (Max-SAT) is more general and more difficult to solve than satisfiability (SAT). In this paper, we first investigate the effectiveness of Walksat, one of the best local search algorithms designed for SAT, on Max-SAT. We show that Walksat is also effective on Max-SAT, while its effectiveness degrades as the problem is more constrained. We then develop a novel method that exploits the backbone information in the local minima from Walksat and applies the backbone information in different ways to improve the performance of the Walksat algorithm. We call our new algorithm backbone guided Walksat (BGWalksat). On large random SAT and Max-SAT problems as well as instances from the SATLIB, BGWalksat significantly improves Walksat's performance.

1 Introduction

Satisfiability or *SAT is* an archetypical combinatorial problem. A SAT involves a set of Boolean variables and a set of clauses in the form of a disjunction of literals (variables or their negations). A clause is satisfied if one of its literals is set to true, and the problem is satisfied if all clauses are satisfied. The problem is to decide if a variable assignment exists that satisfies all the clauses. When not all clauses are satisfiable, the goal is to maximize the satisfied clauses, and the problem becomes *maximum SAT or Max-SAT,* an optimization problem. A SAT (Max-SAT) with *k* literals per clause is denoted as k-SAT (Max-k-SAT). It is known that k-SAT with *k* greater than two is NP-complete and Max-k-SAT with *k* being at least two is NP-hard [7]. Max-SAT is more general and difficult to solve than SAT. Many real-world problems, such as scheduling, pattern recognition, and multi-agent cooperation and coordination [3; 6], can be formulated and solved as SAT or Max-SAT.

Recent years have witnessed significant progress made on SAT and Max-SAT along two directions. The first is the understanding of properties such as phase transitions and backbones [2; 12; 13; 17; 19]. Phase transitions refer to dramatic changes in a problem property when a critical parameter of a problem changes. The most important phase transition results on random 3-SAT are that when the ratio of the number of clauses to the number of variables (C/V ratio) increases beyond a critical value, around 4.3, random 3-SATs begin to become unsatisfiablc, and the computational cost of the problem increases abruptly [2; 12; 13]. The backbone of a problem is a set of variables that have fixed values in all optimal solutions to the problem. The hardness of a SAT or Max-SAT is determined by the backbone size of the problem [13]. In addition, the backbones of random SAT and Max-SAT also have small to large phase transitions [19]. The second direction is focused on developing efficient SAT solvers. The most noticeable results along this line are the Walksat local search algorithm [16] and its variants [11], which are among the best SAT solvers used in practice for problems such as planning.

A problem of fundamental interest and practical importance in algorithm design is how to utilize problem structure properties such as phase transitions and backbones to cope with high complexity and improve algorithm performance. The current research on this problem is limited. [15] developed a transformation method to exploit phase transitions of tree search. [5] designed a method to incorporate estimated backbone variables in a systematic search for SAT. [18] proposed a heuristic backbone sampling method for generating initial assignments for a local search using an estimated backbone. This sampling scheme will also be studied in this research.

In this paper, we develop a novel method to exploit the structure of local minima reached by an effective local search algorithm, and an effective approach to utilize the structure information to improve the performance of a local search algorithm. We focus on Max-SAT in this research.

To develop our new method, we first study the performance of Walksat, which was designed for SAT, on Max-SAT (Section 3). We show that Walksat is effective on random Max-SAT, finding many high quality local minima close to optimal. In addition, the local minima reached by Walksat appear to form large clusters around optimal solutions. Our results also show that the performance of Walksat degrades when the constrainedness of Max-SAT increases. Although we carry out this analysis for the purpose of developing a new method, the result is of interest and significance on its own. To our knowledge, Walksat has only been analyzed on some overconstrained Steiner tree problems [10].

The main contribution of this paper is an innovative method that exploits the solution structure of a combinatorial problem to improve the performance of a local search algorithm such as Walksat (Section 4). It was inspired by the recent research on phase transitions and backbones of SAT and Max-SAT [13;

*This research was supported in part by NSF grants IIS-0196057 and ITR/EIA-0113618, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897.

19] and the performance results of Walksat in the first part of the paper. We directly apply the new method to Walksat. Nevertheless, the ideas and techniques developed here are general and applicable to other combinatorial problems and local search algorithms.

