# Experimental results on the crossover point in random 3-SAT 

James M. Crawford ${ }^{\text {a,*,1 }}$, Larry D. Auton ${ }^{\text {b,2 }}$<br>${ }^{a}$ Computational Intelligence Research Laboratory, 1269 University of Oregon, Eugene, OR 97403-I269, USA<br>${ }^{\mathrm{b}}$ AT\&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974-0636, USA

Received May 1994; revised April 1995


#### Abstract

Determining whether a propositional theory is satisfiable is a prototypical example of an NPcomplete 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 the most extensive set of experiments to date on the location and nature of the crossover point in satisfiability problems. These experiments generally confirm previous results with two notable exceptions. First, we have found that neither of the functions previously proposed accurately models the location of the crossover point. Second, we have found no evidence of any hard problems in the under-constrained region. In fact the hardest problems found in the under-constrained region were many times easier than the easiest unsatisfiable problems found in the neighborhood of the crossover point. We offer explanations for these apparent contradictions of previous results.


Keywords: Search phase transition; Satisfiability; Crossover point in random 3-SAT; Experimental analysis of 3-SAT

## 1. Introduction

A large number of problems that occur in knowledge-representation, learning, planning, and other areas of AI are known to be NP-complete in their most general form.

[^0]Further, many commercially important problems in scheduling, configuration, and planning also appear to be instances of NP-complete problems. The best-known algorithms for solving such problems are known to require exponential run time (in the size of the problem) in the worst case.

However, a worst-case result tells us relatively little about the nature of a problem in practice. It might turn out that almost every practical problem requires exponential run time, or that virtually none do. Similarly, the exponential factor might be so large that a three-variable problem is unsolvable, or so small that the problems do not become intractable in practice until the problem size is larger than we can even write down. Alternatively, there might be a problem parameter such that the hardest problems tend to be those for which the parameter is in a particular range.

Recent experimental evidence indicates that satisfiability problems fall into this last class. Problems with a relatively small number of constraints appear to be easy because they generally have many solutions. Problems with a very large number of constraints appear to be easy because an intelligent algorithm will generally be able to quickly close off most or all of the branches in the search tree. However, the problems in betweenthose with few solutions but lots of partial solutions-seem to be quite hard. Interestingly, for randomly generated 3-SAT problems, these hard problems seem to occur very near the point at which half of the randomly generated problems are satisfiable [16]. We refer to this point as the crossover point. Fig. 1 shows the crossover effect graphically. One line shows the percent satisfiable, and the other shows problem difficulty. Notice that problem difficulty peaks in the region where percent satisfiable suddenly falls from almost one-hundred percent to almost zero.

The crossover point divides the space of satisfiability problems into three regions: the under-constrained region below the crossover point, the critically-constrained region in the neighborhood of the crossover point, and the over-constrained region above the crossover point. Each of these regions is interesting-though for different reasons. Generally the commercially important satisfiability and constraint-satisfaction problems are optimization problems: one wants to minimize costs subject to a given set of constraints. If the cost threshold is set too high then an under-constrained problem results. If the cost threshold is set just right then a critically-constrained problem results. Similarly for over-constrained problems. If we have an optimization problem to solve, and we do not have sufficiently powerful algorithms to solve it in the critically-constrained region (which is usually the case for realistically-complex problems), then our only choice is to loosen the cost threshold and move the problem into the under-constrained region. Thus the under-constrained region is important because in practice this is where optimization problems are usually "solved". Clearly the critically-constrained region is important because this is where we must work if we are to solve optimization problems exactly. Finally, the over-constrained region is important because showing over-constrained problems unsolvable corresponds to showing optimality of solutions to optimization problems (showing unsatisfiability is also the essential task of a theorem prover though there is no a priori reason to expect that theorem proving problems fall into any particular regions).

In this paper we investigate the location of the crossover point and the behavior of a modern systematic satisfiability algorithm in each of the three regions. All of our


Fig. 1. Percent satisfiable and problem difficulty for 200 -variable random 3-SAT as a function of the clause/variable ratio. Problem difficulty is normalized so that the hardest problem is given a difficulty of 100.
experiments are done on randomly generated 3-SAT problems. This choice perhaps deserves some explanation. The first question to be asked is why work with random problems. The immediate answer is that random problems are readily available in any given size and virtually inexhaustible numbers. For example, the experiments reported here required several million problems and it is hard to imagine collecting that many problems any other way. But beyond this, there is an argument that randomly generated problems represent a "core" of hard satisfiability problems. Certainly real problems have structure, regularity, symmetries, etc., and algorithms will have to make use of this structure to simplify the problems. However, once all the structure is "squeezed out", the remainder will be a problem that requires search, and if all the structure is uscd up then the remainder will presumably be a random problem. Clearly it is unlikely that techniques will be developed to squeeze out all structure, but the fact remains that random problems seem to get at some basic domain-independent aspect of the hardness of NP-complete problems.

Even given that we are interested in randomly-generated problems, there are still various choices to be made. First, there are many possible distributions: e.g., the "constant probability model", "random $k$-SAT", etc. We focus on random $k$-SAT because of its simplicity and because past experimental results have indicated that the $k$-SAT model generates problems whose difficulty in the critically-constrained region grows exponen-
tially for all known algorithms. For other distributions, such as that given by the constant probability model, problem difficulty seems to grow much more slowly [15]. Finally, we focus on $k=3$ primarily to limit the scope of the paper to a manageable size. $k=3$ is in a sense the simplest interesting case since (1) if all clauses are of length two then polynomial algorithms are known, and (2) a theory with clauses longer than three can be converted to an equivalent theory with clauses of length three with only a linear increase in the length of the theory.

The rest of this paper is organized as follows. First we present the experimental results broken down into first the results on the location of the crossover point, and then results on the difficulty of problems below, at, and above the crossover point. We then give a detailed description of the satisfiability algorithm used to generate these results.

## 2. Experimental results

In this section we present a series of experimental results on the location of the crossover point and the difficulty of solving satisfiability problems in the under-constrained, critically-constrained, and over-constrained regions. We begin with formal definitions of satisfiability and random 3-SAT.

