Experimental Results on the Crossover Point in Satisfiability Problems

James M. Crawford
Larry D. Auton
AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974-0636
jc@research.att.com, lda@research.att.com

Abstract

Determining whether a propositional theory is satisfiable is a prototypical example of an NP-complete problem. Further, a large number of problems that occur in knowledge representation, learning, planning, and other areas of AI are essentially satisfiability problems. This paper reports on a series of experiments to determine the location of the *crossover point* — the point at which half the randomly generated propositional theories with a given number of variables and given number of clauses are satisfiable — and to assess the relationship of the crossover point to the difficulty of determining satisfiability. We have found empirically that, for 3-SAT, the number of clauses at the crossover point is a *linear* function of the number of variables. This result is of theoretical interest since it is not clear why such a linear relationship should exist, but it is also of practical interest since recent experiments [Mitchell et al. 92; Cheeseman et al. 91] indicate that the most computationally difficult problems tend to be found near the crossover point. We have also found that for random 3-SAT problems below the crossover point, the average time complexity of satisfiability problems seems empirically to grow linearly with problem size. At and above the crossover point the complexity seems to grow exponentially, but the rate of growth seems to be greatest near the crossover point.

Introduction

Many classes of problems in knowledge representation, learning, planning, and other areas of AI are known to be NP-complete in their most general form. The best known algorithms for such problems require exponential time (in the size of the problem) in the worst case. Most of these problems can be naturally viewed as *constraint-satisfaction problems*. Under this view, a problem defines a set of constraints that any solution must satisfy. Most known algorithms essentially search the space of possible solutions for one that satisfies the constraints.

Consider a randomly-generated constraint-satisfaction problem. Intuitively, if there are very few constraints, it should be easy to find a solution (since there will generally be many solutions). Similarly, if there are very many constraints, an intelligent algorithm will generally be able to quickly close off most or all of the branches in the search tree. We thus intuitively expect the hardest problems to be those that are neither over- nor under-constrained. Cheeseman et al. have shown empirically that this is indeed the case [91]. Mitchell et al. take this argument further and show empirically that the most difficult problems occur in the region where there are enough constraints so that half of the randomly generated problems have a solution [92]. We refer to this point as the *crossover point.*

The location of the crossover point is of both theoretical and practical importance. It is theoretically interesting since the number of constraints at the crossover point is an intrinsic property of the language used to express the constraints (and in particular is independent of the algorithm used to find solutions). Further, we have found that in the case of 3-SAT the number of constraints required for crossover is a linear function of the number of variables. This leads one to expect there to be some theoretical method for explaining the location of the crossover point (though no satisfactory explanation has yet been proposed).

The crossover point is of practical interest for several reasons. First, since empirically the hardest problems seem to be found near the crossover point, it makes sense to test candidate algorithms on these hard problems. Similarly, if one encounters in practice a problem that is near the crossover point, one can expect it to be difficult and avoid it or plan to devote extra computational resources to it. Furthermore, several algorithms have been proposed [Selman et al. 92; Minton et al. 90] that can often find solutions to constraint-satisfaction problems, but that cannot show a problem unsolvable (they simply give up after some set number of tries). Accurate knowledge about the location of the crossover point provides a method for partially testing such algorithms on larger problems — models should be found for around half of the randomly generated problems at the projected crossover point. Finally, as problem size increases, the transition from satisfiable to unsatisfiable becomes increasingly sharp (see Figures 3-6). This means that if one knows the location of the crossover point, then for random problem (*i.e.*, problems with no regular structure) the number of clauses can be used as a predictor of satisfiability.

In this paper we focus on the prototypical example of an NP-complete problem: 3-SAT — propositional satisfiability of clausal form theories with three variables per clause. We first survey the algorithm used to carry out our experiments (readers primarily interested in the experimental results may skip this section). We then confirm and extend the results of Mitchell et al. [92] by showing that for up to 200 variables the hardest problems do tend to be found near the crossover point. We then show empirically that at the crossover point the number of clauses is a linear function of the number of variables. Finally we show empirically that below the crossover point the complexity of determining satisfiability seems to grow linearly with the number of variables, but above the crossover point the complexity seems to grow exponentially. The exponential growth rate appears to be steepest near the crossover point.