Briefly, our main ideas are to use backbone information to make biased moves in a local search, and use local minima to approximate optimal solutions for extracting backbone information. The frequencies of literals appearing in all local minima are called *pseudo-backbone frequencies*. Our new approach applies pseudo-backbone frequencies as a heuristic for selecting a variable to flip in each step of the Walksat algorithm. Such biased moves guide the search toward regions possibly having optimal solutions. The effectiveness of this method is partially due to the effectiveness of Walksat on finding optimal and high quality approximate solutions. We call the resulting algorithm *backbone guided Walksat* or BGWalksat. We demonstrate and evaluate experimentally the effectiveness of BGWalksat on large random Max-SAT instances and real problems from SATLIB [9] (Section 5).

2 The Walksat Algorithm

Walksat is a randomized algorithm, it starts by creating an initial random variable assignment (which we call assignment generation). It then repeatedly makes a move by selecting a variable and flipping its value from True' to False or vice versa, until it finds a satisfying assignment or reaches a predefined maximal number of flips. Each such attempt is called a *try* or *restart.* The procedure repeats until a maximal number of tries has been executed.

To select a variable, the effect of flipping a variable is assessed. Flipping a variable may make some unsatisfied clauses satisfied. The number of clauses being made unsatisfied by flipping a variable is call the *break-count* of the variable at the current assignment. Walksat attempts to flip a variable with zero break-count, trying not to make the next assignment worse than the current. To find a variable with zero break-count, Walksat first selects an unsatisfied clause *C, randomly,* from all unsatisfied clauses. This is called clause pick. If *C* has a variable of zero break-count, Walksat then picks such a variable, *randomly,* from the ones that qualify. If no zero break-count variable exists in *C,* Walksat then makes a random choice. With probability *p* it chooses, *randomly,* a variable from the ones involved in *C* (called noise pick); and with probability 1 — *p* it picks a variable with the least break-count, breaking the tie arbitrarily if multiple choices exist (called greedy pick).

A flow chart of one try of Walksat is in Figure 1. The four shaded rectangles contain the random choices we will analyze. Walksat takes three parameters to run, the number of tries, the maximal number of flips in each try or *cutoff parameter,* and a probability or *noise ratio* for noise pick.

3 Performance of Walksat

We now turn to the effectiveness of Walksat on Max-SAT, particularly Max-3-SAT. We measure a local minimum in two ways. The first is the cost difference between a local minimum and an optimal solution (which may have non-zero cost). The second measure is the Hamming distance between a local minimum and the optimal solution nearest to it. We introduce the nearest Hamming distance to capture the minimal number of flips required to turn a local minimum into an optimal solution. Furthermore, to make these two quality measures on

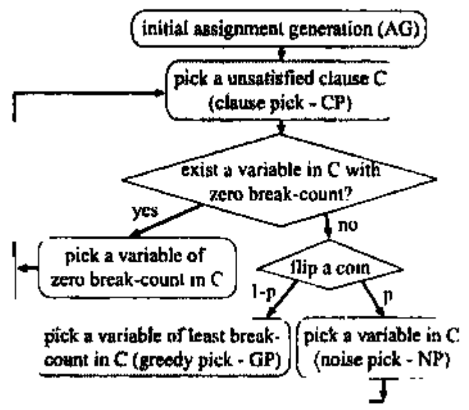
initial assignment generation (AG)
pick a unsatisfied clause C (clause pick - CP)
exist a variable in C with zero break-count?
yes
no
pick a variable of zero break-count in C
flip a coin
1-p
p
pick a variable of least break-count in C (greedy pick - GP)
pick a variable in C (noise pick - NP)

Figure 1: Main operations in a try of Walksat.

two Max-SAT instances of different sizes comparable, we normalize them. For a given problem, we divide the Hamming distance by the number of variables, resulting in the Hamming distance per variable, and divide the cost difference by the number of clauses, giving the cost difference per clause. We then use these two normalized quality measures to construct the landscape of local minima reached by Walksat.

To find the nearest optimal assignment, all optimal solutions are required, which are found by an extended DPLL algorithm [4] for SAT. Since finding all optimal solutions is expensive, we restricted ourselves to relatively small Max-SAT problems. In our experiments, we used problems of 100 variables and C/V ratios of 2.0, 4.3, 6.0 and 8.0 to capture problems in different constrainedness regions. The problems were randomly generated by uniformly picking three literals without replacement for a clause, discarding duplicate clauses. For Walksat, we set the noise ratio at 0.5, the number of tries per problem at 100, and the cutoff parameter at 10,000. We generated 1,000 problems at each C/V ratio.

