An Empirical Comparison of Backtracking Algorithms

by

Cynthia A. Brown

and

Paul Walton Purdom, Jr.

Computer Science Department

Indiana University

Bloomington, Indiana 47405

TECHNICAL REPORT NO. 100

AN EMPIRICAL COMPARISON OF BACKTRACKING ALGORITHMS

CYNTHIA A. BROWN

AND

PAUL WALTON PURDOM, JR.

Revised August 1981

Research reported herein was supported in part by the National Science Foundation under grant number MCS 7906110.

An Empirical Comparison of Backtracking Algorithms

by

Cynthia A. Brown

and

Paul Walton Purdom, Jr.

Abstract. In this paper we report the results of experimental studies of zero-level, one-level, and two-level search rearrangement backtracking. We establish upper and lower limits for the size problem for which one-level backtracking is preferred over zero-level and two-level methods, thereby showing that the zero-level method is best for very small problems. The one-level method is best for moderate size problems, and the two-level method is best for extremely large problems. Together with our theoretical asymptotic formulas, these measurements provide a useful guide for selecting the best search rearrangement method for a particular problem.

Research reported herein was supported by the National Science Foundation under grant number MCS 7906110.

An Empirical Comparison of Backtracking Algorithms

1. Introduction

An important task of computer scientists is devising general algorithms that can be used to solve any problem from a large set of related problems. Such sets of problems can be divided into two classes, sometimes called "easy" and "hard". The easy sets are those for which each problem in the set can be solved within a time which is a polynomial function of the problem size. An example of such an easy set is the computation of the shortest path between two nodes in a graph. An individual problem in the set consists of a graph whose arcs have nonnegative labels, and a distinguished pair of vertices. There are well-known general methods for solving any problem in this set in a time proportional to the number of edges in the graph [7]. For naturally occurring easy problem sets the degree of the polynomial time bound is usually no greater than three, so rapid solution of large problems is possible.

A hard problem set is one for which the best known algorithm takes more than polynomial time for some sequence of problems in the set. (For some hard problem sets there are solution methods with small average time.) Many important hard problem sets are NP complete. Garey and Johnson [8] give a thorough discussion of the NP complete class and list many NP complete problem sets.

An examination of the problems listed by Garey and Johnson shows that most of them have a natural representation as a predicate of the form

$$P \equiv \bigwedge_{1 \le i \le m} R_i(v_1, \ldots, v_n) \qquad (1)$$

where each R is a relation that is simple in the sense that it depends on only a few of the variables, and each v has a finite number of possible values. Moreover, the work of Cook [5] and Karp [15] shows that solving any NP problem is equivalent to determining the satisfiability of a conjunctive normal form (CNF) predicate with three literals per term, where the size of the CNF formula is polynomially related to the size of the original problem. This means that any hard problem in NP can be represented in the form of (1), using simple relations for the R's. Fast general methods for solving problems of form (1) would be useful for any NP complete set. (See also [11].)

Problems of form (1) where each R depends on a small number of variables are called constrained labelling problems by some authors. Each R is then called a constraint, each v a unit, and each possible value for a unit is called a label. The problem is NP complete even when each R depends on only two variables and each variable has only three possible values [18], or when each R depends on three variables and each variable has two possible values [5].

Backtracking is a natural general method for solving problems of form (1) (as well as more difficult problems, such as alternating Turing machine computations [4], that have a similar structure). We have described our theoretical analyses of various backtracking algorithms elsewhere [2,19]. Here we report on empirical studies of these algorithms. The algorithms measured are the most efficient known simple methods for solving many moderate size problems from NP complete sets. The results of these studies, in conjunction with our theoretical work, provide practical criteria for selecting the most appropriate backtracking algorithm for a particular problem. Tree size estimation [16,17] can then be used to judge how long the chosen method will take.

The remainder of this paper is organized as follows. Section 2 indicates some of the types of backtracking that have been studied by us and other investigators and summarizes the current theoretical results on their average running time. Section 3 gives a more detailed description of the search rearrangement algorithms whose performance we measured. Section 4 discusses the model problem set on which the average performance of the algorithms was measured. Section 5 is a discussion of our experimental techniques, and the results are given in Section 6. Section 7 presents our conclusions.