Propositional Satisfiability

The propositional satisfiability problem is the following ([Garey & Johnson 79]):

Instance: A set of clauses[1] C on a finite set U of variables.

Question: Is there a truth assignment[2] for U that satisfies all the clauses in C?

We refer to a set of clauses as a *clausal propositional theory*. 3-SAT is propositional satisfiability for theories in which all clauses have exactly three *terms*.[3]

Tableau Based Satisfiability Checking

Our satisfiability checking program, TABLEAU, began life as an implementation of Smullyan's tableau based inference algorithm [Smullyan, 68]. TABLEAU has since evolved significantly and can most easily be presented as a variant of the Davis-Putnam procedure.

The basic Davis-Putnam procedure is the following [Davis et al. 62]:

```
Find_Model(theory)
  unit_propagate(theory);
  if contradiction discovered return(false);
  else if all variables are valued return(true);
  else {
    x = some unvalued variable;
    return(Find_Model(theory AND x) OR
           Find_Model(theory AND NOT x));
  }
```

Unit Propagation

Unit propagation consists of the repeated application of the inference rule:

$$\frac{\begin{array}{l} x \\ \neg x \vee y_1 \ldots \vee y_n \end{array}}{y_1 \vee \ldots \vee y_n}$$

(similarly for $\neg x$).

Complete unit propagation takes time linear in the size of the theory [Dowling & Gallier 84]. In this section we sketch the data structures and algorithms used for efficient unit propagation in TABLEAU.

We maintain two tables: the *variable table* and the *clause table*. In the variable table, we record the value of each variable (*true*, *false*, or *unknown*) and lists of the clauses in which the variable appears. In the clause table, we keep the text of each clause (*i.e.*, a list of the terms in the clause), and a count of the total number of unvalued variables (*i.e.*, variables whose value is *unknown*) in the clause.

Unit propagation then consists of the following:

1. Whenever a variable's value goes from *unknown* to *true*, decrement the unvalued variable count for all clauses (with non-zero counts) in which the variable appears negated, and set to zero the unvalued variable count for all clauses in which the variable appears positively (this signifies that these clauses are now redundant and can be ignored). Variables going from *unknown* to *false* are treated similarly.

2. Whenever the count for a clause reaches one, walk through the list of variables in the clause and value the one remaining unvalued variable.

The actual implementation has several additional complications for efficiency. Details are given in [Crawford & Auton 93].

Heuristics

On each recursive call to *Find_Model* one must choose a variable to branch on. We have observed that simple variable selection heuristics can make several orders of magnitude difference in the average size of the search tree.

[1] A *clause* is a disjunction of propositional variables or negated propositional variables.

[2] A *truth assignment* is a mapping from U to $\{true, false\}$.

[3] A *term* is a negated or non-negated variable.

The first observation underlying the heuristics used in Tableau is that there is no need to branch on variables which only occur in Horn clauses:[4]

Theorem 1 If a clausal propositional theory consists only of Horn clauses and unit propagation does not result in an explicit contradiction (i.e. x and $\neg x$ for some variable x in the theory) then the theory is satisfiable.

Satisfiability problems are often viewed as constraint-satisfaction problems in which the variables must be given values from $\{true, false\}$, subject to the constraints imposed by the clauses. We take a different approach — we view each non-Horn clause as a "variable" that must take a "value" from among the literals in the clause. The Horn clauses are then viewed as the constraints. It turns out that one can ignore any negated variables in the non-Horn clauses (whenever any one of the non-negated variables is set to $true$ the clause becomes redundant and whenever all but one of the non-negated variables are set to $false$ the clause becomes Horn). Thus the number of "values" a non-Horn clause can take on is effectively the number of non-negated variables in the clause.