The propositional satisfiability problem is the following [9]:

- Instance: A set of clauses ${ }^{3} C$ on a finite set $U$ of variables.
- Question: Is there a truth assignment ${ }^{4}$ for $U$ that satisfies all the clauses in $C$ ? Clearly one can determine whether such an assignment exists by trying all possible assignments. Unfortunately, if the set $U$ is of size $n$ then there are $2^{n}$ such assignments. All known approaches to determining propositional satisfiability are computationally equivalent (asymptotically in the worst case) to such a complete search-they differ only in that they may take time $2^{n / k}$ for some constant $k$ (and in their expected-case time complexity on different classes of problems).

In all our experiments we generate random 3-SAT theories using the method of Mitchell et al. [16]-we generate each clause by picking three different variables at random and negating each with probability 0.5 . We do not check whether clauses are repeated in a theory.

### 2.1. The location of the cross-over point

The location of the crossover point is of both theoretical and practical importance. It is theoretically interesting since the number of constraints required to achieve crossover 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, in the case of 3-SAT the number of constraints required for crossover appears to be almost (but not exactly)

[^1]a linear function of the number of variables in the problem. This leads one to expect there to be some theoretical method for explaining the location of the crossover point (though no satisfactory method 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 thus avoid it (or plan to devote extra computational resources to it). Further, several algorithms have been proposed [14,17] that can often find solutions to constraint-satisfaction problems, but which cannot show a problem unsolvable (they simply give up after a given number of tries). Accurate knowledge about the location of the crossover point would provide a method for testing such algorithms on larger problems than those on which complete methods (i.e., methods which always show problems solvable or unsolvable) can work. Finally, as problem size increases the transition from satisfiable to unsatisfiable becomes increasingly sharp. This means that if one knows the location of the crossover point, then for random problems (i.e., problems with no structure) the number of clauses can be used as a predictor of satisfiability.

We should point out that it is not reasonable to expect to take satisfiability problems drawn from other sources (e.g., satisfiability encodings of scheduling problems) and expect to derive any meaningful information by comparing the clause/variable ratio to the results given in this paper. The crossover point is algorithm-independent but it is heavily distribution-dependent-problems drawn from other distributions are likely to have a crossover point, but the clause/variable ratio at that point is likely to bear little or no relationship to the clause/variable ratio at the crossover point for random 3-SAT.

In the experiments presented in this section we first look generally at how the percent satisfiable changes as a function of the clause/variable ratio. As we shall see, near the crossover point the percent satisfiable curve is nearly linear. The slope of this line is fairly gentle for small numbers of variables (c.g., 20) but gets progressively steeper as the number of variables grows (see Fig. 2). Some past work [13] has suggested that this percent satisfiable curve "rotates" around the point at which the clause/variable ratio is 4.2. In other words, if the clause/variable ratio is fixed at 4.2 and the number of variables is increased, then the percent satisfiable will remain approximately constant. If this were true it would suggest that in the limit the fifty-percent point would approach 4.2. We show, however, that the percent satisfiable at 4.2 clauses/variable is not fixed but actually appears roughly parabolic. As yet we know of no explanation for this phenomenon. We then focus on deriving the best estimate we can for the location of the fifty-percent point. We present data from 20 to 300 variables. It turns out that neither the simple linear function presented in our past work [4], nor the finite-scaling model presented by Selman and Kirkpatrick [12], fit particularly well. We show that a variant of the Kirkpatrick and Selman equation,

$$
c / v=4.258+58.26 v^{-2 / 3}
$$

fits the data fairly well.


Fig. 2. Percent satisfiable as a function of the number of variables and the clause/variable ratio.


Fig. 3. Percent satisfiable for each number of variables as a function of the clause/variable ratio.

### 2.1.1. Experiment 1: the shape of the crossover region

The aim of this experiment is to provide a view of the three-dimensional surface defined by the percent satisfiable as a function of the number of variables and the clause/variable ratio.

## Experimental method

We varied the number of variables from 20 to 260 incrementing by 20 . We also varied the clause/variable ratio from 3.5 to 5.5 incrementing by 0.1 . At each point we ran $10^{3}$ experiments and recorded percent satisfiable and difficulty (measured by the number of leaves in the search tree).


Fig. 4. Percent satisfiable for each clause/variable ratio as a function of the number of variables.

## Results

The results are shown graphically in Fig. 2. Since it's hard to get a feel for threedimensional curves in two dimensions, we also show two projections in Figs. 3 and 4. The first projection shows percent satisfiable as a function of the clause/variable ratio. Each line in this figure represents a different number of variables. In the second projection we show the percent satisfiable as a function of the number of variables. Each line now corresponds to a different clause/variable ratio.

## Discussion

In Figs. 2 and 3 one can see how the slope of the percent satisfiable curve becomes steeper as the number of variables is increased. Notice also that the "beginning" of the crossover region stays at around 4 clauses per variable (moving only slightly toward higher clause/variable ratios as the number of variables is increased). The other end of the crossover region moves more dramatically toward lower clause/variable ratios as the number of variables grows. This effect can also be seen in Fig. 4. The lower lines curve upwards as the number of variables is decreased. These lines represent high clause/variable ratios at which the percent satisfiable increases dramatically for small numbers of variables. The upper lines curve down-at these small clause/variable ratios the percent satisfiable decreases for small numbers of variables. The nearly stationary line is 4.2 clauses/variable. We examine the line more closely in the next experiment.


Fig. 5. Percent satisfiable as a function of the number of variables at 4.2 clauses/variable.
2.1.2. Experiment 2: the behavior of percent satisfiable at 4.2 clauses/variable

The goal of this experiment is to determine the behavior of the percent satisfiable curve when the clause/variable ratio is held fixed at 4.2 clauses/variable.

## Experimental method

We varied the number of variables from 20 to 260 , and fixed the number of clauses at 4.2 times the number of variables. At each point we ran $10^{4}$ experiments (above 200 variables we ran $10^{3}$ experiments at each point).

## Results

The results are shown in Fig. 5.

## Discussion

Past work [13] has suggested that the percent satisfiable curve "rotates" around the point at which the clause/variable ratio is 4.2 . In other words, for any number of variables $v$, if the number of clauses is $4.2 v$ then the percent satisfiable will be approximately constant. Larrabee and Tsuji's [13] experiments only covered the region from 50 to 170 variables and involved 500 experiments at each point. They observed that the percent satisfiable stayed within three percent of 68 percent.


Fig. 6. Percent satisfiable for unique clause model as a function of the number of variables at 4.2 clauses/variable.

