To appear in *Proceedings of IJCAI-93.*
All comments welcome! {selman,kautz}@research.att.com

Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems

Bart Selman and Henry Kautz
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 (Selman *et al.* 1992). 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. GSAT also efficiently solves encodings of graph coloring problems, N-queens, and Boolean induction. However, GSAT does not perform as well on handcrafted encodings of blocks-world planning problems and formulas with a high degree of asymmetry. We present three strategies that dramatically improve GSAT's performance on such formulas. These strategies, in effect, manage to uncover hidden structure in the formula under considerations, thereby significantly extending the applicability of the GSAT algorithm.

1 Introduction

Selman *et al.* (1992) introduce a randomized greedy local search procedure called GSAT for solving propositional satisfiability problems. Experiments show 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.

A common criticism is that GSAT might not do as well on problems with a much more intricate underlying structure. Similar criticism has been raised against other randomized local search type procedures, such as simulated annealing (Johnson *et al.* 1991). In exploring this issue, we found that GSAT indeed has problems on certain classes of highly structured formulas. Examples are propositional encodings of blocks-world planning problems (Kautz and Selman 1992), and graph coloring problems with a high-degree of asymmetry (Ginsberg and Jónsson 1992).[1]

After studying GSAT's difficulty on such problem instances, we discovered general, domain-independent extensions that dramatically improve GSAT's performance. In effect, these extensions manage to uncover the underlying structure in the formulas. For certain problem classes, the extensions push GSAT's performance beyond that of backtrack type procedures, whereas on other classes GSAT becomes competitive with such procedures. The extensions therefore significantly the enlarge space of problems on which GSAT performs well, making it a promising general approach for dealing with hard computational problems in artificial intelligence.

The paper is structured as follows. We first review the basic GSAT procedure and briefly discuss its performance. We then introduce the various extensions and give experimental data showing the improved performance.

2 The GSAT Procedure

GSAT 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 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). Also, see Minton *et al.* (1990) for a very successful application of local search to scheduling problems.)

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. One distinguishing feature of GSAT, however, is the we also allow flips that do not directly improve the assignment (*i.e.,* when the best possible flip

[1]We thank Matt Ginsberg and Ari Jónsson for bringing these instances to our attention.

[2]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.

Procedure GSAT
Input: set of clauses α, MAX-FLIPS, and MAX-TRIES
Output: a satisfying truth assignment of α, if found
begin
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"
end

Figure 1: The GSAT procedure.

does not actually decrease the total number of satisfied clauses. As discussed in Selman *et al.* (1992), such flips are essential in solving hard problems. Another feature of GSAT is that the variable whose assignment is to be changed is chosen at random from those that would give an equally good improvement. Such non-determinism makes it unlikely that the algorithm makes the same sequence of changes over and over.

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 a few 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, which in turn depends on the application. 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 for an application by experimenting with various parameter settings.

2.1 Summary of Previous Results

In Selman *et al.* (1992), we showed how 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); Larrabee and Tsuji (1993); and Crawford and Auton (1993).) Table 1 summarizes the results. The table shows how GSAT is indeed much faster than the Davis-Putnam (DP) procedure on such hard random formulas.[3]

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). Results on encodings of blocks-world planning problems were not as good though (Kautz and Selman 1992), especially compared with DP's performance.

The problem with the blocks-world planning formulas appears to be their highly structured character. Such structure leads to large numbers of very fast unit propagations in backtrack search procedures, which dramatically reduces the search space.[4] GSAT has no explicit mechanism for handling unit propagation. Its bit flipping strategy implicitly propagates unit clauses but not as efficiently as a specialized procedure. We tried to extend GSAT with an explicit mechanism for handling unit propagation. This extension did not improve GSAT's overall performance, presumable because the unit propagation mechanism does not match well with the randomized local search strategy. We did however find other extensions that do dramatically improve GSAT's performance on the planning problems, and other challenging formulas. These extensions are described in the next section.

