Local search characteristics of incomplete SAT procedures

Dale Schuurmans Finnegan Southey

Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada
{dale,fdjsouth}@cs.uwaterloo.ca

Abstract

Effective local search methods for finding satisfying assignments of CNF formulae exhibit several systematic characteristics in their search. We identify a series of measurable characteristics of local search behavior that are predictive of problem solving efficiency. These measures are shown to be useful for diagnosing inefficiencies in given search procedures, tuning parameters, and predicting the value of innovations to existing strategies. We then introduce a new local search method, SDF ("smoothed descent and flood"), that builds upon the intuitions gained by our study. SDF works by greedily descending in an informative objective (that considers how strongly clauses are satisfied, in addition to counting the number of unsatisfied clauses) and, once trapped in a local minima, "floods" this minima by reweighting unsatisfied clauses to create a new descent direction. The resulting procedure exhibits superior local search characteristics under our measures. We show that this method can compete with the state of the art techniques, and reduces the number of search steps of some recent methods by a significant factor.

1 Introduction

Since the introduction of GSAT [24] there has been considerable research on local search methods for finding satisfying assignments for CNF formulae. These methods are surprisingly effective: they can often find satisfying assignments for large CNF formulae that are far beyond the capability of current systematic search methods [24].[1] Of course, local search is incomplete and cannot prove that a formula has no satisfying assignment when none exists. However, despite this limitation, incomplete methods for solving large satisfiability problems are proving their worth in applications ranging from planning to circuit design and diagnosis [12, 13, 23].

Significant progress has been made on improving the speed of these methods since the development of GSAT. In fact, a series of innovations have led to current search methods that are now an order of magnitude faster.

Perhaps the most significant early improvement was to incorporate a "random walk" component where variables were flipped from within random falsified clauses [21]. This greatly accelerated search and led to the development of the very successful WSAT procedure [22]. A contemporary idea was to keep a tabu list [15] or break ties in favor of least recently flipped variables [7, 8] to prevent GSAT from repeating earlier moves. The resulting TSAT and HSAT procedures were also improvements over GSAT, but to a lesser extent. The culmination of these ideas was the development of the Novelty and R_Novelty procedures which combined a preference for least recently flipped variables in a WSAT-type random walk [16], yielding methods that are currently among the fastest known.

A different line of research has considered adding clause-weights to the basic GSAT objective (which merely counts the number of unsatisfied clauses) in an attempt to guide the search from local basins of attraction to other parts of the search space [3, 4, 5, 6, 17, 21]. These methods have proved harder to control than the above techniques, and it has only been recent that clause re-weighting has been developed to a state of the art method. The series of "discrete Lagrange multiplier" (DLM) systems developed in [26, 25, 27] have demonstrated competitive results on benchmark challenge problems in the DIMACS and SATLIB repositories.[2]

Although these developments are impressive, a systematic understanding

[1] However, see [2, 14] for competitive systematic search results.

[2] URLs ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf and http://aida.intellektik.informatik.tu-darmstadt.de/~hoos/SATLIB/ respectively.

of local search methods for satisfiability problems remains elusive. Research in this area has been largely empirical and it is still often hard to predict the effects of a minor change in a procedure, even when this results in dramatic differences in search times.

In this paper we identify three simple, intuitive measures of local search effectiveness: depth, mobility, and coverage. We show that effective local search methods for finding satisfying assignments exhibit all three characteristics. These, however, are conflicting demands, and successful methods are primarily characterized by their ability to effectively manage the tradeoff between these factors—whereas ineffective methods tend to fail on at least one measure. Our goal is to be able to distinguish between effective and ineffective search strategies in a given problem (or diagnose problems with a given method, or tune parameters) without having to run exhaustive search experiments to their completion.

To further justify this endeavor, we introduce a new local search procedure, SDF ("smoothed descent and flood"), that arose from our investigation of the characteristics of effective local search procedures. We show that SDF exhibits uniformly good depth, mobility, and coverage values, and consequently achieves good search performance (in terms of the number of search steps required) on a large collection of benchmark satisfiability problems. This paper revises and extends the earlier work reported in [20].

2 Local search procedures

In this paper we investigate several dominant local search procedures from the current literature. Although many of these strategies appear to be only superficial variants of one another, they demonstrate dramatically different problem solving performance and (as we will see) they exhibit distinct local search characteristics as well.

The local search procedures we consider start with a random variable assignment $\mathbf{x} = \langle x_1, ..., x_n \rangle$, $x \in \{0, 1\}$, and make local moves by flipping one variable x_i at a time (setting $x_i' = 1 - x_i$) until they either find a satisfying assignment or reach a maximum number of moves. For any variable assignment there are a total of n possible variables to consider, and the various strategies differ in how they make this choice. Current methods uniformly adopt the original GSAT objective of simply minimizing the number of

unsatisfied clauses

$$g(\mathbf{x}) \;=\; \sum_{c_j \,\in\, \text{clauses}} 1_{(\mathbf{x} \text{ falsifies } c_j)} \tag{1}$$

perhaps with some minor variation such as introducing clause weights or considering how many new clauses become falsified by a flip (break count) or how many new clauses become satisfied (make count). The specific flip selection strategies we investigate along with their free parameters (shown in parentheses) are as follows.

GSAT() Flip the variable x_i that results in the fewest total number of clauses being unsatisfied. Break ties uniformly at random [24].

HSAT() Same as GSAT, but break ties in favor of the least recently flipped variable [7].

WSAT-G(p) Pick a random unsatisfied clause c_j. With probability p flip a random x_i in c_j. Otherwise flip the variable in c_j that results in the smallest total number of unsatisfied clauses [16].

WSAT-B(p) Same as WSAT-G, except in the latter case, flip the variable that would cause the smallest number of *new* clauses to become unsatisfied [16].

WSAT(p) Same as WSAT-B, except first check whether some variable x_i would not falsify any new clauses if flipped, and always take such a move if available [16, 22].

Novelty(p) Pick a random clause c_j. Flip the variable x_i in c_j that would result in the smallest total number of unsatisfied clauses, unless x_i is the most recently flipped variable in c_j. In the latter case, flip x_i with probability $1-p$ and otherwise flip the variable x_k in c_j that results in the second smallest total number of unsatisfied clauses [16].

Novelty+(p, q) Same as Novelty, except that after the clause c_j is selected, flip a random x_i in c_j with probability q, and otherwise continue with Novelty [11].

The recent **DLM** procedure is much more complex than the above methods. However, it is arguably the most efficient search system for satisfying

assignments currently known. DLM keeps a vector of nonnegative weights w_j on the clauses c_j and then employs a GSAT style local search to reduce the total weight of unsatisfied clauses. Once a local minimum is reached, the weights of the unsatisfied clauses are increased and the search is continued. DLM also employs a tabu list on flipped variables. The exact calculation for the weight updates is very complex and the interested reader is referred to [25, 26, 27] for details. Despite its overall complexity (the publically released implementation contains more than 20 free parameters) the architects of DLM have managed to engineer it into achieving state of the art performance [27]—a fact which we verify below. To run DLM in our experiments we explored the five default parameter sets supplied in the public software release.[3]

Note that, conventionally, all of these local search procedures have an outer loop that places an upper bound, t, on the maximum number of flips allowed before re-starting with a new random assignment. However, we will not focus on random restarts in our experiments below because *any* search strategy can be improved (or at the very least, not damaged) by choosing an appropriate cutoff value t [9, 10]. In fact, it is straightforward and well known how to do this optimally (in principle): For a given search strategy and problem, let the random variable T denote the number of flips needed to reach a solution in a single run, and let T_t denote the number of flips needed when using a random restart after every t flips. We then have the straightforward equality [18]

$$\mathrm{E}(T_t) \;=\; \frac{t}{\mathrm{P}(T \leq t)} - [t - \mathrm{E}(T | T \leq t)] \tag{2}$$

which simply states that $\mathrm{E}(T_t)$ is t times the expected number of restarts needed to find a solution ($t/\mathrm{P}(T \leq t)$), minus a small correction which accounts for the fact that on the last restart the search will generally find the solution before the maximum cutoff value has been reached. Note that this always offers a potential improvement in principle, since one could choose the optimal cutoff value

$$t^* \;=\; \arg \min_{t:0<t\leq\infty} \mathrm{E}(T_t) \tag{3}$$

which will yield a substantial reduction in expected search time for most natural search distributions [10] and, at the very least, will not increase the

[3] Software available at http://manip.crhc.uiuc.edu.

expected search time for any distribution (see Appendix A for a detailed discussion). Therefore, we will focus on investigating the single run characteristics of the various variable selection policies, but be sure to report estimates of what the optimum achievable performance would be using random restarts. We report this optimal quantity for every procedure using the empirical distribution of T over several runs (at least 100) to estimate $\mathrm{E}(T_{t^*})$.