The data in Fig. 5 shows that for less than 50 or more than 200 variables, the percent satisfiable is not constant.

In the random 3-SAT model used for these experiments, we make sure that each clause contains three unique variables, but do not check whether clauses are repcated in a theory. One might argue that the percent satisfiable increases for 20 variables because many duplicate clauses are being generated. To test this hypothesis we re-ran this experiment making sure to never generate duplicate clauses. The results are show in Fig. 6. The percent satisfiable for 20 and 40 variables is slightly lower than in Fig. 5 , but the curve still has the same basic shape.

To understand this shape notice first that for three-variable problems one can show analytically that the percent satisfiable at 4.2 clauses/variable is almost 100 percent. This is because the fifty-percent point is at around 19 clauses. As the number of variables is increased, this fifty-percent point moves toward smaller clause/variable ratios. At 20 variables the fifty-percent point occurs at approximately 91 clauses or 4.55 clauses/variable. At 200 variables the fifty-percent point is at about 854 clauses or 4.27 clauses/variable. Thus the fifty-percent point is moving toward the 4.2 clause/variable point. This tends to decrease the percent satisfiable at 4.2 clauses/variable. This effect seems to dominate up to about 100 variables.

Above 100 variables another effect seems to take over. Recall that the percent satisfiable curve gets stceper as the number of variables increases. This causes the percent


Fig. 7. Percent satisfiable as a function of the clause/variable ratio.
satisfiable at 4.2 clauses/variable to increase. We conjecture that it approaches 100 percent in the limit as the number of variables approaches infinity. ${ }^{5}$

### 2.1.3. Experiment 3: 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 300 , incrementing by 20 . In each case we collected data near where we expected to find the crossover point. For each data point we ran TABLEAU on $10^{4}$ randomly generated 3-SAT problems ( $10^{3}$ for 280 variables and above). The raw data points are given in the appendix.

## Results

The results for 20, 100, 180, and 260 variables are shown in Fig. 7. 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 nearly linear in the neighborhood of the crossover point. To derive a good estimate of the fifty-percent point for each number of variables, we fit a line to the data

[^2]

Fig. 8. Experimental results for the number of clauses required for crossover plotted with the linear model $c=4.24 v+5.55$.
for each number of variables, and then interpolated to get the number of clauses at the fifty-percent point. The resulting points are shown in Fig. 8.

## Discussion

The data in Fig. 8 appears quite linear. A least-square fit to the data yields:

$$
\begin{equation*}
c=4.24 v+5.55 \tag{1}
\end{equation*}
$$

To the eye this appears to be quite a good fit, and in fact the residuals are only one clause or so. However, a close look at the residuals, shown in Fig. 9, reveals a definite pattern. This suggests that there are nonlinearities in the data not captured by the fit, and further suggests that projecting this fit to larger numbers of variables is not likely to be successful.

A different equation is suggested by Kirkpatrick and Selman [12]. They use finite-size scaling methods from statistical physics to derive an equation of the form:

$$
\begin{equation*}
c=\alpha_{0} v+\alpha_{1} v^{1-u} . \tag{2}
\end{equation*}
$$

They estimate $\alpha_{0}=4.17, \alpha_{1}=3.1$, and $u=2 / 3$. The residuals of this fit against our experimental data is shown in Fig. 10.


Fig. 9. Residuals for the fit given by $c=4.24 v+5.55$.

If we stay with an equation of this form, a better fit to the data appears to be given by $\alpha_{0}=4.258, \alpha_{i}=58.26, u=5 / 3$. The residuals for this fit are shown in Fig. 11. Judging by these residuals, the equation

$$
\begin{equation*}
c=4.258 v+58.26 v^{-2 / 3} \tag{3}
\end{equation*}
$$

is our best current estimator for projecting values for the crossover point. ${ }^{6}$
A detailed discussion of the relationship between this data and the theory behind rescaling is beyond the scope of this paper but we should note that these parameters ( $\alpha_{0}=4.258$ and $u=5 / 3$ ) do not work well at all as rescaling parameters. However, $\alpha_{0}=4.258$ and $u=2 / 3$ do work well as rescaling parameters and if we then write Kirkpatrick and Selman's parameter $y_{50}$ as a function of $1 / v$ then we recover an equation of the form of 3 for the location of the crossover point.

[^3]

Fig. 10. Residuals for the fit given by $c=4.17 v+3.1 v^{1 / 3}$.

### 2.2. Problem difficulty at, below, and above the crossover point

### 2.2.I. Experiment 4: problem difficulty in the under-constrained region

Generally speaking problems in the under-constrained region are quite easy. However, some researchers have found rare problems that seem to be harder than any problems in the crossover region [10,11]. The goal of this experiment was to look for such extremely hard problems.

## Experimental method

Following Gent and Walsh [10], we fixed the number of variables but varied the clause/variable ratio from 1.8 to 3.0. In our experiments we took the number of variables to be 200 (Gent and Walsh used 50 -variable problems). Also following Gent and Walsh, we took $10^{6}$ problems at each ratio.

## Results

In Fig. 12 we show the mean, median, and maximum number of branch points as a function of the clause/variable ratio. ${ }^{7}$ For comparison, in the set of 100,000 problems

[^4]

Fig. 11. Residuals for the fit given by $c=4.258 v+58.26 v^{-2 / 3}$.
in the crossover region used for Experiment 3, the mean and median number of branch points is 1290 , and the maximum number of branch points is 7781 . The minimum number of branch points in the crossover region is 9 but this is for a satisfiable problem so it presumably corresponds to a case where TABLEAU happened to go almost directly to a model. The minimum number of branch points for an unsatisfiable problem is 305.

## Discussion

These results show that for tableau the hardest problems in the under-constrained region are many times easier than the easiest unsatisfiable problems in the crossover region. This appears to contradict the results of Gent and Walsh who show that for the Davis-Putnam algorithm there are rare problems in the under-constrained region that are much harder than any problems in the crossover region.

The primary difference between the Davis-Putnam ${ }^{8}$ algorithm and Tableau is that tableau uses dynamic variable ordering. Thus the most likely explanation of the difference between these results and those of Gent and Walsh is that Tableau's variable selection heuristics are working (or equivalently that the scarce, extremely large,

[^5]

Fig. 12. Hardness in the under-constrained region: number of branch points as a function of the clause/variable ratio.