3 Extensions and experimental results

3.1 Adding Weights

Ginsberg and Jónsson (1992) supplied us with some instances of graph coloring where GSAT did not manage to find a solution, even after a large number of tries each with many flips.[5] 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 contain strong asymmetries. The effect of asymmetry is best illustrated with the graph shown in Figure 2. Consider trying to color this graph with three colors, using a local search procedure. Initially each node is randomly assigned one of the three colors. The algorithm then starts adjusting colors in order to minimize the number of conflicts. (A conflict occurs when two connected vertices have the same color.) Since the constraints between the large group of nodes and the single nodes at the bottom far outnumber the single constraint between the two bottom nodes, the algorithm gets quickly stuck in a state where the bottom two nodes have the same

[3] During the last year, we have collected data on other backtrack procedures. Using special heuristics, the most efficient ones can handle up to around 350 variable formulas in about one hour (Buro and Kleine Büning 1992; Crawford and Auton 1993) on a MIPS machine. Nevertheless, the running time clearly scales exponentially, for example, hard 450 variable formulas are undoable. Our current GSAT runs over 10 times faster than the one used for generating the data in Table 1. Moreover, using the random walk option discussed in section 3.3, we can now solve hard 1500 variable formulas in under an hour.

[4] Unit propagation is a efficient mechanism for dealing with Horn clauses. A Horn clause is a clause with at most one positive literal.

[5] A *try* is one execution of the outer-loop of the GSAT procedure; see Figure 1.

formulas		GSAT			DP		
vars	clauses	M-FLIPS	tries	time	choices	depth	time
50	215	250	6.4	0.4s	77	11	1.4s
100	430	500	42.5	6s	84×10^3	19	2.8m
140	602	700	52.6	14s	2.2×10^6	27	4.7h
150	645	1500	100.5	45s	—	—	—
300	1275	6000	231.8	12m	—	—	—
500	2150	10000	995.8	1.6h	—	—	—

Table 1: Results for GSAT and DP on *hard* random 3CNF formulas. Data from Selman *et al.* (1992). ("choices" is the number of nodes in the DP search tree, and "depth" the average depth of the tree.) The timings reported here and in the rest of this paper are based on C implementations running on a MIPS R6000, unless stated otherwise.

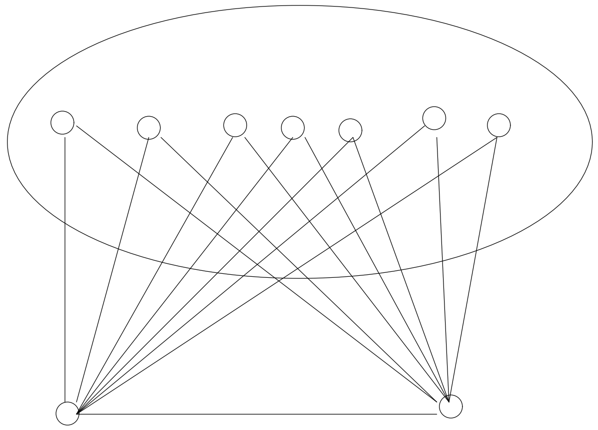

Figure 2: A gerrymandered graph.

# unsat clauses at end of try	# of times reached basic	weights
0	0	80
1	2	213
2	0	0
3	0	0
4	0	0
5	0	2
6	0	2
7	0	168
8	0	2
9	90	127
10–19	0	181
20–29	908	225

Table 2: Unsatisfied clause distribution for GSAT with and without weights on an asymmetrical graph coloring problem (Ginsberg and Jónsson 1992). It is a 50 node graph with a 4 coloring. The encoding has 200 variables and 2262 clauses.

color and each node in the larger set has one of the other two colors. This coloring satisfies all but one constraint! In other words, the single constraint between the bottom two nodes simply gets out-voted by the other nodes. We call such graphs *gerrymandered* graphs.

To overcome asymmetries, we added a weight to each clause (constraint). A 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. Of course, we don't know exactly what weights to assign to the clauses. We use the following strategy for *dynamically* adjusting the clause weights during the search.[6]

Strategy I: Clause Weights
Initialize all clause weights to 1.
At the end of each try, increment by K the weights of those clauses not satisfied by the current assignment.

