In: *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose, CA, July 1992, 440–446.*

A New Method for Solving Hard Satisfiability Problems

Bart Selman
AT&T Bell Laboratories
Murray Hill, NJ 07974
selman@research.att.com

Hector Levesque*
Dept. of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
hector@ai.toronto.edu

David Mitchell
Dept. of Computing Science
Simon Fraser University
Burnaby, Canada V5A 1S6
mitchell@cs.sfu.ca

Abstract

We introduce a greedy local search procedure called GSAT for solving propositional satisfiability problems. Our 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. We also show that GSAT can solve structured satisfiability problems quickly. In particular, we solve encodings of graph coloring problems, N-queens, and Boolean induction. General application strategies and limitations of the approach are also discussed.

GSAT is best viewed as a model-finding procedure. Its good performance suggests that it may be advantageous to reformulate reasoning tasks that have traditionally been viewed as theorem-proving problems as model-finding tasks.

Introduction

The property of *NP-hardness* is traditionally taken to be the barrier separating tasks that can be solved computationally with realistic resources from those that cannot. In practice, to solve tasks that are NP-hard, it appears that something has to be given up: restrict the range of inputs; allow for erroneous outputs; use defaults outputs when resources are exhausted; limit the size of inputs; settle for approximate outputs, and so on. In some cases, this can be done in a way that preserves the essence of the original task. For example, perhaps erroneous outputs occur extremely rarely; perhaps the class of allowable inputs excludes only very large, unlikely, or contrived cases; perhaps the approximate answers can be guaranteed to be close to the exact ones, and so on. In this paper, we propose an algorithm for an NP-hard problem that we believe has some very definite advantages. In particular, it works very quickly (relative to its competition) at the expense of what appears to be statistically minimal errors.

The first computational task shown to be NP-hard by Cook (1971) was propositional satisfiability, or SAT: given a formula of the propositional calculus, decide if there is an assignment to its variables that satisfies the formula according to the usual rules of interpretation. Unlike many other NP-hard tasks (see Garey and Johnson (1979) for a catalogue), SAT is of special concern to AI because of its direct connection to reasoning. Deductive reasoning is simply the complement of satisfiability: Given a collection of base facts Σ, then a sentence α should be deduced iff $\Sigma \cup \{\neg\alpha\}$ is not satisfiable. Many other forms of reasoning (including default reasoning, diagnosis, planning, and image interpretation) also make direct appeal to satisfiability. The fact that these usually require much more than the propositional calculus simply highlights the fact that SAT is both a fundamental task and a major stumbling block to effective reasoners.

Though SAT is originally formulated as decision problem, there are two closely related *search* problems:

1. *model-finding:* find an interpretation of the variables under which the formula comes out true, or report that none exists. If such an interpretation exists, then the formula is obviously satisfiable.
2. *theorem-proving:* find a formal proof (in a sound and complete proof system) of the *negation* of the formula in question, or report that there is no proof. If a proof exists, then the negated formula is valid, and so the original formula is not satisfiable.

Whereas much of the reasoning work in AI has favored theorem-proving procedures (and among these, resolution is the favored method), in this paper, we investigate the behaviour of a new model-finding procedure called GSAT. We will also explain why we think that finding models may be a useful alternative for many AI reasoning problems.

The original impetus for this work was the recent success in finding solutions to very large N-queens problems, first using a connectionist system (Adorf and Johnston 1990), and then using greedy local search (Minton *et al.* 1990). To us, these results simply indicated that N-queens was an *easy* problem. We felt that such techniques would fail in practice for SAT. But this

*Fellow of the Canadian Institute for Advanced Research, and E. W. R. Steacie Fellow of the Natural Sciences and Engineering Research Council of Canada.

appears not to be the case. The issue is clouded by the fact that some care is required to randomly generate SAT problems that are hard for even ordinary backtracking methods.[1] But once we discovered how to do this (and see Mitchell *et al.* (1992) for details), we found that GSAT's local search was very good at finding models for the hardest formulas we could generate.

