Generating Hard Satisfiability Problems ⋆

Bart Selman [b], David Mitchell [a,1] and Hector J. Levesque [c,2]

[a] *Dept. of Computer Science, University of Toronto, Toronto, Canada M5S 1A4*
email: mitchell@ai.toronto.edu

[b] *AI Principles Research Dept., AT&T Bell Laboratories, Murray Hill, NJ 07974*
email: selman@research.att.com

[c] *Dept. of Computer Science, University of Toronto, Toronto, Canada M5S 1A4*
email: 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.

1 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 is based on worst-case scenarios, and that what is

⋆ An earlier version of this paper [26] was presented at AAAI-92. *Correspondence to:* Bart Selman, AT&T Bell Laboratories, Murray Hill, NJ 07974. Telephone: (908) 582-2221. E-mail: selman@research.att.com.

[1] Work carried out while visiting AT&T Bell Laboratories.

[2] Fellow of the Canadian Institute for Advanced Research. Supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

really needed is a better understanding of how these procedures perform "on average".

The first computational task shown to be NP-hard, by Cook [9], 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 [16] 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 by Goldberg [18] suggests that SAT can be readily solved "on average" in polynomial 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 [14] refuted the Goldberg result by showing that it was a direct consequence of the choice of distribution. It is not that Goldberg had a clever algorithm, or that the problem is easy, but that he 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.*, [20,22]), or presenting data suggesting that very large satisfiability problems — with thousands of propositional variables — can be solved. 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 [29], 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 is equivalent to a particular case of resolution [18], the most widely used general reasoning method in AI; second, until recently, 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.

This paper is an extension of a previous report [26]. The primary observations and arguments remain the same, but we have used much larger sample sizes (10,000 formulas per point, rather than 500), which eliminates most of the sampling error and so gives a clearer picture of the behaviors of interest. We have added data in Section 3 on the average cost to find all satisfying truth assignments (rather than just one), and the median number of satisfying truth assignments. We have also improved our data comparing different random formula models (Section 4). Some discussions have been extended or clarified, and some updated references have been added, especially to theoretical results.

The rest of the paper is organized as follows. In Section 2 we describe the Davis-Putnam procedure and in Section 3 we study its performance on one distribution of formulas, the fixed clause length model. We show that with the right choice of parameter values, it produces computationally challenging SAT instances. In Section 4 we consider a second distribution, the constant density model, and argue that it is not useful in the evaluation of satisfiability testing procedures. We briefly review related work in Section 5, and summarize our results in Section 6.

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 $\{p\}$, assign p the value true, 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.

Fig. 1. The DP procedure.

2 The Davis-Putnam Procedure

One of the most widely used methods for propositional satisfiability testing is the Davis-Putnam procedure [11]. Our procedure, which we refer to as DP, is the splitting variant of the Davis-Putnam procedure as described in [12], but without the pure literal rule. (The pure literal rule is: if a literal p occurs in a formula, but its negation does not, then p can be immediately assigned the value true.) DP 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).) DP 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 variable assignment. In our implementation, variables are given an arbitrary ordering, and in the "splitting" step we choose the next variable according to this ordering, and set it first to true. The performance of simple backtracking is greatly improved by employing the unit clause rule: Whenever a clause containing a single literal arises, the variable occurring in that clause is immediately assigned the appropriate truth value. The formula is then simplified, which may lead to new unit clauses, and so on. This process of *unit propagation* can be executed in time linear in the total number of literals.

3 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 M. To keep the volume of data presented manageable and yet give a detailed