We usually set K equal to 1. This strategy automatically "discovers" hidden asymmetries of the formula under consideration. Using the weights, GSAT solves a typical instance of Ginsberg and Jónsson's asymmetrical coloring problems in 1.3 seconds (after only 11 tries with 1000 flips per try). This is comparable with the time used by efficient backtrack style procedures. Table 2 shows the distribution of the number of unsatisfied clauses that remain at the end of each try for GSAT with and without weights. We used a total of 1000 tries with 1000 flips per try. For example, with weights, 213 tries led to an assignment with only one unsatisfied clause, and 80 tries led to a satisfying assignment (no unsatisfied clauses). The results show a substantial improvement when using the weights.

Using this weighing strategy, we can also easily solve the kinds of formulas discussed in section 4 of Selman *et al.* (1992). Those formulas were introduced to show some of the limitations of basic GSAT. They were handcrafted to repeatedly steer GSAT in the wrong direction. The weighing strategy again compensates quite easily.

Figure 3 illustrates the effect the weighing strategy on the search space for GSAT. GSAT searches for a global minimum in the number of unsatisfied clauses as a function of the truth assignments. Figure 3 abstractly represents this function, with the number of unsatisfied clauses along the vertical axis and truth assignments along the horizontal axis. The weights, in effect, are used to "fill in" local minima while the search proceeds. This general strategy may also be useful in avoiding lo-

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

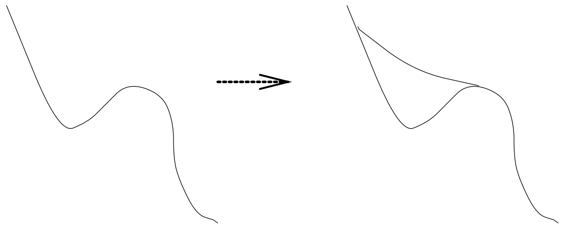

Figure 3: Filling in local minima using clause weights.

cal minima in other optimization methods. The strategy provides an interesting alternative to the standard use of random perturbations ("noise") to escape from local minima, as used in, for example, in simulated annealing. For more discussion on the relation to simulated annealing, see Selman and Kautz (1993).

3.2 Averaging in Previous Near Solutions

After each try, GSAT picks a completely new random assignments as its next starting point. The following strategy is an attempt to use some of the information that may be contained in the previous near-solution (*i.e.,* the truth assignment at the end of the previous try). Let T_i^{init} and T_i^{best} be, respectively, the assignment at the beginning of the i^{th} try and the best assignment (*i.e.,* fewest unsatisfied clauses) found during the i^{th} try.

Strategy II: Averaging in
Set T_1^{init} to a randomly generated assignment.
    Run inner-loop GSAT starting with T_1^{init}
T_2^{init} := bitwise average of T_1^{init} and T_1^{best}
    Run inner-loop GSAT starting with T_2^{init}
For $i := 3$ to RESET-TRIES
    T_i^{init} := bitwise average of T_{i-1}^{best} and T_{i-2}^{best}
    Run inner-loop GSAT starting with T_i^{init}
    End for
Repeat, starting with a new random assignment.

The bitwise average of two truth assignments is an assignment which 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 best states become nearly identical. We therefore reset the initial assignment to a new random assignment every RESET-TRIES (useful reset values are between 10 and 50).[7]

To evaluate this strategy, we consider some of the hardest problems in our test suite. These problems are based on hard graph coloring problems used by Johnson *et al.* (1991) to evaluate specialized graph coloring algorithms. Table 3 shows GSAT's performance on these instances.

We see a marked improvement by using our averaging strategy. In particular, note that we were unable to solve at all two of the instances without using averaging. (Our Davis-Putnam procedure cannot solve any of these instances, and we do not know of any other backtrack style satisfiability procedure that can solve these problems.)

The table also shows that GSAT's performance compares favorably with some of the best specialized graph coloring algorithms as studied by Johnson *et al.*[8] This is quite remarkable because GSAT does not use any special techniques for graph coloring: it is basically unaware of the fact that it is dealing with encodings of graph coloring problems.

3.3 Random Walk

Consider the following algorithm for testing the satisfiability of CNF formulas. Start with a random truth assignment; randomly select an clause not satisfied by this assignment; flip the truth assignment of one of the letters occuring in this clause (the clause becomes satisfied); repeat the last two steps until the assignment satisfies all clauses.