Davis-Putnam search trees are the result of bad choices for branch variables). ${ }^{9}$
It is certainly possible that if we either (1) increased the number of instances, or (2) increased the number of variables, we would find under-constrained problems that are hard for tableau. While speculation is always dangerous, our expectation is that increasing the number of instances would be unlikely to lead to hard under-constrained instances; the current set of $1,000,000$ instances is just too tightly clustered. However, as the number of variables is increased, the amount of information available to TABLEAU's heuristics at the top of the search tree decreases. Thus, as the number of variables in increased it is possible that hard under-constrained problems will emerge.

### 2.2.2. Experiment 5: problem difficulty in the crossover region

Since the crossover region appears to hold the hardest test cases (at least for TABLEAU on problems of this size) it makes sense to compare algorithms on instances drawn from

[^6]Table I
The number of branch points and the run time of tableau in the neighborhood of the crossover point. Run times are in seconds and are for a Sparc 10.51

| $V$ | $C$ | Branch points |  |  |  | CPU time (sec.) |  |  |
| ---: | ---: | ---: | ---: | ---: | ---: | ---: | ---: | ---: |
|  |  | Mean | Variance | Maximum |  | Mean | Variance | Maximum |
| 25 | 112 | 2.50 | 1.11 | 7 |  | 0.00 | 0.00 | 0.0 |
| 50 | 218 | 6.49 | 2.32 | 17 |  | 0.00 | 0.00 | 0.0 |
| 75 | 324 | 15.44 | 6.72 | 44 |  | 0.00 | 0.01 | 0.1 |
| 100 | 430 | 35.77 | 18.99 | 94 |  | 0.05 | 0.05 | 0.1 |
| 125 | 536 | 89.28 | 52.41 | 311 |  | 0.14 | 0.09 | 0.5 |
| 150 | 642 | 225.04 | 133.85 | 739 |  | 0.41 | 0.24 | 1.4 |
| 175 | 748 | 538.19 | 341.04 | 1696 | 1.03 | 0.65 | 3.9 |  |
| 200 | 854 | 1268.02 | 848.67 | 4307 | 2.78 | 1.86 | 9.3 |  |
| 225 | 960 | 3080.98 | 2163.82 | 11141 | 7.62 | 5.47 | 37.1 |  |
| 250 | 1066 | 7477.17 | 5528.35 | 30902 | 20.33 | 15.16 | 80.4 |  |
| 275 | 1172 | 19345.80 | 14048.62 | 78904 | 57.36 | 42.46 | 237.6 |  |
| 300 | 1278 | 44646.20 | 34972.27 | 187257 | 141.48 | 111.50 | 587.0 |  |

this region. In this experiment we compute the rate of growth of the number of branch points and the run time of tableau in the crossover region.

## Experimental method

Following Freeman [8], we varied the number of variables from 25 to 350 by 25 , and choose the number of clauses to give approximately fifty-percent satisfiability. We ran on 1000 instances at each point.

## Results

The results are shown in Table 1. The mean number of branch points is plotted in Fig. 13. Run times here are for a Sparc 10.51 and are "user" times as reported by "/usr/bin/time".

## Discussion

As Fig. 13 shows, the growth rate of the number of branch points is exponential in the number of variables. If we fit this data to an equation of the form branches $=2^{a v+b}$ we get:

$$
\text { branches }=2^{v / 19.5+0.08} .
$$

For comparison, for the algorithm described in our previous work [4], the number of branches grows as $2^{v / 17}$. Freeman [8] gets branches $=2^{v / 18.5-0.02}$.

The run times for TABLEAU are quite competitive. In fact, for large problems near the crossover point, TABLEAU is the fastest algorithm of which we are aware (TABLEAU


Fig. 13. Hardness in the crossover region: Number of branch points as a function of the number of variables. Number of branch points shown on log scale.
is only slightly faster than Freeman's POSIT, but tableau 's run times appear to be growing more slowly).

### 2.2.3. Experiment 6: problem difficulty in the over-constrained region

## Experimental method

For this experiment we fixed the clause/variable ratio at 10 and varied the number of variables from 100 to 1000 , incrementing by 100 . At each point we ran on 1000 instances.

## Results

The results are shown in Table 2. The mean number of branch points is plotted in Fig. 14. Run times here are for a Sparc 10.51.

## Discussion

In this region the number of branch points still seems to grow exponentially with the number of variables but the rate of growth is considerably slower. If we fit this data to an equation of the form branches $=2^{a v+b}$ we get:

$$
\text { branches }=2^{x / 68.2+0.09}
$$

Table 2
The number of branch points and the run time of tableau in the over-constrained region. Run times are in seconds and are for a Spare 10.51

| $V$ | C | Branch points |  |  | CPU time (sec.) |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | Mean | Variance | Maximum | Mean | Variance | Maximum |
| 100 | 1000 | 2.62 | 0.74 | 5 | 0.08 | 0.00 | 0.1 |
| 200 | 2000 | 8.31 | 1.83 | 17 | 0.18 | 0.01 | 0.2 |
| 300 | 3000 | 23.57 | 4.94 | 48 | 0.39 | 0.03 | 0.5 |
| 400 | 4000 | 64.71 | 13.35 | 121 | 0.95 | 0.12 | 1.4 |
| 500 | 5000 | 176.87 | 33.25 | 318 | 2.60 | 0.35 | 4.0 |
| 600 | 6000 | 493.73 | 96.35 | 797 | 7.52 | 1.12 | 11.3 |
| 700 | 7000 | 1331.53 | 242.45 | 2253 | 21.56 | 3.18 | 32.2 |
| 800 | 8000 | 3634.19 | 662.78 | 5931 | 62.28 | 9.46 | 96.0 |
| 900 | 9000 | 9798.53 | 1792.24 | 16269 | 175.26 | 26.42 | 264.4 |
| 1000 | 10000 | 26375.80 | 4722.36 | 43792 | 487.39 | 70.09 | 720.0 |



Fig. 14. Hardness in the over-constrained region: Number of branch points as a function of the number of variables. Number of branch points shown on log scale.

