アルゴリズム 12－37
(1989. 11. 22)

Random Satisfiability Problems

Paul Purdom
Department of Computer Science
Indiana University, Bloomington, Indiana 47405, U. S. A.

Abstract: Various algorithms for satisfiability problems require vastly different times to solve typical problems. The time used when solving random problems is discussed for five algorithms. The results from asymptotic analyses are surveyed. Plots of the average number of nodes per problem are given for random problems with 50 variables. The plots give contours for the number of nodes as a function of the number of clauses and of the probability of that a literal is in a clause. They show the strengths and weaknesses of each algorithm.

Introduction

The best algorithms for NP-complete problems appear to use exponential worst-case time. Yet some algorithms can solve typical problems quite rapidly. This paper summarizes asymptotic studies of five algorithms for the satisfiability problem and gives new curves showing the average time each algorithm uses for solving random satisfiability problems with 50 variables. The number of clauses per problem varies between 1 and 500. The probability that a literal is in a clause also varies. These parameters have a drastic effect on the average running time. No algorithm is best for the entire range of parameters. These studies clearly indicate some of the strengths and weaknesses of the algorithms.

Although the algorithms are evaluated by their speed at solving the satisfiability problem, the algorithms can solve any discrete constraint satisfaction problem, and the algorithms are stated for the broader class of problems. Let $R_1, \ldots, R_t$ be relations on variables $x_1, \ldots, x_v$, where each variable has a finite set of possible values. The constraint satisfaction problem is to set the variables so that

$$R_1(x_1, \ldots, x_v) \wedge \cdots \wedge R_t(x_1, \ldots, x_v) \qquad (1)$$

is *true* or to determine that this can not be done. In the satisfiability problem, each variable can have the value *true* or *false* and each R is a clause (the logical *or* of literals, where a literal is a variable or its negation).

Algorithms

This paper has an informal statement of each algorithm. The original papers have more details.

Backtracking [2]: Select the first remaining variable from those variables without a value. (If all variables have values then the current setting is a solution.) Generate a set of subproblems by assigning the selected variable each possible value. Solve the subproblems recursively, but skip those subproblems where some R_i simplifies to *false* with the current variable setting.

Backtracking is an old algorithm. It is relatively quick when most of the subproblems are skipped. The version of backtracking given here finds all the solutions to a problem, so it needs a lot of time if the problem has a lot of solutions.

Unit clause backtracking [17]: If some relation depends on only one of the unset variables then select that variable, otherwise select the first unset variable. Continue as in backtracking.

The improved variable selection results in much faster backtracking in many cases that arise in practice [1]. The version of unit clause backtracking given here also finds all the solutions.

Clause order backtracking [4]: Select the first relation which can evaluate to both *true* and *false* depending on the setting of the unset variables. Select variables from this relation until its value is determined. Process selected variables in the same way that backtracking does.

By setting only those variables that affect the value of relations, this algorithm sometimes avoids the need of assigning values to all of the variables. It finds all the solutions, but in a compressed form. A single solution with unset variables represents the set of solutions obtained by making each possible assignment to the unset variables.

The pure literal rule algorithm [12]: Select the first variable that does not have a value. (If all variables have values, then the current setting is a solution if it satisfies all the relations.) If some value of the selected variable results in all relations that depend on the selected variable having the value *true* then generate one subproblem by assigning the selected variable that good value. Otherwise generate a subproblem for each value of the selected variable. Solve the subproblems recursively.

This algorithm has the essence of the pure literal rule from the Davis-Putman procedure [6]. By removing most of the good features, an analyzable algorithm is obtained. It can solve a wide class of problems in polynomial average time, but it does not find all solutions.

Iwama's algorithm [15]: For each relation of the problem count the number nonsolutions. For each pair of relations that depend on different variables count the number of nonsolutions. Continue for triples, etc. Use inclusion-exclusion to determine the total number of nonsolutions. Subtract the number of nonsolutions from the number of possible variable settings to obtain the number of solutions.

The counting for this algorithm can often be done quickly if there are few clauses or if the clauses depend on lots of variables.

It is clear that an improved algorithm can be obtained by combining the techniques of the first four algorithms. Such an algorithm would be at least as fast as the fastest of the four. Iwama's algorithm is sometimes much faster than the others and sometimes much slower. It is not so clear how to best combine it with the previous four algorithms.

