This paper appears in
Proceedings of AAAI-94, Seattle, WA, July 1994.

Noise Strategies for Improving Local Search

Bart Selman, Henry A. Kautz, and Bram Cohen
AT&T Bell Laboratories
Murray Hill, NJ 07974
{selman, kautz, cohen}@research.att.com

Abstract

It has recently been shown that local search is surprisingly good at finding satisfying assignments for certain computationally hard classes of CNF formulas. The performance of basic local search methods can be further enhanced by introducing mechanisms for escaping from local minima in the search space. We will compare three such mechanisms: simulated annealing, random noise, and a strategy called "mixed random walk". We show that mixed random walk is the superior strategy. We also present results demonstrating the effectiveness of local search with walk for solving circuit synthesis and circuit diagnosis problems. Finally, we demonstrate that mixed random walk improves upon the best known methods for solving MAX-SAT problems.

Introduction

Local search algorithms have been successfully applied to many optimization problems. Hansen and Jaumard (1990) describe experiments using local search for MAX-SAT, *i.e.*, the problem of finding an assignment that satisfies as many clauses as possible of a given CNF formula. In general, such local search algorithms find good but non-optimal solutions, and thus such algorithms were believed not to be suitable for satisfiability testing, where the objective is to find an assignment that satisfies *all* clauses (if such an assignment exists).

Recently, however, local search has been shown to be surprisingly good at finding completely satisfying assignments for CNF problems (Selman *et al.* 1992; Gu 1992). Such methods outperform the best known systematic search algorithms on certain classes of large satisfiability problems. For example, GSAT, a randomized local search algorithm, can solve 2,000 variable computationally hard randomly-generated 3CNF (conjunctive normal form) formulas, whereas the current fastest systematic search algorithms cannot handle instances from the same distribution with more than 400 variables (Buro and Kleine-Büning 1992; Dubois *et al.* 1993).

The basic GSAT algorithm performs a local search of the space of truth-assignments by starting with a randomly-generated assignment, and then repeatedly changing ("flipping") the assignment of a variable that leads to the largest decrease in the total number of unsatisfied clauses. As with any combinatorial problem, local minima in the search space are problematic in the application of local search methods. A local minimum is defined as a state whose local neighborhood does not include a state that is strictly better. The standard approach in combinatorial optimization of terminating the search when a local minimum is reached (Papadimitriou and Steiglitz 1982) does not work well for Boolean satisfiability testing, since only global optima are of interest. In Selman *et al.* (1992) it is shown that simply continuing to search by making non-improving, "sideways" moves, dramatically increases the success rate of the algorithm.[1] We call the set of states explored in a sequence of sideways moves a "plateau" in the search space. The search along plateaus often dominates GSAT's search. For a detailed analysis, see Gent and Walsh (1992).

The success of GSAT is determined by its ability to move between successively lower plateaus. The search fails if GSAT can find no way off of a plateau, either because such transitions from the plateau are rare or nonexistent. When this occurs, one can simply restart the search at new random initial assignment. There are other mechanisms for escaping from local minima or plateaus, which are based on occasionally making uphill moves. Prominent among such approaches has been the use of simulated annealing (Kirkpatrick *et al.* 1982), where a formal parameter (the "temperature") controls the probability that the local search algorithm makes an uphill move.

Selman and Kautz (1993) proposed another mechanism for introducing such uphill moves. The strategy is based on mixing a random walk over variables that appear in unsatisfied clauses with the greedy local search. The strategy can be viewed as a way of

[1] Minton *et al.* (1990) encountered a similar phenomenon in their successful application of local search in solving large scheduling problems.

introducing noise in a very focused manner — namely, perturbing only those variables critical to to the remaining unsatisfied clauses.

We will present detailed experimental data comparing the random walk strategy, simulated annealing, random noise, and the basic GSAT procedure on computationally difficult random formulas. In doing this comparison, we tuned the parameter settings of each procedure to obtain their best performance. We will see that the random walk strategy significantly outperforms the other approaches, and that all the escape strategies are an improvement over basic GSAT.