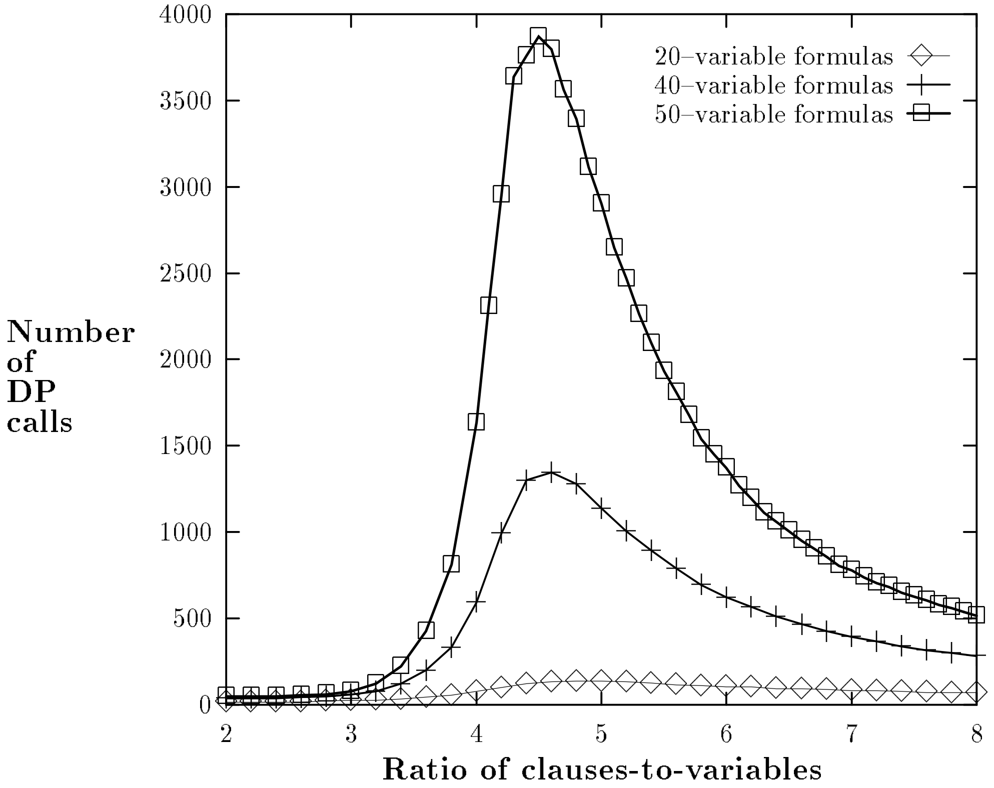

Fig. 2. Median number of recursive DP calls for Random 3-SAT formulas, as a function of the ratio of clauses–to–variables.

picture, we limit our attention to formulas with $K = 3$, that is, random 3-SAT. (Some data on other values of K can be found in [27].) For a given N and M, an instance of random 3-SAT is produced by randomly generating M clauses of length 3. Each clause is produced by randomly choosing 3 distinct variables from the set of N available, and negating each with probability 0.5. (Note that this method of generation allows duplicate clauses in a formula, so that strictly speaking such formulas are sequences of clauses, not sets. However, as N gets large duplicates will become rare because we generally select only a linear number of clauses. Most analytical work on random K-SAT uses this same model).

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 ratio of clauses to variables (ie., the number of clauses normalized through division by the number of variables). Each data point gives the median number of calls for a random sample of 10,000 formulas.

We use medians instead of means here because the means of the number of calls are heavily influenced by a very small number of extremely large values. Although they occur very rarely, these values are large enough to make the variance be as large as the mean. But our current interest is what the "bulk" of instances from the distribution are like, not the unusual or extreme cases. As the median is more robust in the presence of such "outliers" [1], it appears to be a more informative statistic for current purposes. [3]

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 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 [8,30].

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 curve for number of recursive calls and the point where the probability that a formula is satisfiable is about 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, and the location gradually decreases with N: the 50%-point occurs at 4.55

[3] A reasonable question to ask is how big a sample would be required to get a good estimate of the mean. Because of the potentially exponential nature of the problem, as we increase the sample size, we may continue to find ever larger (but ever rarer) samples that could place the mean anywhere.

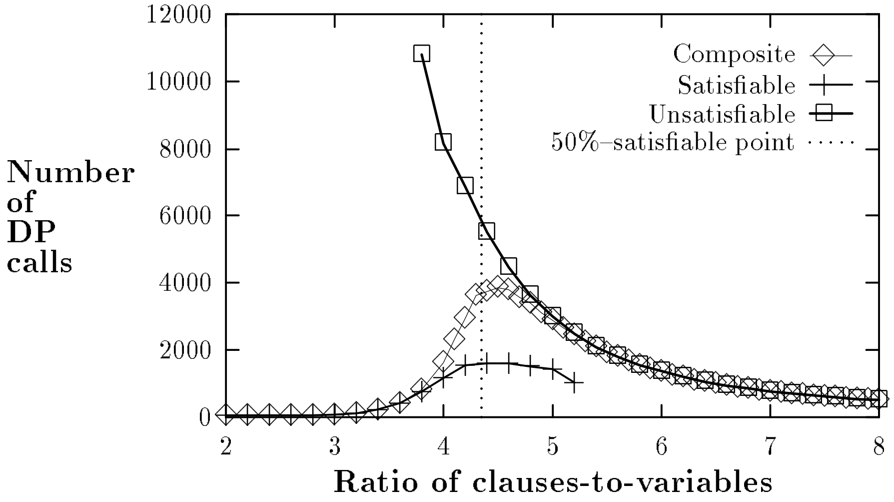

Fig. 3. Median DP calls for 50–variable Random 3-SAT as a function of the ratio of clauses–to–variables.

