Tableau: A Fast Procedure to Test Satisfiability

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

January 20, 1993

Abstract

Tableau is a fast C implementation of a Davis-Putnam type satisfiability testing procedure. If utilizes an efficient unit propagation algorithm and a variety of heuristics for choosing branch variables. We have used tableau to perform a series of experiments. One of our most significant results to date is that for 3-SAT the number of clauses required to reach the *cross-over point* — the point at which half of the randomly generated theories are satisfiable — seems, for up to 260 variables, to be a linear function of the number of variables.

Other submissions: To appear *AAAI-93 Spring Symposium on AI and NP-Hard Problems*. Submitted to *AAAI-93*.

1 Introduction

Tableau is a C implementation of a David-Putnam style satisfiability testing algorithm. We use a set of data structures designed to support very efficient unit-propagation (we can discuss these in the full paper if people would be interested). We also use a series of heuristics for choosing branch variables. These are discussed briefly in section 2. We have performed several sets of experiments. We include a brief description of two of these below. We are quite interested in testing Tableau on additional data sets and problem classes in the challenge.

2 Heuristics

The basic Davis-Putnam procedure simply branches on the variables in some predetermined 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.

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.[1] Formally, one can easily show the following:

Theorem 2.1 *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.*

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 non-negated 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. This heuristic is similar to the one used in [Zabih & McAllester 88].

In cases where several variables occur in the same number of binary clauses, we count the number of unvalued *singleton neighbors*.[2] We refer to two terms as *neighbors*

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

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

if they occur together in some clause.[3] 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 $x \vee y$ and the term x occurs nowhere else in the theory then x is a singleton). Singletons are important because if that one clause becomes redundant, the variable will then occur with only one sign and so can be valued without fear of missing any possible models of the theory:

Theorem 2.2 *If a variable x always occurs negated in a theory Th then Th is satisfiable iff $Th \wedge \neg x$ is satisfiable.*

A similar result holds for variables which only occur non-negated. Counting singleton neighbors is a relatively expensive process but it rarely has to be done (in most cases ordering by the number of occurrences in binary clauses produces a unique best variable).

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.

We evaluate the effectiveness of the heuristics by counting the number of nodes in the search trees they generate.[4] Figure 1 shows the effects of the search heuristics on 1000 randomly generated 3-SAT problems with 100 variables and 429 clauses. These problems are at the cross-over point and thus are some of the most difficult 100 variable 3-SAT problems.

Figure 2 shows a comparison of tableau with Davis-Putnam on 50-150 variable 3-SAT problems near the cross-over point. The Davis-Putnam data in this table is courtesy of David Mitchell. The key point to note is that tableau's heuristics seem to have a greater effect as the size of the problem is increased. The experience of the community has been that the complexity of the Davis-Putnam algorithm on 3-SAT problems near the cross-over 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.

3 The Cross-Over Point

Consider a randomly generated satisfiability 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 then 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 which are

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

[4] 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 no more expensive than the unit-propagation required at each node.

Heuristic	Nodes	Standard Deviation
Davis-Putnam	41320	56808
most-constrained-first	14190	17290
most occurrences in binary clauses	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.

Tableau:

Variables	Clauses	Experiments	Nodes	Standard dev.	Run time (sec.)[a]
50	218	1000	26	11	.02
100	430	1000	204	114	.47
150	635	1000	1532	1098	3.5

Davis-Putnam:

Variables	Clauses	Experiments	Nodes	Standard deviation
50	218	1000	341	335
100	430	1000	42,407	54,418
150	635	164	3,252,280	4,221,380

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

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

neither over nor under constrained. Cheeseman has shown empirically that this is indeed the case [Cheeseman et al. 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 [Mitchell et al. 92]. We refer to the point at which exactly half of the randomly generated problems have solutions as the *cross-over* point.

The location of the cross-over point is of both theoretical and practical importance. It is theoretically interesting since the number of constraints required to achieve cross-over 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 shall show that in the case of 3-SAT the number of constraints required for cross-over is 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 cross-over point (though no satisfactory method has yet been proposed). The cross-over point is a practical interest for several reasons. First, since empirically the hardest problems seem to be found near the cross-over point, it makes sense to test candidate algorithms on these hard problems. Similarly, if one encounters in practice a problem that is near the cross-over 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 [Selman et al. 92; Minton et al. 90] 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 cross-over 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 cross-over point occurs, then for random problems (*i.e.*, problems with no structure) the number of clauses can be used as a predictor of satisfiability.

Figure 3 shows the percent of satisfiable theories, and the average search-tree size, for problems with 200 variables and from zero to 2000 clauses (incremented by 20 with 1000 runs at each point).[5] This graph dramatically illustrates the increase in problem difficulty at the cross-over point observed in [Mitchell et al. 92].

Figure 4 shows our current best results on the location of the cross-over point for 20 to 260 variables. The line shown is a simple a least-squares fit to the data.[6] We have also analytically calculated the cross-over point for three variables (it turns out to be 19.65 clauses). If we add this data to the least-square fit we get:

$$clauses = 4.24vars + 6.21$$

This equation should be regarded as our best current estimate of the location of the cross-over point.

[5] 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.

[6] The fit is relatively good. This is partly due to the large number of experiments run to compute these values. We will include full descriptions of the experiments in the final paper.