One might speculate that the good performance of the random walk strategy is a consequence of our choice of test instances. We therefore also ran experiments using several other classes of problem instances, developed by transforming other combinatorial problems into satisfiability instances. In particular, we considered problems from planning (Kautz and Selman 1992) and circuit synthesis (Kamath *et al.* 1991; 1993). These experiments again demonstrate that mixed random walk is the superior escape mechanism. In addition, we show that GSAT with walk is faster than systematic search on certain circuit synthesis problems (such as adders and comparators) that contain *no* random component. We then present data on experiments with a modified version of the random walk strategy that further improves performance over GSAT with walk. Finally, we demonstrate that mixed random walk also improves upon the best known methods for solving MAX-SAT problems.

Local Search for Satisfiability Testing

GSAT (Selman *at al.* 1992) performs a greedy local search for a satisfying assignment of a set of propositional clauses.[2] The procedure starts with a randomly generated truth assignment. It then changes ("flips") the assignment of the variable that leads to the greatest decrease in the total number of unsatisfied clauses. Note that the greatest decrease may be zero (sideways move) or negative (uphill move). Flips are repeated until either a satisfying assignment is found or a pre-set maximum number of flips (MAX-FLIPS) is reached. This process is repeated as needed up to a maximum of MAX-TRIES times.

In Selman *et al.* (1992), it was shown that GSAT substantially outperforms backtracking search procedures, such as the Davis-Putnam procedure, on various classes of formulas, including hard randomly generated formulas and SAT encodings of graph coloring problems (Johnson *et al.* 1991).

As noted above, local minima in the search space of a combinatorial problem are the primary obstacle to the application of local search methods. GSAT's use of sideways moves does not completely eliminate this problem, because the algorithm can still become stuck on a plateau (a set of neighboring states each with an equal number of unsatisfied clauses). Therefore, it is useful to employ mechanisms that escape from local minima or plateaus by making uphill moves (flips that increase the number of unsatisfied clauses). We will now discuss two mechanisms for making such moves.[3]

Simulated Annealing

Simulated annealing introduces uphill moves into local search by using a noise model based on statistical mechanics (Kirkpatrick *et al.* 1983). We employ the annealing algorithm defined in Johnson *et al.* (1991): Start with a randomly generated truth assignment. Repeatedly pick a random variable, and compute δ, the change in the number of unsatisfied clauses when that variable is flipped. If $\delta \leq 0$ (a downhill or sideways move), make the flip. Otherwise, flip the variable with probability $e^{-\delta/T}$, where T is a formal parameter called the *temperature*. The temperature may be either held constant,[4] or slowly decreased from a high temperature to near zero according to a cooling schedule. One often uses *geometric* schedules, in which the temperature is repeatedly reduced by multiplying it by a constant factor (< 1).

Given a finite cooling schedule, simulated annealing is not guaranteed to find a global optimum — that is, an assignment that satisfies all clauses. Therefore in our experiments we use multiple random starts, and compute the average number of restarts needed before finding a solution. We call this number R.

The basic GSAT algorithm is very similar to annealing at temperature zero, but differs in that GSAT naturally employ restarts and *always* makes a downhill move if one is available. The value of R for GSAT is simply the average number of tries required to find a solution.

The Random Walk Strategy

Selman and Kautz (1993) introduce several extensions to the basic GSAT procedure. One of those extensions mixes a random walk strategy with the greedy local search. More precisely, they propose the following **mixed random walk strategy:**

With probability p, pick a variable occuring in some unsatisfied clause and flip its truth assignment.
With probability $1-p$, follow the standard GSAT scheme, *i.e.*, make the best possible local move.

[2] A clause is a disjunction of literals. A literal is a propositional variable or its negation. A set of clauses corresponds to a CNF formula: a conjunction of disjunctions.

[3] If the *only* possible move for GSAT is uphill, it will make such a move, but such "forced" uphill moves are quite rare, and are not effective in escaping from local minima or plateaus.

[4] This form of annealing corresponds to the Metropolis algorithm (Jerrum 1992). See Pinkas and Dechter (1992), for an interesting modification of the basic annealing scheme.