Because model-finding is NP-hard, we cannot expect GSAT to solve it completely and exactly within tolerable resource bounds. What we will claim, however, is that the *compromises* it makes are quite reasonable. In particular, we will compare GSAT to another procedure DP (which is, essentially, a version of resolution adapted to model-finding) and demonstrate that GSAT has clear advantages. But there is no free lunch: we can construct satisfiable formulas for which GSAT may take an exponential amount of time, unless told to stop earlier. However, these satisfiable counter-examples do appear to be extremely rare, and do not occur naturally in the applications we have examined.

In the next section, we give a detailed description of the GSAT procedure. We then present test results of GSAT on several classes of formulas. This is followed by a discussion of the limitations of GSAT and some potential applications. In the final section, we summarize our main results.

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.

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 presence of sideways moves, dis-

[1]After the current paper was prepared for publication, we were surprised to discover that a procedure very similar to ours had been developed independently, and was claimed to solve instances of SAT substantially larger than those discussed here (Gu 1992). It is tempting, however, to discount that work since the large instances involved are in fact easy ones, readily solvable by backtracking procedures like DP in a few seconds.

[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. Thus, GSAT handles CNF SAT.

procedure GSAT
Input: a 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 procedure GSAT.

cussed below. 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 very 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.

It should be clear that GSAT could fail to find an assignment even if one exists, *i.e.* GSAT is incomplete. We will discuss this below.

Experimental results

We tested GSAT on several classes of formulas: random formulas, graph coloring encodings, N-queens encodings, and Boolean induction problems. For purposes of comparison, we ran the tests with the Davis-Putnam procedure (DP) (Davis and Putnam 1960).

The DP procedure

DP is in essence a resolution procedure (Vellino 1989). It performs a backtracking search in the space of all truth assignments, incrementally assigning values to

formulas		GSAT			DP		
vars	clauses	M-FLIPS	tries	time	choices	depth	time
50	215	250	6.4	0.4s	77	11	1.4s
70	301	350	11.4	0.9s	42	15	15s
100	430	500	42.5	6s	84×10^3	19	2.8m
120	516	600	81.6	14s	0.5×10^6	22	18m
140	602	700	52.6	14s	2.2×10^6	27	4.7h
150	645	1500	100.5	45s	—	—	—
200	860	2000	248.5	2.8m	—	—	—
250	1062	2500	268.6	4.1m	—	—	—
300	1275	6000	231.8	12m	—	—	—
400	1700	8000	440.9	34m	—	—	—
500	2150	10000	995.8	1.6h	—	—	—

Table 1: Results for GSAT and DP on *hard* random 3CNF formulas.

variables and simplifying the formula. If no new variable can be assigned a value without producing an empty clause, it backtracks. The performance of the basic DP procedure is greatly improved by using *unit propagation* whenever unit clauses arise:[3] variables occurring in unit clauses are immediately assigned the truth value that satisfies the clause, and the formula is simplified, which may lead to new unit clauses, *etc.* This propagation process can be executed quite efficiently (in time linear in the total number of literals). DP combined with unit propagation is one of the most widely used methods for propositional satisfiability testing.

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 some care is taken in sampling formulas, random satisfiability testing can end up looking surprisingly easy. For example, Goldberg (1979) showed experimentally how DP runs in polynomial average time on a class of random formulas. However, Franco and Paull (1983) demonstrated that the instances considered by Goldberg were so satisfiable that an algorithm that simply guessed truth assignments would find a satisfying one just as quickly as DP! This issue is discussed in detail in (Mitchell *et al.* 1992).

Formulas are generated using the uniform distribution or fixed-clause length model. For each class of formulas, we choose the number of variables N, the number of literals per clause K, and the number of clauses L. 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. As discussed in Mitchell *et al.* (1992), the difficulty of such formulas critically depends on the ratio between N and L. The hardest formulas appear to 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 $L \approx 4.3N$.[4] We should stress that for different ratios of clauses to variables, formulas can become very easy. For example, DP solves 10,000 variable 20,000 clause 3SAT instances in a few seconds, whereas it cannot in practice solve 250 variable 1062 clause instances. In this paper, when we speak of random formulas we mean those in the hardest region only.