2. Backtracking

Consider a problem that is expressed as a predicate P in form (1). A solution of the problem is a set of values $W_1,\ldots W_n$ for $w_1,\ldots,w_v$ that make P true. An <u>intermediate predicate</u> $P_j(w_1,\ldots,w_j)$ for P is a predicate such that $P_j(W_1,\ldots,W_j)$ is false only if P does not have a solution with $w_1 = W_1, \ldots, w_j = W_j$. (When P_j is true, P still may not have such a solution.) An obvious choice for P_j is the conjunction of all the relations R that depend only on $w_1,\ldots,w_j$. If the intermediate predicates are frequently false, they can be used to greatly increase the efficiency of a search for solutions to P, using simple backtracking. In simple backtracking, we begin with all the variables unset (they may be regarded as having a special value, "undefined," at this point.) Each variable in order beginning with the first is set to its first value. As variable w_j is set, the intermediate predicate P_j is tested. If the value of the predicate is false, the variable is set to its next value, until a value for which P_j is true is found. If the variable has no such value, it is returned to the list of unset variables (set to "undefined") and the previous variable is set to its next value. This continues recursively until all the values of variable w_1 have been tried.

In a problem with v binary variables, the number of potential solutions is 2^v, so an exhaustive search for solutions would take time exponential in v. Backtracking examines partial potential solutions as well as complete ones, so in the worst case it examines twice as many sets of values as the exhaustive search method. In practice, however, backtracking usually performs much better than exhaustive search.

In [2] we analyzed the behavior of simple backtracking over an NP complete set of problems consisting of conjunctive normal form formulas over v variables with s literals per term and v^α terms, $1<\alpha<s$. (The parameters s and α are arbitrary numbers; varying them varies the characteristics of the problems being considered.) The average time for such problems is $\exp \Theta(v^{\frac{s-\alpha}{s-1}})$ [2]. (To say that an item $g(v)$ is $\Theta(f(v))$ means that there exist positive constants C_1 and C_2 such that $C_1|f(v)| \leq g(v) \leq C_2|f(v)|$ for all values of v greater than some v_0. The related notation $O(f(v))$ means that there exists a positive constant C such that $g(v) \leq C|f(v)|$ for all values of v greater than some v_0.) Since exhaustive search requires time $\exp \Theta(v)$, and since $\frac{s-\alpha}{s-1}<1$, simple backtracking saves an exponential amount of time. (See also Haralick and Elliot [14] for a non-asymptotic analysis on a slightly different problem set). Figure 1 shows$_2$ an example of a backtrack tree generated by simple backtracking.

While simple backtracking represents a huge improvement over exhaustive search, it can still be extremely slow. This has led to the development of a number of methods for improving the efficiency of backtracking. These methods can be divided into two categories: <u>basic backtracking</u> and <u>predicate analysis backtracking</u>.

Basic backtracking methods derive all their information through the evaluation of intermediate predicates. One advantage of these methods is their generality. Once a program for a particular method is written, it can be adapted to any problem set by adding routines to evaluate the appropriate intermediate predicates.

A simple, effective basic backtracking technique was studied by Bitner and Reingold [1]. It involves testing each unset variable to find one with the fewest remaining values (the fewest values for which the intermediate predicate is not false) and introducing that variable next. Thus, instead of following a fixed search order, the order of variables is determined dynamically, and may vary from branch to branch of the backtrack tree. We analyzed the performance of a simple search rearrangement algorithm over the same problem set that we used for the analysis of simple backtracking [18]. The average time is $\exp \Theta(v^{\frac{s-\alpha-1}{s-2}})$ for $1<\alpha\leq\frac{s}{2}$ and $\exp O((\ln v)^{\frac{s-1}{s-2}} v^{\frac{s-\alpha-1}{s-2}})$ for $\frac{s}{2}<\alpha<s-1$. Thus, search rearrangement represents an exponential improvement over simple backtracking. To obtain more detailed information on the performance of search rearrangement we carried out extensive measurements on the behavior of three search rearrangement algorithms, including the analyzed algorithm. This paper reports the results of these measurements.

More complex basic backtracking algorithms are described by Purdom, Brown and Robertson [18]. In this paper we report preliminary measurements that indicate that the more complex algorithms are more efficient for solving large problems. The algorithms that Haralick et. al. develop using Ψ_P [11] can also be programmed as basic backtracking algorithms.

The second approach to improving backtracking involves a direct analysis of the structure of P. Such predicate analysis methods are probably significantly faster than the basic methods, although they can also be more difficult to use.

