In: *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose, CA, July 1992, 459–465.*

Hard and Easy Distributions of SAT Problems

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

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

Abstract

We report results from large-scale experiments in satisfiability testing. As has been observed by others, testing the satisfiability of random formulas often appears surprisingly easy. Here we show that by using the right distribution of instances, and appropriate parameter values, it is possible to generate random formulas that are hard, that is, for which satisfiability testing is quite difficult. Our results provide a benchmark for the evaluation of satisfiability-testing procedures.

Introduction

Many computational tasks of interest to AI, to the extent that they can be precisely characterized at all, can be shown to be NP-hard in their most general form. However, there is fundamental disagreement, at least within the AI community, about the implications of this. It is claimed on the one hand that since the performance of algorithms designed to solve NP-hard tasks degrades rapidly with small increases in input size, something will need to be given up to obtain acceptable behavior. On the other hand, it is argued that this analysis is irrelevant to AI since it based on worst-case scenarios, and that what is really needed is a better understanding of how these procedures perform "on average".

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 makes the formula true according to the usual rules of interpretation. Subsequent tasks have been shown to be NP-hard by proving they are at least as hard as SAT. Roughly, a task is NP-hard if a good algorithm for it would entail a good algorithm for SAT. 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 relationship to deductive reasoning (*i.e.*, given a collection of base facts Σ, a sentence α may 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 a fundamental task, and that developing SAT procedures that work well in AI applications is essential.

We might ask when it is reasonable to use a sound and complete procedure for SAT, and when we should settle for something less. Do hard cases come up often, or are they always a result of strange encodings tailored for some specific purpose? One difficulty in answering such questions is that there appear to be few applicable *analytical* results on the expected difficulty of SAT (although see below). It seems that, at least for the time being, we must rely largely on empirical results.

A number of papers (some discussed below) have claimed that the difficulty of SAT on randomly generated problems is not so daunting. For example, an often–quoted result (Goldberg, 1979; Goldberg *et al.* 1982) suggests that SAT can be readily solved "on average" in $O(n^2)$ time. This does not settle the question of how well the methods will work *in practice*, but at first blush it does appear to be more relevant to AI than contrived worst cases.

The big problem is that to examine how well a procedure does on average one must assume a distribution of instances. Indeed, as we will discuss below, Franco and Paull (1983) refuted the Goldberg result by showing that it was a direct consequence of their choice of distribution. It's not that Goldberg had a clever algorithm, or that the problem is easy, but that they had used a distribution with a preponderance of easy instances. That is, from the space of all problem instances, they sampled in a way that produced almost no hard cases.

Nevertheless, papers continue to appear purporting to empirically demonstrate the efficacy of some new procedure, but using just this distribution (*e.g.*, Hooker, 1988; Kamath *et al.* 1990), or presenting data suggesting that very large satisfiability problems — with thousands of propositional variables — can be solved. In fact, we are presenting one of the latter kind our-

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

PROCEDURE DP

Given a set of clauses Σ defined over a set of variables V:

- If Σ is empty, return "satisfiable".
- If Σ contains an empty clause, return "unsatisfiable".
- (Unit-Clause Rule) If Σ contains a unit clause c, assign to the variable mentioned the truth value which satisfies c, and return the result of calling DP on the simplified formula.
- (Splitting Rule) Select from V a variable v which has not been assigned a truth value. Assign it a value, and call DP on the simplified formula. If this call returns "satisfiable", then return "satisfiable". Otherwise, set v to the opposite value, and return the result of calling DP on the re-simplified formula.

Figure 1: The DP procedure with unit propagation.

selves (Selman *et al.*, 1992)! How are we to evaluate these empirical results, given the danger of biasing the sample to suit the procedure in question, or of simply using easy problems (even if unwittingly)?

