This papers appears in the *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, Washington, D.C., 1993.

An Empirical Study of Greedy Local Search for Satisfiability Testing

Bart Selman and
Henry A. Kautz
AI Principles Research Department
AT&T Bell Laboratories
Murray Hill, NJ 07974
{selman, kautz}@research.att.com

Abstract

GSAT is a randomized local search procedure for solving propositional satisfiability problems. GSAT can solve hard, randomly generated problems that are an order of magnitude larger than those that can be handled by more traditional approaches, such as the Davis-Putnam procedure. This paper presents the results of numerous experiments we have performed with GSAT, in order to improve our understanding of its capabilities and limitations.

We first characterize the space traversed by GSAT. We will see that for nearly all problem classes we have encountered, the space consists of a steep descent followed by broad flat plateaus. We then compare GSAT with simulated annealing, and show how GSAT can be viewed as an efficient method for executing the low-temperature tail of an annealing schedule. Finally, we report on extensions to the basic GSAT procedure. We discuss two general, domain-independent extensions that dramatically improve GSAT's performance on structured problems: the use of clause weights, and a way to average in near-solutions when initializing the procedure before each try.

1 Introduction

Selman *et al*. (1992) introduced a randomized greedy local search procedure called GSAT for solving propositional satisfiability problems. Experiments showed that this procedure can be used to solve hard, randomly generated problems that are an order of magnitude larger than those that can be handled by more traditional approaches, such as the Davis-Putnam procedure or resolution. GSAT was also shown to perform well on propositional encodings of the N-queens problem, graph coloring problems, and Boolean induction problems.

This paper presents the results of numerous experiments we have performed with GSAT, in order to improve our understanding of its capabilities and limitations. We will begin with an exploration of the shape of the search space that GSAT typically encounters. We will see that for nearly all problem classes we have examined, the space consists of a steep descent followed by broad plateaus. We then compare GSAT with simulated annealing, and show how GSAT can be viewed as a very efficient method for executing the low-temperature tail of an annealing schedule.

A common criticism of randomized algorithms like GSAT is that they might not do as well on problems that have an intricate underlying structure as they do on randomly generated problems. Based on our understanding of the shape of GSAT's search space, we developed two general, domain-independent extensions that dramatically improve its performance: the use of clause weights, and a way to average in near-solutions when initializing the procedure before each try. We will also describe other local search heuristics which appear promising, but did not improve performance on our test problems.

This paper is unabashidly empirical. Although we will point to relevant results in the theoretical literature, we will not present an abstract analysis of our results. It would obviously be highly desirable to characterize precisely the class of problems for which GSAT succeeds, and to provide precise bounds on its running time. Unfortunately, such results are extremely rare and difficult to obtain in work on incomplete algorithms for NP-hard problems. The situation is similar, for example, in research on simulated annealing, where the formal results show convergence in the limit (*i.e.*, after an arbitrary amount of time), but few address the *rate* of convergence to a solution. In fact, a good, general characterization of the rate of convergence appears to be beyond the current state of the art of theoretical analysis (Bertsimas and Tsitsiklis 1992; Jerrum 1992). Current theory does, however, explain why GSAT performs well on certain limited classes of formulas

```
Procedure GSAT
Input: a set of clauses α, MAX-FLIPS, and MAX-TRIES
Output: a satisfying truth assignment of α, if found
  for i := 1 to MAX-TRIES
      T := a randomly generated truth assignment
      for j := 1 to MAX-FLIPS
          if T satisfies α then return T
          p := a propositional variable such that a change
                in its truth assignment gives the largest
                increase in the total number of clauses
                of α that are satisfied by T
          T := T with the truth assignment of p reversed
      end for
  end for
  return "no satisfying assignment found"
```

Figure 1: The GSAT procedure.

(*e.g.* 2-SAT and over-constrained formulas), and the range of applicability of such formal results will certainly increase over time (Papadimitriou 1991; Koutsoupias and Papadimitriou 1992). We believe that experimental work should proceed in parallel with theoretical work, because real data can point out the problem-solving techniques that are worthy of formal analysis, and can help distinguish the asymptotic results that carry over to practical cases from those that do not.