The quality of local minima from Walksat are summarized in Figure 2. The X-Y planes show the correlation between the normalized Hamming distance and the normalized cost difference. The origins of the figures correspond to global minima. Each point on the X-Y plane represents a possible local minimum that may be visited by Walksat. The vertical Z axis measures the number of local minima reached by Walksat, averaged over 1,000 instances on each of which 100 tries were run.

As shown in Figures 2(a) and (b), Walksat performs well on underconstrained and critically constrained problems, in that it can find global minima very often. This is shown by the points on Z axes indicated by the arrows in the figures. Therefore, Walksat is effective at finding optimal solutions on underconstrained and critically constrained problems. However, the number of local minima which are also global minima decreases from 66,677 to 8,616, on average, as the C/V ratio increases from 2.0 to 4.3, indicating that the effectiveness of Walksat decreases. This number decreases further to 201 and 0 on overconstrained problems with C/V ratios equal to 6.0 and 8.0 (Figures 2(c) and (d)). This result indicates that Walksat becomes less effective in finding optimal solutions as problem constraints increase.

The distribution of local minima of Walksat exhibits a *bell surface* on overconstrained problems (Figures 2(c) and (d)). More importantly, the summit of such a bell surface shifts away

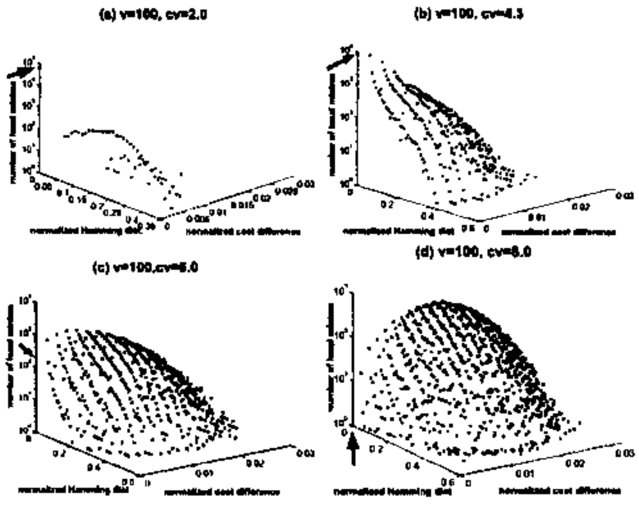

Figure 2: Landscape of local minima from Walksat.

from optima] solutions, the (0,0) point on the X-Y plane, as the problems are more constrained. As the C/V ratio increases, the number of optimal solutions decreases [19], which can become indistinguishable from a large number of suboptimal solutions. As a result, the search space becomes rugged, with more local minima in which Walksat may be trapped.

Despite the difficulty of overconstrained problems, Walksat is still fairly effective in that it is able to reach local minima close to optimal solutions. For a C/V ratio of 8.0 (Figure 2(d)), for instance, a large number of local minima landed on by Walksat (the peak point of the bell surface) have a normalized cost difference to optimal solutions of 0.014 for 100 variable problems, which is equivalent to approximately eleven more constraints violated than an optimal solution on such overconstrained problems. Since Walksat is normally executed in multiple tries, the best local minimum it can reach is usually much better than such most frequent local minima.

A closer examination of the results in Figures 2(c) and (d) indicates a nearly linear correlation between the cost difference and the Hamming distance. This observation implies that a local minimum with a small cost is more likely to share a larger common structure with an optimal solution. A similar phenomenon on the Traveling Salesman Problem was called the "big valley" [1]. We will exploit this in our new search algorithm described in the next section.

4 Backbone Guided Local Search

4.1 The main idea

If all optimal solutions to a problem were known, they could provide a useful clue to how a variable should be set. For example, if a variable is a part of the backbone, i.e., it has a fixed value in all optimal solutions, obviously the variable should be set to that value. In fact, we can extend the concept of backbone to backbone frequency. The frequency of a variable-value pair (or literal in SAT) in all optimal solutions is an indication of how often the variable should take that particular value. This can be exploited as a heuristic for selecting variables and values in a local search such as Walksat. In general, a variable-value pair will be less likely to be swapped out of the current state if it has a higher backbone frequency than another pair.