In this paper, we present empirical results showing that random instances of satisfiability can be generated in such a way that easy and hard sets of instances (for a particular SAT procedure, anyway) are predictable in advance. If we care about the *robustness* of the procedures we develop, we will want to consider their performance on a wide spectrum of examples. While the easy cases we have found can be solved by almost *any* reasonable method, it is the hard cases ultimately that separate the winners from the losers. Thus, our data is presented as challenging test material for developers of SAT procedures (see Selman *et al.*, 1992, for example).

The SAT procedure we used for our tests is the Davis-Putnam procedure, which we describe below. We believe this was a good choice for two reasons: First, it has been shown to be a variant of resolution (Vellino 1989, Galil 1977), the most widely used general reasoning method in AI; second, almost all empirical work on SAT testing has used one or another refinement of this method, which facilitates comparison. We suspect that our results on hard and easy areas generalize to *all* SAT procedures, but this remains to be seen.

The rest of the paper is organized as follows. In the next two sections we describe the Davis-Putnam procedure, and consider it's performance on one distribution of formulas, the fixed clause-length model. We show that with the right parameter values, it produces computationally challenging SAT instances. In the following section we consider a second distribution, the constant-probability model, and argue that it is not useful in the evaluation of satisfiability-testing procedures. We then briefly review related work, and summarize our results.

The Davis-Putnam Procedure

The Davis-Putnam (DP) procedure (Davis and Putnam 1960) is sketched in Figure 1. It takes as input a set of clauses Σ over a set of variables V, and returns either "satisfiable" or "unsatisfiable." (A clause is a disjunction of literals. A set of clauses represents a conjunction of disjunctions, *i.e.*, a formula in conjunctive normal form (CNF).) It performs a backtracking depth-first search in the space of all truth assignments, incrementally assigning truth values to variables and simplifying the formula. If no new variable can be assigned a value without producing an empty clause, it backtracks by changing a previously made assignment. The performance of simple backtracking is greatly improved by employing *unit propagation*: Whenever a unit clause (one containing a sinle literal) arises, the variable occurring in that clause is immediately assigned the truth value that satisfies it. The formula is then simplified, which may lead to new unit clauses, and so on. This propagation process can be executed 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. (It is also common to include the "pure literal rule", which we excluded from this implementation because it is relatively expensive, and seemed to provide only a modest improvement on the formulas we consider here.)

The fixed clause-length model

In this section, we study formulas generated using the fixed clause-length model, which we call *Random K-SAT*. There are three parameters: the number of variables N, the number of literals per clause K, and the number of clauses L. To keep the volume of data presented manageable and yet give a detailled picture, we limit our attention to formulas with $K = 3$, that is, random 3-SAT. For a given N and L, an instance of random 3-SAT is produced by randomly generating L clauses of length 3. Each clause is produced by randomly choosing a set of 3 variables from the set of N available, and negating each with probability 0.5.

We now consider the performance of DP on such random formulas. Figure 2 shows the total number of recursive calls by DP to find one satisfying assignment, or to determine that the formula is unsatisfiable. There are three curves, for formulas with 20, 40, and 50 variables. Along the horizontal axis is the number of clauses in the formulas tested, normalized through division by the number of variables. Each data point gives the median number of calls for a sample size of 500.[1]

In figure 2, we see the following pattern: For formulas that are either relatively short or relatively long, DP finishes quickly, but the formulas of medium length take much longer. Since formulas with few clauses are *under-constrained* and have many satisfying assignments, an

[1]The means of the number of calls are influenced by a very small number of very large values. As the median is less sensitive to such "out-liers", it appears to be a more informative statistic.