Our first variable-selection heuristic is to concentrate on the non-Horn clauses with a minimal number of non-negated variables. Basically this is just a most-constrained-first heuristic (since the non-Horn clauses with a small number of non-negated variables are viewed as "variables" that can take on a small number of "values"). The first step of Tableau's variable-selection algorithm is thus to collect a list, V, of all variables that occur positively in a non-Horn clause with a minimal number of non-negated variables.

The remainder of the variable selection heuristics are used to impose a priority order on the variables in V. Our first preference criterion is to prefer variables that would cause a large number of unit-propagations. We have found that it is not cost-effective to actually compute the number of unit-propagations. Instead we approximate by counting the number of (non-redundant) binary clauses in which the variables appear.[5] This heuristic is similar to one used by Zabih and McAllester [88].

In cases where several variables occur in the same number of binary clauses, we count the number of unvalued *singleton neighbors*.[6] We refer to two terms as *neighbors* if they occur together in some clause.[7] A term is a *singleton* iff it only occurs in one clause in the theory (this does *not* mean that the clause is of length one — if a theory contains $x \vee y \vee z$ and the term x occurs nowhere else in the theory then x is a singleton). Singleton terms are important because if their clause becomes redundant, their variable will then occur with only one sign and so can be valued:

Theorem 2 If a variable x always occurs negated in a clausal propositional theory Th, then Th is satisfiable iff $Th \wedge \neg x$ is satisfiable.

A similar result holds for variables which only occur positively.

In some cases there is still a tie after the application of both metrics. In these cases we count the total number of occurrences of the variables in the theory.

Heuristic	Nodes	Standard Dev.
Fixed order (no heuristics)	41320	56808
most-constrained-first	14190	17290
binary clauses occurrences	231	145
most singleton neighbors	225	141
most occurrences	195	118

Figure 1: Effects of the search heuristics. The data on each line is for the heuristic listed added to all the previous heuristics.

Figure 1 shows the effects of the search heuristics on 1000 randomly generated 3-SAT problems with 100 variables and 429 clauses.[8] Notice that counting the number of occurrences in binary clauses has quite a dramatic impact on the size of the search space. We have noticed that Tableau usually branches down some number of levels (depending on the problem size) and then finds a large number of unit-propagations that lead to a contradiction relatively quickly. This heuristic seems to work by decreasing the depth in the search tree at which this cascade occurs (thus greatly decreasing the number of nodes in the tree).

Figure 2 shows a comparison of Tableau with Davis-Putnam on 50-150 variable 3-SAT problems near the crossover point.[9] The experience of the community has been that the complexity of the Davis-Putnam algorithm on 3-SAT problems near the crossover point grows at about $2^{n/5}$ (where n is the number of variables). Our experiments indicate that the complexity of Tableau grows at about $2^{n/17}$, thus allowing us to handle problems about three times as large.

[4] A clause is *Horn* iff it contains no more than one positive term.

[5] At this point we count appearances in non-Horn *and* Horn clauses since the intent is to approximate the total number of unit-propagations that would be caused by valuing the variables.

[6] This heuristic was worked out jointly with Haym Hirsh.

[7] Recall that a term is a negated or non-negated variable. Thus x and $\neg x$ are different terms but the same variable.

[8] Some of the heuristics are more expensive to compute than others, but we will ignore these differences since they tend to become less significant on larger problems, and since even our most expensive heuristics are empirically only a factor of about three to four times as expensive to compute.

[9] The Davis-Putnam data in this table is courtesy of David Mitchell.

Tableau:

Variables	Clauses	Experiments	Nodes	Time[a]
50	218	1000	26	.02
100	430	1000	204	.47
150	635	1000	1532	3.50

Davis-Putnam:

Variables	Clauses	Experiments	Nodes
50	218	1000	341
100	430	1000	42,407
150	635	164	3,252,280

Figure 2: A Comparison of Davis-Putnam to TABLEAU on 3-SAT problems.