3 Measuring local search performance

In order to tune the parameters of a search strategy, determine whether a strategic innovation is helpful, or even debug an implementation, it would be useful to be able to measure how well a search is progressing without having to run it to completion on large, difficult problems.

3.1 Depth

To begin, we consider a simple and obvious measure of local search performance that has no doubt been used to tune and debug many search strategies in the past.

Depth measures how many clauses remain unsatisfied as the search proceeds. Intuitively, this indicates how deep in the objective g the search is remaining. To get an overall summary, we take a depth average over all search steps (after the first t_d steps[4]). Note that it is desirable to obtain a small value of depth.

Although simple minded, and certainly not the complete story, it is clear that effective search strategies do tend to descend rapidly in the objective function and remain at good objective values as the search proceeds. By contrast, strategies that fail to persistently stay at good objective values usually have very little chance of finding a satisfying assignment in a reasonable number of flips [16].

To demonstrate this rather obvious effect, consider the problem of tuning the noise parameter p for the WSAT procedure on a given problem. Here

[4]Note that a random initial assignment will generally have poor depth, but most search methods will nevertheless settle into a lower steady state behavior after the initial transient phase. In our experiments we simply ignore the first $t_d = 100$ flips to calculate the average depth to avoid this initial transient phase.

we use the uf100-0953 problem from the SATLIB repository to demonstrate our point.[5] Table 1 shows that higher noise levels cause WSAT to stay higher in the objective function and significantly increases the numbers of flips needed to reach a solution for this problem. (The SDF procedure in the table is introduced in Section 4 below.) This result holds for the raw average number of flips and also for the optimal expected number of flips using an optimal maximum flips cutoff with random restarts, $\hat{\mathrm{E}}(T_{t^*})$ (estimated by choosing the t^* that minimizes (2) under the empirical distribution). The explanation is obvious: by repeatedly flipping an arbitrary random variable in an unsatisfied clause, WSAT is frequently "kicked up" to higher objective values—to the extent that it begins to spend significant search time simply re-descending to a lower objective value, only to be prematurely kicked up again.

Although depth is a simplistic measure, it proves to be very useful for tuning noise and temperature parameters in local search procedures. By measuring depth, one can determine if the search is spending too much time recovering from large increases in the objective function and not enough time exploring near the bottom of the objective. More importantly, maintaining depth appears to be *necessary* for achieving reasonable search times. Table 2 shows the results of a large experiment conducted on the entire collection of 2700 uf problems in SATLIB. This comparison ranked four comparable methods—DLM, Novelty, Novelty+, and WSAT—in terms of their search depth and average flips. For each problem, the four methods were run 100 times and the optimal expected flip count (using an ideal restart scheme) was estimated for each, and then the methods were ranked in terms of their estimated optimal flip count and average depth. Each (flips rank, depth rank) pair was then recorded in a table. The relative frequencies of these pairs is summarized in Table 2. This table shows that the highest ranked method in terms of search efficiency was almost always ranked near the top in terms of search depth, and almost never near the bottom rank.

[5] The uf series of problems are randomly generated 3-CNF formulae that are generated at the phase transition ratio of 4.3 clauses to variables. Such formulae have roughly a 50% chance of being satisfiable, but uf contains only verified satisfiable instances. For a recent approach to generating hard random *satisfiable* instances see [1].

100 runs on uf100-0953	Average depth	Average flips	Est. optimal average flips using restarts
WSAT(.5)	5.16	11,757	9,802
WSAT(.7)	8.60	18,042	17,282
WSAT(.8)	10.3	25,543	18,698
WSAT(.9)	12.3	59,042	53,451
Novelty(.5)	3.75	4,916	4,563
Novelty+(.5, .01)	3.68	3,965	3,965
DLM(pars4)	5.40	2,182	1,984
SDF(.00085)	3.03	1,192	1,174

Table 1: Average number of unsatisfied clauses obtained by various search procedures over the length of a search run (depth), along with the average number of flips needed to find a solution and the estimated average number of flips needed under an optimal re-start scheme. Results are averaged over 100 runs on problem uf100-0953 from SATLIB.

Flips rank	Depth rank			
	best 1	2	3	worst 4
best 1	.74	.09	.13	.05
2	.10	.28	.38	.25
3	.14	.42	.34	.11
worst 4	.03	.22	.16	.60

Table 2: Frequencies of (flips rank, depth rank) pairs from among four search procedures, DLM, Novelty, Novelty+, and WSAT. Frequencies measured over the entire collection of uf problems in SATLIB (2700 problems in total).

3.2 Mobility

Although useful, depth alone is clearly not a *sufficient* criterion for ensuring good search performance. A local search could easily become stuck at a good objective value, and yet fail to explore widely. To account for this possibility we introduce another measure of local search effectiveness.

Mobility measures how rapidly a local search moves in the search space. We measure mobility by calculating the Hamming distance[6] between variable assignments that are k steps apart in the search sequence and average this quantity over the entire sequence to obtain average distances at time lags $k = 1, 2, 3, ...$, etc. It is desirable to obtain a large value of mobility since this indicates that the search is moving rapidly through the space.

Mobility is obviously very important in a local search. In fact, most of the significant innovations in local search methods over the last decade appear to have the effect of substantially improving mobility without significantly damaging depth. This is demonstrated clearly in Figure 1 and Table 3 for the uf100-0953 problem. It appears that the dramatic improvements of these methods could have been predicted from their improved mobility scores (while maintaining comparable depth scores).

Figure 1 and Table 3 cover several highlights in the development of local search methods for satisfiability. For example, one of the first useful innovations to GSAT was to add a preference for least recently flipped variables, resulting in the superior HSAT procedure. Table 3 shows that one benefit of this change is to increase mobility without damaging search depth, which corresponds to improved solution times. (GSAT failed to solve the problem within 500,000 steps on 100 trials, whereas HSAT solved it once.[7]) Another early innovation was to incorporate "random walk" in GSAT. Figure 1 shows that WSAT-G also delivers a noticeable increase in mobility—again resulting in a dramatic reduction in solution times (Table 3). It is interesting to note that the apparently subtle distinction between WSAT-G and WSAT in terms of their definition is no longer subtle here: WSAT offers a dramatic improvement in mobility, along with an accompanying improvement

[6] The Hamming distance between two Boolean vectors is simply the number of positions where the vectors disagree.

[7] HSAT's improvement over GSAT is even more apparent in Table 4 below.

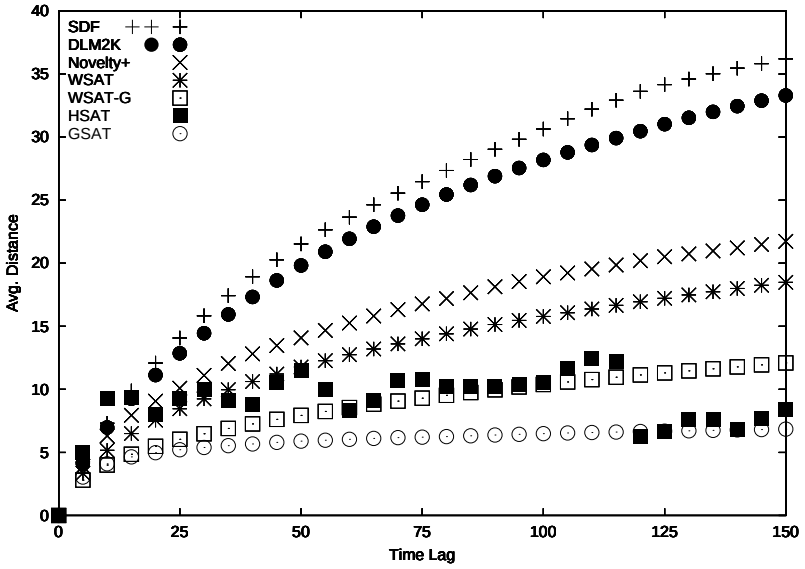

Figure 1: Average Hamming distance between variable assignments at different time lags obtained by various search procedures. Distances are averaged over the length of each search run and then averaged across 100 repetitions of the uf100-0953 problem in SATLIB.

100 runs on uf100-0953	Average mobility	Average depth	Average flips	Est. optimal average flips using restarts
GSAT()	6.0	2.19	500,000	na
HSAT()	9.0	2.06	495,003	16,700
WSAT-G(.5)	10.1	4.20	29,661	15,442
WSAT(.5)	16.1	5.16	11,757	9,802
Novelty(.5)	19.0	3.75	4,916	4,563
Novelty+(.5, .01)	18.7	3.68	3,965	3,965
DLM(pars4)	28.6	5.40	2,182	1,984
SDF(.00085)	29.7	3.03	1,192	1,174

Table 3: Hamming distance between assignments 100 steps apart in the search, averaged over the length of a search run (along with depth and search step measures). Results are averaged over 100 runs on problem uf100-0953 in SATLIB.