Papadimitriou (1991) shows that such a surprisingly simple randomized strategy finds assignments of 2CNF formulas (satisfiable ones, of course), in $O(n^2)$ steps with probability approaching one, where n is the number of propositional letters. His proof exploits properties of random walks (Feynman *et al.* 1989). Note the difference with GSAT: simply flipping the assignment of any variable in some unsatisfied clause may actually *increase* the total number of unsatisfied clauses. We found that this strategy by itself does not solve the hard satisfiability problems, but it does provide a another useful mechanism for escaping local minima.

Strategy III: Random walk
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., pick randomly from the list of variables
    that gives the largest decrease in the total number
    of unsatisfied clauses.

The probability p is fixed in advance; we used $p = 0.35$ in our experiments.[9] We first consider the effect of the walk strategy on the Boolean induction problems as studied by Kamath *et al.* (1992). The task under consideration is to derive ("induce") a logical circuit from its input-output behavior. Kamath *et al.* encode this problem as a satisfiability problem. They give test results for various instances of this problem using a satisfiability

[7] 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.

[8] We should note that we have not yet been able to solve the hardest coloring problem in Johnson *et al.* (1991). This problem involves a 500 node graph, and was solved by only one of the Johnson's coloring algorithms using many hours of cpu time. The very large size of the Boolean encoding of this instance exhausts memory of our current hardware.

[9] Note the difference between random walk and random noise as used in simulated annealing. Random noise can perturb the truth assignments of any of the variables. In the random walk strategy, the perturbation is closely tied to the unsatisfied clauses (the "problem spots"). Preliminary experiments show the latter to be much more effective.

graph		formula		GSAT		Best methods
nodes	colors	vars	clauses	basic	average in	(Johnson *et al.*)
125	18	2125	70,163	55 *sec*	44 *sec*	18 / 36 *sec*
125	17	2125	66,272	Not Solved	30 *min*	5.4 / 64 *min*
250	15	3750	233,965	7.5 *sec*	7 *sec*	4.2 / 12.5 *sec*
250	29	7250	454,622	Not Solved	1.1 *hrs*	0.3 / 0.3 *hrs*

Table 3: The effect of averaging in of previous near-solutions (strategy II). The problems considered are propositional encodings of graph coloring problems (Johnson *et al.* 1991). The "best methods" times are for the best and the second best specialized graph coloring algorithm as reported by Johnson *et al.* We have reduced Johnson's times by a factor of 20 to compensate for differences in cpu speed. (Johnson used a Sequent Balance 21000.)

formula			Int. Prog.	GSAT	
id	vars	clauses		basic	walk
16A1	1650	19368	2039	274	19
16B1	1728	24792	78	2540	58
16C1	1580	16467	758	4	2
16D1	1230	15901	1547	112	14
16E1	1245	14766	2156	2	2

Table 4: Using the random walk stategy on Boolean induction problems. (Timings in seconds.)

Wff	vars	Original		Weights		+ Walk	
		secs	tries	secs	tries	secs	tries
med.	273	3,093	10^4	19	47	10	27
rev.	244	2,607	10^4	34	108	13	41
hanoi	417	Failed		3,517	5,000	1,455	2,000
huge	1075	Failed		6,518	2,500	904	303

Table 5: Using the random walk stategy on planning formulas.

algorithm based on integer programming. We consider some of their hardest instances as are given in Table 4.5 in Kamath *et al.* (1992). In Selman *et al.* (1992), we showed how GSAT's performance is competitive with their results. However, using our walk strategy, we can still substantially improve GSAT's performance on these formulas, as is shown in Table 4.

As a second example of the effectiveness of the walk strategy, we consider encodings of various blocks-world planning problems (Kautz and Selman 1992). As we discussed above, 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, in the "towers of Hanoi" problem, the initial state is encoded by a conjunction of propositional literals corresponding to the assertion that block A is on B, B is on C, and C is on Peg 1; the final state is encoded by asserting that the blocks are stacked in the same order, except that C is on Peg 3. In addition, there are clauses that ensure that all moves are legal: for example, that only clear blocks are moved, that a larger block is never placed on a smaller one, and so on. The basic GSAT procedure would often find assignments that violated very few constraints, but were quite far from a satisfying assignment. For example, one such assignment would correspond to a plan which simply moved C directly to Peg 3, with all the other blocks on top of it; this would only violate the clause that asserted that C must be clear in order to be moved. Another such assignment just moves the top block back and forth a few times, and ends up with the final state the same as the initial state. This assignment violates the single clause asserting that C is on Peg 3 in the final state.