[a] Run times are in seconds and are for the C version of TABLEAU running on a MIPS RC6280, R6000 uni-processor with 128MB ram.

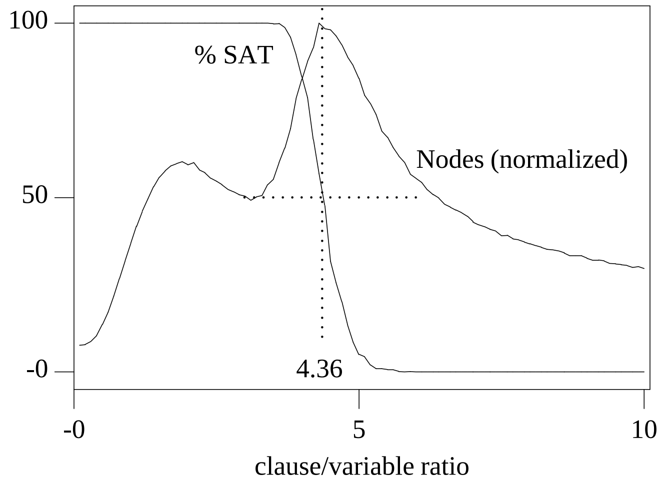

Figure 3: Percent satisfiable and size of the search tree — 50 variables.

Probabilistic Analysis of Subsumption

The original Davis-Putnam procedure included a test for subsumed clauses (*e.g.*, if the theory includes $x \vee y$ and $x \vee y \vee z$ then the larger clause was deleted). We have found that this test does not seem to be useful near the crossover point.[10]

Consider a clause $x \vee y \vee z$. If we derive $\neg x$, we then unit propagate and conclude $y \vee z$. What is the chance that this new clause subsumes some other clause in the theory ? A simple probabilistic analysis shows that the expected number of subsumed clauses is $3c/2(v-1)$ (where v is the number of variables, and c is the ratio of clauses to variables — empirically this is about 4.24 at the crossover point). The expected number of subsumptions is thus relatively small (.06 for $v = 100$ near the crossover point) and falls as the size of the problem increases.

It seems likely that a similar, but more complex, analysis would show that the expected benefit from enforcing arc and path consistency on 3-SAT problems near the crossover point also decreases with increasing problem size.

Experiment 1: The Relationship Between Crossover and Problem Difficulty

This experiment is intended to give a global view of the behavior of 3-SAT problems as the clause/variable ratio is changed.

Experimental Method

We varied the number of variables from 50 to 200, incrementing by 50. In each case we varied the clause/variable ratio from zero to ten, incrementing the number of clauses by 5 for 50 and 100 variables, by 15 for 150 variables, and by 20 for 200 variables. At each setting we ran TABLEAU on 1000 randomly generated 3-SAT problems.[11]

Results

The graphs for 50, 100, 150, and 200 variables are shown in Figures 3-6. In each case we show both the average number of nodes in the search tree and the percentage of the theories that were found to be satisfiable.

Discussion

Our data corroborates the results of Mitchell et al. [92] — the most difficult problems tend to occur near the crossover point, and the steepness of both curves increases as the size of the problem is increased. Our data shows a small secondary peak in problem difficulty at about two clauses per variable. This peak does not seem to occur with the Davis-Putnam algorithm and is probably an artifact of the heuristics used to choose branch variables in TABLEAU.

For 3-SAT theories with only three variables, one can analytically derive the expected percent satisfiable for a given clauses/variable ratio. The clauses in a theory with three variables can differ only in the placements of negations (since we require each clause to contain three distinct variables). A theory will thus be unsatisfiable iff it contains each of the eight possible clauses (*i.e.*, every possible combination of negations). If the theory contains n randomly chosen clauses, then the chance that it will be unsatisfiable is equivalent to the chance that picking n times from a bag of eight objects (with replacement) results in getting at least one of each objects. The expected percent satisfiable as a function of the clauses/variables ratio is shown if Fig-

[10] We do, of course, do "unit subsumption" — if we generate x, we remove all clauses containing x.