2 The GSAT Procedure

GSAT performs a greedy local search for a satisfying assignment of a set of propositional clauses.[1] The procedure starts with a randomly generated truth assignment. It then changes ('flips') the assignment of the variable that leads to the largest increase in the total number of satisfied clauses. Such 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. See Figure 1. (For a related approach, see Gu (1992).)

[1] A clause is a disjunction of literals. A literal is a propositional variable or its negation. A set of clauses corresponds to a formula in conjunctive normal form (CNF): a conjunction of disjunctions. Thus, GSAT handles CNF–SAT.

GSAT mimics the standard local search procedures used for finding approximate solutions to optimization problems (Papadimitriou and Steiglitz 1982) in that it only explores potential solutions that are "close" to the one currently being considered. Specifically, we explore the set of assignments that differ from the current one on only one variable. The GSAT procedure requires the setting of two parameters, MAX-FLIPS and MAX-TRIES, which determine, respectively, how many flips the procedure will attempt before giving up and restarting, and how many times this search can be restarted before quitting. As a rough guideline, setting MAX-FLIPS equal to about ten times the number of variables is sufficient. The setting of MAX-TRIES will generally be determined by the total amount of time that one wants to spend looking for an assignment before giving up.

In our experience so far, there is generally a good setting of the parameters that can be used for all instances of an application. Thus, one can fine-tune the procedure by experimenting with various parameter settings. It is important to understand that we are *not* suggesting that the parameters need to be reset for each individual problem — only for a broad class, for example, coloring problems, random formulas, *etc*. Practically all optimization algorithms for intractable problems have parameters that must be set this way,[2] so this is not a particular disadvantage of GSAT. Furthermore, one could devise various schemes to automatically choose a good parameter setting by performing a binary search on different parameter settings on a sequence of problems.

2.1 Summary of Previous Results

In Selman *et al*. (1992), we showed that GSAT substantially outperforms backtracking search procedures, such as the Davis-Putnam procedure, on various classes of formulas. For example, we studied GSAT's performance on hard randomly generated formulas. (Note that generating hard random formulas for testing purposes is a challenging problem by itself, see Cheeseman *et al*. (1991); Mitchell *et al*. (1992); Williams and Hogg (1992); Larrabee and Tsuji (1993); and Crawford and Auton (1993).) The fastest backtrack type procedures, using special heuristics, can handle up to 350 variable hard random formulas in about one hour on a MIPS workstation (Buro and Kleine Büning 1992; Crawford and Auton 1993). Nevertheless, the running time clearly scales exponentially, for example, hard 450 variable formulas are undoable. Our current implementation of GSAT,

[2]For example, see the discussion on integer programming methods in Fourer (1993).

using the random walk option discussed in Selman and Kautz (1993), solves hard 1500 variable formulas in under an hour. Selman *et al*. also showed that GSAT performs well on propositional encodings of the N-queens problem, hard instances of graph coloring problems (Johnson *et al*. 1991), and Boolean induction problems (Kamath *et al*. 1992).

3 The Search Space

Crucial to a better understanding of GSAT's behavior is the manner in which GSAT converges on an assignment. In Figure 2, we show how the GSAT's search progresses on a randomly generated 100 variable problem with 430 clauses. Along the horizontal axis we give the number of flips, and along the vertical axis the number of clauses that still remained unsatisfied. (The final flip reduces this number to zero.) It is clear from the figure that most of the time is spent wandering on large plateaus. Only approximately the first 5% of the search is spent in pure greedy descent. We have observed qualitatively similar patterns over and over again. (See also the discussion on "sideway" moves in Selman *et al*. (1992), and Gent and Walsh (1993).)

The bottom panel in Figure 2 shows the search space for a 500 variable, 2150 clause random satisfiable formula. The long plateaus become even more pronounced, and the relative size of the pure greedy descent further diminishes. In general, the harder the formulas, the longer the plateaus.