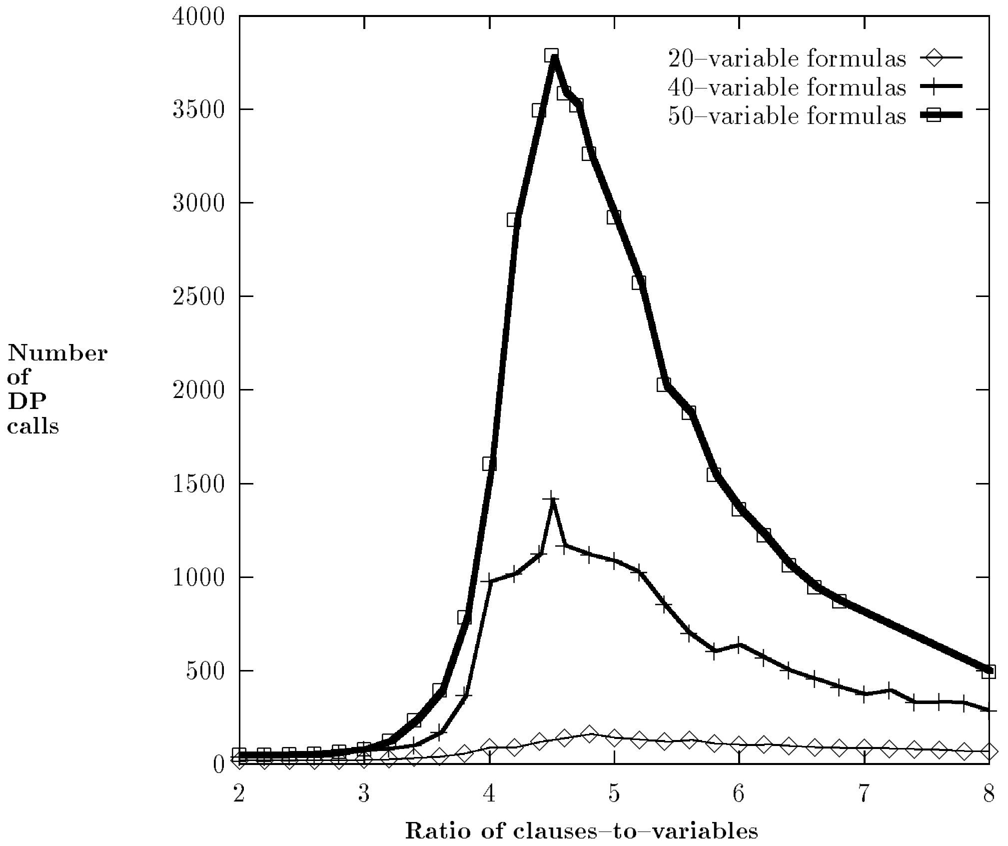

Figure 2: Median number of recursive DP calls for Random 3-SAT formulas, as a function of the ratio of clauses–to–variables.

assignment is likely to be found early in the search. Formulas with very many clauses are *over-constrained* (and usually unsatisfiable), so contradictions are found easily, and a full search can be completed quickly. Finally, formulas in between are much harder because they have relatively few (if any) satisfying assignments, but the empty clause will only be generated after assigning values to many variables, resulting in a deep search tree. Similar under- and over-constrained areas have been found for random instances of other NP-complete problems (see the discussion of the work of Cheeseman *et al.* (1991), below).

The curves in figure 2 are for *all* formulas of a given size, that is they are composites of satisfiable and unsatisfiable subsets. In figure 3 the median number of calls for 50-variable formulas is factored into satisfiable and unsatisfiable cases, showing that the two sets are quite different. The extremely rare unsatisfiable short formulas are very hard, whereas the rare long satisfiable formulas remain moderately difficult. Thus, the easy parts of the composite distribution appear to be a consequence of a relative abundance of short satisfiable formulas or long unsatisfiable ones.

To understand the hard area in terms of the likelihood of satisfiability, we experimentally determined the probability that a random 50-variable instance is satisfiable (figure 4). There is a remarkable correspondence between the peak on our recursive calls curve and the point where the probability that a formula is satisfiable is 0.5. The main empirical conclusion we draw from this is that *the hardest area for satisfiability is near the point where 50% of the formulas are satisfiable.*

This "50% satisfiable" point seems to occur at a fixed ratio of the number of clauses to the number of variables: when the number of clauses is about 4.3 times the number of variables. There is a boundary effect for small formulas: for formulas with 20 variables, the point occurs at 4.55; for 50 variables, at 4.31; and for 140 variables, at 4.3. While we conjecture that this ratio approaches about 4.25 for very large numbers of variables, it remains a challenging open problem to *analytically* determine the "50% satisfiable" point as a function of the number of variables.