Random satisfiability problems

In the random clause model there are v variables available for forming CNF predicates. A literal is a variable or its negation. A clause consists of the logical *or* of a set of literals. A random clause is formed by including each of the $2v$ possible literals with probability p. It is possible for a clause to contain both a variable and its negation, but this is not likely when p is well below $v^{-1/2}$. A random predicate is formed by taking the logical *and* of t random clauses.

The characteristics of the typical predicate varies with the values of v, t, and p. One of the most important characteristics is the average number of solutions per problem. When solutions are rare, the running time of a satisfiability algorithm depends mainly upon how quickly it can prove that a problem has no solution. When solutions are common, the time depends mainly upon how quickly an algorithm can find some solution.

Asymptotic studies of the algorithms have considered the running time of these algorithms as a function of t and p as v tends to infinity, where t and p are both functions of v. This paper includes contour plots for the case $v = 50$. The number 50 is large enough that the main features of the asymptotic analyses are clearly demonstrated.

Results

The average number of solutions for a random satisfiability problem [20] is

$$S = 2^v[1 - (1 - p)^v]^t. \qquad (2)$$

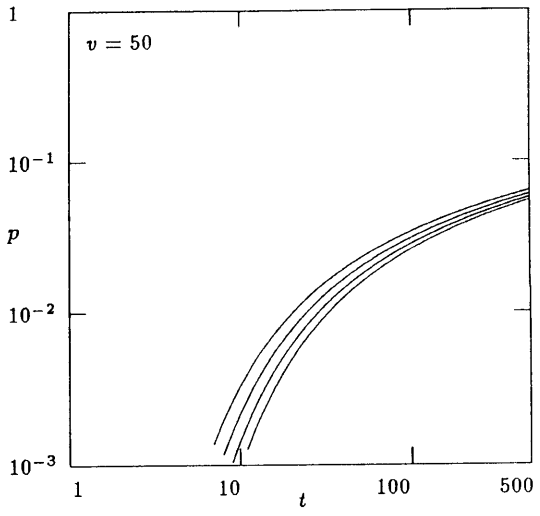

Fig. 1. Contours for the number of solutions.

The lower contour in Figure 1 shows the value of p (as a function of t) that results in an average of v solutions per problem when $v = 50$. Above it are the contours for v^2, v^3, and v^4 solutions. Both the p and t scales are logarithmic. The number of solutions is exponentially small in the lower right corner, where p is small and there are a lot of clauses. The number of solutions approaches 2^v in the upper left corner, where p is near 1 and there are few clauses. When v goes to infinity and p goes to zero, the dividing line between an exponential and polynomial number of solutions is given by [17]

$$\frac{t}{v} = \frac{\ln 2}{-\ln(1 - e^{-pv})}. \qquad (3)$$

The contours are defined only for integer values of t, consecutive points on the contour are connected by straight lines. Since the scale is logarithmic, this is most evident for small values of t. See Figure 2.

Figure 2 shows the average number of solutions when the settings of irrelevant variables are not given (as with clause order backtracking). The outer contour is for an average of v solutions. Contours for v^2, v^3 and v^4 solutions are also given. The line from the left side to the right side shows, for each t, that value of p which results in the largest number of solutions. Listing solutions in this compact form does not have much effect when t is large and p is small or moderate, but it drastically reduces the number of separate solutions in all other cases.

The contours in Figure 2 are given by [4]

$$S = S(t, 0), \qquad (4)$$

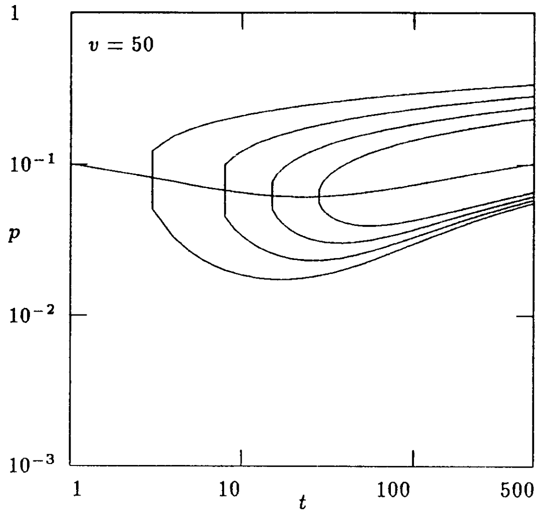

Fig. 2. Contours for compacted solutions.

where $S(t, i)$ is the solution to the recurrence

$$S(0, i) = 2^i, \tag{5}$$

$$S(t, i) = \sum_j a_j(i) S(t-1, i+j) \tag{6}$$

for $t > 0$, where

$$a_0(j) = 1 - (1-p)^v (1+p)^{v-i}, \tag{7}$$

$$a_i(j) = \sum_k \binom{v-i}{k} 2^{k-j} p^k (1-p)^{2v-i-k} \tag{8}$$

for $i > 0$. The asymptotic analysis of eq. 3 has not been completed.