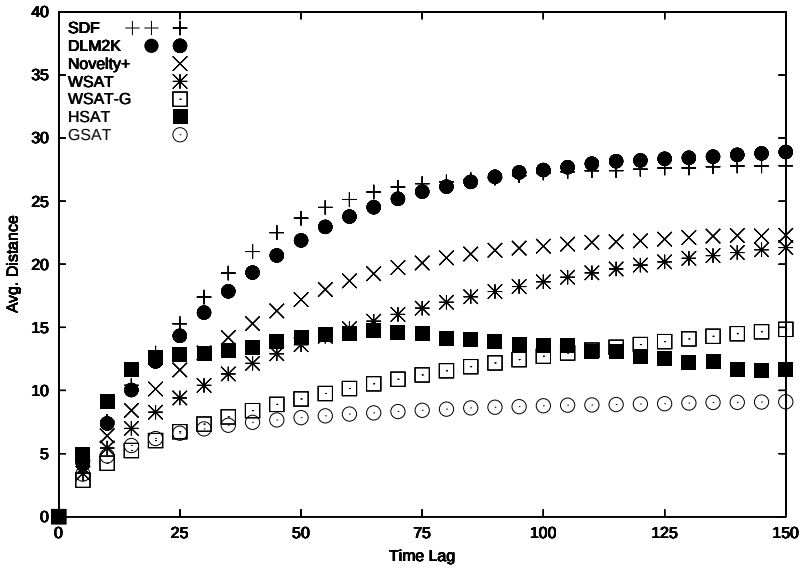

Figure 2: Same as Figure 1, but results are averaged over 100 runs on all 1000 uf100 problems from SATLIB.

100 runs on 1000 uf100 problems	Average mobility	Average depth	Average flips	Est. optimal average flips using restarts
GSAT()	8.8	1.62	74,595	10,823
HSAT()	15.8	1.35	73,510	2,503
WSAT-G(.5)	12.8	3.13	7,695	4,431
WSAT(.5)	18.9	3.49	3,582	2,828
Novelty(.5)	23.4	1.90	2,586	1,473
Novelty+(.5, .05)	23.6	1.93	2,224	1,393
DLM(pars4)	32.4	6.36	1,020	800
SDF(.00085)	31.8	1.41	876	755

Table 4: Same as Table 3, but results averaged over 100 runs on all 1000 uf100 problems from SATLIB.

Flips rank	Mobility rank best 1	2	3	worst 4
best 1	.92	.07	.01	.00
2	.06	.72	.21	.01
3	.03	.20	.75	.03
worst 4	.00	.01	.03	.96

Table 5: Frequencies of (flips rank, mobility rank) pairs from among four search procedures, DLM, Novelty, Novelty+, and WSAT. Frequencies measured over the entire collection of uf problems in SATLIB (2700 problems in total). Mobility is measured between assignments 100 steps apart in the search, averaged over the length of a search run.

in efficiency. The culmination of novelty and random walk in the Novelty procedure achieves even a further improvement in mobility, and, therefore it seems, solution time. Finally, the recent DLM procedure clearly exhibits superior mobility and search step scores to all the other methods (except perhaps for the SDF procedure, which we introduce below).

We have observed this relationship between mobility and solution time consistently over the entire range of problems we have investigated. In fact, Figure 2 and Table 4 show that similar results are obtained by averaging over the entire collection of uf100 problems in the SATLIB repository (1000 problems in total). From these results it appears that, in addition to depth, mobility also is a necessary characteristic of an effective local search for satisfiability problems. To establish this further, Table 5 shows the results of a large experiment conducted on the entire collection of 2700 uf problems in SATLIB. The same four procedures were tested as before (DLM, Novelty, Novelty+, WSAT) and ranked in terms of their mobility scores and number of flips to find a solution. The results show that the highest ranked in terms of mobility is almost always ranked near the top in problem solving efficiency, and that low mobility tends to be correlated with inferior search efficiency.

3.3 Coverage

A final characteristic of local search behavior that we consider is easily demonstrated by a simple observation: Hoos [11] presents a small satisfiable CNF formula with five variables and six clauses that causes Novelty to sometimes get stuck (permanently) in a local basin of attraction that prevents

it from solving an otherwise trivial problem. The significance of this example is that Novelty exhibits good depth and mobility on this instance and yet fails to solve what is otherwise an easy problem. This concern led Hoos to develop the slightly modified procedure Novelty+ in [11]. The characteristic that the original Novelty procedure is missing in this case is coverage.

Coverage measures how systematically the search explores the entire space. We compute a *rate* of coverage by first estimating the size of the largest unexplored "gap" in the search space (given by the maximum Hamming distance between any unexplored assignment and the nearest explored assignment) and measuring how rapidly the largest gap size is being reduced. In particular, we define the coverage rate to be $(n - max\ gap)/(search\ steps \times n)$. Note that it is desirable to have a high rate of coverage as this indicates that the search is systematically exploring new regions of the space as it proceeds.

Table 6 shows that Hoos's modified Novelty+ procedure improves the coverage rate of Novelty on the uf100-0953 problem. Table 7 shows that this effect is systematic over a large number of uf100 problems from the SATLIB repository. In these experiments Novelty+ demonstrates better coverage than Novelty while maintaining similar values on other measures, and thereby achieves better performance on almost every problem in the benchmark collections.

From these results it appears that coverage too is a necessary trait for effective local search in satisfiability problems. To further support this assertion Table 8 shows the results of a large experiment conducted on the entire collection of 1000 uf100 problems in SATLIB. Once again, we tested the four procedures, DLM, Novelty, Novelty+, and WSAT, and measured the (flips rank, coverage rank) frequencies they obtained. Table 8 shows that the highest ranked in terms of coverage is almost always ranked near the top in problem solving efficiency, and again almost never near the bottom rank.

It is important to emphasize that coverage captures a different aspect of local search behavior than mobility. Mobility characterizes how rapidly the search moves away from recently explored assignments. However, this does not necessarily imply that the search is exploring significant new regions of the space. Moving rapidly within a large but confined region is quite different from systematically moving into new regions—even an extremely mobile search could be managing to ignore significant regions of the search

100 runs on uf100-0953	Average coverage rate	Average mobility	Average depth	Average flips	Est. optimal average flips using restarts
GSAT()	.00003	6.0	2.19	500,000	na
HSAT()	.00028	9.0	2.06	495,003	16,700
WSAT-G(.5)	.00011	10.1	4.20	29,661	15,442
WSAT(.5)	.00013	16.1	5.16	11,757	9,802
Novelty(.5)	.00013	19.0	3.75	4,916	4,563
Novelty+(.5, .01)	.00017	18.7	3.68	3,965	3,965
DLM(pars4)	.00047	28.6	5.40	2,182	1,984
SDF(.00085)	.00053	29.7	3.03	1,192	1,174

Table 6: Average coverage rates obtained over entire search runs by various search procedures—along with depth, mobility, and number of search steps needed to find a solution. Results are averaged over 100 runs on problem uf100-0953 in SATLIB.

100 runs on 1000 uf100 problems	Average coverage rate	Average mobility	Average depth	Average flips	Est. optimal average flips using restarts
GSAT()	.0002	8.8	1.62	74,595	10,823
HSAT()	.0007	15.8	1.35	73,510	2,503
WSAT-G(.5)	.0004	12.8	3.13	7,695	4,431
WSAT(.5)	.0005	18.9	3.49	3,582	2,828
Novelty(.5)	.0010	23.4	1.90	2,586	1,473
Novelty+(.5, .01)	.0012	23.6	1.93	2,224	1,393
DLM(pars4)	.0014	32.4	6.36	1,020	800
SDF(.00085)	.0016	33.4	1.52	870	725

Table 7: Same as Table 6, but results averaged over 100 runs on all 1000 uf100 problems from SATLIB.

Flips rank	Coverage rank			
	best 1	2	3	worst 4
best 1	.67	.22	.10	.01
2	.19	.43	.32	.06
3	.13	.32	.45	.11
worst 4	.00	.04	.14	.82

Table 8: Frequencies of (flips rank, coverage rank) pairs from among four search procedures, DLM, Novelty, Novelty+, and WSAT. Frequencies measured over the entire collection of uf100 problems in SATLIB (1000 problems in total).

space, which clearly is not a desirable characteristic in a general search procedure. Coverage, on the other hand, measures precisely how systematically the search is uncovering new regions of the space that have not been previously explored.