Note that the "walk" moves can be uphill.

A natural and simpler variation of the random walk strategy is not to restrict the choice of a randomly flipped variable to the set of variables that appear in unsatisfied clauses. We will refer to this modification as the *random noise* strategy. Note that random walk differs from both simulated annealing and random noise, in that in random walk upward moves are closely linked to unsatisfied clauses. The experiments discussed below will show that the random walk strategy is generally significantly better.

Experimental Results

We compared the basic GSAT algorithm, simulated annealing, random walk, and random noise strategies on a test suite including both randomly-generated CNF problems and Boolean encodings of other combinatorial problems. The results are given in the Tables 1, 2, and 3. For each strategy we give the average time in seconds it took to find a satisfying assignment,[5] the average number of flips it required, and R, the average number of restarts needed before finding a solution. For each strategy we used at least 100 random restarts (MAX-TRIES setting in GSAT) on each problem instance; if we needed more than 20 restarts before finding a solution, the strategy was restarted up to 1,000 times. A "*" in the tables indicates that no solution was found after running for more than 10 hours or using more than 1,000 restarts.

The parameters of each method were varied over a range of values, and only the results of the best settings are included in the table. For basic GSAT, we varied MAX-FLIPS and MAX-TRIES; for GSAT with random walk, we also varied the probability p with which a non-greedy move is made, and similarly for GSAT with random noise. In all of our experiments, the optimal value of p was found to be between 0.5 and 0.6. For constant temperature simulated annealing, we varied the temperature T from 5 to 0 in steps of 0.05. (At $T = 5$, uphill moves are accepted with probability greater than 0.8.) For the random formulas, the best performance was found at $T = 0.2$. The planning formulas required a higher temperature, $T = 0.5$, while the Boolean circuit synthesis were solved most quickly at a low temperature, $T = 0.15$.

We also experimented with various geometric cooling schedules. Surprisingly, we did not find any geometric schedule that was better than the best constant-temperature schedule. We could not even significantly improve the average number of restarts needed before finding a solution by extremely slow cooling schedules, regardless of the effect on execution time. An possible explanation for this is that almost all the work in solving CNF problems lies in satisfying the last few unsatisfied clauses. This corresponds to the low-temperature tail of a geometric schedule, where the temperature has little variation.

Hard Random Formulas

Random instances of CNF formulas are often used in evaluating satisfiability procedures because they can be easily generated and lack any underlying "hidden" structure often present in hand-crafted instances. Unfortunately, unless great care is taken in specifying the parameters of the random distribution, the problems so created can be trivial to solve. Mitchell *et al.* (1992) show how computationally difficult random problems can be generated using the fixed-clause length model. Let N be the number of variables, K the number of literals per clause, and L the number of clauses. Each instance is obtained by generating L random clauses each containing K literals. The K literals are generated by randomly selecting K variables, and each of the variables is negated with a 50% probability. The difficulty of such formulas critically depends on the ratio between N and L. The hardest formulas lie around the region where there is a 50% chance of the randomly generated formula being satisfiable. For 3CNF formulas ($K = 3$), experiments show that this is the case for $L \approx 4.3N$. (For larger N the the critical ratio for the 50% point converges to 4.25.) We tested the algorithms on formulas around the 4.3 point ranging in size from 100 to 2000 variables.

Table 1 presents our results. For the smallest (100-variable) formula, we observe little difference in the running times. As the number of variables increase, however, the random walk strategy significantly dominates the other approaches. Both random noise and simulated annealing also improve upon basic GSAT, but neither of these methods found solutions for largest three formulas.[6] The performance of GSAT with walk is quite impressive, especially consider that fact that fastest current systematic search methods cannot solve hard random 3CNF instances with over 400 variables (Dubois *et al.* 1993).

The columns marked with "flips" give the average number of flips required to find an assignment. (A "flip" in our simulated annealing algorithm is an actual change in the truth assignment. We do not count flips that were considered but not made.) When comparing the number of flips required by the various strategies, we arrive at the same conclusion about the relative efficiencies of the methods. This shows that our observations based on the running times are not simply a consequence of differences in the relative efficiencies of our implementations.