Another interesting property of the graphs is that we see no upwards moves. An upward move would occur when the best possible flip increases the number of unsatisfied clauses. This appears to be extremely rare, especially for the randomly generated instances.

The search pattern brings out an interesting difference between our use of GSAT and the standard use of local search techniques for obtaining good *approximate* solutions to combinatorial optimization problems (Lin and Kernighan 1973; Papadimitriou and Steiglitz 1982; Papadimitriou *et al*. 1990). In the latter, one generally halts the local search procedure as soon as no more improvement is found. Our figure shows that this is appropriate when looking for a *near-solution*, since most of the gain lies in the early, greedy descent part. On the other hand, when searching for a global minimum (*i.e*., a satisfying assignment) stopping when flips do not yield an immediate improvement is a poor strategy — *most* of the work occurs in satisfying the last few remaining clauses.

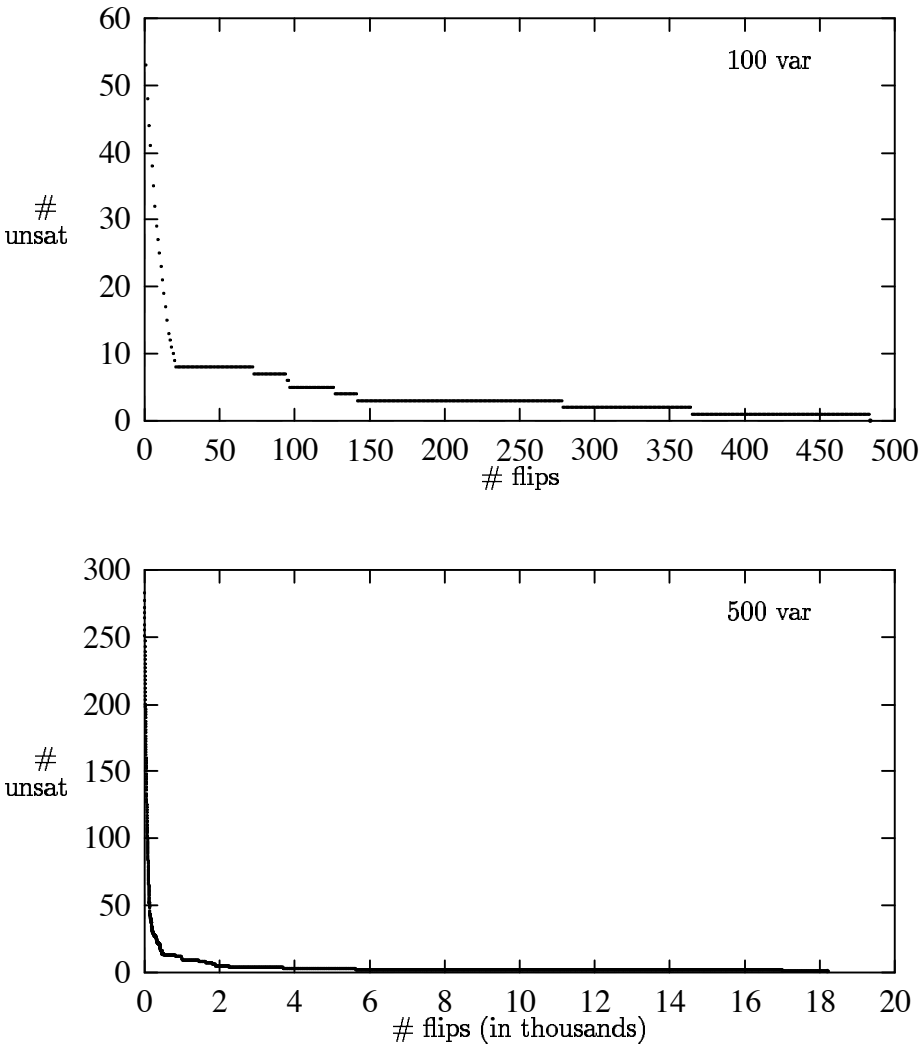

Figure 2: GSAT's search space on a 100 and 500 variables formulas.