3.4 Sufficiency

Up to this point we have investigated the *necessity* of the three proposed measures: depth, mobility and coverage. Necessity means that poor performance under any one measure leads to poor performance in overall problem solving behavior. We have demonstrated the apparent necessity of each of depth, mobility and coverage in the previous sections. However, our observations raise the question of the *sufficiency* of the three measures. That is, are simultaneously good depth, mobility, and coverage scores sufficient to ensure that effective problem solving performance is obtained (independent of other algorithmic details)? To present some evidence that this might at least be empirically the case, we conducted the following experiment: We ran all of the local search methods that we had at our disposal (DLM, SDF, Novelty+, Novelty, WSAT, HSAT, GSAT) with three parameter perturbations each (for a total of 21 methods) on all 2700 uf problems in the SATLIB repository. Our intent was to measure the correlation between each performance measure and problem solving efficiency while keeping the other search measures fixed, but allowing all other aspects of algorithmic detail to vary as widely as possible.

Specifically, we ran all 21 methods on the 2700 uf problems and recorded their depth, mobility, coverage, and flips scores, averaged over 100 runs. This

	depth	mobility	coverage
correlation	0.21	-0.31	-0.49
entropy	2.20	2.07	1.23

Table 9: Correlations of proposed measures with number of search steps (flips), holding the other two measures roughly constant. Numbers are averaged across 625 buckets of controlled tuples obtained by running 21 methods over the entire collection of 2700 uf problems. The entropy numbers indicate the average diversity of methods in each bucket supporting the estimates.

yielded 56,700 ⟨*depth, mobility, coverage, flips*⟩ tuples. For each of the candidate measures (say depth) we controlled the two other measures by placing the tuples into buckets defined by small intervals of the two remaining measures (say mobility and coverage); thereby attempting to keep the two other measures approximately constant. For example, in one experiment the 56,700 tuples were first partitioned into 25 buckets determined by mobility values in the 0-4, 5-8, 9-12,...,97-100 percentiles, and then each of these buckets was further subdivided into 25 sub-buckets determined by coverage values in the 0-4,5-8,...,97-100 percentiles, so that in the end we obtained 625 buckets that each contained tuples with similar mobility and coverage scores. We then investigated the relationship between the depth and flip scores within each bucket. The idea was to keep mobility and coverage fixed and see how systematically depth might be correlated with search steps (flips). Note that, on average, each bucket should contain about $56,700/625 \approx 91$ ⟨*depth, flips*⟩ pairs, which hopefully vary enough to allow us to detect any relationship in their behavior. In each bucket we measured the correlation between depth and flips using the standard sample correlation statistic, and also measured the diversity of the bucket by the entropy of the distribution of methods therein.[8]

In this experiment, supporting evidence for the sufficiency of the set of criteria (independent of other algorithmic details) would be given by an appropriate correlation of each measure with flips while the other measures are controlled, along with a diversity of methods supporting the inference. Such an outcome would suggest that, the two other measures being equal (but little else), improving any one criterion would generally result in a reduction

[8]Hence, the maximum entropy is obtained by a uniform distribution over the 21 methods for a value of 4.39 bits.

in search steps. In fact, this is roughly what we observe. Table 9 shows that, the other two measures being held equal (but the algorithms varying otherwise), each one of the three criteria is correlated in the appropriate direction with search steps (positive with depth, negative with mobility and coverage), and each of these inferences is supported by a nontrivial diversity of search methods.

Although these results are far from completely convincing, we believe that this experiment does provide some weak evidence that the three proposed criteria might at least be partially sufficient for yielding effective local search performance, in addition to their more obvious necessity.

3.5 Summary

Overall, our results lead us to hypothesize that local search procedures work effectively because they descend quickly in the objective, persistently explore variable assignments with good objective values, and do so while moving rapidly through the search space and visiting very different variable assignments without returning to previously explored regions. That is, we surmise that good local search methods do not possess any special ability to predict whether a local basin of attraction contains a solution—rather they simply descend to promising regions and explore near the bottom of the objective as rapidly, broadly, and systematically as possible, until they stumble across a solution. Although this is a rather simplistic view, it seems supported by our data, and moreover it has led to the development of a new local search technique that we introduce below. Our new procedure achieves good characteristics under these measures and, more importantly, exhibits good search-step performance in comparison to existing methods.

4 A new local search strategy: SDF

Although the previous measures provide useful diagnostic information about local search performance, one of the the main contributions of this paper is a new local search procedure, which we call SDF for "smoothed descent and flood." Our procedure has two main components that distinguish it from previous approaches. First, we perform steepest descent in a more informative objective function than earlier methods. Second, we use multiplicative clause re-weighting to rapidly move out of local minima and efficiently travel

to promising new regions of the search space.

Recall that the standard GSAT objective g (defined in (1)) simply counts the number of unsatisfied clauses for a given variable assignment. We instead consider a smoothed objective f (defined below) which takes into account how many variables satisfy each clause. To explain this objective, it will be more convenient to think of a reversed goal where we seek to *maximize* the number of satisfied clauses rather than minimize the number of unsatisfied clauses. Our smoothed objective f works by favoring variable assignments that satisfy more clauses, but all things being equal, favoring assignments that satisfy more clauses twice (subject to satisfying the same number of clauses once), and so on. In effect, we introduce a tie-breaking criterion that decides, when two assignments satisfy the same number of clauses, that we should prefer the assignment which satisfies more clauses on two distinct variables, and if the assignments are still tied, that we should prefer the assignment that satisfies more clauses on three distinct variables, etc. This tie-breaking scheme can be expressed in a scalar function that gives a large increment to the first satisfying variable, and then gives exponentially diminishing increments for subsequent satisfying variables for a given clause. For k-CNF formulas with m clauses such a scoring function is

$$f(\mathbf{x}) \;=\; \sum_{c \,\in\, \text{clauses}} score(\#\ x_i\text{'s that satisfy } c) \tag{4}$$

where

$$\begin{aligned} score(0) &= 0 \\ score(1) &= m^{k-1} \\ score(2) &= m^{k-1} + m^{k-2} \\ &\vdots \\ score(\mathrm{k}) &= m^{k-1} + m^{k-2} + \cdots + 1 \end{aligned}$$

The key property of this scoring function is that the increment obtained by increasing the number of satisfying variables in one clause from j to $j+1$ is greater than the increment obtained by increasing the number of satisfying variables in all remaining $m-1$ clauses from $j+1$ all the way to k. That is

$$\begin{aligned} score(\mathrm{j{+}1}) - score(\mathrm{j}) &= m^{k-j-1} \\ &> (m-1)\left(\frac{m^{k-j-1}-1}{m-1}\right) \end{aligned}$$

$$
\begin{aligned}
&= (m-1)\left(m^{k-j-2} + m^{k-j-3} + \cdots + 1\right) \\
&= (m-1)\left(score(\mathrm{k}) - score(\mathrm{j{+}1})\right)
\end{aligned}
$$

Our intuition is that performing steepest ascent in this objective should help build robustness in the current variable assignment which the search can later exploit to satisfy new clauses. That is, once the search has reached an assignment where no additional clauses can be satisfied immediately, we can still proceed productively by finding assignments that satisfy more clauses on two distinct variables, or failing that, finding assignments that satisfy more clauses on three distinct variables, etc.—with the hope that this will create the opportunity to satisfy more clauses (once) later in the search. We have clearly observed this phenomenon in our experiments. In fact, Figure 3 shows that following a steepest ascent in f descends deeper in the original GSAT objective g than the GSAT procedure itself (before either procedure reaches a local extrema or plateau). This happens because plateaus in the GSAT objective g are not plateaus in f—such plateaus are usually opportunities to build up robustness in satisfied clauses which can be later exploited to satisfy new clauses. This effect is clearly demonstrated in Figure 3, and has been observed over the range of problems we have considered. This gives our first evidence that the SDF procedure, by descending deeper in the GSAT objective, has the potential to improve the performance of existing local search methods.