The remaining figures have the same general form as Figures 1 and 2, but they give the number of nodes generated by the various algorithms while solving random satisfiability problems. Each algorithm generates nodes in time that is bounded by a low degree polynomial function of the number of clauses and variables. The exact relation between the number of nodes and the running time varies with the skill of the person programming the algorithm and with the speed of his computer. Usually these details are not as important as selecting an algorithm with the appropriate features to reduce the number of nodes.

The average number of nodes for backtracking is [20]

$$N = 1 + \sum_{1 \le i \le v} 2^i [1 - (1-p)^{2v-i+1}]^t. \tag{9}$$

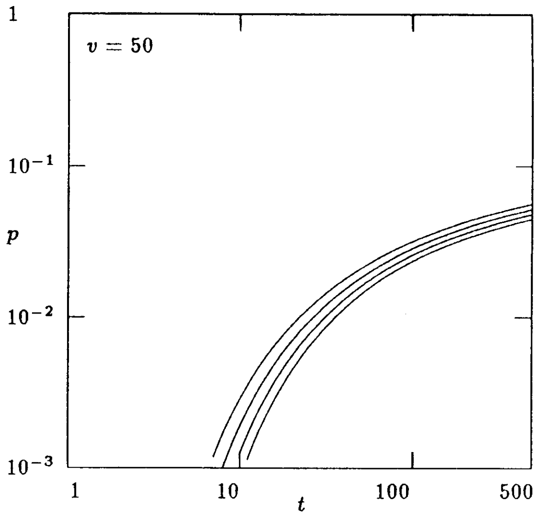

Fig. 3. Contours for backtracking.

The lower contour in Figure 3 shows the value of p (as a function of t) that results in an average of v nodes per problem. Above it are the contours for v^2, v^3, and v^4. The general shape of these contours is the same as the contours for the number of solutions. However, the contours for backtracking do not increase as rapidly, particularly above $t = v$. By $t = 500$ the value of p associated with 50 *solutions* results in nearly 50^4 *nodes*. Asymptotic analysis [20] shows when v goes to infinity the boundary between polynomial and exponential time follows eq. (3) for p below $(\ln 2)/v$. For larger values of p the boundary for exponential time diverges from that for an exponential number of solutions (See the equations in [20] and the curve in [17]). In the limit as pv goes to infinity (with p going to zero), the boundary is given by

$$\frac{t}{v} = \frac{\ln 2}{pv} e^{2pv}. \tag{10}$$

In the limit as v goes to infinity when $t > v$, there is a region where the average number of nodes is an exponential function of v while the average number of solutions is near zero [20]. This region is of considerable interest because people often use backtracking to solve problems with just a few solutions [1].

The average number of nodes for unit-clause backtracking is

$$N = 1 - [1 - (1-p)^{2v}]^t + \sum_u \binom{t}{u} [p(1-p)^{2v-1}]^u N(t-u, u, v), \tag{11}$$

where $N(t,u,v)$ is the solution to the recurrence

$$N(t,u,0) = \delta_{t0}\delta_{u0}, \tag{12}$$

$$\begin{aligned} N(l,0,v) = {} & [1-(1-p)^{2v} - 2vp(1-p)^{2v-1}]^l \\ & + 2\sum_{l',u'} \binom{l}{l'}\binom{l-l'}{u'} \\ & \times p^{l-l'+u'}(1-p)^{2(v-1)u'+l'} \\ & \times [1-(1-p)^{2v-1}]^{l-l'-u'} \\ & \times N(l',u',v-1), \end{aligned} \tag{13}$$

$$\begin{aligned} N(l,u,v) = {} & 2[1-(1-p)^{2v} - 2vp(1-p)^{2v-1}]^l(2v)^u \\ & + \sum_{l',u'} \binom{l}{l'} \sum_j \binom{l-l'}{j}\binom{u-1}{u'-j} \\ & \times 2vp^{l-l'+j}(1-p)^{2(v-1)j+l'} \\ & \times [1-(1-p)^{2v-1}]^{l-l'-j} \\ & \times \{N(l',u',v-1) \\ & \quad - [1-(1-p)^{2v-2} \\ & \qquad - 2(v-1)p(1-p)^{2v-3}]^{l'} \\ & \times (2v-2)^{u'}\}, \end{aligned} \tag{14}$$

for $u > 0$. This equation comes from an unpublished extension of the work in [17] and [18].