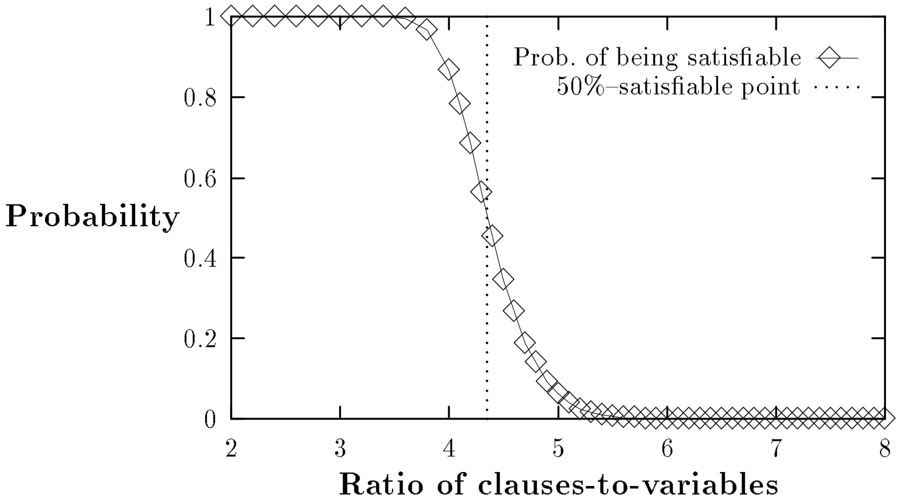

Fig. 4. Probability of satisfiability of 50–variable formulas, as a function of the ratio of clauses–to–variables.

for formulas with 20 variables; 4.36 for 50 variables; 4.31 for 100 variables and 4.3 for 150 variables (all empirically determined). We conjecture that this ratio approaches about 4.25 for very large numbers of variables. The peak hardness for DP exhibits the same behavior that we have just described for the 50-% satisfiable point. These observations about the 50%-satisfiable point are confirmed by more detailed experiments [10,25].

Since empirical tests measure performance of a particular algorithm, we expect the exact location of peak hardness to be somewhat procedure dependent. For example, since the proportion of satisfiable formulas is changing in the vicinity of the peak, a change to a SAT procedure which alters the relative performance on satisfiable and unsatisfiable instances will shift the location of peak hardness. While the performance of DP can be improved by using clever variable selection heuristics, (*e.g.*, [32,4]), it seems unlikely that such heuristics will *qualitatively* alter the 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. This conjecture is supported by other workers, for example Larrabee and Tsuji [25], who used a satisfiability procedure quite different from the Davis-Putnam procedure, but found the same hard and easy areas.

The previous paragraph will have made clear that our use of 50% and its relationship to peak difficulty is primarily heuristic: Most procedures will not peak at exactly this point, and indeed DP actually peaks at about the 43% point. Moreover, while there seems to be a fundamental relationship between the satisfiable-to-unsatisfiable transition and hardness, the data presented here does not suggest any specific relationship between peak hardness and any conjectured "critical point" [25,24], or threshold ratio.

In terms of what is known theoretically about the probability of satisfiability for random 3-SAT, the threshold conjecture is that there is some specific ratio of clauses to variables above which the probability of satisfiability approaches 1, and below which it approaches 0. For the case of 2-SAT it has been shown true, and the threshold ratio is 1 [17,6,13]. The general case for $K > 2$ is a challenging open problem, although there has been substantial recent progress in narrowing the theoretical bounds on the transition. As N gets large the probability that an instance of Random 3-SAT is satisfiable approaches 0 whenever the ratio of clauses to variables is less than 3.003 [15], and approaches 1 when this ratio is greater than 4.758 [23]. As can be seen, our empirical results are in agreement with, though much more fine-grained than, the best theoretical bounds.