Finally, note that we did not specify a method for choosing which variable to guess in the "splitting" step of DP. In our implementation, we simply set variables in lexical order (except when there are unit clauses.) DP can be made faster by using clever selection strategies (*e.g.*, Zabih and McAllester 1988), but it seems unlikely that such heuristics will *qualitatively* alter the

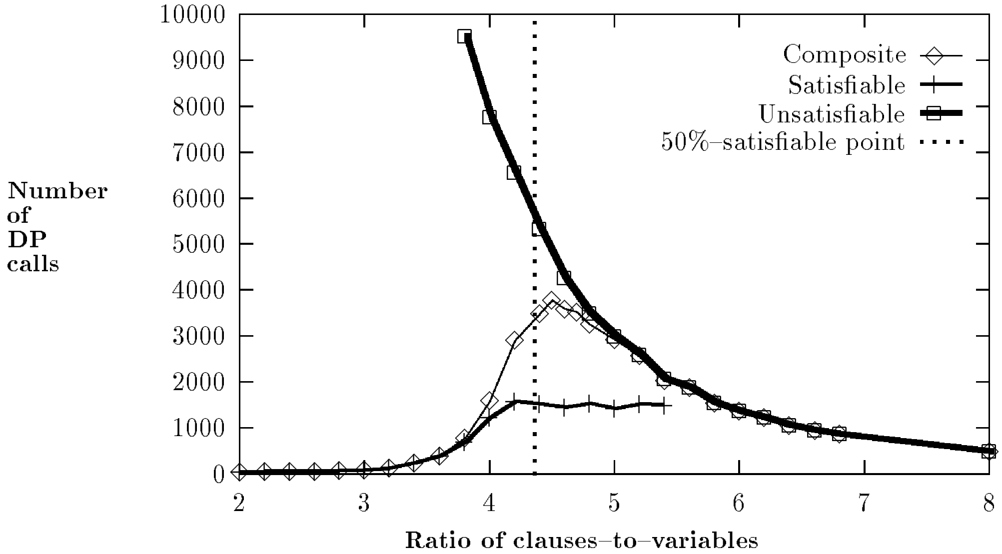

Figure 3: Median DP calls for 50–variable Random 3-SAT as a function of the ratio of clauses–to–variables.

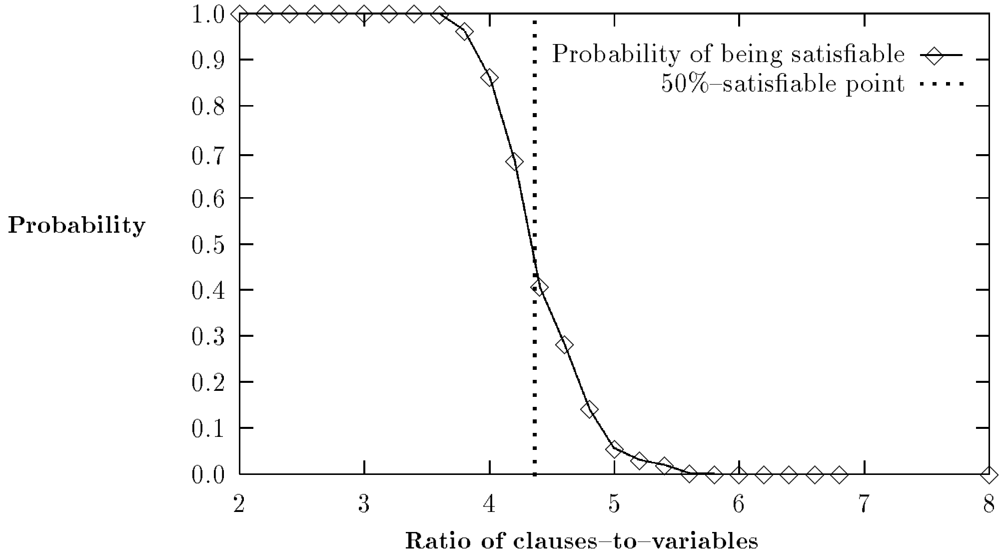

Figure 4: Probability of satisfiability of 50–variable formulas, as a function of the ratio of clauses–to–variables.

easy-hard-easy pattern. The formulas in the hard area appear to be the most challenging for the strategies we have tested, and we conjecture that they will be for every (heuristic) method.