[11] In all our experiments we generated random theories using the method of Mitchell et al. [92] — we made sure that each clause contained three unique variables but did not check whether clauses were repeated in a theory.

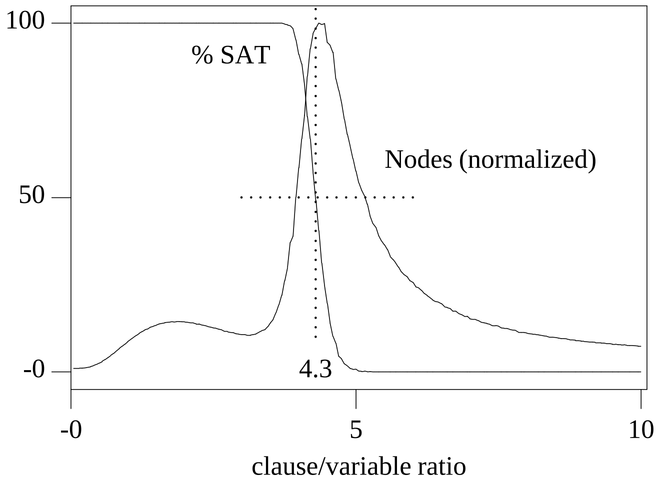

Figure 4: Percent satisfiable and size of the search tree — 100 variables.

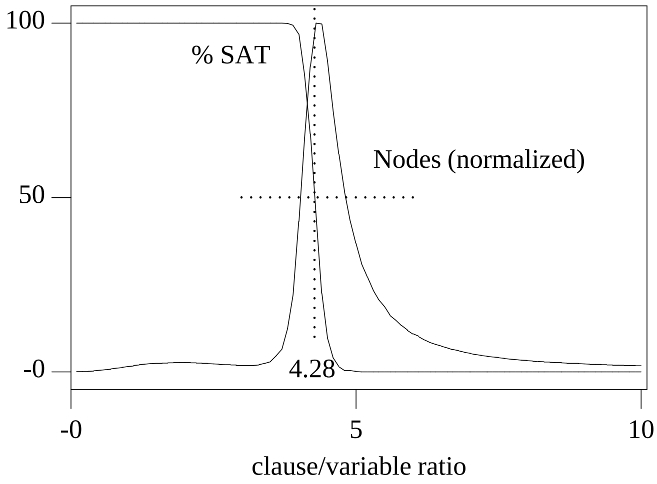

Figure 5: Percent satisfiable and size of the search tree — 150 variables.

ure 7. Notice that the shape of this curve is similar to the experimentally derived curves in Figures 3-6.

Experiment 2: The Location of the Crossover Point

The aim of this experiment is to characterize as precisely as possible the exact location of the crossover point and to determine how it varies with the size of the problem.

Experimental Method

We varied the number of variables from 20 to 260, incrementing by 20. In each case we collected data near where we expected to find the crossover point. We then focused on the data from five clauses below the crossover point to five clauses above the crossover point.[12] For each data point we ran Tableau on 10^4 randomly generated 3-SAT problems (10^3 for 220 variables and above).

Results

The results for 20, 100, 180, and 260 variables are shown in Figure 8. Each set of points shows the percentage of theories that are satisfiable as a function of the clause/variable ratio. Notice that the relationship between the percent satisfiable and the clause/variable ratio is basically linear (this is only true, of course, for points very close to the crossover point).

We took the 50 percent point on each of these lines as our experimental value for the crossover point. The resulting points are shown in Figure 9.

Discussion

From the analytical analysis for 3 variables, one can show that the crossover point for 3 variables is at 19.65 clauses. If we add this data to the least-squares fit in Figure 9 we get:

$$clauses = 4.24 vars + 6.21$$

This equation is our best current estimate of the location of the crossover point.

Experiment 3: The Run Time of Tableau

The goal of this experiment is to characterize the complexity of Tableau below, at, and above the crossover point.