Unsatisfiable formulas are of little interest when testing GSAT, since it will always (correctly) return "no satisfying assignment found" in time directly propositional to (MAX-FLIPS $\times$ MAX-TRIES). So we first used DP to select satisfiable formulas to use as test cases. This approach is feasible for formulas containing up to 140 clauses. For longer formulas, DP simply takes too much time, and we can no longer pre-select the satisfiable ones. In such cases, GSAT is tested on both satisfiable and unsatisfiable instances.

Table 1 summarizes our results: first the number of variables and clauses in each formula, and then statistics for GSAT and DP. For formulas containing up to 120 variables, the statistics are based on averages over 100 satisfiable instances; for the larger formulas, the average is based on 10 satisfiable formulas. For GSAT, we report the setting of MAX-FLIPS (in the header shortened to M-FLIPS), how many tries GSAT took before an assignment was found, and the total time used in finding an assignment.[5] The fractional part of the number of tries indicates how many flips it took on the final successful one. So, for example, 6.4 tries in the first row means that an assignment was not found in the first 6

[3] A unit clause is a clause that contains a single literal.

[4] For more than 150 variables per formula, the ratio seems to converge to $4.25N$. In table 1, we have used this ratio for the higher values of N. The exact ratio is not known; the theoretical derivation of the "50% satisfiable" point is a challenging open problem.

[5] Both GSAT and DP were written in C and ran on a MIPS machine under UNIX.

tries of 250 flips, but on the 7th try, one was found after $0.4 \times 250 = 100$ flips. For DP, we give the number of binary choices made during the search, the average depth of the search tree (ignoring unit propagation), and the time it took to find an assignment.

First, note that for each satisfiable formula found by DP, GSAT had no trouble finding an assignment. This is quite remarkable in itself, since one might expect it to almost always hit some local minimum where at least a few clauses remain unsatisfied. But apparently this is not the case. Moreover, as is clear from table 1, the procedure is substantially faster than DP.

The running time of DP increases dramatically with the number of variables with a critical increase occurring around 140 variables. This renders it virtually useless for formulas with more than 140 variables.[6] The behavior of GSAT, on the other hand, is quite different: 300 variable formulas are quite manageable, and even 500 variable formulas can be solved. As noted above, the satisfiability status of these large test cases was initially unknown. Nonetheless, GSAT did still manage to find assignments for a substantial number of them. (See Selman *et al.* (1992) for more details.)

Now consider in table 1 the total number of flips used by GSAT to find an assignment and the total number of binary choices in the DP search tree. Again, we see a dramatic difference in the growth rates of these numbers for the two methods. This shows that the difference in running times is not simply due to some peculiarity of our implementation.[7] So, GSAT appears to be well-suited for finding satisfying assignments for hard random formulas. Moreover, the procedure can handle much larger formulas (up to 500 variables) than DP (up to around 140 variables). Again, we should stress that we have shown these results for the hardest region of the distribution. Like most other procedures, GSAT also solves the "easy" cases quickly (Selman *et al.* 1992).

Graph coloring

In this section, we briefly discuss the performance of GSAT on graph coloring. Consider the problem of coloring with K colors a graph with V vertices such that no two nodes connected by an edge have the same color. We create a formula with K variables for each node of the graph, where each variable corresponds to assigning one of the K possible colors to the node. We have clauses that state that each node must have at least one color, and that no two adjacent nodes have the same color.

Johnson *et al.* (1991) evaluate state-of-the-art graph-coloring algorithms on instances of random graphs. We considered one of the hardest instances discussed: a 125 vertex graph for which results are given in table II of Johnson *et al.* (1991). The encoding that allows for 18 colors consists of 89,288 clauses with 2,250 variables, and an encoding that allows for only 17 colors consists of 83,272 clauses with 2,125 variables. GSAT managed to find the 18-coloring in approximately 5 hours. (DP ran for many more hours but did not find an assignment.) This is quite reasonable given that the running times for the various *specialized* algorithms in Johnson *et al.* ranged from 20 minutes to 1.7 hours. Unfortunately, GSAT did not find a 17-coloring (most likely optimal; Johnson (1991)).[8] This is perhaps not too surprising given that some of the methods in Johnson *et al.* couldn't find one either, while another took 21.6 hours, and the fastest took 1.8 hours. Interestingly, some of the best graph-coloring methods are based on simulated annealing, an approach that shares some important features with GSAT.