Note that finding an assignment that satisfies all clauses of a logical theory is essential in many reasoning and problem solving situations. For example, in our work on planning as satisfiability, a satisfying assignment correspond to a correct plan (Kautz and Selman 1992). The near-satisfying assignments are of little use;

they correspond to plans that contain one of more "magical" moves, where blocks suddenly shift positions.

3.1 Simulated Annealing

Simulated annealing is a stochastic local search method. It was introduced by Kirkpatrick *et al*. (1983) to tackle combinatorial optimization problems. Instead of pure greedy local search, the procedure allows a certain amount of "noise" which enables it to make modifications that actually *increase* the cost of the current solution (even when this is not the best possible current modification). In terms of finding satisfying assignments, this means that the procedure sometimes allows flips that actually increase the total number of unsatisfied clauses. The idea is that by allowing random occurrences of such upwards moves, the algorithm can escape local minima. The frequency of such moves is determined by a parameter T, called the temperature. (The higher the temperature, the more often upward moves occur.)

The parameter T is set by the user. Normally, one follows a so-called annealing schedule in which one slowly decreases the temperature until T reaches zero. It can be shown formally that provided one "cools" slowly enough, the system will find a global minimum. Unfortunately, the analysis uses an exponentially long annealing schedule, making it only of theoretical interest (Hajek 1988). Our real interest is in the rate of convergence to a global minimum for more practical annealing schedules. Current formal methods, however, appear too weak to tackle this question.[3]

It is interesting to compare the plot for GSAT (Figure 2) with that for annealing (Figure 3) on the same 100 variable random formula.[4] In the early part of the search, GSAT performs pure greedy descent. The descent is similar to the initial phase of an annealing schedule, although more rapid, because GSAT performs no upward moves. In the next stage, both algorithms must search along a series of long plateaus. GSAT makes mostly sideways moves, but takes advantage of

[3]Recent work by Pinkas and Dechter (1992) and Jerrum (1992) provides some interesting formal convergence results for a special class of optimization problems.

[4]We use the annealing algorithm given in Johnson *et al*. (1991). Start with a randomly generated truth assignment; repeatedly pick a random variable, and compute how many more clauses become satisfied when the truth value of that variable is flipped — call this number δ. If $\delta \geq 0$, make the flip. Otherwise, flip the variable with probability $e^{\delta/T}$. We slowly decrease the temperature from 10 down to 0.05.

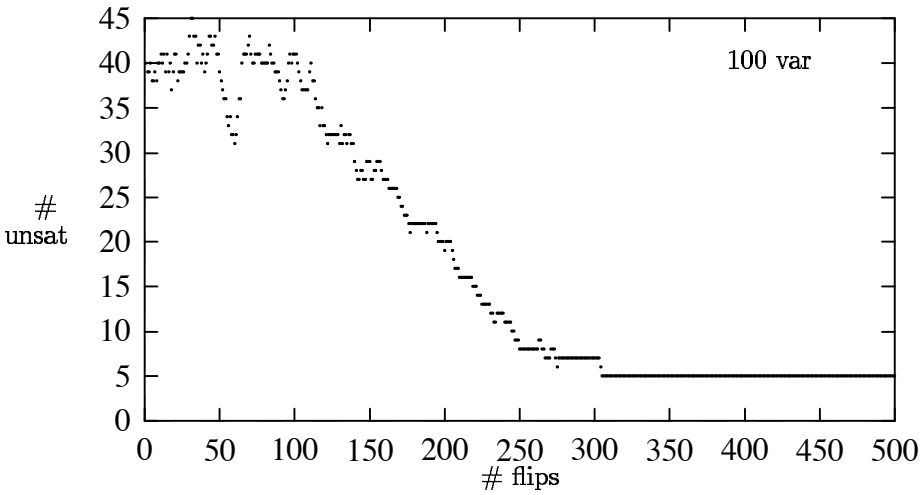

Figure 3: Simulated annealing's search space on a 100 variable formula.

a downward move whenever one arises. Annealing has reached the long, low-temperature "tail" of its schedule, where it is very unlikely to make an upward move, but allows both sideways and downward moves. Because much of the effort expended by annealing in the initial high temperature part of the schedule is wasted, it typically takes longer to reach a solution. Note, for example, that after less than 500 moves GSAT has reached a satisfying assignment, while the annealing algorithm still has 5 unsatisfied clauses. A more rapid cooling schedule would, of course, more closely mimick GSAT.