## 3. The TABLEAU algorithm

The basic algorithm underlying TABLEAU is depth-first search with unit-propagation. This combination can be traced back at least as far as the work of Davis, Logemann, and Loveland [5]. To this basic framework Tableau adds a highly-optimized unitpropagation algorithm, and a set of special-purpose heuristics for selecting branch variables. Section 3.1 describes the basic algorithm, and Sections 3.2 and 3.3, respectively, describe the unit-propagation algorithm and the heuristics. These sections are fairly detailed and are probably primarily of interest to those actually building satisfiable or constraint-satisfaction algorithms.

### 3.1. Basic algorithm

```
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 consists of the repeated application of the inference rule:

$$
\begin{aligned}
& x \\
& \frac{\neg x \vee y_{1} \cdots \vee y_{n}}{y_{1} \vee \cdots \vee y_{n}}
\end{aligned}
$$

(similarly for $\neg x$ ).
Complete unit-propagation takes time linear in the size of the theory [6].

### 3.2. Fast unit-propagation

The computational bottlenecks for TABLEAU are the unit-propagator and the machinery needed to save the state of the search for backtracking. TABLEAU 's data-structures are designed to simultaneously allow efficient unit-propagation and inexpensive backtracking.

The key to inexpensive backtracking is being able to describe the state of the search as concisely as possible [8]; more concise descriptions require less copying and use less memory. The current version of tableau maintains three arrays. These arrays record for each variable: (1) its current assignment (if any). (2) the number of binary clauses in which it occurs positively (i.e., not negated), and (3) the number of binary clauses in which it occurs negatively. The binary clause counts are critical for the heuristics described below.

To speed up unit-propagation, for each variable we maintain a list of the clauses in which it occurs. Then whenever a variable is valued we walk down its list. For each clause in the list there are several operations we might have to perform: value some other variable in the clause, update the binary clause counts on the other variables, etc. It is straightforward to enumerate the possible conditions and the actions nccessary for each condition. For example, for a clause of the form $x \vee y \vee z$ these include:
(1) If $x$ is assigned false, and $y$ and $z$ are unassigned then increment the binary clause counts on $y$ and $z$.
(2) If $y=F$, and $z$ is unassigned then set $z=T$. Similarly, if $z=F$ and $y$ is unassigned then set $y=T$.
(3) If $x$ is assigned true, $y=F$, and $z$ is unassigned then decrement the binary clause count for $z$.
Etc.
It turns out to be more efficient to unit-propagate breadth-first (this is mostly because of the relatively high cost of procedural calls and stack-based recursion in C). Breadthfirst unit-propagation introduces one additional rather technical complication. Assume $x$ is assigned $F$, and we have two clauses of the form $x \vee y_{1}$ and $x \vee y_{2}$. We then assign values to $y_{1}$ and $y_{2}$, and push $y_{1}$ and $y_{2}$ onto the unit-resolution queue. Now assume there is some other clause of the form $y_{1} \vee \neg y_{2} \vee z$. When we unit-propagate $y_{1}$ we decrement the binary clause count for $z$ (by case (3) above). When we later unitpropagate $y_{2}$ we do nothing (this assignment does not create a binary clause because $y_{1}=T$ ). If the binary clause count for $z$ was zero originally it may now be -1 ! The problem is that we somehow skipped the intermediate state that would have contained a binary clause: $y_{2}=T$ and $y_{1}$ unassigned. If we had seen this state then this would have incremented the binary count for $z$ and the count would have ended up zero. There are many possible fixes for this problem. The current version of tableau explicitly examines the unit-resolution queue to check for conditions of this kind. In this case, when propagating $y_{2}$, TABLEAU would see that $y_{1}$ is still on the queue and would thus realize that this clause was not really a binary clause (since $y_{1}$ was valued but not propagated) and so would not decrement the binary clause count for $z$.

### 3.3. Heuristics

There are two choices that must be made on each recursive call to the Find_Model routine. First, one must decide which variable to branch on, and second one must decide which value (i.e., TRUE or FALSE) to try first. The basic Davis-Logemann-Loveland procedure simply branches on the variables in some pre-determined order (independent of the problem). We have found that simple variable selection heuristics can make a significant difference in the average size of the search tree (however, for random 3-SAT we have yet to find any useful heuristics for deciding which value to try first).

Our primary preference criterion is to prefer variables that would cause a large number of unit-propagations. This heuristic is similar to the one used in [7,19]. We have found that it is not cost-effective to actually compute the number of unit-propagations that would result from valuing a variable. Instead we approximate the number of unitpropagations by counting the number of (non-redundant) binary clauses in which the
variable appears. In cases where there are no binary clauses we simply choose the variables occurring most often in the theory. This is similar to the heuristic used by Dubois in which preference is given to the variable that occurs most often in the shortest clauses in the theory.