Unfortunately, exact backbone frequencies are hard to come by without finding at least one optimal solution. The idea to bypass this problem was inspired by the effectiveness of high performance local search algorithms, such as Walksat. Our analysis on the performance of Walksat (Section 3) showed that Walksat can find optimal solutions very often on underconstrained and critically constrained Max-SAT, and can also reach near optimal solutions very close to the optimal. Such near optimal local minima also form large clusters around optimal solutions. Thus, the local minima of Walksat must have common structures, which may also be shared by optimal solutions. Therefore, we can use the local minima as if they were optimal solutions to compute *pseudo-backbone frequencies* as an estimation of the true backbone frequencies. We call this general approach *backbone guided local search* or BGLS.

The above ideas can be applied to almost all local search algorithms. However, actual application is problem and algorithm specific, since local search algorithms operate very differently on different problems. In this research, we consider Max-SAT and the Walksat algorithm.

4.2 Backbone guided Walksat: BGWalksat

The application of BGLS to Max-SAT and Walksat leads to the backbone guided Walksat (BGWalksat) algorithm. Similar to Walksat, BGWalksat runs a total *K* tries. In addition, BG-Walksat splits the total of *K* tries into two phases. The first, estimation phase, runs K_1 tries of original Walksat to collect local minima and compute pseudo-backbone frequencies (see Section 4.3). The second, backbone guided search phase runs $K_2 = K - K_1$ tries, in which variable selection is guided by pseudo-backbone frequencies. The newly discovered local minima can also be added to the pool of all local minima to update the pseudo-backbone frequencies.

We now consider backbone guided moves or biased moves. As discussed in Section 2 and shown in Figure 1, Walksat makes five uniformly random choices. The first is the random initial assignment generation. Instead of uniformly randomly choosing a variable and setting it to an arbitrary value, we can view the pseudo-backbone frequencies as an approximate probability distribution of variable assignments and take samples of variable assignments as initial assignments from this distribution. This scheme was called heuristic backbone sampling in [18].

The second random choice in Walksat is the random clause pick for choosing an unsatisfied clause. To apply the idea of backbone frequencies, we extend it to the concept of *pseudo-backbone clause frequencies*, which measure the frequencies that clauses are satisfied in all local minima. When picking a clause from the set of unsatisfied clauses, we choose one based on its pseudo-backbone clause frequency. For example, given three unsatisfied clauses C_1, C_2 and C_3 with pseudo-backbone frequencies 0.3,0.05 and 0.15, respectively, we choose C_1 with probability $0.3/(0.3+0.05+0.15) = 0.6, C_2$ with probability $0.05/0.5 = 0.1$ and C3 with probability $0.15/0.5 = 0.3$.

The remaining three random choices involved selecting a variable, uniformly, from a particular set. The first choice picks a variable from the set of zero break-count variables in the chosen clause *C* (the unshaded rectangle in Figure 1), the second directly from the variables involved in C, and the third from the set of variables in *C* with the least break-count. Rather than picking one variable uniformly from a particular set, we make a biased selection based on variables' pseudo-backbone frequencies. Consider an example of three literals $x_1, \neg x_2$ and x_3 in the chosen clause *C* with backbone frequencies 0.1,0.3 and 0.6, respectively. We want to flix_1 more than x_3 because literal $\neg x_1$ appears more often than $\neg x_3$ in all local minima (frequency 0.9 versus 0.4). Therefore, we pick $x_1, \neg x_2$ and x_3 with frequencies $0.9/(0.9+0.7+0.4) = 0.45, 0.7/2.0 = 0.35$ and $0.4/2.0 = 0.2$, respectively.

4.3 Pseudo-backbone frequencies

The performance of backbone guided local search depends largely upon the quality of pseudo-backbone frequencies. There are at least two ways to compute pseudo-backbone frequencies. The first and simplest method is to consider all the available local minima as if they were global optima, and take the frequency of a variable-value pair $l = (x_i, v_i)$ that appears in all local minima *S* as its pseudo-backbone frequency. Specifically, we have frequency $p(l) = \sum_{\forall s_i \in S, l \in s_i} 1/|S|$. We call this method *average counting.*