Thus we can view GSAT as a very efficient method for executing the low-temperature tail of an annealing schedule. Furthermore, our experiments with several annealing schedules on hard, random formulas confirmed that most of the work in finding a true satisfying assignment is in the tail of the schedule. In fact, we were unable to find an annealing schedule that performed better than GSAT, although we cannot rule out the possiblity that such a schedule exists. This is an inherent difficulty in the study of annealing approaches (Johnson *et al*. 1991).

4 Extensions

The basic GSAT algorithm is quite elementary, and one might expect that more sophisticated algorithms could yield better performance. We investigated several extensions to GSAT, and found a few that were indeed successful. But it is important to stress that the basic GSAT algorithm is very robust, in the sense that many intuitively appealing modifications do not in fact change its performance. We have found that experimental study can help reveal the assumptions, true or false, implicit in such intuitions, and can lead to interesting questions for further empirical or theoretical research.

4.1 Improving the Initial Assignment

One natural intuition about GSAT is that it would be better to start with an initial assignment that is "close" to a solution, rather than with a totally random truth assignment. Indeed, the theoretical analysis of general greedy local search presented in (Minton *et al*. 1992) shows that the closer the initial assignment is to a solution, the more likely it is that local search will succeed.

Therefore we tried the following method for creating better initial assignments: First, a variable is assigned a random value. Next, all clauses containing that variable are examined, to see if the values of any unassigned variables are then determined by unit propagation. If so, these variables are assigned, and again unit propagation is performed. (If a clause is unsatisfied by the current partial assignment, it is simply ignored for the time being.) When no more propagations are possible, another unassigned variable is given a random value, and the process repeats.

Experiments revealed that this strategy did *not* significantly reduce the time required to find a solution. In retrospect, this failure can be explained by the shape of the search space, as discussed above. The descent from an initial state in which many clauses are unsatisfied to one which only a few are unsatisfied occupies only a tiny fraction of the overall execution time, and initial unit propagation helps only in this phase of the search.

The problem is that the number of unsatisfied clauses is a fairly crude measure of the distance to a solution, measured in terms of the number of flips required to reach a satisfying assignment. (Minton *et al*. (1992) make a similar observation regarding coloring problems. See also Gent and Walsh (1993).) This led us to consider another strategy for generating good initial assignments. Since GSAT

typically performs many tries before finding a solution, we make use of the information gained from previous tries to create an initial assignment that is already some distance out on a low plateau, and thus actually closer to a solution. We do this by initializing with the *bitwise average* of the best assignment found in the two previous tries.

The bitwise average of two truth assignments is an assignment that agrees with the assignment of those letters on which the two given truth assignments are identical; the remaining letters are randomly assigned truth values. After many tries in which averaging is performed, the initial and final states become nearly identical. We therefore reset the initial assignment to a new random assignment every 10 to 50 tries.[5]

In Selman and Kautz (1993), we give an empirical evaluation of the averaging strategy. We considered propositional encodings of hard graph coloring problems used by Johnson *et al*. (1991) to evaluate specialized graph coloring algorithms. Our experiments show that GSAT with the averaging strategy compares favorably with some of the best *specialized* graph coloring algorithms as studied by Johnson. This is quite remarkable because GSAT does not use any special techniques for graph coloring.

4.2 Handling Structure with Clause Weights

As we noted in the introduction, the fact that GSAT does well on randomly-generated formulas does not necessarily indicate that it would also perform well on formulas that have some complex underlying structure. In fact, Ginsberg and Jónsson (1992) supplied us with some graph coloring problems that GSAT could not solve, even with many tries each with many flips. Their dependency-directed backtracking method could find solutions to these problems with little effort (Jónsson and Ginsberg 1993). In running GSAT on these problems, we discovered that at the end of almost every try the same set of clauses remained unsatisfied. As it turns out, the problems contained strong *asymmetries*. Such structure can lead GSAT into a state in which a few violated constraints are consistently "out-voted" by many satisfied constraints.