The main issue is to cope with local maxima in the new objective. That is, although f does not contain many plateaus, the local search procedure now has to deal with legitimate (and numerous) local maxima in the search space. While this means that plateau walking is no longer a significant issue, it creates the difficulty of having to escape from true traps in the objective function. Our strategy for coping with local maxima involves the second main idea behind the SDF procedure: *multiplicative* clause re-weighting. That is, we first consider a clause-weighted version of our smoothed objective

$$
h(\mathbf{x}, \mathbf{w}) = \sum_{c \in \text{ clauses}} w(c)\ \ score(\#\ x_i\text{'s that satisfy } c) \tag{5}
$$

(where the clause weights are initially assigned to be uniform, $w(c) = 1/m$) and then use multiplicative weight updates to escape local maxima. Note that when a search is trapped at a local maxima the current variable assignment must leave some subset of the clauses unsatisfied. Many authors have observed that such local extrema can be "filled in" by increasing the weight

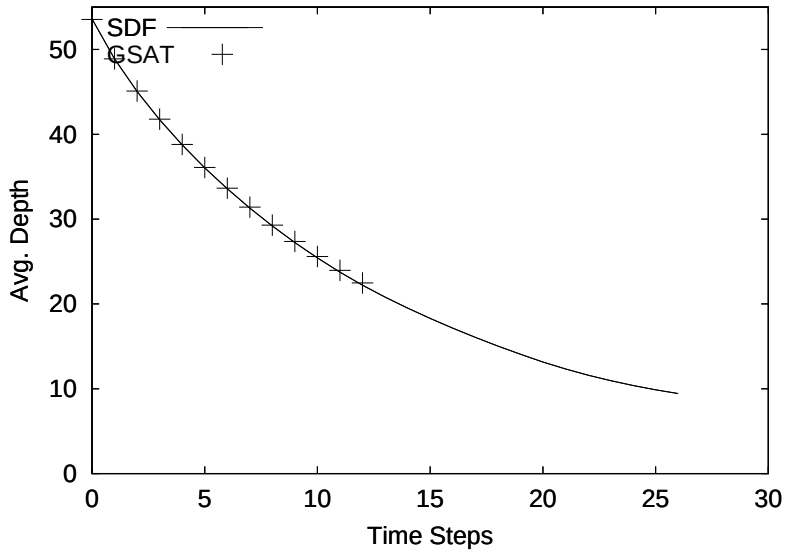

Figure 3: Average number of unsatisfied clauses (depth in g) achieved before reaching a local optimum or plateau. Results are obtained by running initial descents in g and $-f$ on uf100-0953 until the first local optimum or plateau point is reached, and then reporting the average g value at a given time point if at least 95 of 100 runs have successfully descended that many steps.

of the unsatisfied clauses to create a new search direction that allows the procedure to escape [5, 17, 21, 26, 27]. That is, one permits a few low weight clauses to be sacrificed in the interests of satisfying an important high weight clause. However, previous published re-weighting schemes all use *additive* updates to increment the clause weights. Unfortunately, additive updates do not work very well on difficult search problems because the clauses develop large weight differences over time, and this causes the update mechanism to lose its ability to rapidly adapt the weight profile to new regions of the search space. Multiplicative updating has the advantage of maintaining the ability to swiftly change the weight profile whenever necessary.

However, a final issue we faced in devising our update procedure was that persistently satisfied clauses would often lose their weight to the extent that they would become frequently falsified, and consequently the depth of search (as measured in g) would deteriorate. To cope with this effect, we flattened the weight profile of the satisfied clauses at each re-weighting by shrinking them towards their common mean. This increased the weights of satisfied clauses without requiring them to be explicitly falsified, and had the overall effect of restoring search depth and improving performance. The final SDF procedure we tested is summarized as follows.[9]

SDF(δ, ρ) Flip the variable x_i that leads to the greatest increase in the weighted objective $h(\mathbf{x}, \mathbf{w})$. If the current variable assignment is a local maximum and not a solution, then re-weight the clauses to create a new ascent direction and continue (see Figure 4).

Re-weight(δ, ρ) Multiplicatively re-weight the unsatisfied clauses and re-normalize the clause weights $\mathbf{w}$ so that the resulting largest difference in the $h(\mathbf{x}, \mathbf{w})$ objective (when flipping any one variable) is δ. (That is, create a new greedy search direction.) Then flatten the weight profile of the *satisfied* clauses by shrinking them $1 - \rho$ of the distance towards their common mean (to prevent the weights from becoming too small and causing clauses to be gratuitously falsified in the future). The technical details of this algorithm are outlined in Figure 5.

One interesting aspect of this procedure is that it is almost completely deterministic (given that ties are rare in the objective, and ignoring the application of random re-starts) and yet seems to perform very well in comparison

[9] Our code is publically available at http://ai.uwaterloo.ca/~dale/software/sdf/.

procedure SDF(δ, ρ)

Initialize the weight $w(c) := \frac{1}{m}$ for each clause c
Start with a random initial variable assignment $\mathbf{x}$
Compute $\Delta_{x_i}(\mathbf{x}, \mathbf{w}) := h(\mathbf{x}_{(x_i:=1-x_i)}, \mathbf{w}) - h(\mathbf{x}, \mathbf{w})$ for each variable x_i

while $\mathbf{x}$ is not a satisfying assignment
    $x_{i^*} := \arg\max_{x_i} \Delta_{x_i}(\mathbf{x}, \mathbf{w})$
    if $\Delta_{x_{i^*}}(\mathbf{x}, \mathbf{w}) < 0$
        $\mathbf{w} :=$ **Re-weight**$(\delta, \rho)(\mathbf{x}, \mathbf{w})$
        Recompute $\Delta_{x_i}(\mathbf{x}, \mathbf{w})$ for each x_i
        $x_{i^*} := \arg\max_{x_i} \Delta_{x_i}(\mathbf{x}, \mathbf{w})$
    end
    $x_{i^*} := 1 - x_{i^*}$
    Update $\Delta_{x_i}(\mathbf{x}, \mathbf{w})$ for each x_i that shares a common clause with x_{i^*}
end

Figure 4: Simple outline of the SDF procedure. Note that this description does not mention an outer re-start loop because we assume this can be implicitly added to any local search procedure, as discussed in Section 2. (An efficient implementation of this procedure requires the quantities Δ_{x_i} to be maintained implicitly and updated in a lazy manner that avoids explicitly looping through the entire set of clauses; see http://ai.uwaterloo.ca/~dale/software/sdf/ for details.)

procedure Re-weight(δ, ρ)

Let F = total weight of false clauses
S = total weight of satisfied clauses (note that $F + S = 1$)

For each variable x_i let

f_i^+ = total increment from currently false clauses (i.e., satisfied by $1 - x_i$)
s_i^+ = total increment from currently satisfied clauses (satisfied by $1 - x_i$)
s_i^- = total decrement from currently satisfied clauses (i.e., satisfied by x_i)

Note that $\Delta_{x_i} = f_i^+ + s_i^+ - s_i^-$ (defined in Figure 4) and also by the assumption of re-weighting we have $\Delta_{x_i} < 0$ for all x_i.

For each variable x_i such that $f_i^+ > 0$, compute

$$\alpha_i := \frac{s_i^- - s_i^+ + S\delta}{F(s_i^- - s_i^+) + S f_i^+} , \quad \beta_i := \frac{1 - F\alpha_i}{S}$$

This ensures the following two properties (proved in Appendix B):

Property 1: $\alpha_i F + \beta_i S = 1$, $\alpha_i > 1$, and $\beta_i < 1$
Property 2: if $f_i^+ > 0$, then $\Delta'_{x_i} = \alpha_i f_i^+ + \beta_i s_i^+ - \beta_i s_i^- = \delta$

Assign $\alpha_* := \min_i \alpha_i$, $\beta_* := \dfrac{1 - F\alpha_*}{S}$,

$$w(c) := \begin{cases} \alpha_* \, w(c) & \text{if } c \text{ currently false} \\ \beta_* \, w(c) & \text{if } c \text{ currently satisfied} \end{cases}$$

This ensures that $\Delta_{x_*} = \delta$ by Property 2, and $F + S = 1$ by Property 1.

To shrink the satisfied clause weights towards their common mean, assign

$$\overline{sat} := \frac{1}{\#sat} \sum_{c \in \text{sat}} w(c)$$

$$w(c) := (1 - \rho)\, \overline{sat} + \rho \, w(c)$$

Note that this still ensures $F + S = 1$ (Property 3, proved in Appendix B).

Figure 5: Conceptual outline of the multiplicative clause re-weighting procedure, demonstrating the required calculations and key properties. (An efficient implementation of this procedure requires the clause weights to be maintained implicitly and only updated when directly connected to a flipped variable; see http://ai.uwaterloo.ca/~dale/software/sdf/ for details.)

to the best current methods, all of which are highly stochastic. We claim that much of the reason for this success is that SDF maintains good depth, mobility, and coverage in its search. This is clearly demonstrated in all of the experiments of Section 3, where it is shown that SDF obtains superior measurements under every criteria.