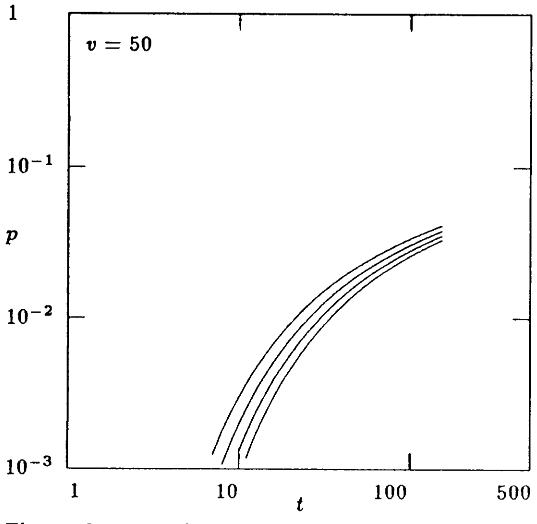

Fig. 4. Contours for unit clause backtracking.

The lower contour in Figure 4 is for an average of v nodes. Contours for v^2, v^3, and v^4 are also given. (These contours stop at $t = 153$ because of floating point overflow and because of the large amount of time required to solve the recurrence for large t.) Each contour is between the corresponding contour for the number of solutions (Figure 1) and the one for the number of nodes for backtracking (Figure 3), but much closer to the one for the number of solutions. The formulas for the asymptotic analysis [17] are too long to repeat here, but they show that when t/v is large enough, the boundary for exponential time is closer to that for backtracking than it is to that for the number of solutions. Figure 4 suggests that unit clause backtracking does much better than the lower bound analysis in [17] would suggest, but the calculations do not go to a large enough value of t/v to comment on the quality of the upper bound analysis. Among the algorithms of this paper, unit clause backtracking is the fastest when t is large and p is small.

The average number of nodes for the clause order backtracking is [4]

$$N = N(t,0), \tag{15}$$

where

$$N(0,i) = 1, \tag{16}$$

$$\begin{aligned} N(t,i) = {} & 1 + b_0(i)[N(t-1,i) - 1] \\ & + \sum_{j\geq 1} b_j(i) \\ & \quad \times \{2[1-(1-p)^{2v-i-j+1}]^{t-1} \\ & \qquad + N(t-1,i+j) - 1\}, \end{aligned} \tag{17}$$

and

$$b_0(i) = 1 - (1-p)^v(1+p)^{v-i}, \tag{18}$$

$$b_j(i) = \sum_{k\geq j} \binom{v-i}{k} 2^k p^k (1-p)^{2v-i-k} \tag{19}$$

for $j \geq 1$.

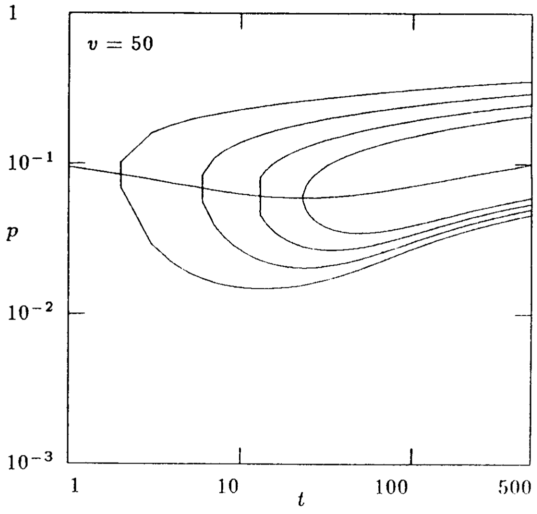

Fig. 5. Contours for clause order backtracking.