To get some further insight into the search performed by DP, we now consider what happens when we let DP search the full space, *i.e.,* we do *not* stop the procedure as soon as one assignment is found. Figure 5 gives the total number of recursive calls. (Again, each data point gives the median value of a sample of 10,000 formulas.) We see that the size of the search space monotonically increases for decreasing ratios of clauses to variables. This is consistent with our expectation that the fewer clauses in the formula the longer it takes before DP will run into a contradiction, and the deeper the search tree. We also see that the search tree is somewhat larger for satisfiable formulas, at least on average. This can be intuitively explained by the fact that when an assignment is found DP tends to assign many variables in order to satisfy all clauses, so it

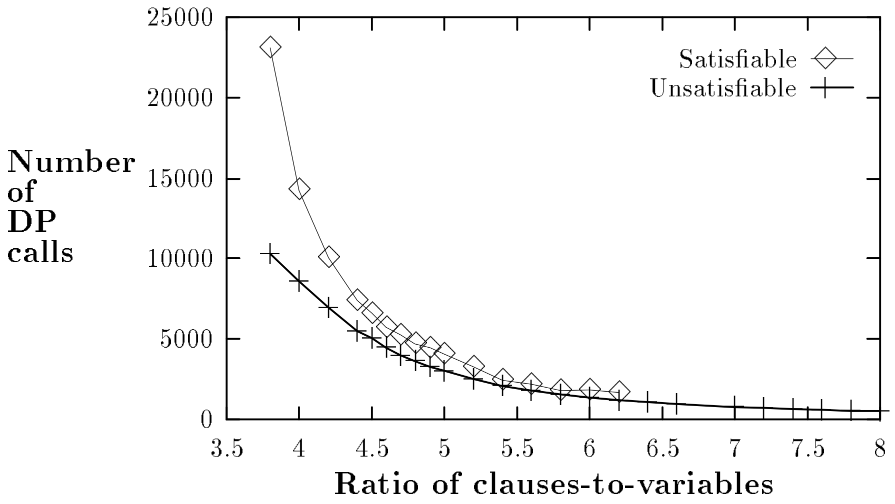

Fig. 5. Searching for all assignments. Median DP calls for 50–variable Random 3-SAT as a function of the ratio of clauses–to–variables.

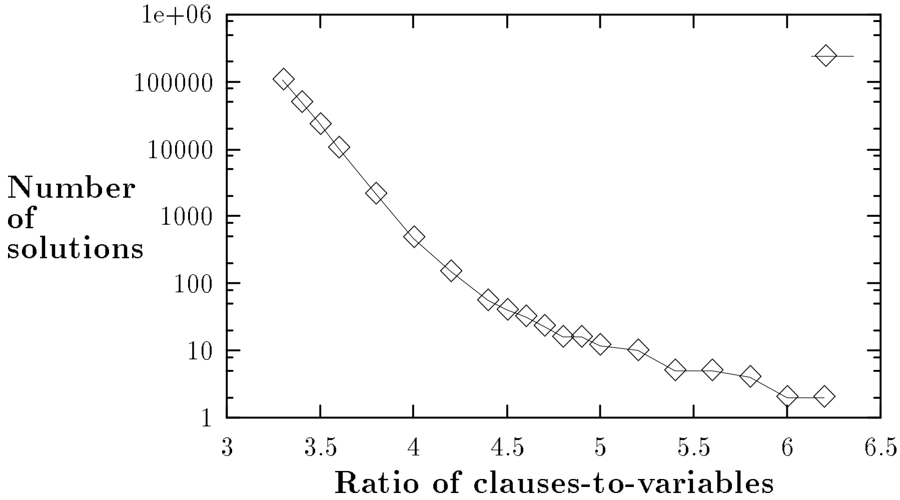

Fig. 6. Median number of satisfying assignments for satisfiable 50–variable Random 3-SAT formulas as a function of the ratio of clauses–to–variables. Note the logarithmic scale.

generates relatively long branches for satisfying assignments. The full search space for under-constrained formulas quickly becomes extremely large. For example, at a ratio of 3.3, we have over 200,000 recursive calls for satisfiable instances. Note however that such a large search space is not a problem when searching for a *single* assignment.