Experimental Method

In these experiments we fixed the ratio of clauses to variables (at 1,2,3 and 10) and varied the number of variables from 100 to 1000 incrementing by 100. At each point we ran 1000 experiments and counted the number of nodes in the search tree.[13] We also used the results from Experiment 2 to calculate the size of the search tree near the crossover point (thus each data point represents an average over 10^5 runs for 20 to 200 variables, and over 10^4 runs for 220 to 260 variables).

Results

The graphs for clause/variable ratios of 1,2, and 3 are shown in Figure 10. The results near the crossover point, and at 10 clauses/variable are shown in Figure 11.

Discussion

[12] To determine whether a point is above or below the crossover point we rounded to one place beyond the decimal point — points at 50.0 were taken to be *at* the crossover point. In most cases we found five points clearly above and five points clearly below the crossover point. The exceptions were: 60 variables — 5 above, 1 at, 4 below, 100 variables — 5 above, 1 at, 4 below, 140 variables — 5 above, 1 at, 4 below, 240 variables — 4 above, 1 at, 5 below, and 260 variables — 4 above, 1 at, and 5 below.

[13] At 10 clauses/variable we currently only have data up to 600 variables.

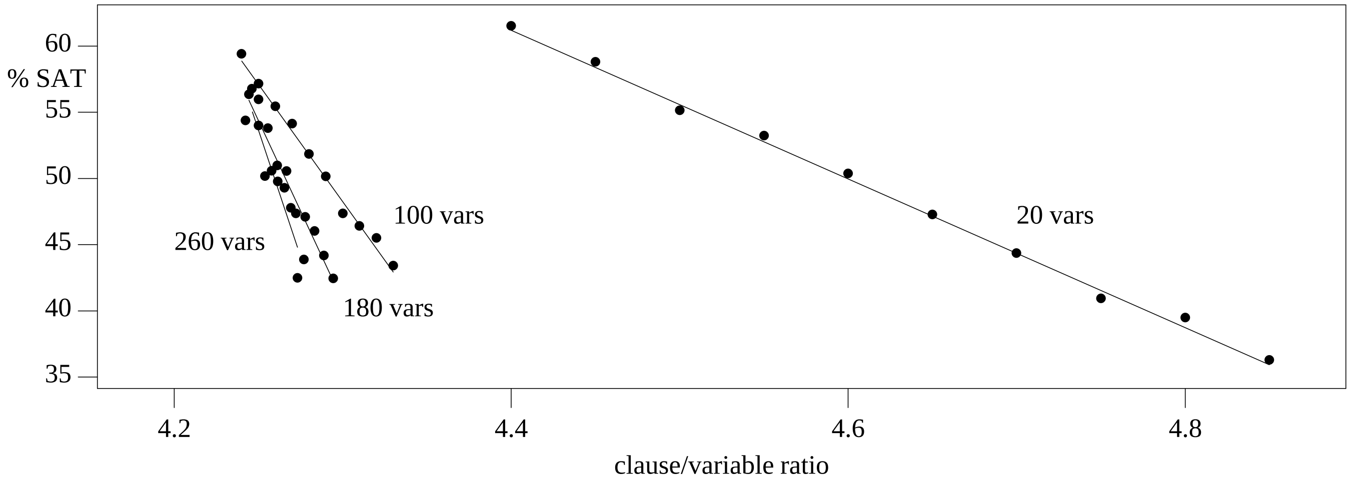

Figure 8: Percent satisfiable as a function of the number of clauses.