Finally, let us consider R, the average number of restarts needed before finding a solution. Basic GSAT

[5] The algorithms were implemented in C and ran on an SGI Challenge with a 70 MHz MIPS R4400 processor. For code and experimental data, contact the first author.

[6] GSAT with walk finds approximately 50% of the formulas in the hard region to be satisfiable, as would be expected at the transition point for SAT.

formula		GSAT									Simul. Ann.		
		basic			walk			noise					
vars	clauses	time	flips	R	time	flips	R	time	flips	R	time	flips	R
100	430	.4	7554	8.3	.2	2385	1.0	.6	9975	4.0	.6	4748	1.1
200	860	22	284693	143	4	27654	1.0	47	396534	6.7	21	106643	1.2
400	1700	122	2.6×10^6	67	7	59744	1.1	95	892048	6.3	75	552433	1.1
600	2550	1471	30×10^6	500	35	241651	1.0	929	7.8×10^6	20	427	2.7×10^6	3.3
800	3400	*	*	*	286	1.8×10^6	1.1	*	*	*	*	*	*
1000	4250	*	*	*	1095	5.8×10^6	1.2	*	*	*	*	*	*
2000	8480	*	*	*	3255	23×10^6	1.1	*	*	*	*	*	*

Table 1: Comparing noise strategies on hard random 3CNF instances.

easily gets stuck on plateaus, and requires many random restarts, in particular for larger formulas. On the other hand, GSAT with walk is practically guaranteed to find a satisfying assignment. Apparently, mixing random walk over variables in the unsatisfied clauses with greedy moves allows one to escape almost always from plateaus that have few or no states from which a downhill move can be made. The other two strategies also give an improved value of R over basic GSAT but the effect is less dramatic.

Planning Problems

As a second example of the effectiveness of the various escape strategies, we consider encodings of blocks-world planning problems (Kautz and Selman 1992). Such formulas are very challenging for basic GSAT. Examination of the best assignments found when GSAT fails to find a satisfying assignment indicates that difficulties arise from extremely deep local minima. For example, the planning problem labeled "Hanoi" corresponds to the familiar "towers of Hanoi" puzzle, in which one moves a stack of disks between three pegs while never placing a larger disk on top of a smaller disk. There are many truth assignments that satisfy *nearly* all of the clauses that encode this problem, but that are very different from the correct satisfying assignment; for example, such a near-assignment may correspond to slipping a disk out from the bottom of the stack.

As seen in Table 2, GSAT with random walk is far superior. As before, basic GSAT fails to solve the largest problems. GSAT with walk is about 100 times faster than simulated annealing on the two largest problems, and over 200 times faster than random noise. The random noise and annealing strategies on the large problems also require many more restarts than the random walk strategy before finding a solution.

Circuit Synthesis

Kamath *et al.* (1991) developed a set of SAT encodings of Boolean circuit synthesis problems in order to test a satisfiability procedure based on integer programming. The task under consideration was to derive a logical circuit from its input-output behavior. Selman *et al.* (1992) showed that basic GSAT was competitive with their integer-programming method. In Table 3, we give our experimental results on five of the hardest instances considered by Kamath *et al.* As is clear from the table, both the random walk and the simulated annealing strategies significantly improve upon GSAT, with random walk being somewhat better than simulated annealing. For comparison, we also included the original timings reported by Kamath *et al.*[7] In this case, the random noise strategy does not lead to an improvement over basic GSAT. In fact mixing in random noise appears to degrade GSAT's performance. Note that the basic GSAT procedure already performs quite well on these formulas, which suggests that they are relatively easy compared to our other benchmark problems.