The constant-probability model

We now examine formulas generated using the constant-probability model. The model has three parameters: the number of variables N, and number of clauses L as before; but instead of a fixed clause length, clauses are generated by including a variable in a clause with some fixed probability P, and then negating it with probability 0.5. Large formulas generated this way very often have at least one empty clause and several unit clauses, so that they tend to be either trivially unsatisfiable, or easily shown satisfiable. Thus, the more interesting results are for the modified version in which empty and unit clauses are disallowed. This distribution we call *Random P-SAT.*

Analytic results by Franco and Paull (1983) suggest that one probably cannot generate computationally challenging instances from this model, and our experiments confirm this prediction. In Figure 5, we compare the number of recursive DP calls to solve instances of random P-SAT with the same figures for random 3-SAT. Although we see a slight easy-hard-easy pattern (previously noted by Hooker and Fedjki, 1989), the hard area is not nearly so pronounced, and in absolute terms

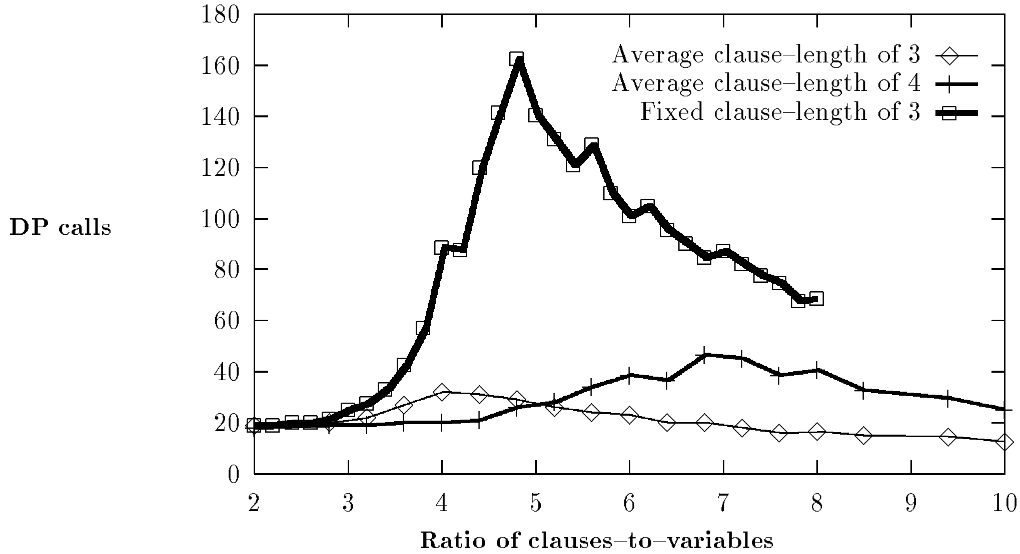

Figure 5: Comparison of median DP calls for 20–variable fixed–length and constant–probability formulas.

the random P-SAT formulas are much easier than random 3-SAT formulas of similar size. Franco and Paull's analysis (discussed below) and our experimental results show that the constant-probability model is not suitable for the evaluation of satisfiability testing procedures.

Related work

There is a large body of literature on testing the satisfiability of random formulas, reporting mostly analytic results. The main impetus for this research was early experimental results by Goldberg (1979), which suggested that SAT might in fact be easily solvable, on average, using DP. Franco and Paull (1983) showed that Goldberg's positive results were a direct consequence of the distribution used — a variant of the constant-probability model — and thus overly optimistic. Goldberg's formulas were so easily satisfiable that an algorithm which simply tried randomly generated assignments would, with probability 1, find a satisfying assignment in a constant number of guesses. Further analytic results for the constant-probability model can be found in Franco (1986), Franco and Ho (1988).