5 Evaluation

We have conducted an evaluation of SDF on several thousand benchmark satisfiability problems from the SATLIB and DIMACS repositories. The results we have obtained appear to be interesting and encouraging. Comparing SDF to the very effective Novelty+ and WSAT procedures we find that SDF typically reduces the number of flips needed to find a satisfying assignment over the best of Novelty+ and WSAT by a factor of two to three on random satisfiable CNF formulae (in the uf, flat, and aim collections) and by a factor of five to ten on non-random CNF formulae (in the SATLIB blocks-world and ais problem sets). These results are consistent across the range of the problems we have investigated, and hold up even when considering the mean flips without restart, median flips, and optimal expected flips using restarts, $\hat{\mathrm{E}}(T_{t^*})$, estimated from (1). Although much simpler, SDF appears to be competitive with DLM in terms of search steps as well (although not in terms of CPU time).

In our experiments, each search procedure was run 100 times on each problem and the results were averaged over these runs. The numbers reported are average number of flips needed to find a solution, the estimated optimal number of flips needed under an optimal restart scheme (estimated using (3)), the percentage of failures over 100 repetitions (cut off at 500,000 flips),[10] and the average CPU time per flip (in milliseconds on a PIII 450MHz processor). The methods we tested were SDF, DLM, Novelty, Novelty+, and WSAT, and on smaller problems HSAT, GSAT, and simulated annealing. To ensure that we compared SDF to other state of the art clause re-weighting schemes, we included the most recent version of DLM [27] in our experiments. All parameter settings used for each procedure are indicated in parentheses.[11]

[10] All problems were cut off at 500,000 flips except bw_large.c (1,000,000 flips) and bw_large.d (5,000,000 flips).

[11] All parameter settings are shown explicitly, with the exception of SDF where only the value if δ is given (since ρ is fixed to 0.995 in all cases), and DLM where the 22 parameters

There are two broad classes of experiments reported in Tables 10 to 17. The first class, Tables 10 to 14, covers a wide array of *random* satisfiability problems from both the SATLIB and DIMACS repositories. The second class of experiments, Tables 15 to 17, covers large *structured* problems in these collections. The results in all cases are averaged over all problems in the respective problem set.

The results on the second class of (structured) problems are particularly striking. These problems challenge the previous generation of local search methods (WSAT, Novelty and Novelty+) and yet SDF and DLM both appear to solve them with significantly fewer search steps. This suggests that, although SDF and DLM shares many similarities to other local search methods currently in use, they might offer a qualitatively different approach that could yield benefits in real world problems.

To compare SDF to DLM directly, we find that they generally achieved solutions in about the same number of steps. SDF has a significant flips advantage on flat and ais-12, but DLM holds a significant advantage on aim-100. However, SDF is consistently a factor of three to four times slower than DLM in terms of CPU overhead, and therefore SDF is not competitive with DLM in terms of CPU time. Nevertheless, SDF is competitive with the previous generation systems (WSAT, Novelty+) in CPU time on large structured problems such as ais, bw_large and logistics.[12]

It is obvious that our current implementation of SDF is not without its limitations. We are currently using a floating-point implementation that maintains a few additional quantities over DLM, and therefore even though SDF executes a comparable number of search steps (flips) to solve most problems, each search step is more expensive to compute. (DLM almost entirely avoids the use of floating point arithmetic.) However, the complexity of DLM's main loop is in principle no simpler than SDF's, and we are hopeful that further optimizations might improve the relative performance. Unfortunately, our floating point implementation creates the additional disadvantage of round off errors which eventually cause the variable scores and clause weights to drift out of agreement with one another (primarily due to

were set by testing each of the 5 default parameter sets given at http://manip.crhc.uiuc.edu and choosing the best of these sets.

[12]The similarity in flips performance between SDF and DLM suggests that the simpler SDF procedure might be capturing some essense of the complex DLM system that accounts for its performance advantage over WSAT and Novelty. We hope to investigate this question further in future research.

the use of an efficient lazy weight updating scheme). To compensate, we have to run periodic correction loops in SDF to explicitly re-calculate these quantities—which adds to the overall computational overhead. It remains an open question as to whether a fundamentally faster algorithm in terms of CPU time can be developed from the ideas presented in this paper.

6 Conclusion

We have introduced three simple measures of local search performance—depth, mobility, and coverage—and demonstrated how achieving adequate measures under these criteria appears to be necessary for achieving effective problem solving performance using greedy local search for satisfiability problems. We provided two general pieces of evidence to support each claim of necessity: First, we demonstrated that superior local search procedures exhibit uniformly good scores under all three criteria, whereas inferior search procedures always exhibit significantly weaker scores on at least one criterion. Second, even among superior search procedures, there is a strong positive correlation between problem solving performance on one hand, and good measures under each of the search criteria on the other—and this holds for each of the three measures independently. Finally, to address the question of sufficiency, we provided experimental evidence that good performance under all three measures simultaneously appears to be (at least weakly) sufficient for effective local search performance. Our evidence for this claim is given by the observation that an improved rating under any one measure (holding the others fixed) systematically yields a predictive correlation with reduced number of search steps, roughly independent of other algorithmic details.

With the intent of creating a local search procedure that exhibits superior measures on all three criteria, we developed the new procedure SDF, which employs a smoothed version of the standard GSAT objective to obtain better depth scores, and uses multiplicative clause-weight updating to obtain better mobility and coverage scores than existing methods. The outcome is a simple search procedure that systematically reduces the number of search steps (flips) needed to find a solution over older methods, and competes very closely with the state of the art DLM system. Although SDF does not yet deliver an improvement in CPU time (owing to its greater computational overhead per flip) the improvement in terms of flips does vindicate our claim that improvements under the three measurable search criteria should predict

reductions in the number of search steps needed to find satisfying assignments.

By no means does SDF close any line of research on local search procedures for satisfiability, nor does it settle the question about which are the best methods for simultaneously optimizing the three proposed criteria. One of the important directions for future research is to investigate other forms of global search strategies for satisfiability (for example, population-based and global-stochastic search methods) and see if any significant further gains can be made. Even for the SDF procedure itself, we continue to investigate simplifications that might allow reductions in CPU time in addition to improvements in overall numerical stability. Finally, it remains future work to conduct a theoretical analysis of the search criteria we have proposed and verify whatever guarantees they might offer for local search performance, as well as discover what the ultimate limitations of these measures are.

Acknowledgements

Research supported by NSERC, MITACS, CITO and Bell University Labs. Thanks to Fahiem Bacchus, Rob Holte, Holger Hoos, and Bart Selman for very helpful comments.

flat100	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	44,467	34,391	.4	2.9
Novelty(.5)	27,120	13,689	.5	2.7
Novelty+(.5, .01)	23,995	13,893	.2	2.9
DLM(pars4)	9,571	8,314	0	4.8
SDF(.0002)	7,207	6,478	0	21.8

flat125	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	74,517	62,390	1.5	2.9
Novelty(.5)	42,276	27,660	1.3	2.8
Novelty+(.5, .01)	37,408	24,440	.8	3.0
DLM(pars4)	26,182	21,506	0	4.8
SDF(.0002)	15,277	13,313	0	22.0

flat150	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	148,095	136,207	8.6	5.8
Novelty(.5)	91,004	62,985	3.2	5.0
Novelty+(.5, .01)	79,601	59,928	2.3	5.2
DLM(pars4)	69,779	54,637	1.0	8.9
SDF(.0001)	35,967	30,556	.05	26.4

flat200	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	300,471	297,584	36	3.1
Novelty(.5)	240,829	206,455	27	3.0
Novelty+(.5, .01)	221,257	192,575	22	3.2
DLM(pars4)	280,401	242,439	31	5.8
SDF(.0001)	142,277	115,440	7	26.9

Table 10: flat problems (100 problems each)

uf50	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	652	502	0	4.2
Novelty(.5)	837	270	.08	3.6
Novelty+(.5, .01)	411	272	0	4.1
DLM(pars4)	186	160	0	6.4
SDF(.003)	154	139	0	12.3

uf75	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	1841	1424	0	4.2
Novelty(.5)	1798	693	.15	3.7
Novelty+(.5, .01)	1001	685	0	4.1
DLM(pars4)	515	404	0	5.7
SDF(.0015)	441	377	0	14.1

uf100	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	3655	2862	0	4.3
Novelty(.5)	3801	1467	.25	3.8
Novelty+(.5, .01)	2298	1448	0	4.2
DLM(pars4)	1020	800	0	5.8
SDF(.00085)	870	725	0	16.3

uf125	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	8926	5772	.05	4.5
Novelty(.5)	6737	2951	.19	4.0
Novelty+(.5, .01)	5174	2964	0	4.3
DLM(pars4)	2096	1573	0	5.8
SDF(.0006)	1880	1534	0	18.4