The instances from Kamath *et al.* (1991) were derived from randomly wired Boolean circuits. So, although the SAT encodings contain some intricate structure from the underlying Boolean gates, there is still a random aspect to the problem instances. Recently, Kamath *et al.* (1993) have generalized their approach, to allow for circuits with multiple outputs. Using this formulation, we can encode Boolean circuits that are useful in practical applications. Some examples are adder and comparator circuits. We encoded the I/O behavior of several of such circuits, and used GSAT with walk to solve them. Table 4 shows our results. ("GSAT+w" denotes GSAT with walk. We used $p = 0.5$. We will discuss the "WSAT" column below.) The type of circuit is indicated in the table. For example, every satisfying assignment for the formula 2bitadd_11 corresponds to a design for a 2-bit adder using a PLA (Programmable Logic Array). The suffix "11" indicates that the circuit is constrained to use only 11 AND-gates. We see from the table that GSAT with walk can solve the instances in times that range from less than a second to a few minutes. We also included the timings for the Davis-Putnam (DP) procedure. We used a variant of this procedure developed by Crawford and Auton (1993). This procedure

[7] Kamath *et al.*'s satisfiability procedure ran on a VAX 8700 with code written in FORTRAN and C.

formula			GSAT									Simul. Ann.		
			basic			walk			noise					
id	vars	clauses	time	flips	R	time	flips	R	time	flips	R	time	flips	R
med.	273	2311	7.5	70652	125	0.4	3464	1.0	4.5	41325	1.1	4.5	12147	1.0
rev.	201	1382	3.7	41693	100	0.3	3026	1.0	2.7	29007	1.1	2.7	9758	1.0
hanoi	417	2559	*	*	*	39	334096	2.6	20017	16×10^6	100	3250	4.1×10^6	25
huge	937	14519	*	*	*	38	143956	1.1	9648	37×10^6	200	8302	4.4×10^6	13

Table 2: Comparing noise strategies on SAT encodings of planning problems.

formula		Int.P.	GSAT									Simul. Ann.		
			basic			walk			noise					
id	vars	time	time	flips	$R.$	time	flips	$R.$	time	flips	$R.$	time	flips	$R.$
f16a1	1650	2039	58	709895	5	2	3371	1.1	375	1025454	6.7	12	98105	1.3
f16b1	1728	78	269	2870019	167	12	25529	1.0	1335	2872226	167	11	96612	1.4
f16c1	1580	758	2	12178	1.0	1	1545	1.0	5	14614	1.0	5	21222	1.0
f16d1	1230	1547	87	872219	7.1	3	5582	1.0	185	387491	1.0	4	25027	1.0
f16e1	1245	2156	1	2090	1.0	1	1468	1.0	1	3130	1.0	3	5867	1.0

Table 3: Comparing noise strategies on the circuit synthesis problem instances as studied in Kamath *et al.* (1991).

is currently one of the fastest complete methods, but it is quite surprising to see that it only solves two of the instances.[8] (A "$\star$" indicates that the method ran for 10 hrs without finding an assignment.) The good performance of GSAT with walk on these problems indicates that local search methods can perform well on structured problems that do not contain any random component.

Circuit Diagnosis

Larrabee (1992) proposed a translation of the problem of test pattern generation for VLSI circuits into a SAT problem. We compared the performance of GSAT with walk and that of DP on several of Larrabee's formulas. Our results are in table 5.[9] We see that GSAT with walk again works very well, especially compared to DP's systematic search. These results and the ones for circuit synthesis are of particular interest because they involve encodings of problems with clear practical applications, and are not just useful as benchmark problems for testing satisfiability procedures.

Modifying the Random Walk Strategy

We have recently begun to experiment with a new algorithm that implements GSAT's random walk strategy with subtle but significant modifications. This new algorithm, called WSAT (for "walk sat"), makes flips by first randomly picking a clause that is not satisfied by the current assignment, and then picking (either at random or according to a greedy heuristic) a variable within that clause to flip. Thus, while GSAT with walk can be viewed as adding "walk" to a greedy algorithm, WSAT can be viewed as adding greediness as a heuristic to random walk. The "WSAT" columns in Tables 4 and 5 shows that WSAT can give a substantial speed up over GSAT with walk. Whether or not WSAT outperforms GSAT with walk appears to depend on the particular problem class. We are currently studying this further.