So, although it is not as fast as the specialized graph-coloring procedures, GSAT can be used to find near optimal colorings of hard random graphs. Moreover, the problem reformulation in terms of satisfiability does *not* result in a dramatic degradation of performance, contrary to what one might expect. The main drawback of such an encoding appears to be the inevitable polynomial increase in problem size.

N-queens

In the N-queens problem one has to find a placement of N queens on a N $\times$ N chess board such that no queen attacks another. Although a generic solution to the problem is known (Falkowski *et al.* 1986), it is based on placing the queens in a very specific, regularly repeated pattern on the board. The problem of finding *arbitrary* solutions has been used extensively to test constraint satisfaction algorithms.

Using standard backtracking techniques, the problem appears to be quite hard. But in a recent paper, Minton *et al.* (1990) show how one can generate solutions by starting with a random placement of the queens (one in each row) and subsequently moving the queens around within the rows, searching for a solution. This method works remarkably well: their method appears to scale linearly with the number of queens.[9]

To test GSAT on the N-queens problem, we first translate the problem into a satisfiability question: we

[6] A recent implementation of a highly optimized variant of DP incorporating several special heuristics is able to handle hard random formulas of up to 200 variables (Crawford and Auton, personal communication 1991).

[7] If the depth continues to grow at its current rate, the DP search tree for 500 variable formulas could have as many as 2^{100} nodes. Even when processing 10^{10} nodes per second, DP could take 10^{12} years to do a complete search.

[8] By using initial assignments that are not completely random, as suggested by Geoff Hinton, we have recently been able to solve also this instance (Selman *et al.* 1992).

[9] There are, it should be mentioned, notable differences in Minton's and our approaches. One is the use of sideways moves. This appears essential in satisfiability testing, discussed below. Also, GSAT chooses the variable that gives the best possible improvement, while Minton's program selects an arbitrary queen and moves it to reduce conflicts.

formulas			GSAT		
Queens	vars	clauses	flips	tries	time
8	64	736	105	2	0.1s
20	400	12560	319	2	0.9s
30	900	43240	549	1	2.5s
50	2500	203400	1329	1	17s
100	10000	1.6×10^6	5076	1	195s

Table 2: Results for GSAT on CNF encodings of the N-queens problem.

use one variable for each of the N^2 squares of the board, where intuitively, a variable is true when a queen is on the corresponding square. To encode the N-queens problem, we use N disjunctions (each with N variables) stating that there is at least one queen in each row, and a large number of binary disjunctions stating that there are no two queens in any row, column, or diagonal.

Table 2 shows the performance of GSAT on these formulas.[10] For N larger than 30, a solution is always found on the first try.[11] Also, the number of flips is roughly 0.5 N^2. This is near optimal, since a random truth assignment places about that many queens on the board, and most of them must be removed. (On the order of N flips are needed if one starts with approximately N queens randomly placed on the board in the initial state (Selman *et al.* 1992).) One of the most interesting aspects of this approach is that so few natural constraints (such as the obvious one of using only N queens) are maintained during the search. Nonetheless, solutions are found quickly.

Boolean induction

Promising results have recently been obtained using integer programming techniques to solve satisfiability problems (Hooker 1988; Kamath *et al.* 1991). Most of the experimental evaluations of these methods have been based on the constant-density random clause model, which unfortunately under-represents hard instances (Mitchell *et al.* 1992). To compare GSAT and these methods, we considered the formulas as studied by Kamath *et al.* (1991) in their work on Boolean induction. In Boolean induction, the task is to derive ("induce") a logical circuit from its input-output behavior. Kamath *et al.* give a translation of this problem into a satisfiability problem. They present test results for their algorithm on these formulas. We considered the formulas presented in table 4.4 in Kamath *et al.* (1991).

Table 3 shows our results. The performance of GSAT is comparable to the integer programming method, which is somewhat surprising given its relative simplicity. Further testing is needed to determine whether there are classes of formulas on which the methods behave very differently.

[10]The size of our propositional encodings prevented us from considering problems with more than 100 queens.