The use of weights and averaging greatly improves the performance of GSAT on these formulas, as many of the local minima are filled in or at least elevated. However, it still requires many more retries to solve these problems than is required for similar-sized random or coloring problems. As shown in Table 5, combining the random walk strategy with both weights and averaging allows GSAT to almost always climb out of the local minima and find a solution in a reasonable amount of time. The performance is comparable to that of basic backtracking search procedures, though highly optimized procedures with special heuristics can still solve these problems about a 10 times as fast. Nevertheless, the table shows that these strategies come a long way in improving GSAT's performance on problems that appear much more suited for the use of backtrack style search algorithms.[10]

4 Conclusions

We have described three strategies that greatly enhance the power and applicability of GSAT, a randomized greedy local search procedure for propositional satisfiability testing. The effectiveness of these strategies was empirically determined. (We also tested a number of other intuitively plausible strategies which in practice failed to improve GSAT's performance.) The clause weighing strategy and the random walk strategy are useful in escaping from local minima. The weighing strategy is particularly well-suited for dealing with problems with hidden asymmetries. The averaging-in strategy reduces the number of tries necessary to solve certain classes of

[10] Currently we are investigating alternative representations of planning problems that would be more amenable to a local search procedure, in that assignments that violated few constraints would be likely to be close to a satisfying assignment, in terms of the number of flips necessary to reach it. The overall goal is to develop representations that are less fragile than the "classic" logical representations developed by McCarthy and Hayes (1969) and extended to planning as satisfiability by Kautz and Selman (1992).

instances. It does so 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. 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, such as, simulated annealing and genetic algorithms (Davis 1987).

In our future research we hope to develop a precise formal understanding of the benefits and applicability of each technique. While some theoretical results, such as those of Papadimitriou (1991) on random walks, suggest that the strategies are plausible, it is likely to be extremely difficult to prove *formally* that they are effective in practice.[11]

Finally, we should note that we do not claim that GSAT will be able to outperform backtracking search methods on all possible problems. We do, in fact, believe that certain highly structured problems lend themselves better for exhaustive search approaches and domain-specific heuristics (such as means-end analysis in planning). Nevertheless, with the extensions described here, we have considerably enlarged the class of problems on which GSAT performs well, thereby making GSAT a viable general procedure for solving hard computational problems in artificial intelligence.[12]

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, November 1992

Cheeseman, P. and Kanefsky, B. and Taylor, W.M. (1991). Where the Really Hard Problems Are. *Proceedings IJCAI–91,* 1991, 163–169.

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

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

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

[11]A similar situation exists with regard to the huge body of research on simulated annealing. To this day there is no convincing theoretical explanation as to why simulated annealing is as effective as it is *in practice*, and not just in the asymptotic limit (Bertsimas and Tsitsiklis 1992).

[12]Experimental data and code is available via ftp. For requests, contact the first author at selman@research.att.com. Related papers are available via anonymous ftp to research.att.com in dist/ai.

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

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.

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 *Proceedings of the AAAI Spring Symposia*, 1993.

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1991). A continuous approach to inductive inference. *Mathematical Programming*, 57, 1992, 215–238.

Kautz, H.A. and Selman, B. (1992). Planning as satisfiability. *Proceedings ECAI92*, Vienna, Austria.

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.

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

Minton, S., Johnston, M.D., Philips, A.B., and Laird, P. (1990) Solving large-scale constraint satisfaction an 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 AAAI92,* San Jose, CA, 459–465.

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

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

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

Selman, B. and Kautz, H.A. (1993). An Empirical Study of Greedy Local Search for Satisfiability Testing. *Proceedings AAAI93,* to appear.

Selman, B. and Levesque, H.J., and Mitchell, D.G. (1992). A New Method for Solving Hard Satisfiability Problems. *Proceedings AAAI92,* San Jose, CA, 440–446.