One unexpected and interesting observation we have already made is that there can be a great variance between running GSAT with 100% walk (*i.e.*, $p = 1.0$) and running WSAT where variables are picked within an unsatisfied clause at random. At first glance, these options would appear to be identical. However, there is a subtle difference in the probability that a given variable is picked to be flipped. GSAT maintains a list (without duplicates) of the variables that appear in unsatisfied clauses, and picks at random from that list; thus, every variable that appears in an unsatisfied clause is chosen with equal probability. WSAT employs the two-step random process described above (first picking a clause, and then picking a variable), that favors variables that appear in many unsatisfied clauses. For many classes of formulas, the difference does not appear to be significant. However, GSAT with 100% walk does not solve the circuit diagnosis problems, whereas WSAT with random picking can solve all of them.

Maximum Satisfiability

Finally, we compare the performance of GSAT with walk to the methods studied by Hansen and Jau-

[8] Preliminary experiments indicate that some of these formulas can also be solved by combining DP with multiple starts that randomly permute variables. Details will appear in the full version. We thank Jimi Crawford for discussions on this issue.

[9] The table contains some typical satisfiable instances from a collection made available by Allan van Gelder and Yumi Tsuji at the University of California at Irvine.

formula			DP	GSAT+w	WSAT
id	vars	clauses	time	time	time
2bitadd_12	708	1702	*	0.081	0.013
2bitadd_11	649	1562	*	0.058	0.014
3bitadd_32	8704	32316	*	94.1	1.0
3bitadd_31	8432	31310	*	456.6	0.7
2bitcomp_12	300	730	23096	0.009	0.002
2bitcomp_5	125	310	1.4	0.009	0.001

Table 4: Comparing an efficient complete method (DP) with local search strategies on circuit synthesis problems. (Timings in seconds.)

formula			DP	GSAT+w	WSAT
id	vars	clauses	time	time	time
ssa7552-038	1501	3575	7	129	2.3
ssa7552-158	1363	3034	*	90	2
ssa7552-159	1363	3032	*	14	0.8
ssa7552-160	1391	3126	*	18	1.5

Table 5: Comparing DP with local search strategies on circuit diagnosis problems by Larrabee (1989). (Timings in seconds.)

mard (1990) for MAX-SAT. Our results appear in Table 6. Hansen and Jaumard compared five different algorithms for MAX-SAT. They considered a basic local search algorithm called "zloc", two deterministic algorithms proposed by David Johnson (1974) called "zjohn1" and "zjohn2", a simulated annealing approach called "anneal", and their own "steepest ascent, mildest descent" algorithm, "zsamd". The last one is similar to basic GSAT with a form of a tabu list (Glover 1989). They showed that "zsamd" consistently outperformed the other approaches.

We repeated their main experiments using GSAT with walk. The problem instances are randomly generated 3SAT instances. For each problem size, 50 problems were randomly generated, and each problem was solved 100 times using different random initial assignments. The mean values of the best, and mean number of unsatisfied clauses found during the 100 tries are noted in the table. For example, on the 500 variable, 5000 clause 3SAT instances, the best assignments GSAT with walk found contained an average of 161.2 unsatisfied clauses. As we can see from the table, GSAT with walk consistently found better quality solutions than any other method. Note that there is only a small difference between the best and mean values found by GSAT with walk, which may indicate that the best values are in fact close to optimal.

Conclusions

We compared several mechanisms for escaping from local minima in satisfiability problems: simulated annealing, random noise, and mixed random walk. The walk strategy introduces perturbations in the current state that are directly relevant to the unsatisfied constraints of the problem. Our experiments show that this strategy significantly outperforms simulated annealing and random noise on several classes of hard satisfiability problems. Both of the latter strategies can make perturbations that are in a sense less focused, in that they may involve variables that do not appear in any unsatisfied clauses. The relative improvement found by using random walk over the other methods increases with increasing problem size. We also showed that GSAT with walk to be remarkably efficient in solving basic circuit synthesis problems. This result is especially interesting because the synthesis problems do not have any random component, and are very hard for systematic methods. Finally, we demonstrated that GSAT with walk also improves upon the best MAX-SAT algorithms. Given the effectiveness of the mixed random walk strategy on Boolean satisfiability problems, an interesting direction for future research would be to explore similar strategies on general constraint satisfaction problems.