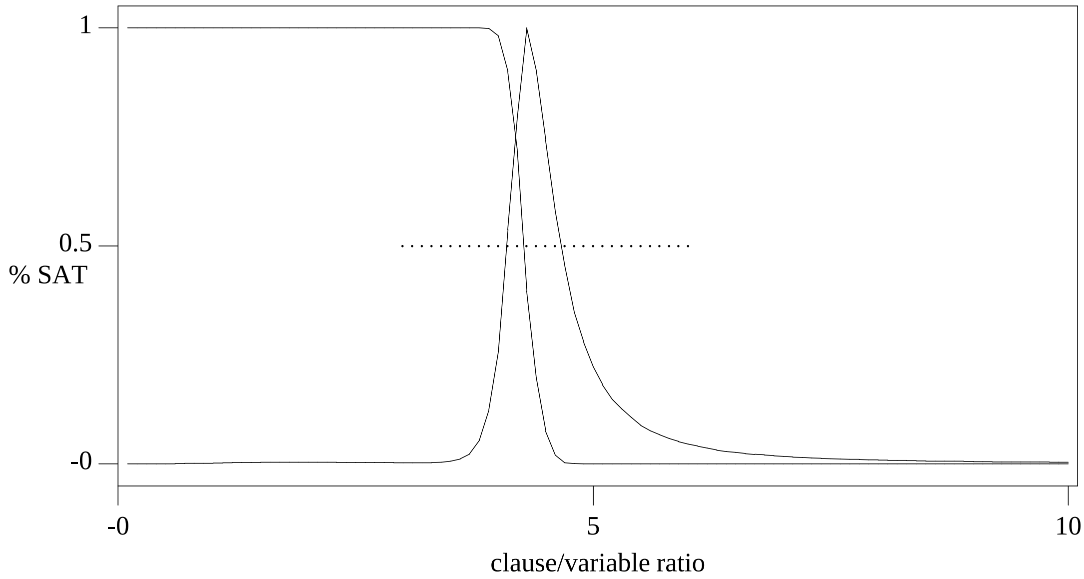

Figure 3: Percent satisfiable and size of the search tree — 200 variables.

4 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 the results of [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 cross-over point. We have determined that the number of clauses required for cross-over seems to be a *linear* function of the number of variables. Our best empirical estimate of this function is $clauses = 4.24vars + 6.21$. Finally, the complexity of our satisfiability algorithm seems to be on average roughly linear below the cross-over point (though even here there are some problems that are as hard as those at the cross-over point). At the cross-over point the complexity seems to be about $2^{n/17}$ (where n is the number of variables). Above the cross-over point we conjecture that the growth rate will be of form $2^{n/k}$ where k is less than 17 and decreases as one gets further from the cross-over point.

Acknowledgments

This work could only have been done at Bell Labs. Many people participated in discussions of this material, but we would like to particularly thank Haym Hirsh,

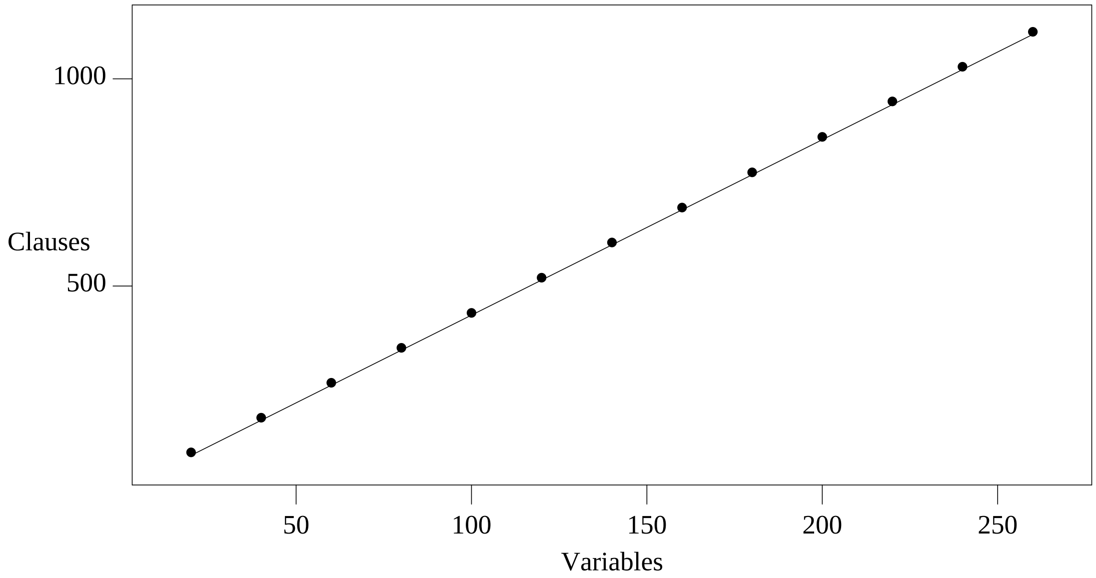

Figure 4: Empirical values for the number of clauses required for cross-over as a function of the number of variables.

Bart Selman, Henry Kautz, and David Mitchell.

References

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

[Chvátal & Szemerédi 88] Chvátal, V. and Szemerédi, E. (1988). Many Hard Examples for Resolution. JACM 35:4, pp. 759-768.

[Crawford & Auton 93] Crawford, J.M. and Auton L.D. (1993). Pushing the edge of the satisfiability envelope. In preparation.

[Davis & Putnam 60] Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7, 201-215.

[Garey & Johnson 79] Garey, M.R. and Johnson D.S. (1979). Computers and Intractability. W.H. Freeman and Company, New York.

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

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

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

[Smullyan, 68] Smullyan, R. M. (1968) First Order Logic Springer-Verlag New York Inc.

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