Table 11: Small uf problems (100 problems each, except uf50 and uf100 which have 1000 problems each)

uf150	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	14,331	9504	.3	4.5
Novelty(.5)	9573	4604	.15	4.2
Novelty+(.5, .01)	8331	4456	.03	4.5
DLM(pars4)	3263	2455	0	6.0
SDF(.00065)	3164	2357	0	19.6

uf175	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	27,136	18,112	.71	4.6
Novelty(.5)	19,649	8223	.70	4.4
Novelty+(.5, .01)	18,609	9879	.68	4.6
DLM(pars4)	6819	4923	0	6.1
SDF(.0005)	6714	5109	0	22.3

uf200	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	41,377	32,366	2.9	4.7
Novelty(.5)	31,794	25,127	2.5	4.7
Novelty+(.5, .01)	28,529	17,953	2.3	4.9
DLM(pars4)	12,215	9,020	.1	6.2
SDF(.0003)	14,484	11,304	.4	23.5

uf225	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	44,908	34,729	2.4	4.8
Novelty(.5)	41,794	22,953	4.0	4.6
Novelty+(.5, .01)	38,158	23,046	3.3	4.8
DLM(pars4)	16,098	10,072	0	6.4
SDF(.00025)	16,708	10,478	.3	25.0

Table 12: Larger uf problems (100 problems each)

uf250	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	41,049	34,212	1.6	4.9
Novelty(.5)	32,864	24,439	2.1	4.7
Novelty+(.5, .01)	31,560	21,434	2.2	4.9
DLM(pars4)	22,635	12,387	.3	6.6
SDF(.0002)	20,331	13,573	.3	27.0

Table 13: Largest uf problems (uf250), 100 problems

jnh	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	4682	3655	0	8.4
Novelty(.5)	4128	2227	.2	7.8
Novelty+(.5, .01)	3469	2227	0	8.2
DLM(pars4)	879	723	0	11.4
SDF(.0005)	890	712	0	49.9

aim-100	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	252,380	251,864	50	2.1
Novelty(.5)	251,489	251,100	50	1.7
Novelty+(.5, .01)	251,337	250,771	50	1.9
DLM(pars4)	34,721	32,814	4.1	2.6
SDF(.0005)	106,514	104,304	16	15.5

aim-200	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	271,141	258,947	51	2.3
Novelty(.5)	270,153	255,546	50	1.9
Novelty+(.5, .01)	267,611	255,071	50	2.1
DLM(pars4)	233,651	233,557	42	2.9
SDF(.00025)	251,358	251,263	50	22.5

Table 14: jnh and aim problems (16 problems each)

ais6	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	1377	1246	0	6.4
Novelty(.5)	460,007	1299	92	6.1
Novelty+(.5, .01)	12,031	794	0	6.3
DLM(pars4)	410	406	0	9.2
SDF(.0015)	435	434	0	20.6

ais8	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	36,499	33,845	0	9.7
Novelty(.5)	495,003	18,800	99	9.7
Novelty+(.5, .01)	169,626	137,436	8	9.9
DLM(pars4)	5376	5326	0	11.3
SDF(.0004)	4490	4419	0	36.5

ais10	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	317,323	317,323	37	13.2
Novelty(.5)	500,000	na	100	13.9
Novelty+(.5, .01)	451,222	273,300	84	14.1
DLM(pars4)	18,420	14,306	0	16.4
SDF(.00013)	15,000	13,941	0	57.3

ais12	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	486,698	486,698	95	17.9
Novelty(.5)	500,000	na	100	18.7
Novelty+(.5, .01)	497,518	497,518	99	18.6
DLM(pars4)	215,360	179,383	7	17.3
SDF(.0001)	132,021	97,600	6	76.9

Table 15: ais (All-Interval series) problems (1 problem each)

bw_large.a	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	20,336	18,452	0	8.2
Novelty(.5)	9695	9310	0	8.3
Novelty+(.5, .01)	10,788	10,323	0	8.4
DLM(pars4)	3712	3701	0	14.9
SDF(.0001)	2993	2970	0	46.8

bw_large.b	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	377,348	377,348	58	11.7
Novelty(.5)	343,078	343,078	48	11.4
Novelty+(.5, .01)	373,001	217,394	53	11.7
DLM(pars4)	44,361	39,216	0	18.3
SDF(.00005)	33,442	33,255	0	85.4

bw_large.c	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	1,000,000	na	100	20.3
Novelty(.5)	1,000,000	na	100	19.9
Novelty+(.5, .01)	1,000,000	na	100	20.6
DLM(pars4)	895,213	895,213	77	33.3
SDF(.00002)	921,179	921,971	85	171

bw_large.d	Avg. Flips	Opt. Flips	Fail%	msec. per flip
DLM(pars4)	4,635,200	4,635,200	80	57.0
SDF(.00002)	4,909,150	4,909,150	90	217

Table 16: bw_large problems (1 problem each)

medium	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	1167	1064	0	5.5
Novelty(.5)	392	392	0	6.3
Novelty+(.5, .01)	537	532	0	6.1
DLM(pars4)	265	265	0	12.6
SDF(.0005)	265	263	0	19.2

huge	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	20,211	20,161	0	11.1
Novelty(.5)	11,382	10,774	0	10.9
Novelty+(.5, .01)	12,419	10,342	0	11.2
DLM(pars4)	3658	3532	0	20.6
SDF(.0001)	3002	3002	0	47.9

logistics.c	Avg. Flips	Opt. Flips	Fail%	msec. per flip
WSAT(.5)	332,822	332,822	42	12.1
Novelty(.5)	135,382	135,382	2	8.9
Novelty+(.5, .01)	163,622	163,622	1	9.0
DLM(pars4)	12,101	11,805	0	24.7
SDF(8.76×10^{-6})	15,939	15,939	0	173

Table 17: Other structured problems (1 problem each)

A Effects of random re-starts

It can be shown that imposing an ideal re-start value t^* on the random search time T yields an improvement in expected time for most natural search distributions. To see this, note that for any random variable T we have

$$\mathrm{E}(T) \;=\; \mathrm{E}(T|T\leq t)\;\mathrm{P}(T\leq t) \;+\; \mathrm{E}(T|T>t)\;\mathrm{P}(T>t) \tag{6}$$

From (2) we know that the expected search time when using a random restart after t flips is

$$\begin{aligned}
\mathrm{E}(T_t) &= \frac{t}{\mathrm{P}(T\leq t)} - [t - \mathrm{E}(T|T\leq t)] \\
&= t\,\left(\frac{1}{\mathrm{P}(T\leq t)} - 1\right) + \mathrm{E}(T|T\leq t) \\
&= t\,\frac{\mathrm{P}(T>t)}{\mathrm{P}(T\leq t)} + \mathrm{E}(T|T\leq t)
\end{aligned} \tag{7}$$

To determine whether (7) offers an improvement over (6) first note that for an exponential random variable T (i.e., such that $\mathrm{P}(T>x) = e^{-x/\mu}$ for all $x>0$) we actually have $\mathrm{E}(T) = \mathrm{E}(T_t)$ for every cutoff value $t>0$. This follows from the "memoryless property" of exponential random variables which states that $\mathrm{P}(T>t+x|T>t) = \mathrm{P}(T>x)$ for all $t>0$ and $x>0$ [19], and immediately implies that

$$\mathrm{E}(T|T>t) \;=\; t + \mathrm{E}(T) \tag{8}$$

for all $t>0$. Thus

$$\begin{aligned}
\mathrm{E}(T) &= \mathrm{E}(T|T>t)\;\mathrm{P}(T>t) \;+\; \mathrm{E}(T|T\leq t)\;\mathrm{P}(T\leq t) \\
&= t\;\mathrm{P}(T>t) \;+\; \mathrm{E}(T)\;\mathrm{P}(T>t) \;+\; \mathrm{E}(T|T\leq t)\;\mathrm{P}(T\leq t)
\end{aligned} \tag{9}$$

from (8), and therefore for exponential random variables we always have

$$\mathrm{E}(T)\;\mathrm{P}(T\leq t) \;=\; t\;\mathrm{P}(T>t) \;+\; \mathrm{E}(T|T\leq t)\;\mathrm{P}(T\leq t)$$

and hence

$$\begin{aligned}
\mathrm{E}(T) &= t\frac{\mathrm{P}(T>t)}{\mathrm{P}(T\leq t)} \;+\; \mathrm{E}(T|T\leq t) \\
&= \mathrm{E}(T_t)
\end{aligned} \tag{10}$$

for all $t > 0$, as stated.

So imposing a random restart after a cutoff value t does not affect the expectation of any exponential random variable T. It might therefore appear that random restarts may not offer a useful improvement in general. However, the equality (10) only holds because of the memoryless property (8), and it turns out that this property holds for all $t > 0$ *only* for exponential random variables [19].