The outer contour in Figure 5 is for an average of v nodes. Contours for v^2, v^3 and v^4 are also shown. The line from the left side to the right side shows, for each t, the value of p that results in the largest number of nodes. This algorithm is much better than simple backtracking, particularly when t is not large or p is large. A comparison of Figures 2 and 5 shows that much of the speed of this algorithm is associated with the compact representation of solutions. When t is large and p is small the algorithm is only slightly better than backtracking. Among the algorithms in this paper, clause order backtracking is the fastest when t is moderate and p is small. The analysis of a simpler algorithm in [11] suggests that clause order backtracking should run in polynominal time when pv is below some constant.

The average number of nodes for the pure literal rule algorithm is [13]

$$N(0,v) = N(t,0) = 1 \tag{20}$$

$$N(t,v) = 1 + (1-p)^{2t}a(t,v-1) + \sum_{i\geq 1} \binom{t}{i} p^i (1-p)^{t-i} N(t-i, v-1) \tag{21}$$

for $t > 0$.

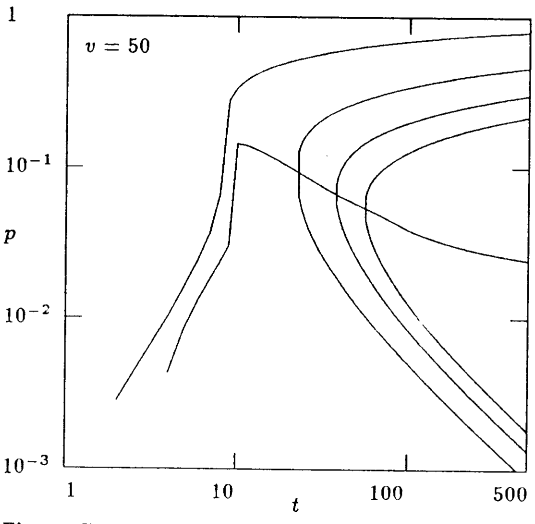

Fig. 6. Contours for the pure literal rule algorithm.

The outer contour of Figure 6 is for an average of v nodes. Contours for v^2, v^3 and v^4 are also shown. The line that is near the middle on the right side shows, for each t, the value of p that results in the largest number of nodes. Asymptotic upper bound analysis [19] show that this algorithm uses polynomial time when

$$t < \frac{\ln v}{\epsilon}, \tag{22}$$

$$p > \epsilon, \tag{23}$$

or

$$tp \leq \epsilon \sqrt{\frac{\ln v}{v}}, \tag{24}$$

where ϵ is any fixed small number. Among the algorithms in this paper, the pure literal algorithm is fastest when t is small and p is not large.

The average number of sets of clauses considered by Iwama's algorithm is given by [15]

$$N = \sum_{0\leq i\leq t} \binom{t}{i} [(1-p)^i(1-(1-p)^i)]^v. \tag{25}$$

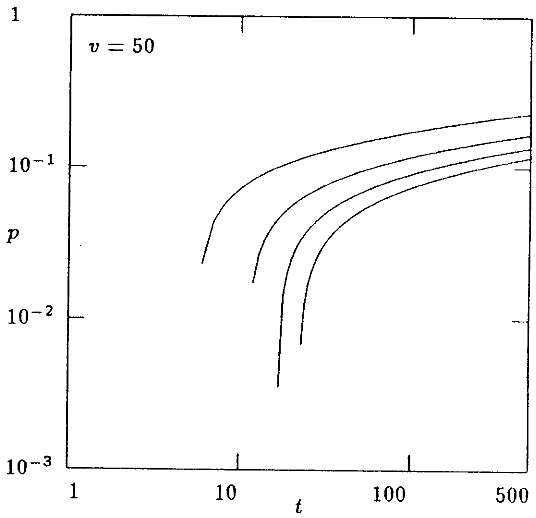

Fig. 7. Contours for Iwama's algorithm.

The *upper* contour in Figure 7 is for v nodes. Contours for v^2, v^3, and v^4 are also given. The average time [15] for Iwama's algorithm is polynomial when

$$p > \sqrt{\frac{\ln t}{v}}. \tag{26}$$

Among the algorithms of this paper, Iwama's algorithm is fastest for large p.

Discussion

The algorithms in this paper have drastic variations in their average running time for random satisfiability problems, depending on the parameters used to generate the problems. Each algorithm has a region where it is best (except backtracking is never quite as good as either unit clause or clause order backtracking). It is straightforward to add the unit clause rule and the pure literal rule to clause order backtracking to obtain an algorithm that has the good features of all the algorithms except Iwama's. The problem of how to best combine the ideas of Iwama's algorithm with the backtracking type algorithms is more challenging, and it is important.