One question here is how to combine the counts of the number of positive and negative occurrences of variables. Following Freeman [8] we use the equation

$$
\begin{equation*}
\operatorname{score}(x)=p c(x) * n c(x) * 1024+p c(x)+n c(x)+1, \tag{4}
\end{equation*}
$$

where $p c(x)(n c(x))$ is the number of positive (negative) occurrences of $x$ in binary clauses. This metric gives preference to variables that lead to significant numbers of unit-propagations in both branches of the search tree.

As suggested by Dubois, we collect a list of the top $k$ variables under this metric and then explicitly compute the effect of valuing each of these variables and unit-propagating. Setting the value for $k$ is something of a black art. Following Freeman, we currently use:

$$
k=v-21 * \text { vars_valued }
$$

where vars.valued is the number of variables assigned by the current assignment. Unlike Dubois and Freeman we count the number of new binary clauses produced (rather than the number of variables valued). We believe this gives a more accurate picture of the impact of valuing each variable. We use Eq. (4) to combine this count for valuing the variable true and false. We break ties by counting the number of occurrences in the theory (combining the number of positive and negative occurrences using Eq. (4)).

## 4. Conclusion

Our experimental results show that the hardest satisfiability problems are 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 [2,16]. For randomly-generated problems, these critically-constrained problems are found in a narrow band near the crossover point. Empirically, the number of clauses required for crossover seems to be best modeled by the equation $c=4.258 v+58.26 v^{-2 / 3}$. We thus predict that the asymptotic valuc of the clause/variable ratio will be near 4.258. Fig. 15, giving the clause/variable ratio at crossover for 20 to 300 variables, shows that our existing experimental data is consistent with this prediction. Below the crossover point we have failed to find any hard problems for tableau. At the crossover point the size of the tree searched by TABLEAU seems to grow at about $2^{v / 19.5}$. For large random 3-SAT problems near the crossover point the run time for TABLEAU seems to be slightly faster than the best previously published times. Above the crossover point the run time appears to grow as $2^{n / 68}$. This slower growth rate has allowed us to solve 1000 -variable over-constrained problems.


Fig. 15. Clause/variable ratio at the crossover point as a function of the number of variables.

## Acknowledgments

The experiments reported here were made possible by the use of three SGI (Silicon Graphic Inc.) Power Challenge systems purchased by the University of Oregon Computer Science Department through NSF grant STI-9413532.

Much of the early work on TABLEAU was done at Bell Labs. Many people there participated in discussions of this material, but we would like to particularly thank Haym Hirsh, Bart Selman, Henry Kautz, and David Mitchell. More recently, tableau has been completely rewritten with help and advice from Matt Ginsberg and John Freeman. Andrew Parkes contributed heavily to the analysis of the crossover point data. We would also like to thank the anonymous reviewers of this paper for their useful comments.

## Appendix A

Below is the raw data from our experiments on the location of the crossover point. These experiments were run on SGI (Silicon Graphic Inc.) Power Challenge machines. Run times are the "user" times reported by the command "/usr/bin/time". The load on these machines varied over the course of the experiments so there may be noise in these run times.

| Variables | Clauses | Percent SAT | Branches | Run time | Instances |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 20 | 88 | 60.29 | 2 | 0.009952 | 10000 |
| 20 | 89 | 58.33 | 1 | 0.009952 | 10000 |
| 20 | 90 | 56.25 | 1 | 0.009957 | 10000 |
| 20 | 91 | 52.56 | 1 | 0.009941 | 10000 |
| 20 | 92 | 50.43 | 1 | 0.009929 | 10000 |
| 20 | 93 | 46.86 | 1 | 0.009942 | 10000 |
| 20 | 94 | 44 | 1 | 0.009931 | 10000 |
| 20 | 95 | 40.72 | 1 | 0.009946 | 10000 |
| 20 | 96 | 38.27 | 1 | 0.009948 | 10000 |
| 20 | 97 | 35.62 | 1 | 0.009957 | 10000 |
| 40 | 171 | 61.38 | 4 | 0.019056 | 10000 |
| 40 | 172 | 59.69 | 4 | 0.01957 | 10000 |
| 40 | 173 | 57.38 | 4 | 0.019644 | 10000 |
| 40 | 174 | 55.33 | 4 | 0.019865 | 10000 |
| 40 | 175 | 52.2 | 4 | 0.019855 | 10000 |
| 40 | 176 | 49.84 | 4 | 0.01989 | 10000 |
| 40 | 177 | 47.25 | 4 | 0.01993 | 10000 |
| 40 | 178 | 45.07 | 4 | 0.019942 | 10000 |
| 40 | 179 | 42.95 | 4 | 0.019953 | 10000 |
| 40 | 180 | 40.46 | 4 | 0.01996 | 10000 |
| 60 | 255 | 60.89 | 9 | 0.025987 | 10000 |
| 60 | 256 | 58.78 | 9 | 0.026548 | 10000 |
| 60 | 257 | 55.99 | 9 | 0.026907 | 10000 |
| 60 | 258 | 55.07 | 9 | 0.027038 | 10000 |
| 60 | 259 | 51.82 | 9 | 0.027252 | 10000 |
| 60 | 260 | 49.12 | 9 | 0.027433 | 10000 |
| 60 | 261 | 47.61 | 9 | 0.027573 | 10000 |
| 60 | 262 | 46.57 | 9 | 0.027749 | 10000 |
| 60 | 263 | 44.02 | 9 | 0.0279 | 10000 |
| 60 | 264 | 41.98 | 8 | 0.028036 | 10000 |
| 80 | 340 | 58.28 | 18 | 0.04171 | 10000 |
| 80 | 341 | 57.51 | 17 | 0.041838 | 10000 |
| 80 | 342 | 54.66 | 18 | 0.042614 | 10000 |
| 80 | 343 | 52.79 | 18 | 0.042812 | 10000 |
| 80 | 344 | 50.94 | 18 | 0.043278 | 10000 |
| 80 | 345 | 49.65 | 18 | 0.043545 | 10000 |