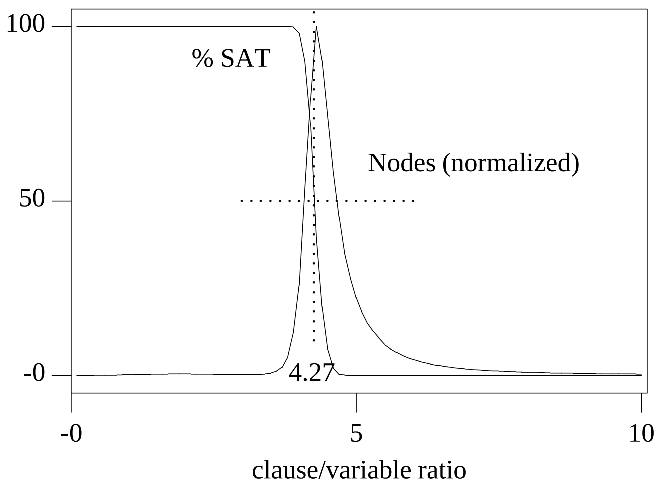

Figure 6: Percent satisfiable and size of the search tree — 200 variables.

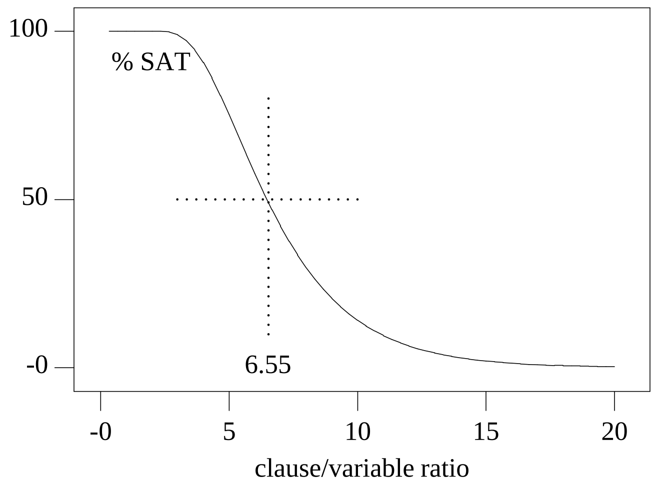

Figure 7: Analytical results for the three variable case.

Below the crossover point, the size of the search tree seems to grow roughly linearly with the number of variables. This is consistent with the results of Broder et al. [93]. However, even in this range there are clearly some problems as hard as those at the crossover point (we observed such problems while trying to gather data beyond 1000 variables). Near the crossover point, the complexity of TABLEAU seems to be exponential in the number of variables. The growth rate from 20 to 260 variables is approximately $2^{n/17}$. Above the crossover point, the complexity appears to grow exponentially (this is consistent with the results of Chvátal and Szemerédi [88]), but the exponent is lower than at the crossover point. The growth rate from 100 to 600 variables at 10 clauses/variable is approximately $2^{n/57}$.

Conclusion

Our experimental results show that the hardest satisfiability problems seem to be those that are *critically constrained* — *i.e.*, those that are neither so under-constrained that they have many solutions nor so over-constrained that the search tree is small. This confirms past results [Cheeseman et al. 91; Mitchell et al. 92]. For randomly-generated problems, these critically-constrained problems seem to be found in a narrow band near the crossover point. Empirically, the number of clauses required for crossover seems to be a *linear* function of the number of variables. Our best current estimate of this function is $clauses = 4.24vars + 6.21$. Finally, the complexity of our satisfiability algorithm seems to be on average

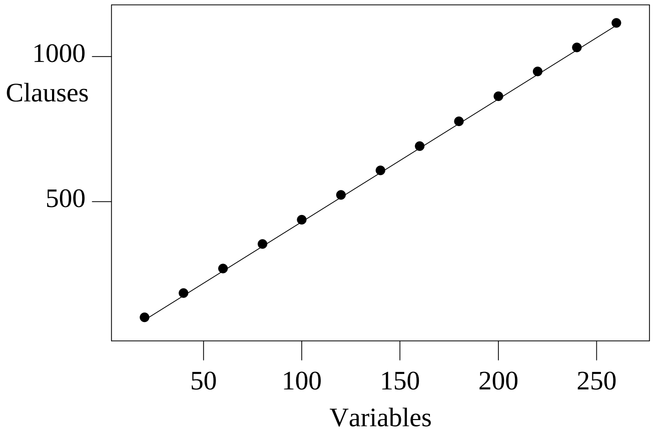

Figure 9: The number of clauses required for crossover.