It is imperative to note that not all local minima are of equal quality. A lower quality local minimum usually contains less backbone variables than one of a higher quality. Therefore, the former is less reliable and should contribute less to the pseudo-backbone frequencies than the latter. Thus, we must discount the contribution of a local minimum based on its cost. If a local minimum s_i has cost c_i, we can compute the pseudo-backbone frequency of $l = (x_i, v_i)$ as $p(l) = (\sum_{\forall s_i \in S, l \in s_i} 1/c_i)/(\sum_{\forall s_i \in S} 1/c_i)$. In other words, the pseudo-backbone frequency of *l* is reciprocally weighted by the costs of the local minima that *l* was involved with. We call this method *cost reciprocal average counting.*

4.4 Walksat and BGWalksat with Dynamic Noise

One limitation of the Walksat family of algorithms is their dependence on a manually set noise parameter. So far, two mechanisms have been proposed to resolve this issue in SAT. Auto-Waksat [14] uses a preliminary probing stage to estimate the optimal value for the noise parameter, while Walksat with dynamic noise [8] automatically adjusts the noise level as the search progresses. We have successfully combined our method with the dynamic noise strategy, for SAT and Max-SAT.

The idea of dynamic noise is simple: start search with the noise parameter equal to zero, and examine the number of violations in the current state every $\theta \cdot m$ flips, where m is the number of clauses in the instance. If the number of violations has not decreased since the last time we checked (n flips ago), the search is assumed to have stagnated, and the noise level is increased to $wp + (1 - wp) \cdot \phi$, where wp is the current noise level. Otherwise, the noise level is decreased to $wp - wp \cdot 2\phi$. The discrepancy between the formulas for increasing and decreasing the noise level are based on the observations of how Walksat behaves when the noise parameter is too high, compared with how it behaves when the parameter is too low [8]. Following [8], we have set $\theta = 1/6$ and ϕ = 0.2, which have been found to be effective over a wide range of SAT and Max-SAT instances.

Dynamic noise was designed and tested with Walksat's cutoff parameter set to infinity; i.e., no random restarts. This is the setting we use for Walksat with dynamic noise for all of our experiments in the next section. Unfortunately, this setting is incompatible with backbone guided Walksat, which requires random restarts in order to construct the pseudo-backbone. To overcome this, we have devised "compromise" parameter settings, which allow dynamic noise to function effectively, while still constructing the pseudo-backbone. Specifically, we run Walksat with dynamic noise for 30 short runs to construct the pseudo-backbone, followed by 7 long runs of backbone guided local search, each of which is 10 times as long as a short run.

5 Experimental Results

We now analyze the performance of BGWalksat on Max-SAT The benchmark suite for empirical evaluation consists of two different types of problems: randomly generated problems and problems converted from real-world applications in SATLIB [9]. We included satisfiable and unsatisfiable problem instances. We used Walksat implementation from Henry Kautz and developed BGWalksat on top of it.

5.1 Random ensembles

In these experiments, we investigate the effects of biased moves on random choices in Walksat (Section 2). There are five places where Walksat makes uniformly random choices (the five rectangles in Figure 1). Our experimental results showed that the biased random choice of picking a variable of zero break-count in a selected clause *C* (the unshaded rectangle in Figure 1) has almost no effect. The reason is that when a zero break-count variable exists, there is usually one available, meaning that a very few such biased moves occur. We will not consider this biased random choice in the rest of the paper.

In our experiments, we generated random MAX-3-SAT instances with 2,000 variables and three different C/V ratios of 2.0, 4.3 and 8.0. We generated 100 instances for each of the C/V ratios. We also tested various noise ratios, from 0 to 0.9 with an increment of 0.1, as well as Walksat with dynamic noise. Each run, regardless of its configuration, was allowed a total of one million flips.

For Walksat with static noise, the cutoff parameter was set to 10,000 flips (so the algorithm was run from 100 different starting points). We tested different ratios of the lengths of the backbone estimation phase and the actual backbone search phase in BGWalksat. We found that a ratio of about 0.3 seems to provide the best results. In the results shown below, Walksat was run on the first 30 tries to collect the initial set of local minima and obtain the pseudo-backbone frequencies. BGWalksat was run on the remaining 70 tries. New local minima were added to update pseudo-backbone frequencies every 5 tries.

We experimentally compared the two methods for computing pseudo-backbone frequencies (Section 4.3). The average counting (AC) method is generally worse than the cost reciprocal average counting (CRAC) method on random problems. Due to space limitations here, we will only report the results from the CRAC method below.