Franco and Paull (1983) also investigated the performance of DP on random 3-SAT, and suggested that it might be more useful for generating random instances, an hypothesis we have confirmed experimentally here. They showed that for any fixed ratio of clauses to variables, if DP is forced to find *all* satisfying truth assignments (rather than stopping at the first one found, as our version does), its expected time will be exponential in the number of variables, with probability approaching 1 as the number of variables approaches infinity. Unfortunately, this result does not directly tell us much about the expected time to find a single assignment.

A recent result by Chvatal and Szemeredi (1988) can be used to obtain some further insight. Extending a ground-breaking result by Haken (1985), they showed that any resolution strategy requires exponential time with probability 1 on formulas where the ratio of clauses to variables is a constant greater than 5.6. (They also show that with probability approaching 1 such formulas are unsatisfiable.) Given that DP corresponds to a particular resolution strategy, as mentioned above, it follows that on such formulas the average time complexity of DP is exponential.

This may appear inconsistent with our claim that over-constrained formulas are easy, but it is not. Chvatal and Szemeredi's result holds for constant ratios of clauses to variables as *both* to go to infinity. So for any fixed ratio of clauses to variables well beyond the 50%-satisfiable point, there is some (possibly very large) number such that, whenever the number of variables exceeds this number, DP is very likely to take exponential time. For formulas with fewer variables (and clauses), however, DP may still finish quickly. For example, our experiments show that DP consistently takes only several seconds to determine the unsatisfiability of 1000-variable, 50,000-clause instances of 3-SAT (even though there are some 250-variable 1062-clause formulas that it cannot practically solve). For this large ratio of 50, the exponential behavior does not appear to occur for formulas with 1000 or fewer variables: those formulas lie in what we have termed the "easy area." But for formulas with larger numbers of variables, eventually the truly easy area will occur only at ever higher ratios of clauses to variables. Of course, even without increasing this ratio, we suspect that the formulas will nonetheless be relatively easy in comparison to those at the 50% satisfiable point.

Turning to under-constrained formulas, the behavior of DP can be explained by the fact that they tend to have very many satisfying truth assignments — a fact we have verified experimentally — so that the procedure almost always finds one early in the search. Other analytic results for random 3-SAT are reviewed in Chao and Franco (1990), and Franco (1986).

Not only are our experimental results consistent with the analytic results, they also provide a more fine-grained picture of how 3-SAT may behave in practice. One reason for the limitations of analytic results is the complexity of the analyses required. Another is that they are asymptotic, *i.e.*, they hold in the limit as the number of variables goes to infinity, so do not necessarily tell us much about formulas that have only a modest number of variables (say, up to a few thousand) as encountered in practice.

Finally, we would like to mention the valuable contribution of a recent paper by Cheeseman *et al.*, (1991). This paper explores the hardness of random instances of various NP-complete problems, their main results being for graph coloring and Hamiltonian circuit problems. They observe a similar easy-hard-easy pattern as a function of one or more parameters of instance generation. They also give some preliminary results for satisfiability. Unfortunately, they do not describe exactly how the formulas are generated, and their findings are based on relatively small formulas (up to 25 variables). Also, their backtrack search procedure does not appear to incorporate unit resolution, which would limit its ability to find quick cutoff points. Possibly because of the preliminary nature of their investigation, they observe that they do not know how to generate hard SAT instances except via transformation of hard graph coloring problems. Although there are similarities in pattern, their hard area appears to be at a different place than in our data, suggesting that the mapping process generates a distribution somewhat different than random 3-SAT. We also note that when formulas are not generated randomly, but encode some other NP-complete problem, such as graph coloring, the ratio of clauses to variables may not be a good indicator of expected difficulty. Moreover, it is possible to arbitrarily change the clause-to-variable ratio of a formula by "padding" it, without substantially affecting its difficulty.