To overcome asymmetries, we added a weight to each clause (constraint).[6] A

[5] We thank Geoffrey Hinton and Hector Levesque for suggesting this strategy to us. The strategy has some of the flavor of the approaches found in genetic algorithms (Davis 1987).

[6] Morris (1993) has independently proposed a similar approach.

weight is a positive integer, indicating how often the clause should be counted when determining which variable to flip next. Stated more precisely, having a clause with weight L is equivalent to having the clause occur L times in the formula. Initially, all weights are set to 1. At the end of each try, we increment by 1 the weights of those clauses not satisfied by the current assignment. Thus the weights are dynamically modified *during* problem solving, again making use of the information gained by each try.

# unsat clauses at end of try	# of times reached	
	basic	weights
0	0	80
1	2	213
2–4	0	0
5–9	90	301
10+	908	406

Table 1: Comparison of GSAT with and without weights on a highly asymmetrical graph coloring problem (see text for explanation).

Using weights, GSAT solves a typical instance of these coloring problems in a second or two. This is comparable with the time used by efficient backtrack-style procedures. Table 1 shows the distribution of the number of unsatisfied clauses after each try for GSAT with and without weights on Ginsberg and Jónsson's 50 node graph (200 variables and 2262 clauses). We used a total of 1000 tries with 1000 flips per try. For example, basic GSAT never found an assignment that had no unsatisfied clauses, but GSAT with weights found one in 80 tries out of 1000. Similarly, basic GSAT found an assignment with one unsatisfied clause only twice, while GSAT with weights found such an assignment 213 times.

The weight strategy turns out to help not only on problems handcrafted to fool GSAT (including the similarly "misleading" formulas discussed in Selman *et al*. (1992)), but also on many *naturally occuring* classes of structured satisfiability problems. A case in point are formulas that encode *planning problems*. As we reported in Kautz and Selman (1992), the basic GSAT algorithm had difficulty in solving formulas that encoded blocks-world planning problems. However, using weights GSAT's solution times are comparable with those of the Davis-Putnam procedure on these formulas. Details appear in Selman and Kautz (1993).

The regularities that appear in certain non-random classes of generated formulas tend to produce local minima that can trap a simple greedy algorithm. The weights, in effect, are used to *fill in* local minima while the search proceeds, and thus *uncover* the regularities. Note that this general strategy may also be useful in avoiding local minima in other optimization methods, and provides an interesting alternative to the use of random noise (as in simulated annealing).

5 Conclusions

The experiments we ran with GSAT have helped us understand the nature of the search space for propositional satisfiability, and have led us to develop interesting heuristics that augment the power of local search on various classes of satisfiability problems. We saw that the search space is characterized by plateaus, which suggests that the crucial problem is to develop methods to quickly traverse broad flat regions. This is in contrast, for example, to much of the work on simulated annealing algorithms, which support the use of slow cooling schedules to deal with search spaces characterized by jagged surfaces with many deep local minima.

We discussed two empirically successful extensions to GSAT, averaging and clause weights, that improve efficiency by re-using some of the information present in previous near-solutions. Each of these strategies, in effect, helps uncover hidden structure in the input formulas, and were motivated by the shape of GSAT's search space. Given the success of these strategies and the fact that they are not very specific to the GSAT algorithm, it appears that they also hold promise for improving other methods for solving hard combinatorial search problems. In our future research we hope to improve our formal understanding of the benefits and applicability of these techniques.

Finally, we should note that we do not claim that GSAT and its descendants will be able to efficiently solve all interesting classes of satisfiability problems. Indeed, no one universal method is likely to prove successful for all instances of an NP-complete problem! Nonetheless, we believe it is worthwhile to develop techniques that extend the *practical* range of problems that can be solved by local search.