There is little impact of biased moves when the C/V ratio is 2.0, because Walksat is able to find optimal solutions in the first 30 tries in almost all problems. Figure 3 shows the results on random Max-SAT with 2,000 variable and C/V ratios of 4.3 and 8.0, comparing Walksat and BGWalksat with biased noise pick, biased greedy pick, and bias moves combining biased noise pick, greedy pick, clause pick, and assignment generation. The error bars in the figures correspond to 95% confidence intervals. As shown, biased greedy pick is effective at low noise levels, but leads to degraded performance on higher noise levels. Biased noise pick, on the other hand, has a greater effect as the noise level increases, due to the direct relationship between the noise level and the frequency of noise picking. Biased assignment generation and clause pick on their own, have

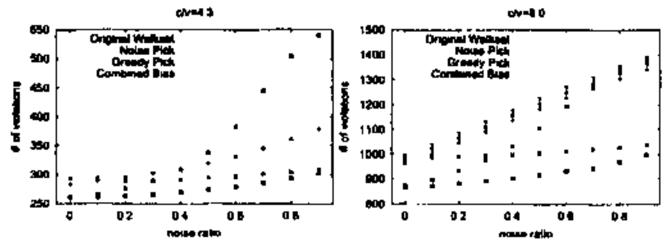

Figure 3: Quality improvement on random Max-SAT using biased noise pick, biased greedy pick, and the combination of biased noise pick, greedy pick, clause pick, and assignment generation (combined bias).

little effect of Walksat's performance (data not shown). However, when combined with other strategies, they can provide improvements, as shown.

We also tested Walksat and BGWalksat with dynamic noise on the same instances, with good results. This is fairly unsurprising, since on this class of instances, Walksat performs best with a low noise ratio, and dynamic noise begins with the noise ratio set to zero, and only raises it as needed. We found that the noise parameter fluctuated at low levels, with an average of 0.2 for both C/V=8 and C/V=4.3. The noise level never exceeded 0.6 at any time. The solutions found by Walksat with dynamic noise were as good as the best solutions found by Walksat with static noise for C/V=8, and only about 3.5% worse for C/V=4.3. When combining dynamic noise with BGWalksat, the average noise level was slightly lower, giving 0.13 for C/V=8 and 0.15 for C/V=4.3, due to the increased number of random restarts. This is because after every restart there is a period of rapid improvement, when the noise level remains at zero. The performance of BGWalksat with dynamic noise is similar to that of BGWalksat with best static noise for C/V=8 and slightly worst than that for C/V=4.3.

5.2 Real problem instances

We now investigate how BGWalksat improves upon Walksat on the problems from SATLIB [9], when used in conjunction with dynamic noise. The test problems include SAT-encoding instances from a random hard graph coloring problems and SAT-encoded blocks world planning, bounded model checking, and the all interval series problem. The details of these problems can be found on the website. We only chose problems with more than 350 variables, and discarded those that can be easily solved by Walksat and BGWalksat.

We considered satisfiable and unsatisfiable problems. We ran both Walksat and BGWalksat, both using dynamic noise, with a total of 10 million flips (compared with 1 million for our results for random instances). Interestingly, BGWalksat exhibited superior performance, over a wide range of real instances, with some of the methods of biased-moves utilizing the pseudo-backbone disabled. For the results presented here, we use backbone-guided noise pick and backbone-guided clause pick, with the average-counting (AC) method. Results are for twenty runs for each algorithm and instance. Experiments with the best static noise (not shown) produced similar results.

In viewing the results, we found it useful to divide the satisfiable instances into two categories, the easier instances, which were solved at least once (Table 1), and the harder ones, which

Table 1: Walksat vs. BGWalksat on easier satisfiable problems. Walksat and BGWalksat are the runs resulting in a satisfying solutions (out of 20) by these algorithms. The better results are underlined and in bold.

problem	#Var	#Clause	Walksat	BGWalksat
bw_large.c	3016	50457	1	2
bw.large.d	6325	131973	1	0
par8-1	350	1149	6	19
par8-2	350	1157	6	19
par8-3	350	1171	7	17
par8-4	350	1155	0	16
par8-5	350	1171		
qg1-08	512	148957	8	12
qg2-08	512	148957	1	4
qg3-08	512	10469	11	20
qg6-09	729	21844	0	5
qg7-09	729	22060	4	5
g125.17	2125	66272	5	5
g250.29	7250	454622	4	2 1

were not solved by either method, in any of their runs (Table 2). Results for unsatisfiable instances are presented in Table 3.