| 80 | 346 | 47.9 | 18 | 0.043396 | 10000 |
| 80 | 347 | 44.46 | 18 | 0.043666 | 10000 |
| 80 | 348 | 43.68 | 18 | 0.043714 | 10000 |
| 80 | 349 | 40.91 | 18 | 0.043831 | 10000 |


| Variables | Clauses | Percent SAT | Branches | Run time | Instances |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 100 | 424 | 58.62 | 35 | 0.069854 | 10000 |
| 100 | 425 | 57.2 | 35 | 0.070392 | 10000 |
| 100 | 426 | 54.53 | 36 | 0.071596 | 10000 |
| 100 | 427 | 53.79 | 35 | 0.071464 | 10000 |
| 100 | 428 | 51.87 | 36 | 0.072399 | 10000 |
| 100 | 429 | 49.69 | 36 | 0.073214 | 10000 |
| 100 | 430 | 48.22 | 36 | 0.073184 | 10000 |
| 100 | 431 | 46.73 | 36 | 0.073393 | 10000 |
| 100 | 432 | 45.21 | 36 | 0.073603 | 10000 |
| 100 | 433 | 42.93 | 36 | 0.074104 | 10000 |
| 120 | 509 | 58.15 | 70 | 0.123834 | 10000 |
| 120 | 510 | 55.33 | 71 | 0.124714 | 10000 |
| 120 | 511 | 53.95 | 72 | 0.125722 | 10000 |
| 120 | 512 | 52.54 | 72 | 0.126598 | 10000 |
| 120 | 513 | 50.77 | 73 | 0.127967 | 10000 |
| 120 | 514 | 50.09 | 73 | 0.12755 | 10000 |
| 120 | 515 | 47.82 | 74 | 0.129513 | 10000 |
| 120 | 516 | 46.89 | 73 | 0.128761 | 10000 |
| 120 | 517 | 43.69 | 75 | 0.132049 | 10000 |
| 120 | 518 | 42.98 | 75 | 0.131482 | 10000 |
| 140 | 594 | 57.53 | 143 | 0.242835 | 10000 |
| 140 | 595 | 55.38 | 145 | 0.244796 | 10000 |
| 140 | 596 | 53.64 | 147 | 0.24808 | 10000 |
| 140 | 597 | 52.37 | 149 | 0.251002 | 10000 |
| 140 | 598 | 50.67 | 147 | 0.249652 | 10000 |
| 140 | 599 | 49.47 | 148 | 0.251184 | 10000 |
| 140 | 600 | 47.94 | 150 | 0.254217 | 10000 |
| 140 | 601 | 45.8 | 150 | 0.254696 | 10000 |
| 140 | 602 | 44.31 | 151 | 0.256326 | 10000 |
| 140 | 603 | 42.09 | 154 | 0.25965 | 10000 |
| 160 | 679 | 56.6 | 294 | 0.497074 | 10000 |
| 160 | 680 | 55.64 | 292 | 0.494609 | 10000 |
| 160 | 681 | 53.38 | 298 | 0.505744 | 10000 |
| 160 | 687 | 488 | 43.36 | 309 | 0.52497 |
| 160 | 682 | 52.02 | 301 | 0.510479 | 10000 |
| 160 | 683 | 49.97 | 305 | 0.516951 | 10000 |
| 160 | 684 | 49.18 | 303 | 0.51516 | 10000 |
| 160 | 685 | 47.49 | 306 | 0.519636 | 10000 |
| 160 | 686 | 45.25 | 311 | 0.528903 | 10000 |
| 160 |  | 309 | 0.52479 | 10000 |  |
|  |  |  |  |  |  |


| Variables | Clauses | Percent SAT | Branches | Run time | Instances |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 180 | 764 | 57.02 | 596 | 1.04088 | 10000 |
| 180 | 765 | 54.75 | 604 | 1.05387 | 10000 |
| 180 | 766 | 53.07 | 616 | 1.07616 | 10000 |
| 180 | 767 | 52.54 | 605 | 1.05978 | 10000 |
| 180 | 768 | 50.33 | 620 | 1.0829 | 10000 |
| 180 | 769 | 48.63 | 629 | 1.09834 | 10000 |
| 180 | 770 | 47.06 | 631 | 1.10119 | 10000 |
| 180 | 771 | 46.2 | 622 | 1.08833 | 10000 |
| 180 | 772 | 44.52 | 633 | 1.10704 | 10000 |
| 180 | 773 | 42 | 640 | 1.1184 | 10000 |
| 200 | 850 | 55.29 | 1238 | 2.26319 | 10000 |
| 200 | 851 | 52.98 | 1256 | 2.29539 | 10000 |
| 200 | 852 | 52.02 | 1249 | 2.25742 | 10000 |
| 200 | 853 | 50.14 | 1264 | 2.28371 | 10000 |
| 200 | 854 | 49.53 | 1266 | 2.28977 | 10000 |
| 200 | 855 | 47.74 | 1279 | 2.32607 | 10000 |
| 200 | 856 | 46.63 | 1283 | 2.32193 | 10000 |
| 200 | 857 | 44.67 | 1281 | 2.31943 | 10000 |
| 200 | 858 | 43.56 | 1296 | 2.35121 | 10000 |
| 200 | 859 | 41.47 | 1304 | 2.3677 | 10000 |
| 220 | 934 | 56.24 | 2513 | 4.82845 | 10000 |
| 220 | 935 | 54.44 | 2543 | 4.8885 | 10000 |
| 220 | 936 | 52.2 | 2564 | 4.93092 | 10000 |
| 220 | 937 | 51.37 | 2603 | 5.02871 | 10000 |
| 220 | 938 | 51.12 | 2593 | 5.013 | 10000 |
| 220 | 939 | 49.45 | 2575 | 4.9825 | 10000 |
| 220 | 940 | 47.85 | 2637 | 5.10846 | 10000 |
| 220 | 941 | 45.58 | 2648 | 5.12738 | 10000 |
| 220 | 942 | 44.54 | 2647 | 5.1338 | 10000 |
| 220 | 943 | 43.56 | 2642 | 5.11471 | 10000 |
| 240 | 1020 | 53.4 | 5217 | 10.4304 | 10000 |
| 240 | 1021 | 53.5 | 5167 | 10.3346 | 10000 |
| 240 | 1022 | 51.54 | 5270 | 10.6345 | 10000 |
| 240 | 1028 | 43.87 | 5390 | 10.9021 | 10000 |
| 240 | 1023 | 51.14 | 5210 | 10.5113 | 10000 |
| 240 | 1024 | 49.65 | 5323 | 10.7333 | 10000 |
| 240 | 1025 | 47.18 | 5348 | 10.7901 | 10000 |
| 240 | 1026 | 46.84 | 5364 | 10.8267 | 10000 |
| 240 | 1027 | 44.54 | 5375 | 10.8581 | 10000 |
| 240 | 43.46 | 5389 | 10.9062 | 10000 |  |
|  |  |  |  |  |  |


| Variables | Clauses | Percent SAT | Branches | Run time | Instances |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 260 | 1103 | 56.6253 | 10477 | 22.0023 | 9992 |
| 260 | 1104 | 54.9304 | 10613 | 22.2801 | 10060 |
| 260 | 1105 | 54.8761 | 10549 | 22.1547 | 10008 |
| 260 | 1106 | 53.1589 | 10680 | 22.4421 | 10051 |
| 260 | 1107 | 51.5561 | 10879 | 23.1446 | 10057 |
| 260 | 1108 | 51.3549 | 10691 | 22.7078 | 10038 |
| 260 | 1109 | 49.8014 | 10904 | 23.2187 | 10072 |
| 260 | 1110 | 48.0433 | 11044 | 23.5086 | 10068 |
| 260 | 1111 | 46.0679 | 11036 | 23.3709 | 11279 |
| 260 | 1112 | 46.0132 | 11056 | 23.4536 | 10008 |
| 280 | 1188 | 57.6 | 21449 | 47.4166 | 1000 |
| 280 | 1189 | 53.9 | 21725 | 48.0683 | 1000 |
| 280 | 1190 | 55.3 | 21783 | 48.2112 | 1000 |
| 280 | 1191 | 50.2 | 22923 | 50.771 | 1000 |
| 280 | 1192 | 51.6 | 22109 | 49.0018 | 1000 |
| 280 | 1193 | 51.7 | 22254 | 49.0067 | 1000 |
| 280 | 1194 | 50.2 | 22304 | 49.1085 | 1000 |
| 280 | 1195 | 49.6 | 21903 | 48.2329 | 1000 |
| 280 | 1196 | 46.7 | 23003 | 50.6697 | 1000 |
| 280 | 1197 | 46.9 | 23021 | 50.7528 | 1000 |
| 300 | 1274 | 57 | 43298 | 99.8636 | 1000 |
| 300 | 1275 | 55.3 | 44240 | 102.05 | 1000 |
| 300 | 1276 | 51.6 | 44718 | 103.172 | 1000 |
| 300 | 1277 | 52.8 | 45979 | 106.443 | 1000 |