The most interesting of the predicate analysis methods is the Putnam-Davis procedure [6]. Goldberg [10] analyzed the average behavior of a simplified version of the Putnam-Davis procedure on an NP complete problem set. For predicates where there is a fixed probability that each literal appears in a given clause, it was shown that the Davis-Putnam procedure takes polynomial average time.

Some of the algorithms of Gaschnig [9] and of Haralick and coworkers [11,12,13,14] are also predicate analysis methods. We hope to study the performance of various predicate analysis methods over the problem set we have used for our other analyses in the near future.

3. One-level Search Rearrangement

The search rearrangement algorithm of Bitner and Reingold [1] examines each unset variable, and selects for introduction one with the fewest remaining values. A more sophisticated search rearrangement method would consider sets of unset variables of some predetermined size (say k), and introduce the variable which is the root of the smallest k-level subtree [18]. We call an algorithm that considers k element sets a k-level algorithm. Bitner and Reingold's method is thus a one-level algorithm and simple backtracking is 0-level.

There are several distinct one-level algorithms, which vary in the details of how ties are broken and in the order in which variables are tested, but whose general form is the same. We present the general form first, and then discuss the individual differences.

To use search rearrangement it is necessary to have, for each subset S^* of the set S of predicate variables, an intermediate predicate P_{S^*}. This predicate must be consistent with P in the sense that it is false only for sets of values that cannot be extended to a solution to P. If P is in the form of Eq. 1, the natural choice for P_{S^*} is the conjunction of all the relations that depend only on the variables in S*. In the following specifications let S′ be the set of variables whose value is "undefined" (the unset variables) and S″ the set of variables with defined values. Let w range over the predicate variables (units). We denote the value of predicate variable w by Value[w]. The set S″ is maintained as a stack. In this specification it is assumed that the predicate variables are Boolean; trivial changes are needed to accommodate a larger set of values.

One-Level Search Rearrangement

Step 1. (Initialize.) Set S″ to empty and S′ to S.

Step 2. (Solution?) If S′ is not empty, go to Step 3. Otherwise, the current values in Value constitute a solution. Go to Step 6.

Step 3. (Find best variable.) For each variable w in S′ do the rest of this step. (The order in which the variables are tested depends on the specific algorithm.) For both Value[w] ← false and Value[w] ← true, compute $P_{S'' \{w\}}$. If the result is false in both cases, exist the loop and go to Step 6. If the result is false in one and true in the other, remember the value that gives true, exist the loop and go to Step 5. If the result is true in both cases, continue with testing the next value of w.

<u>Step 4.</u> (Binary node.) Choose an element w of S'. (The method for selecting this element depends on the specific algorithm). Set $S'' \leftarrow S'' \ \{w\}$ and $S' \leftarrow S' - \{w\}$. Set Value[w] $\leftarrow$ false, mark w as binary, and go to Step 2.

<u>Step 5.</u> (Unary node.) Set Value[w] to the unique value that makes $P_{S'' \quad \{w\}}$ true. (This value is remembered from Step 3). Set $S'' \leftarrow S'' \ \{w\}$ and $S' \leftarrow S' - \{w\}$. Mark w as unary, and go to Step 2.

<u>Step 6.</u> (Next value.) If S'' is empty, stop. Otherwise, set $w \leftarrow \text{top}(S'')$. If w is marked as binary and Value[w] = false, set Value[w] $\leftarrow$ true and go to Step 2.

<u>Step 7.</u> (Backtrack.) Set $S'' \leftarrow S'' - \{w\}$, $S' \leftarrow S' \ \{w\}$, and go to Step 6.

In our implementations of one-level search rearrangement we found it convenient to encode the following four states into Value[w]: no value (w is an element of S'), unary false, binary false, and true. (There is no need to know whether a true node is unary or binary, since in either case it has no next value).

We measured the performance of three one-level search rearrangement algorithms. The specification given above for one-level search rearrangement left certain details involving the order of testing and selecting variables vague. Specifying how these details are to be handled completes a precise statement of a particular one-level search rearrangement algorithm. In the following paragraphs we provide these specifications for the three one-level algorithms whose performance we measured.

We call the best of these algorithms the Fast algorithm. In this version S' is treated as a stack with regard to insertions (deletions can be done anywhere). Thus in Step 7 all insertions are at the top, and in Step 3 the variables are tested from top to bottom. Normally the testing in Step 3 starts at the top of the stack, but when Step 3 is entered from Step 5 (via Step 2) the search starts just after the former position of the newly-discovered unary node and wraps around from bottom to top if necessary, continuing until all variables have been tested. At Step 4 the top element of S' is chosen.