[11]For fewer queens it may sometimes take a second try. This happens rarely though; about 1 in every 100 tries.

formula			time	
id	vars	clauses	Int. progr.	GSAT
16A1	1650	19368	2039s	1061s
16B1	1728	24792	78s	2764s
16C1	1580	16467	758s	7s
16D1	1230	15901	1547s	63s
16E1	1245	14766	2156s	5s

Table 3: Results for GSAT on encodings of Boolean induction problems as given table 4.4 of in Kamath *et al.* (1991).

Limitations and sideways moves

So far, we have concentrated mainly on the strengths of GSAT. But it does also have some important limitations. The following conjunction of clauses shows that it can be "misled" into exploring the wrong part of the search space (numbers stand for propositional variables):

$$\begin{array}{llll}
(1 \vee \neg 2 \vee 3) & \wedge & (1 \vee \neg 3 \vee 4) & \wedge \\
(1 \vee \neg 4 \vee \neg 2) & \wedge & (1 \vee 5 \vee 2) & \wedge \\
(1 \vee \neg 5 \vee 2) & \wedge & (\neg 1 \vee \neg 6 \vee 7) & \wedge \\
(\neg 1 \vee \neg 7 \vee 8) & \wedge & \ldots & \wedge \\
(\neg 1 \vee \neg 98 \vee 99) & \wedge & (\neg 1 \vee \neg 99 \vee 6) &
\end{array}$$

Note that although most of clauses here contain a negative occurrence of variable 1, the formula can only be satisfied if variable 1 is assigned positively (see the first 5 clauses). The problem is that the greedy approach repeatly steers the search towards a negative assignment, since this does satisfy so many of the clauses. The only way GSAT will solve this example is if starts a search very close to a satisfying assignment, which could take an exponential number of tries.

Finally, we consider sideways moves. In a departure from standard local search algorithms, GSAT continues flipping variables even when this does not increase the total number of satisfied clauses.[12] To show why this is important, we re-ran some experiments, but only allowing flips that *increase* the number of satisfied clauses, restarting otherwise.

Table 4 gives the results. All formulas considered were satisfiable. We tried 100 instances of the random formulas. The %-solved column shows what percentage of those instances was solved. Note that quite often *no* assignment was found, despite a very large number of tries. For comparison, we included our previous data on these formulas. It is clear that finding an assignment becomes much harder without the use of sideways moves.

[12]We have also seen cases where an assignment was found after a sequence of flips containing some that *decreased* the number of satisfied clauses, but these are very rare. Here we ignore such flips.

type	formulas		M-TRIES	no sideway moves			all moves		
	vars	clauses		%-solved	tries	time	%-solved	tries	time
random	50	215	1000	69%	537	10s	100%	6	1.4s
random	100	430	100,000	39%	63,382	15m	100%	81	2.8m
30-queens	900	43240	100,000	100%	50,000	30h	100%	1	2.5s

Table 4: Comparing GSAT with and without sideway moves. (MAX-TRIES is shortened to M-TRIES.)

Applications

As we noted above, GSAT is a sound but incomplete model-finding procedure: when it succeeds in finding an interpretation, we know that it is correct; but negative answers, although perhaps suggestive, are not conclusive. The practical value of GSAT for theorem-proving purposes, where the concern is precisely for *unsatisfiability*, is therefore limited. Fortunately, certain AI tasks have naturally been characterized as model-finding tasks, for example, the visual interpretation task (Reiter and Mackworth 1990). In addition, it is often possible to reformulate tasks that have traditionally been viewed as theorem-proving problems as model-finding ones. One example is the formulation of planning as a model-finding task (Kautz and Selman 1992), and we suspect that there will be many others.