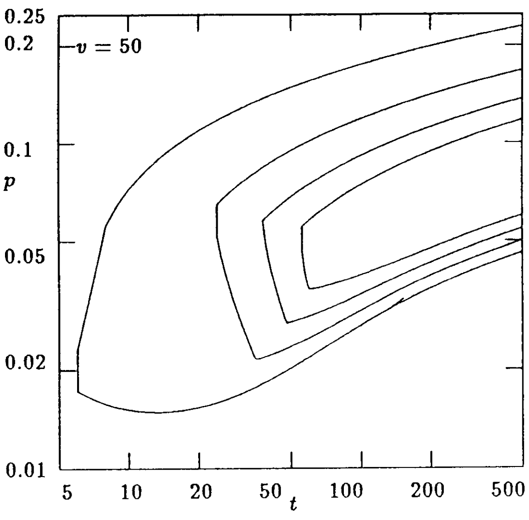

Fig. 8. The contours for a combined algorithm.

Figure 8 shows the contours that result if the best algorithm is used in each region. The outer contour is for v nodes. Contours for v^2, v^3 and v^4 nodes are also given. The upper part of each contour comes from Iwama's algorithm (except the $t = 7$ point on the upper v contour comes from the pure literal rule). The rapidly sloping lower left part of each contour (except for the v contour) comes from the pure literal algorithm. The lower part of each contour comes from clause order backtracking. The unit clause algorithm is becoming best just as it is becoming to hard to compute the contour. On the v contour, it is best for $t > 135$. Starting at $t = 135$ the v contours for both clause order and unit clause backtracking are given. (Due to the small scale this extra contour looks like a thick place in the v contour). For the higher contours it is not best until after $t = 153$ (where the calculation for unit clause backtracking was stopped).

There are a number of algorithms for satisfiability and constraint satisfaction problems that appear to have promising average time performance but which appear hard to analyze (for example [14, 16]). The contours in this paper suggest useful places to measure the preformance of these algorithms.

Other analyses

The type of model for random clauses used in this paper is often called a random clause model. Both backtracking [2] and unit clause backtracking [18] have also been analysed with a fixed clause model. In this type of model each clause is a random selection of s literals.

Some analyses concern algorithms that *always* run in polynomial time and *often* solve the problem. Sometimes, however, they give up. Analyses of these algorithms give the conditions under the probability of finding solutions is high. In the random clause model, the algorithm that tries a random truth assignment solve can a large fraction of the problems in the region where most problems have solutions [7], i.e.

$$p > \frac{\ln t}{v}. \qquad (27)$$

(It takes a somewhat larger value of p for most problems to have solutions than it does to cause the average number of solutions per problem to be near one. Solutions occur in bunches, so a random problem with a solution in one branch of a search tree is more likely to have solutions in other branches [21].) The algorithm that says the problem has no solutions if it has an empty clause has a high probability of detecting the lack of solutions [7] if

$$p < \frac{\ln t}{2v}. \qquad (28)$$

The unit clause rule is often effective for values of p between the two bounds in eq. (27) and eq. (28) [8].

Probabilistic analyses with a fixed clause model have found conditions where many resolution proofs have exponential length [5] and where all search rearrangement backtracking algorithms often require exponential time [9]. Also backtracking where variables are selected from the shortest clause has been analyzed in this model [10].

Numerical techniques

The function associated with each figure was evaluated algebraically at $p = 0$ and $p = 1$. The values at intermediate points were found numerically. In those cases where the peak value was not at an end point, the peak was found using a combination of quadratic interpolation and bisection. At all times the program remembered the value of the function for three values of p, where the peak was known to lie between the two extreme values. Each new value of p was requireed to differ from the previous values by at least a factor of 1.0001 (except on the last step), and the search was stopped when the peak was located within this accuracy. Once the peak value was known, the contours were found by using similar techniques.