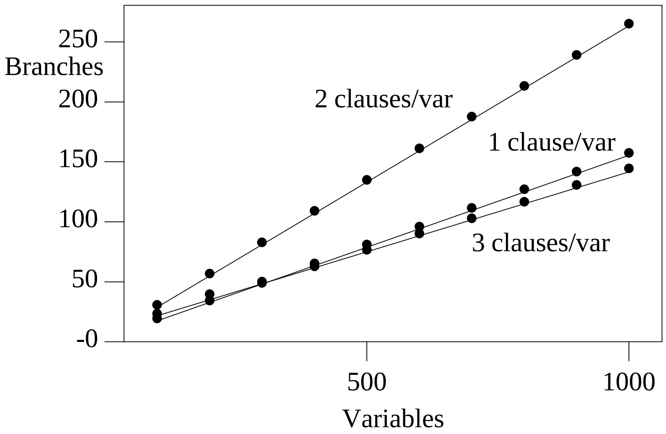

Figure 10: The size of the search tree below the crossover point.

roughly linear on randomly generated problems below the crossover point (though even here there are some problems that are as hard as those at the crossover point). At the crossover point the complexity seems to be about $2^{n/17}$ (where n is the number of variables). Above the crossover point we conjecture that the growth rate will be of form $2^{n/k}$ where k is less than 17 and increases as one gets further from the crossover point. At 10 clauses/variable k seems to be approximately 57.

Acknowledgments

Many people participated in discussions of this material, but we would like to particularly thank Haym Hirsh, Bart Selman, Henry Kautz, David Mitchell, David Etherington, and the participants at the 1993 AAAI Spring Symposium on "AI and NP Hard Problems".

References

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

Cheeseman, P., Kanefsky, B., and Taylor, W.M. (1991). Where the really hard problems are. *IJCAI-91*, pp. 163-169.

Chvátal, V. and Szemerédi, E. (1988). Many Hard Examples for Resolution. *JACM* 35:4, pp. 759-768.

Crawford, J.M. and Auton L.D. (1993). Pushing the edge of the satisfiability envelope. In preparation.

Davis, M., Logemann, G., and Loveland, D. (1962). "A machine program for theorem proving", *CACM*, 5, 1962, 394-397.

Dowling, W.F. and Gallier, J.H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, **3**, 267–284.

Garey, M.R. and Johnson D.S. (1979). *Computers and Intractability.* W.H. Freeman and Co., New York.

Minton, S., Johnson, M.D., Philips, A.B. and Laird, P. (1990). Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. *AAAI-90*, pp. 17-24.

Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and easy distributions of SAT problems. *AAAI-92*, pp. 459-465.

Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. *AAAI-92*, pp. 440-446.

Smullyan, R. M. (1968) *First Order Logic.* Springer-Verlag New York Inc.

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

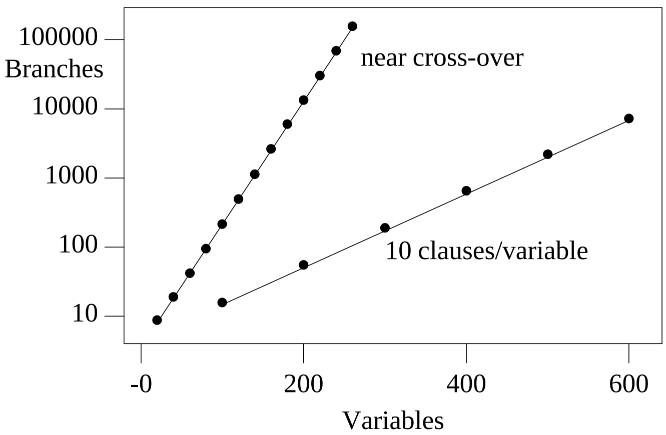

Figure 11: The size of the search tree at and above the crossover point (log scale).