| 300 | 1278 | 48.6 | 46815 | 108.553 | 1000 |
| 300 | 1279 | 53.4 | 42558 | 98.7138 | 1000 |
| 300 | 1280 | 44.7 | 47840 | 110.981 | 1000 |
| 300 | 1281 | 45.7 | 46439 | 107.761 | 1000 |
| 300 | 1282 | 47.7 | 45969 | 106.081 | 1000 |
| 300 | 1283 | 45.4 | 47156 | 108.884 | 1000 |
|  |  |  |  |  |  |

## References

[1] A. Broder, A. Frieze and E. Upfal, On the satisfiability and maximum satisfiability of random 3-CNF formulas, in: Proceedings Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (1993).
[2] P. Cheeseman, B. Kanefsky and W.M. Taylor, Where the really hard problems are, in: Proceedings IJCAI-9I, Sydney, Australia (1991) 163-169.
[3] V. Chvátal and E. Szemerédi, Many hard examples for resolution, J. ACM 35 (4) (1988) 759-768.
14| J.M. Crawford and L.D. Auton, Experimental results on the crossover point in satisfiability problems, in: Proceedings AAAI-93, Washington, DC (1993) 21-27.
[5] M. Davis, G. Logemann and D. Loveland, A machine program for theorem proving, Commun. ACM 5 (1962) 394-397.
[6] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Logic Programming 3 (1984) 267-284.
$17 \mid$ O. Dubois, P. Andre, Y. Boufkhad and J. Carlier, SAT versus UNSAT, in: Proceedings Second DIMACS Challenge: Cliques, Coloring and Satisfiability, Rutgers University, NJ (1993).
[8| J.W. Freeman, Improvements to propositional satisfiability search algorithms, Doctoral Dissertation, University of Pennsylvania, Philadelphia, PA (1994).
19| M.R. Garey and D.S. Johnson, Computers and Intractability (Freeman, New York, 1979).
|10| I.P. Gent and T. Walsh, Easy problems are sometimes hard, Artif. Intell. 70 (1994) 335-345.
[11] T. Hogg and C.P. Williams, The hardest constraint problems: a double phase transition. Artif. Intell. 69 (1994) 359-377.
|12| S. Kirkpatrick and B. Selman, Critical behavior in the satisfiability of random boolean expressions, Science 264 (1994) 1297-1301.
$113 \mid$ T. Larrabee and Y. Tsuji, Evidence for a satisfiability threshold for random 2CNF formulas, in: Working Notes AAAI Spring Symposium on AI and NP-Hard Problems, Stanford, CA (1993).
$|14|$ S. Minton, M.D. Johnson, A.B. Philips and P. Laird, Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. in: Proceedings AAAI-90, Boston, MA (1990) 17-24.
115| D. Mitchell, An empirical study of random SAT, Master's Thesis, Department of Computing Science, Simon Fraser University, Vancouver, BC (1993).
[16] D. Mitchell, B. Selman and H.J. Levesque, Hard and easy distributions of SAT problems, in: Proceedings AAAI-92, San Jose, CA (1992) 459-465.
|17| B. Selman, H.J. Levesque and D. Mitchell, A new method for solving hard satisfiability problems, in: Proceedings AAAI-92, San Jose, CA (1992) 440-446.
[18] R.M. Smullyan, First Order Logic (Springer-Verlag, New York, 1968).
[19] R.D. Zabih and D.A. McAllester, A rearrangement search strategy for determining propositional satisfiability, in: Proceedings AAAI-88, St. Paul, MN (1988) 155-160.


[^0]:    * Corresponding author. Fax: (503) 346-0474. E-mail: jc@cs.uoregon.edu.
    ${ }^{1}$ This work has been supported by the Air Force Office of Scientific Research under grant number 92-0693 and by ARPA/Rome Labs under grant numbers F30602-91-C-0036 and F30602-93-C-00031. Some of this work was done while the first author was at AT\&T Bell Laboratories.
    ${ }^{2}$ E-mail: Ida@research.att.com.

[^1]:    ${ }^{3}$ A clause is a disjunction of variables or negated variables.
    ${ }^{4}$ A truth assignment is a mapping from $U$ to $\{$ true, false $\}$.

[^2]:    ${ }^{5}$ This explanation of Fig. 5 is due to David Mitchell.

[^3]:    ${ }^{6}$ We give four significant figures in this equation only because using three significant figures leads to significantly worse behavior on the part of the residuals. Deriving meaningful bounds for these constants is a nontrivial exercise because this fit is to the crossover point data which is itself the result of interpolating from a least-square fit to experimental data that has a certain uncertainty to it. Further, the residuals in Fig. 11 reveal that there is some additional correction term needed for small values of $v$ that is also certainly skewing these constant values by some amount.

[^4]:    ${ }^{7}$ To avoid any possible ambiguity we measure the size of the search tree by counting branch points. A branch point is a point at which tableau is recursively called twice, setting some variable to true and then false. We count the number of these pairs of recursive calls since they are in a sense the root of the exponential complexity of the algorithm.

[^5]:    ${ }^{8}$ The algorithm Gent and Walsh refer to as "Davis-Putnam" always picks branch variables according to a priority scheme that is fixed, essentially randomly, before the search begins. TABLEAU uses a variety of heuristics (described in Section 3.3 below) to choose branch variables.

[^6]:    ${ }^{9}$ Gent and Walsh also show that branch variable selection heuristics like those used in tableau fail to prevent the occurrence of hard problems in the under-constrained region for the constant probability model. These results are less relevant to our results because the constant probability model leads to a much different distribution of instances (and in particular in the constant probability model the mean of the difficulty of the problems in the crossover region does not seem to be that much higher than the mean of the difficulty of the problems in the under-constrained region [15]).