Some of the recurrences took a long time to solve. The recurrence for unit clause backtracking took time $O(t^4v)$ (the time would have been $O(vt^5)$ except that direct computation of the sum over j can be avoided), the one for clause order backtracking took time $O(tv^2)$, and that for the pure literal rule took time $O(t^2v)$. Therefore several additional techniques were used in most programs to speed up the finding of contours. The evaluations done to find one contour were sometimes useful for finding other contours. Also, in some cases, the calculation for one value of t resulted in values for all smaller values of t. For each t and for each result (contour or peak) the programs remembered the three points closest to the contour or peak (subject to the requirement that one point be on each side of the result). Once the contour was found for three values of t, additional points on the contour were computed by extrapolation. The extrapolation was checked by evaluating the function at the extrapolated point and at a point a factor of 1.0001 away. When the extrapolation was outside the permitted tolerance, the step size was reduced. When it was inside the tolerance, the estimated error was used to predict the next step size. In the case of the unit clause rule, the procedure was refined by extrapolating based on the ratio of the expected number of nodes to expected number of solutions. The expected number of solutions is easy to compute, and using this ratio permitted larger step step sizes. The extrapolation of contours resulting in the need to compute only about one twentieth of the t values for large t. Usually only two function evaluations were needed for each contour.

References

1. James Bitner and Edward Reingold, *Backtrack Programming Techniques*, Comm. ACM **18** (1975) pp 651–655.
2. Cynthia Brown and Paul Purdom, *An Average Time Analysis of Backtracking*, SIAM J. Comput. **10** (1981) pp 583–593.
3. Khaled Bugrara, Youfang Pan, Paul Purdom, *Exponential Average Time for the Pure Literal Rule*, SIAM J. Comput. **18** (1989) pp 409–418.
4. Khaled Bugrara and Paul Purdom, *One Clause at a Time Backtracking*, in prepartion.
5. Vašek Chvátal and Endre Szemerédi, *Many Hard Examples for Resolution*, JACM **35** (1988) pp 759–770.
6. Martin Davis, George Logeman, and Donald Loveland, *A Machine Program for Theorem Proving*, Comm. ACM **5** (1962) pp 394–397.
7. John Franco, *On the Probabilistic Performance of Algorithms for the Satisfiability Problem*, Information Processing Letters **23** (1986) pp 103–106.
8. John Franco, *Probabilistic Performance of a Heuristic for the Satisfiability Problem*, Discrete Applied Math. **22** (1988/1989) pp 35–51.
9. John Franco, *Search Rearrangement Backtracking Often Requires Exponential Time to Verify Unsatisfiability*, Indiana University Technical Report 210 (1987).
10. John Franco, *Probabilistic Analysis of a Generalization of the Unit-Clause Literal Selection Heuristic for the k-Satisfiability Problem*, Information Sciences (to appear).
11. John Franco, *On the Occurrence of Null Clauses in Random Instances of Satisfiability*, Indiana University Technical Report 291 (1989).
12. Allen Goldberg, *Average Case Complexity of the Satisfiability Problem*, Proc. Fourth Workshop on Automated Deduction (1979) pp 1–6.
13. Allen Goldberg, Paul Purdom, and Cynthia Brown, *Average Time Analysis of Simplified Davis-Putnam Procedures*, Information Processing Letters, **15** (1982) pp 72–75. Printer errors corrected in **16** (1983) p 213.
14. John Hooker, *Resolution vs. Cutting Plane Solution of Inference Problems: Some Computational Experience*, Operations Research Letters 7 (1988) pp 1–7.
15. Kazuo Iwama, *CNF Satisfiability Test by Counting and Polynomial Average Time*, SIAM J. Comput. **18** (1989) pp 385–391.
16. Burkhard Monien and Ewald Speckenmeyer, *Solving Satisfiability in Less Than 2^n Steps*, Discrete Applied Math. **10** (1985) pp 287–295.
17. Paul Purdom, *Search Rearrangement Backtracking and Polynomial Average Time*, Artificial Intelligence **21** (1983) pp 117–133.
18. Paul Purdom and Cynthia Brown, *An Analysis of Backtracking with Search Rearrangement*, SIAM J. Comput. **12** (1983) pp 717–733.
19. Paul Purdom and Cynthia Brown, *The Pure Literal Rule and Polynomial Average Time*, SIAM J. Comput. **14** (1985) pp 943–953.
20. Paul Purdom and Cynthia Brown, *Polynomial Average Time Satisfiability Problems*, Information Sciences **41** (1987) pp 23–42.
21. Ewald Speckenmeyer, Burkhard Monien, and Oliver Vornberger, *Superlinear Speedup for Parallel Backtracking*, Proc. of Supercomputing 87, Lecture Notes in Computer Science 297 pp 985–993.