Part of the reason for this is that there are a tremendous number of satisfying assignments for the under-constrained formulas. This is shown in Figure 6. In

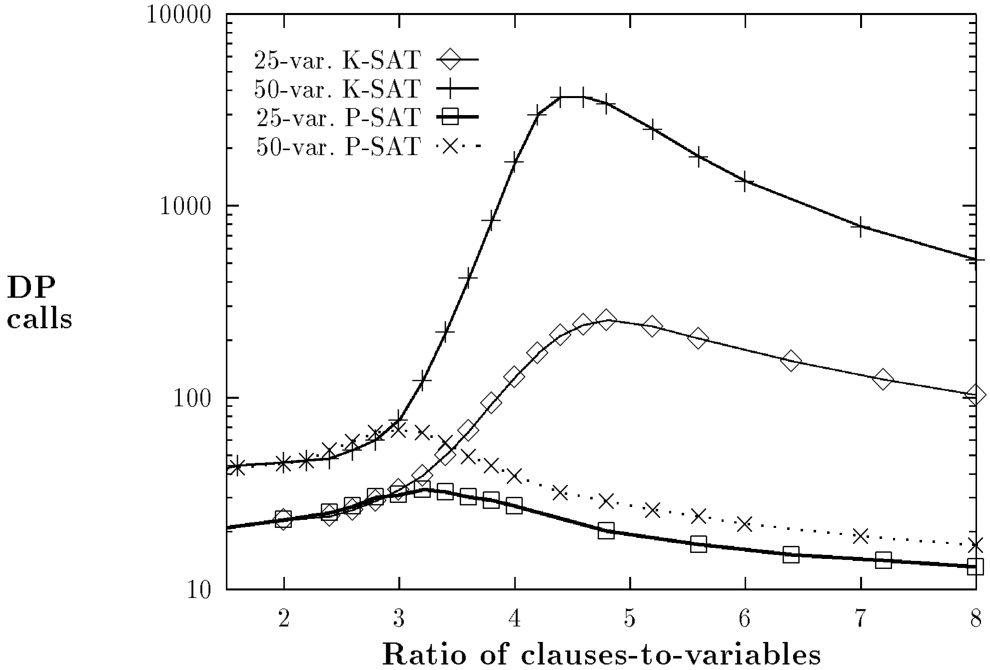

Fig. 7. Comparison of median DP calls for fixed–length (3-SAT) and constant–density formulas (average clause length 3), with 25 and 50 variables.

this figure, we give the median number of satisfying assignments for satisfiable random 3-SAT formulas as a function of the ratio of clauses to variables. (Each data point is based on 10,000 instances.) For example, at a ratio of 3.3, the formulas have around 110,000 satisfying assignments. When searching for a single satisfying assignments, DP may be able to find one early on in the search.

4 The constant-density model

We now examine formulas generated using the constant-density model. The model has three parameters: the number of variables N, and number of clauses M as before; but instead of a fixed clause length, clauses are generated by including a variable in a clause with some given 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 [14] suggest that one probably cannot generate computationally challenging instances from this model, and our ex-

vars	formula distribution		
	K-SAT (3)	P-SAT (3)	P-SAT (4)
25	253	33	53
50	3,683	68	157
75	46,915	111	392

Table 1
Number of DP calls at hardest point, for fixed–length (K-SAT) and constant–density (P-SAT) formulas, with 25, 50, and 75 variables. The numbers in brackets indicate the number of literals per clause (for the P-SAT model, this is the average number of literals per clause).

periments confirm this prediction. In Figure 7, we compare the number of recursive DP calls to solve instances of random P-SAT and random 3-SAT with the same number of variables. In this case, we have set the probability P to give an average clause length of 3. Although we see a slight easy-hard-easy pattern (previously noted by Hooker [21]), the hard area is not nearly as pronounced as that for random 3-SAT formulas of similar size, and in absolute terms the random P-SAT formulas are much easier. Note that we are using a logarithmic scale for the number of DP calls, so the greater height of the 3-SAT curve represents much greater difficulty.

Further, the rate of growth in difficulty of the random 3-SAT formulas is much higher. Table 1 gives the median number of DP calls at the hardest point on the curve for random 3-SAT and random P-SAT formulas, as a function of the number of variables (For comparison, we also include here the numbers for random P-SAT with expected clause length of 4). Again, the table clearly shows a large difference in growth rate for the two random formula models.