As the results show, BGWalksat significantly outperforms Walksat in most cases. On easier satisfiable instances (Table 1), BGWalksat finds more satisfying solutions than Walksat for all parity *(par)* and quasigroup *(qg)* classes, and produces similar results to Walksat on blocksworld instances. On harder satisfiable instances (Table 2), BGWalksat outperforms Walksat in all but two of 34 instances, where it is less than half a percent worse. In contrast, the overall average gain is 30%, and is over 50% in 11 of them. On unstatisfiable instances (Table 3), BGWalksat produces impressive gains on longmult instances, and on ssa6288-047, with an overall average gain of 20%. On unsatisfiable quasigroup instances (not shown), performance was similar to that of Walksat. Performance of BGWalksat is never more than 10% worse than Walksat on any of the unsatisfiable instances we have studied.

The most glaring failure of BGWalksat is on the instances pl25.17 and g250.29. These instances are SAT-encoded graph coloring problems, and serve to illustrate an important point. As described in Section 3, we believe that our method is effective because it exploits the "big valley" structure of the solution space. However, graph coloring problems exhibit a particular type of symmetry in their solution structures which is opaque to local search methods such as Walksat. For example, coloring all red nodes green, and all green nodes red, results in an identical solution, with a radically different encoding. Thus, there is not a single "big valley", but several, which can bury the true backbone information and thus lead to degraded performance. Presumably, BGWalksat's performance will suffer on all instances with this type of symmetry.

6 Conclusions

We developed a novel and general method that exploits backbone information for improving the performance of a local search algorithm. We demonstrated and analyzed the new method, called BGWalksat, on Max-SAT using the Walksat algorithm, with both static and dynamic noise strategies. The main ideas are to extract backbone information from local minima and use it directly to fix the discrepancy between the current state and optimal solutions, so as to guide Walksat toward the regions of the search space more likely to contain optimal solutions. In comparison, almost all existing local search methods focus on the costs of the states in the search space. There-

Table 2: Walksat vs. BGWalksat on harder satisfiablc problems. Walksat and BGWalksat are the average numbers of violations in the best solutions found by the algorithms for a given problem, averaged over 20 runs. Gain is the percentage improvement of BGWalksat over Walksat. The better results are underlined and in bold.

problem	#Var	#Clause	Walksat	BGWalksat	gam(%)
bmc-ibm-1	9685	55870	25.3	4.	—83.60
bmc-ibm-2	3628	14468	5.4	12	77.78
bmc-ibm-3	14930	72106	115.25	19.7	82.91
bmc-ibm-4	28161	139716	118.15	38.9	67.08
bmc-ibm-5	9396	41207	12.95	1.25	90.35
bmc-ibm-6	51654	368367	358.25	103.6	71.08
bmc-ibm-7	8710	39774	17.4	6.4	63.22
bmc-galileo-8	58074	294821	65.65	15.5	76.39
bmc-galileo-9	63624	326999	95.95	17.3	81.97
bmc-ibm-10	61088	334861	406.15	162.45	60.00
bmc-ibm-11	32109	150027	439.8	358.45	18.50
bmc-ibm-12	39598	19477	554.65	445.25	19.72
bmc-ibm-13	13215	6572	88.05	2.7	96.93
f2000	2000	8500	2.2	2.05	6.82
par16-1-c	317	1264	*5.45*	5.35	1.83
par16-1	1015	3310	10.45	9.45	9.57
par16-2-c	349	1392	6.2	5.9	4.84
par16-2	1015	3374	10.6	10.4	1.89
par16-3-c	334	1332	6	5.65	5.83
par16-3	1015	3344	10.45	9.75	6.70
par16-4-c	324	1292	6.15	5.5	10.57
par16-4	1015	3324	10.4	9.55	8.17
par16-5-c	341	1360	6.25	6.05	3.20
par16-5	1015	3358	10.45	9.85	5.74
par32-1-c	1315	5254	21.7	20.85	3.92
par32-1	3176	10277	30.95	30.25	2.26
par32-2-c	1303	5206	21.15	21.2	-0.24
par32-2	3176	10253	32.1	28.35	11.68
par32-3-c	1325	5294	22.05	21.3	3.40
par32-3	3176	10297	32.95	28.55	13.35
par32-4-c	1333	5326	21.3	21.4	-0.47
par32-4	3176	10313	33.65	29.4	12.63
par32-5-c	1339	5350	23.15	22.05	4.75
par32-5	3176	10325	32.9	30.3	7.90
Average					29.82

fore, the new algorithm focuses more on where the current state is within the search space and tries to fix possible problems in the structures of the state directly.