By contrast, it is widely believed that the search time distributions for heuristic search algorithms exhibit a "heavy tailed" behavior on difficult constraint satisfaction problems [10]. That is, the tail of the search time distribution $\mathrm{P}(T > x)$ is not exponential, but instead is typically a much slower converging function such as a power law $\mathrm{P}(T > x) = Cx^{-\alpha}$ for $\alpha > 0$, $C > 0$ (where the tails become heavier for smaller α) [10]. A simple example of a power law distribution is a Pareto density $p(x) = \alpha(1+x)^{-\alpha-1}$, $x > 0$, $\alpha > 0$, which defines a random variable T such that $\mathrm{P}(T > x) = (1+x)^{-\alpha}$. The kth moments of power law distributions are only defined for $k < \alpha$, and in particular their expected values become infinite for $\alpha \leq 1$.

For heavy tailed distributions in general, it is easy to demonstrate that a random re-start strategy will always yield a reduction in expected search time. To see this, first consider the case of a heavy tailed distribution with a finite expected search time; that is, a power law distribution such that $\alpha > 1$. An interesting property of such a distribution is that it is no longer memoryless. In fact, the additional expected search time actually *increases* given that an early solution has not been found:

$$\mathrm{E}(T|T>t) \quad > \quad t + \mathrm{E}(T) \tag{11}$$

for $t > 0$, which is in direct contrast to (8). From this property it immediately follows that a random re-start strategy yields an improvement in expected solution time, since

$$\begin{aligned}
\mathrm{E}(T) &= \mathrm{E}(T|T>t)\ \mathrm{P}(T>t) \ +\ \mathrm{E}(T|T\leq t)\ \mathrm{P}(T\leq t) \\
&> t\ \mathrm{P}(T>t) \ +\ \mathrm{E}(T)\ \mathrm{P}(T>t) \ +\ \mathrm{E}(T|T\leq t)\ \mathrm{P}(T\leq t) \\
&= \mathrm{E}(T_t)
\end{aligned}$$

For example, for a Pareto distribution with $\alpha > 1$ we have $\mathrm{E}(T|T > f) = t + \mathrm{E}(T) + t\mathrm{E}(T)$ for all $t > 0$ and hence immediately obtain $\mathrm{E}(T_t) < \mathrm{E}(T)$.

For the more extreme case of a heavy tailed distribution with an *infinite* expected value (i.e., when $\alpha \leq 1$) it is obvious that a random re-start strategy significantly improves search time, since $\mathrm{E}(T_t)$ is clearly finite (7) for any t such that $\mathrm{P}(T \leq t) > 0$ by inspection.

Therefore, in general, using a random restart strategy with a cutoff value

$$t^* \quad = \quad \arg \min_{t:0<t\leq\infty} \mathrm{E}(T_t)$$

(where $\mathrm{E}(T_\infty)$ is defined to be $\mathrm{E}(T)$) should improve expected search time, since if (11) is satisfied for any t we will immediately obtain $\mathrm{E}(T_{t^*}) < \mathrm{E}(T)$, and otherwise if no such value of t exists we obtain $\mathrm{E}(T_{t^*}) = \mathrm{E}(T)$ (and hence cause no harm). Therefore, under any circumstance, employing a random restart strategy after an ideal cutoff value t^* is never a losing strategy.

B Properties of clause re-weighting scheme

Proof of Property 1. First, to show that $\alpha_i F + \beta_i S = 1$, note

$$\begin{aligned} \alpha_i F + \beta_i S &= \alpha_i F + \frac{1 - F\alpha_i}{S} S \\ &= \alpha_i F + 1 - F\alpha_i \\ &= 1 \end{aligned}$$

Next, to verify that $\alpha_i > 1$, recall that by assumption $\Delta_{x_i} = f_i^+ + s_i^+ - s_i^- < 0 < \delta$, and therefore

$$\begin{aligned} \alpha_i &= \frac{s_i^- - s_i^+ + S\delta}{F(s_i^- - s_i^+) + Sf_i^+} \\ &> \frac{s_i^- - s_i^+ + S(f_i^+ + s_i^+ - s_i^-)}{F(s_i^- - s_i^+) + Sf_i^+} \\ &= \frac{(1-S)(s_i^- - s_i^+) + Sf_i^+}{F(s_i^- - s_i^+) + Sf_i^+} \\ &= \frac{F(s_i^- - s_i^+) + Sf_i^+}{F(s_i^- - s_i^+) + Sf_i^+} \\ &= 1 \end{aligned}$$

Finally, to verify that $\beta_i < 1$, note that $\beta_i = (1-F\alpha_i)/S < (1-F)/S = 1$.

Proof of Property 2. Assume $F > 0$ and $f_i^+ > 0$. First note that

$$\begin{aligned}
\alpha_i &= \frac{s_i^- - s_i^+ + S\delta}{Sf_i^+ + F(s_i^- - s_i^+)} \\
&= \frac{\frac{s_i^- - s_i^+}{Sf_i^+} + \frac{\delta}{f_i^+}}{1 + \frac{F(s_i^- - s_i^+)}{Sf_i^+}} \\
&= \frac{s_i^- - s_i^+}{Sf_i^+} + \frac{\delta}{f_i^+} - \frac{F\alpha_i(s_i^- - s_i^+)}{Sf_i^+} \qquad (12) \\
&= \left(\frac{1 - F\alpha_i}{S}\right)\frac{s_i^- - s_i^+}{f_i^+} + \frac{\delta}{f_i^+} \\
&= \frac{\beta_i(s_i^- - s_i^+) + \delta}{f_i^+}
\end{aligned}$$

where the second step (12) uses the fact that $\alpha = \frac{a}{1+b}$ if and only if $\alpha = a - b\alpha$. Thus $\alpha_i f_i^+ + \beta_i(s_i^+ - s_i^-) = \beta_i(s_i^- - s_i^+) + \delta + \beta_i(s_i^+ - s_i^-) = \delta$.

Proof of Property 3. Recall that before shrinking the satisfied clause weights towards their common mean we have $S = \sum_{c\in\mathrm{sat}} w(c)$ and $\overline{sat} = S/(\#sat)$. Let $w'(c) = (1-\rho)\,\overline{sat} + \rho\, w(c)$. We then obtain

$$\begin{aligned}
S' &= \sum_{c\in\mathrm{sat}} w'(c) \\
&= (1-\rho)\overline{sat}(\#sat) + \rho \sum_{c\in\mathrm{sat}} w(c) \\
&= (1-\rho)\sum_{c\in\mathrm{sat}} w(c) + \rho \sum_{c\in\mathrm{sat}} w(c) \\
&= S
\end{aligned}$$

and therefore $S' + F = S + F = 1$.

References

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 256–261, 2000.

[2] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 203–208, 1997.

[3] B. Cha and K. Iwama. Performance test of local search algorithms using new types of random CNF formulas. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 304–310, 1995.

[4] B. Cha and K. Iwama. Adding new clauses for faster local search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 332–337, 1996.

[5] J. Frank. Weighting for Godot: Learning heuristics for GSAT. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 338–343, 1996.

[6] J. Frank. Learning short-tem weights for GSAT. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 384–391, 1997.

[7] I. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-93)*, pages 28–33, 1993.

[8] I. Gent and T. Walsh. Unsatisfied variables in local search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions (AISB-95)*. IOS Press, Amsterdam, 1995.

[9] C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437, 1998.

[10] C. Gomes, B. Selman, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Automat. Reas.*, 24(1):67–100, 2000.

[11] H. Hoos. On the run-time behavior of stochastic local search algorithms for SAT. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 661–666, 1999.

[12] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, 1996.

[13] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Trans. Computer-Aided Design*, 11(1):4–15, 1992.

[14] C. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, 1997.

[15] B. Mazure, L. Saïs, and É. Grégoire. Tabu search for SAT. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 281–285, 1997.

[16] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 321–326, 1997.

[17] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-93)*, pages 40–45, 1993.

[18] A. Parkes and J. Walser. Tuning local search for satisfiability testing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 356–362, 1996.

[19] S. Ross. *Introduction to Probability Models*. Academic Press, San Diego, 6th edition, 1997.

[20] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 297–302, 2000.

[21] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1993.

[22] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, 1994.

[23] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 50–54, 1997.

[24] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

[25] Y. Shang and W. Wah. A discrete Lagrangian based global search method for solving satisfiability problems. *J. Global Optimization*, 12(1):61–99, 1998.

[26] Z. Wu and W. Wah. Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 673–678, 1999.

[27] Z. Wu and W. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 310–315, 2000.