Another potential application of GSAT lies in the generation of "vivid" representations (Levesque 1986) as a way of dealing with the computational problems encountered in knowledge representation and reasoning systems. Determining what can be deduced from a knowledge base is intractable in general, but not if the knowledge is vivid in form. So, instead of relying on general theorem-proving in a knowledge-based system, one could use a two-step operation: first, use a model-finding procedure like GSAT off-line to generate one or more vivid representations (or models) of what is known; then, as questions arise, answer them efficiently by appealing to these vivid representations. Efficient model-finding procedures like GSAT have therefore the potential of making the vivid reasoning approach and the related model-checking proposal by Halpern and Vardi (1991) workable.[13]

Conclusions

We have introduced a new method for finding satisfying assignments of propositional formulas. GSAT performs a greedy local search for a satisfying assignment. The method is simple, yet surprisingly effective. We showed how the method outperforms the Davis-Putnam procedure by an order of magnitude on hard random formulas. We also showed that GSAT performs well on graph coloring problems, N-queens encodings, and Boolean induction problems. The price we pay is that GSAT is incomplete.

Currently, there is no good explanation for GSAT's performance. Some recent results by Papadimitriou (1991) and Koutsoupias and Papadimitriou (1992) do, however, provide some initial theoretical support for the approach. Our sense is that the crucial factor here is having some notion (however crude) of an approximate solution that can be refined iteratively. In these terms, model-finding has a clear advantage over theorem-proving, and may lead us to AI methods that scale up more gracefully in practice.

[13] Most applications of GSAT would require formulas of first-order logic. If the Herbrand universe in question is finite, the generalization is straightforward. Otherwise, one approach we intend to investigate is to use a form of *iterative deepening* by searching for models in ever larger Herbrand universes.

Acknowledgments

The second author was funded in part by the Natural Sciences and Engineering Research Council of Canada, and the Institute for Robotics and Intelligent Systems.

We thank David Johnson for providing us with the hard instances of graph coloring and Anil Kamath for the inductive inference problems. We also thank Larry Auton, Ron Brachman, Jim Crawford, Matt Ginsberg, Geoff Hinton, David Johnson, Henry Kautz, David McAllester, Steve Minton, Christos Papadimitriou, Ray Reiter, Peter Weinberger, and Mihalis Yannakakis for useful discussions.

References

Adorf, H.M., Johnston, M.D. (1990). A discrete stochastic neural network algorithm for constraint satisfaction problems. *Proc. of the Int. Joint Conf. on Neural Networks*, San Diego, CA, 1990.

Cook, S.A. (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing,* 1971, 151–158.

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

Falkowski, Bernd-Jurgen and Schmitz, Lothar (1986). A note on the queens' problem. *Information Process. Lett.*, 23, 1986, 39-46.

Franco, J. and Paull, M. (1983). Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Math.* 5, 1983, 77–87.

Garey, M.R. and Johnson, D.S. (1979). *Computers* *and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman, New York, NY, 1979.

Goldberg, A. (1979). *On the complexity of the satisfiability problem.* Courant Computer Science Report. No. 16, New York University, NY, 1979.

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

Halpern, J.Y. and Vardi, M.Y. (1991) Model checking vs. theorem proving: a manifesto. Proceedings KR-91, Boston, MA, 325–334.

Hooker, J.N. (1988) Resolution vs. cutting plane solution of inference problems: Some computational experience. *Operations Research Letter*, 7(1), 1988.

Johnson, D.S. (1991) Personal communication, 1991.

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.

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1991). A continuous approach to inductive inference. Submitted for publication.

Kautz, H.A. and Selman, B. (1992). Planning as satisfiability. Forthcoming.

Koutsoupias, E. and Papadimitriou C.H. (1992) On the greedy algorithm for satisfiability. Forthcoming.

Levesque, H.J. (1986). Making believers out of computers. *Artificial Intelligence*, **30**, 1986, 81–108.

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.

Mitchell, D., Selman, B., and Levesque, H.J. (1992). Hard and easy distributions of SAT problems. Forthcoming.

Papadimitriou, C.H. (1991). On selecting a satisfying truth assignment. *Proc. of 32th 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.

Reiter, R. and Mackworth, A. (1989). A logical framework for depiction and image interpretation. *Artificial Intelligence*, **41**, No. 2, 1989, 125-155.

Selman, B., Levesque, H.J., Mitchell, D. (1992) GSAT: A new method for solving hard satisfiability problems. Technical Report, AT&T Bell Laboratories, 1992.

Vellino, A. (1989) The complexity of automated reasoning. Ph.D. thesis, Dept. of Philosophy, University of Toronto, Toronto, Canada (1989).