BGWalksat can significantly outperforms Walksat on SAT and Max-SAT, with both static and dynamic noise. On random Max-3-SAT problems, the more constrained the problems, the more improvement BGWalksat can provide. Specifically, on random Max-3-SAT with 2,000 variables and a C/V ratio of 8.0, BGWalksat can satisfy approximately 10% more clauses on average than Walksat (using dynamic noise). On satisfiable problems from the SATLIB, BGWalksat almost always significantly improves on Walksat's results, with the exception of graph coloring problems which contains a type of symmetry (see Section 5). On unsatisfiablc SATLIB problems, BG-Walksat substantially reduces the number of unsatisfied clauses across several instance classes.

In the process of developing BGWalksat, we also carried out a systematic experimental analysis of Walksat on Max-SAT (again with both static and dynamic noise). Our results show that although designed originally for SAT, Walksat is also effective on Max-SAT, even though its effectiveness degrades as the problem becomes more constrained.

Table 3: Walksat vs. BGWalksat on unsatisfiable problems. The legend is the same as that in Table 2.

problem	#Var	#Clause	Walksat	BGWalksat	gam(%)
longmult06	2848"	—8853	1.5	—1.65	-16.00
longmult07	3319	10335	2.05	2.2	-7.32
longmult08	3810	11877	3.65	2.65	27.40
longmult09	4321	13479	6.75	2.9	57.04
longmult10	4852	15141	10.25	5.6	45.37
longmult11	5403	16863	15.05	9.2	38.87
longmult12	5974	18645	17.8	16.2	8.99
longmultl 3	6565	20487	23.25	21.4	7.96
longmultl 4	7176	22389	32.6	24.6	24.54
longmultl 5	7807	24351	41.5	30	27.71
ssa6288-047	10410	34238	100.25	89.7	16.62
Average					20.01

References

[1] K. D. Boese. *Models for Iterative Global Optimization.* PhD thesis, UCLA/Computer Science Department, 1996.

[2] P. Chccscman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. IJCAI-91,* pages 331-337.

[3] P. Codognet and F Rossi. Notes for the ECAI2000 tutorial on Solving and Programming with Soft Constraints: Theory and Practice. available at http://www.math.unipd.it/frossi/papers.html.

[4j M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *CACM,* 5:394-397, 1962.

[5] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formula. In *Proc. IJCAI-OI,* pages 248-253,2001.

[6] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence,* 58:21-70, 1992.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

18] H. H. Hoos. An adaptive noise mechanism for walksat. In *Proc. AAAI-02.*

[9] H. H. Hoos and T. Stuzle. SATLIB - the satisfiability library. http://www.informatik.tu-darmstadt.de/AI/SATLIB, 1999.

[10] Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochstic algorithm for MAX-SAT. In *Proc. 1st Workshop on AI and OR,* Timberinc, OR, 1995.

III] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. AAAI-97,* pages 321-326, 1997.

[12] D. Mitchell, B. Selman, and H. Levesquc. Hard and easy distributions of SAT problems. In *Proc. AAAI-92,* 1992.

[13] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature,* 400:133-137, 1999.

[14] D. J. Patterson and H. Kautz. Auto-Walksat: A self-tuning implementation of Walksat. *Electronic Notes in Discrete Mathematics,* 9, 2001.

[15] J. C. Pemberton and W. Zhang. Epsilon-transformation: Exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence,* 81:297 325, 1996.

[16] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. AAAI-94.*

[17] J. Slaney and T. Walsh. Backbones in optimization and approximation. *In Proc. IJCAI-01,* 2001.

[18] O. Telelis and P. Stamatopoulos. Heuristic backbone sampling for maximum satisfiability. In *Proc. 2nd Hellenic Conf. on AI,* pages 129-139, 2002.

[19] W. Zhang. Phase transitions and backbones of 3-SAT and Maximum 3-SAT. In *Proc. 7th Intern. Conf. on Principles and Practice of Constraint Programming (CP-200I),* pages 153-167,2001.