References

Bertsimas, D. and Tsitsiklis, J. (1992). Simulated Annealing, in *Probability and Algorithms*, National Academy Press, Washington, D.C., 17–29.

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

Cheeseman, Peter and Kanefsky, Bob and Taylor, William M. (1991). Where the Really Hard Problems Are. *Proc. IJCAI-91,* 1991, 163–169.

Crawford, J.M. and Auton, L.D. (1993) Experimental Results on the Cross-Over Point in Satisfiability Problems. *Proc. AAAI-93,* to appear.

Davis, E. (1987) *Genetic Algorithms and Simulated Annealing*, in Pitman Series of Research Notes in Artificial Intelligence, London: Pitman; Los Altos, CA: Morgan Kaufmann.

Davis, M. and Putnam, H. (1960). A Computing Procedure for Quantification Theory. *J. Assoc. Comput. Mach.*, 7, 1960, 201–215.

Feynman, R.P., Leighton, R.B. , and Sand, M. (1989). *The Feynman Lectures on Physics, Vol. I*, Addison-Wesley Co, Reading, MA.

Fourer, R. , Gay, D.M., and Kernighan, B.W. (1993). *AMPL: A Modeling Language for Mathematical Programming*, San Francisco, CA: The Scientific Press.

Gent, I.P. and Walsh, T. (1993). Towards an Understanding of Hill-Climbing Procedures for SAT. *Proc. AAAI-93*, to appear.

Ginsberg, M. and Jónsson, A. (1992). Personal communication, April 1992.

Gu, J. (1992). Efficient Local Search for Very Large-Scale Satisfiability Problems. *Sigart Bulletin*, vol. 3, no. 1, 1992, 8–12.

Hajek, B. (1988). Cooling Schedules for Optimal Annealing. *Math. Oper. Res.*, 13, 311–329.

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

Johnson, J.L. (1989). A Neural Network Approach to the 3-Satisfiability Problem. *Journal of Parallel and Distributed Computing*, 6, 435–449.

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, 1991.

Jónsson, A.K., and Ginsberg, M.L. (1993) Experimenting with New Systematic and Nonsystematic Search Techniques. *Working Notes of the AAAI Spring Symposia*, 1993.

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1992). A Continuous Approach to Inductive Inference. *Mathematical Programming*, 57, 215–238.

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

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

Koutsoupias, E. and Papadimitriou C.H. (1992) On the Greedy Algorithm for Satisfiability. *Information Processing Letters*, 43, 53-55.

Larrabee, T. and Tsuji, Y. (1993) Evidence for a Satisfiability Threshold for Random 3CNF Formulas. *Working Notes AAAI Spring Symposia*, 1993.

Lin, S. and Kernighan, B.W. (1973). An Efficient Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research,* 21, 498-516.

McCarthy, J. and Hayes, P.J. (1969) Some Philosophical Problems From the Standpoint of Artificial Intelligence, in *Machine Intelligence 4*, Chichester, England: Ellis Horwood, 463ff.

Minton, S., Johnston, M.D., Philips, A.B., Johnson, M.D., and Laird, P. (1992) Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence,* (58)1–3, 1992, 161–205.

Mitchell, D., Selman, B., and Levesque, H.J. (1992). Hard and Easy Distributions of SAT Problems. *Proc. AAAI-92,* San Jose, CA, 459–465.

Morris, P. (1993). Breakout Method for Escaping from Local Minima. *Proc. AAAI-93*, to appear.

Papadimitriou, C.H. (1991). On Selecting a Satisfying Truth Assignment. *Proc. FOCS-91*, 163–169.

Papadimitriou, C.H., Shaffer, A., and Yannakakis, M. (1990). On the Complexity of Local Search. *Proc. STOC-90*.

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

Pinkas, G. and Dechter, R. (1992). *Proc. AAAI-92,* 434–439.

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.

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

William, C.P. and Hogg, T. (1992) Using Deep Structure to Locate Hard Problems. *Proc. AAAI-92*, 472–477.