It might be argued that larger P-SAT formulas could be sufficiently challenging. Some data on larger formulas of this type can be found in [27,22]. While indeed there are difficult formulas, they are easy in comparison to comparably sized random K-SAT formulas. Moreover, a host of analytical results (see below) strongly suggest that as N gets large, almost all instances of this family of distributions will be easy to solve. Our experimental results and the analytical results strongly suggest that the constant-density model is not suitable for evaluating satisfiability testing procedures.

5 Related work

There is a large body of literature on testing the satisfiability of random formulas, which until recently consisted mostly of analytic results. The main impetus for this research was early results by Goldberg [18], which suggested that SAT might in fact be efficiently solvable, on average, using DP. Franco and Paull [14] showed that Goldberg's positive results were a direct consequence of the distribution used — a variant of the constant-density model — and thus overly optimistic. Goldberg's formulas were so easy to satisfy that an algorithm which simply tried randomly generated assignments would, with probability approaching 1, find a satisfying assignment in a constant number of guesses. Further analytic results for the constant-density model can be found in [5,28,31].

Franco and Paull [14] also investigated the performance of DP on random formulas with a fixed clause length of 3, 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, its expected time will be exponential in the number of variables, with probability approaching 1 as the number of variables approaches infinity while keeping the ratio of clauses to variables fixed. Unfortunately, this result does not directly tell us much about the expected time to find a single assignment.

A result of Chvatal and Szemeredi [7] gives further insight. Extending a ground-breaking result by Haken [19], they showed that any resolution strategy requires exponential time with probability approaching 1 on unsatisfiable random 3-SAT formulas, when the ratio of clauses to variables is held constant. Random 3-SAT formulas are unsatisfiable with probability approaching 1 whenever the ratio of clauses to variables is greater than 4.758 [23]. So, given that DP corresponds to a particular resolution strategy as mentioned above, it follows that the average time complexity on such formulas is exponential.

This result may appear inconsistent with our claim that over-constrained formulas are easy, but it is not. Our results show that there is an easy-hard-easy pattern for formulas with a given (fixed) number of variables when varying the number of clauses and thus the ratio of clauses to variables. Chvatal and Szemeredi's result applies to the case where we increase *both* the number of variables and clauses, while keeping the ratio of clauses to variables fixed.[4] They show that growth of the DP tree is an exponential function in the number of variables n. Our results suggest that the exponential for ratios in the hard area grows much faster than in the easy (over-constrained) area. This has been confirmed by recent results for a highly-optimized variant of DP developed by Crawford and Auton [10]. Experiments show that in the hard area, at a ratio of 4.3, their procedure scales with $2^{(n/17)}$, whereas at a ratio of 10, the scaling is with $2^{(n/57)}$. This difference in growth rate has important practical

[4] To state this differently, consider Figure 2. Chvatal and Szemeredi analyze the scaling behavior going in the vertical direction at a given fixed ratio of clauses to variables.

consequences. For example, our 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 300-variable 1290-clause formulas that it cannot practically solve. So, what we have called the "easy area," is definitely easy compared to the formulas in the hard area (around the 50% point), and for lower values of n is often much too easy to be useful as test formulas. For formulas with larger numbers of variables, eventually the truly easy area will occur only at ever higher ratios of clauses to variables.

Turning to under-constrained formulas, the behavior of DP can be at least partially explained by the fact that they tend to have very many satisfying truth assignments — see Figure 6 — so that the procedure almost always finds one early in the search. For ratios below 1.63, a simple heuristic algorithm finds satisfying assignments to random 3-SAT with high probability, in polynomial time [3]. Other analytic results for random 3-SAT are reviewed in [23,5].

As we mentioned in Section 3, establishing theoretically the exact nature the satisfiability transition for random K-SAT, when $K > 2$, is a challenging open problem. The experimental data suggests that there may be a threshold for satisfiability, at about 4.25 clauses per variable in the case of 3-SAT. Threshold phenomena are common in some other types of combinatoric structures, such as random graphs [2].

Not only are our experimental results consistent with the analytic results, they also provide a much more fine-grained picture of how 3-SAT behaves 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 they 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.

A valuable contribution was made by Cheeseman et al. [8], who explored the hardness of random instances of various NP-complete problems. They observed a similar easy-hard-easy pattern as a function of one or more problem parameters for instances of graph coloring and Hamiltonian circuit. They also give some preliminary results for satisfiability. Unfortunately, they do not describe exactly how the formulas are generated, or fully specify their test procedure, and their findings are based on relatively small formulas (up to 25 variables). 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, so their instance distribution is almost certainly somewhat different than random 3-SAT.