References

Buro, M. and Kleine-Büning, H. (1992). Report on a SAT competition. Technical Report # 110, Dept. of Mathematics and Informatics, University of Paderborn, Germany.

Crawford, J.M. and Auton, L.D. (1993) Experimental Results on the Cross-Over Point in Satisfiability Problems. *Proc. AAAI-93*, Washington, DC, 21–27.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7, 201–215.

Dubois, O., Andre, P., Boufkhad, Y., and Carlier, J. (1993). SAT versus UNSAT. DIMACS Workshop on Sat-

method #vars		100	100	100	300	300	300	300	500
#clauses		200	500	700	600	800	1500	2000	5000
zloc	best	0.4	10.8	21.0	2.8	8.3	35.4	64.4	233.8
zjohn1	mean	2.3	14.9	28.2	4.7	10.6	45.3	74.1	268.8
zjohn2	mean	1.4	13.5	26.9	2.7	9.0	44.1	76.5	257.4
anneal	best	0	5.6	15.1	0.7	3.1	22.6	47.6	215.7
zsamd	best	0	3.7	13.4	0.5	1.2	10.6	34.0	174.6
	mean	0.3	5.1	14.7	2.4	4.3	15.3	39	182.8
GSAT+	best	0	2.8	12.9	0	0	7.6	31.8	161.2
	mean	0	2.9	12.9	0	0	8.1	34.9	163.6

Table 6: Experimental results for MAX-3SAT. The data for the first five methods are from Hansen and Jaumard (1990).

isfiability Testing, New Brunswick, NJ, Oct. 1993.

Glover, F. (1989). Tabu search — Part I. *ORSA Journal of Computing*, 1, 190–206.

Gent, I.P. and Walsh, T. (1992). The enigma of SAT hill-climbing procedures. Techn. report 605, Department of Computer Science, University of Edinburgh. Revised version appeared in the *Journal of Artificial Intelligence Research*, Vol. 1, 1993.

Gu, J. (1992). Efficient local search for very large-scale satisfiability problems. *Sigart Bulletin*, Vol. 3, no. 1, 8–12.

Hansen J. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, 44, 279–303.

Jerrum, M. (1992) Large Cliques Elude the Metropolis Process. *Random Structures and Algorithms,* Vol. 3, no. 4, 347-359.

Johnson, D.S. (1974) Optimization algorithms for combinatorial problems. *J. of Comp. and Sys. Sci.*, 9:256–279.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C. (1991) Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partioning. *Operations Research*, 39(3):378–406.

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1991). A continuous approach to inductive inference. *Mathematical Programming*, 57, 215–238.

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1993). An Interior Point Approach to Boolean Vector Function Synthesis. Technical Report, AT&T Bell Laboratories, Nov. 1993.

Kautz, H.A. and Selman, B. (1992). Planning as satisfiability. *Proceedings ECAI-92*, Vienna, Austria.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220, 671–680.

Larrabee, T. (1992). Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 1992.

Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1990) Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. *Proceedings AAAI-90*, 1990, 17–24. Extended version appeared in *Artificial Intelligence*, 1992.

Mitchell, D., Selman, B., and Levesque, H.J. (1992). Hard and easy distributions of SAT problems. *Proceedings AAAI-92,* San Jose, CA, 459–465.

Papadimitriou, C.H. (1991). On Selecting a Satisfying Truth Assignment. *Proc. of the Conference on the Foundations of Computer Science*, 163–169.

Papadimitriou, C.H., Steiglitz, K. (1982). *Combinatorial optimization.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Pinkas, G. and Dechter, R. (1992). An Improved Connectionist Activation Function for Energy Minimization. *Proc. AAAI-92,* San Jose, CA, 434–439.

Selman, B. and Kautz, H.A. (1993). Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *Proc. IJCAI-93,* Chambery, France.

Selman, B. and Levesque, H.J., and Mitchell, D.G. (1992). A New Method for Solving Hard Satisfiability Problems. *Proc. AAAI-92,* San Jose, CA, 440–446.