Conclusions

There has been much debate in AI on the importance of worst-case complexity results, such as NP-hardness results. In particular, it has been suggested that satisfiability testing might be quite easy on average.

We have carried out a detailed study of the average-case difficulty of SAT testing for random formulas. We confirmed previous observations that many instances are quite easy, but we also showed how hard instances can be generated. The fixed clause-length model with roughly 4.3 times as many clauses as variables gives computationally challenging instances which have about a 0.5 probability of being satisfiable. Randomly generated formulas with many more or fewer clauses are quite easy.

Our data provide two important lessons. The first is that the constant-probability model is inappropriate for evaluating satisfiability procedures, since it seems to be dominated by easy instances for all values of the parameters. The second is that it is not necessarily the case that generating larger formulas provides harder formulas. For example, one can solve 1000 variable 3000 clause (under-constrained) and 1000 variable 50000 clause (over-constrained) random 3-SAT instances in seconds using DP. On the other hand, even a highly optimized variant of DP[2] cannot solve random 3-SAT instances with 250 variables and 1075 clauses.

Because random 3-SAT instances can be readily generated, those from the hard area can be very useful in the evaluation of satisfiability testing procedures, and algorithms for related tasks such as Boolean constraint satisfaction. We hope that our results will help prevent further inaccurate or misleading reports on the average case performance of SAT procedures.

Acknowledgments

We thank Henry Kautz for many useful discussions and comments, and Fahiem Bacchus for helpful comments on an earlier draft. The first and third authors were funded in part by the Natural Sciences and Engineering Research Council of Canada, and the Institute for Robotics and Intelligent Systems.

References

Chao, Ming-te and Franco, John (1990). Probabilistic Analysis of a Generalization of the Unit-Clause Literal Selection Heuristics for the k Satisfiability Problem. *Inform. Sci.*, 51, 1990, 23–42.

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

Chvatal,V. and Szemeredi, E. (1988). Many hard examples for resolution. *JACM*, val. 35, no. 4, 1988, 759–208.

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.*, 7, 1960, 201–215.

Franco, J. (1986). Probabilistic analysis of algorithms for NP-complete problems. United States Air Force Annual Scientific Report, 1986.

Franco, J. and Ho, Y.C. (1988). Probabilistic performance of a heuristic for the satisfiability problem. *Discrete Applied Math.*, 22, 1988, 35-51.

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.

[2] Recently developed by Jim Crawford and Larry Auton (1992, personal communication).

Galil, Zvi (1977). On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science*, 4, 1977, 23–46.

Gallo, Giorgio and Urbani, Giampaolo (1989). Algorithms for Testing the Satisfiability of Propositional Formulae. *J. Logic Programming*, 7, 1989, 45–61.

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

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

Goldberg, A., Purdom, Jr. P.W., and Brown, C.A. (1982). Average time analysis of simplified Davis-Putnam procedures. *Information Process. Lett.*, 15, 1982, 72–75; see also "Errata", vol. 16, 1983, p. 213.

Kamath, A.P., Karmarker, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1990). Computational experience with an interior point algorithm on the satisfiability problem. *Proceedings, Integer Programming and Combinatorial Optimization*, Waterloo, Canada: Mathematical Programming Society, 1990, 333–349.

Haken, A. (1985). The intractability of resolution. *Theoretical Computer Science*, **39**, 1985, 297–308.

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

Hooker, J.N. and Fedjki, C. (1989). *Branch-and-cut solution of inference problems in propositional logic*, Working paper 77-88-89, Graduate School of Industrial Administration, Carnegie Mellon University, 1989.

Selman, B. and Levesque, H.J., and Mitchell, D.G. (1992). A New Method for Solving Hard Satisfiability Problems. This proceedings.

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

Zabih, R. and McAllester, D. (1988). A rearrangement search strategy for determining propositional satisfiability. *Proc. AAAI-88*, 1988, 155–160.