6 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-density 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, our DP can solve 1000 variable 3000 clause (under-constrained) and 1000 variable 50000 clause (over-constrained) random 3-SAT instances in seconds. On the other hand, it cannot consistently solve random 3-SAT instances with 300 variables and 1290 clauses.

It is worth mentioning two tempting conclusions that we feel are *unwarranted*. First, it does not follow from our results that any particular formula in the hard region will be difficult. Our results apply only to *sets* of randomly generated formulas. Second, our data says nothing about encodings of real problem instances. Our results apply only to instances of random 3-SAT, not to formulas generated from encodings or by any other means.

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.

Acknowledgement

We thank Jim Crawford and Larry Auton for providing us with their very efficient tableau code, which we modified to obtain a fast implementation of DP. 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

[1] Beckman, R.J. and Cook, R.D., Outlier.........s, *Technometrics*, 25(2):119–149, (1983).

[2] Bollobas, B., *Random Graphs.* Academic Press, London (1985).

[3] Broder, A., Frieze, A., and Upfal, E., On the Satisfiability and Maximum Satisfiability of Random 3-CNF Formulas. *Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (1993) 322–330.

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

[5] Chao, M. and Franco, J., Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k satisfiability problem. *Inform. Sci.*, **51** (1990) 23–42.

[6] Chvatal,V. and Reed, B., Mick gets some (the odds are on his side). *Proc. FOCS–92* (1992) 620–627.

[7] Chvatal,V. and Szemeredi, E., Many hard examples for resolution. *J. of the ACM*, **35**(4) (1988) 759–208.

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

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

[10] Crawford, J.M. and Auton, L.D., Experimental results on the cross-over point in satisfiability problems. *Proc. AAAI93* (1993).

[11] Davis, M. and Putnam, H., A computing procedure for quantification theory. *Journal of the ACM*, **7** (1960) 201–215.

[12] Davis, M., Logemann, G., and Loveland, D., A Machine Program for Theorem-Proving, *Communications of the ACM*, 5:394-397 (1962).

[13] de la Vega, W.F., On random 2SAT. Manuscript 1992.

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

[15] Frieze, A. and Suen, S., *Analysis of three simple heuristics on a random instance of k-SAT*, Manuscript, 1992.

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

[17] Goerdt, A., A threshold for unsatisfiability. Manuscript 1991.

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

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

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

[21] Hooker, J.N. and Fedjki, C., *Branch-and-cut solution of inference problems in propositional logic*, Annals of Math. and Artif. Intell., 1:123-139, (1990).

[22] Kamath, A.P., Karmarker, N.K., Ramakrishnan, K.G., and Resende, M.G.C., 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.

[23] Kamath, A.P., Motwani, A., Palem, R., and Spirakis, P., Tail Bounds for Occupancy and the Satisfiability Threshold Conjecture. *Proc. FOCS-94*, (1994).

[24] Kirkpatrick, Scott and Selman, Bart, Critical Behavior in the Satisfiability of Random Boolean Expressions, *Science*, 264:1297-1301, (1994).

[25] Larrabee, T. and Tsuji, Y., Evidence for a Satisfiability Threshold for Random 3CNF Formulas, *Proc. of the AAAI Spring Symposium on AI and NP-hard problems*, Palto Alto, CA (1993).

[26] Mitchell, D., Selman, B., and Levesque, H.J., Hard and easy distributions of SAT problems. *Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA (1992) 459–465.

[27] Mitchell, D. and Levesque H.J., Some pitfalls for experimenters with random SAT. This volume.

[28] Purdom, P., A Survey of Average Time Analyses of Satisfiability Algorithms. *Journal of Information Processing*, 13(4), (1990).

[29] Selman, B., Levesque, H.J., Mitchell, D., GSAT: A new method for solving hard satisfiability problems. *Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA (1992) 440–446.

[30] Williams, C.P. and Hogg, T., Using deep structure to locate hard problems. *Proc. AAAI-92,* San Jose, CA (1992) 472–277.

[31] Wu, L.C., and Tang, C.Y., Solving the satisfiability problem by using randomized approach. *Information Processing Letters*, 41:187-190, (1992).

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