Figure 2 shows a backtrack tree generated by the Fast algorithm. The moderate improvement in speed obtained by the Fast algorithm (in comparison with other one-level algorithms) is the result of two factors. First, starting the searches in Step 3 just after the point where a unary node was discovered often reduces the time to find another unary node, since the part of the stack that has not been tested recently is more likely to contain a unary node. Second, using a stack for S' permits Step 4 to select for a binary node a variable that was promising on a branch of the tree searched earlier. Since there is a slight correlation among the branches of the tree, this is better than selecting binary nodes in an arbitrary order.

The Analyzed one-level algorithm has those modifications to the Fast algorithm required to ensure that the number of binary nodes in a search tree generated by that algorithm will be exactly the same as the number of binary nodes generated by the one-level algorithm we analyzed in [19]. To accomplish this, we change the Fast algorithm so that at Step 4 the variable with smallest index is chosen (i.e., select v_i before v_j if $i<j$). The Analyzed algorithm tests for unary nodes in a different order

from the algorithm in [19] and usually requires fewer predicate evaluations, but it selects binary nodes in exactly the same order. Figure 3 shows a backtrack tree generated by the Analyzed algorithm.

The Simple one-level algorithm is the same as the Fast algorithm except that at Step 3 the search <u>always</u> begins at the top of the stack. This is the first algorithm we measured. Since it is slower than the Fast algorithm and since no theoretical work has been done on its performance, we did not study it as thoroughly as the other two algorithms.

Our measurements suggest that the asymptotic formula in [19] (roughly $\exp O(v^{\frac{s-\alpha-1}{s-2}})$) for the exponential part of the performance of the Analyzed algorithm applies to all the one-level algorithms considered in this paper. We have no mathematical proof of this, however.

4. <u>The Problem Set</u>

Since backtracking is a general strategy appliable to a wide variety of problems, it was necessary to choose a representative problem set on which to study its behavior. We performed our measurements on the same problem set used in our theoretical analyses: conjunctive normal form formulas with literals chosen from a set of v variables and their negations, with s literals per term and $t = v^{\alpha}$ terms. The values of the parameters v, s, and α determine the nature of the problem set.

Conjunctive normal form formulas are a convenient choice for several reasons. They have natural intermediate predicates: for each subset S^* of the variables, P_{S^*} is the conjunction of the clauses of P that contain only literals of the variables in S^*. For fixed $\alpha > 1$, fixed $s \geq 3$, and increasing v the problem set is NP complete. We have done extensive analytic studies of these problem sets [2,19], and they have many characteristics in common with problem sets for which backtracking is typically used. An interesting empirical question is the choice of the parameterization that is most representative of common problems. Choices that other investigators may wish to consider are $t = av$ or $s = \log v$. The analysis by Goldberg [10] of a simplified Putnam-Davis procedure uses a problem set similar to the one obtained by using a fixed value of t and $s = av$ for fixed a. We used the problem set parameterized by $t = v^{\alpha}$, s fixed for our measurements in order to be able to relate them to our theoretical work. We believe this is a realistic problem set.

5. Experimental Techniques

We studied the performance of the Fast, Analyzed, and Simple versions of one-level backtracking. For comparison we also made some measurements on zero-level (simple backtracking) and two-level algorithms [18]. We studied each algorithm for $s = 3$ and $s = 4$. Starting with $n = 1$ we set $v = n^2$ and $t = n^3$, and increased n in steps of 1. The upper limit on n was determined by the memory capacity of our computer ($n = 16$ for $s = 3$) or by our patience. Our 48 bit linear congruential random number generator, used to generate random literals, was always started at the same initial value, so the various algorithms were tested on exactly the same problems when the runs were of the same length. Individual runs were for 10, 100, 1000, 10000, or 2^{16} problems of each size. The results reported in Tables 1-5 were obtained by combining the data from one to three runs.

The measurements were done over several months using a dedicated TI980 minicomputer. In order to obtain the largest possible number of data points, a great deal of effort was devoted to making the program fast. For large problems nearly all the time was spent doing intermediate predicate evaluations, so optimization of predicate evaluations received particular attention. Backtracking proceeds by setting or changing one variable at a time. For our purposes a clause is true if at least one literal in it is true or unknown. We keep a list for each literal. Each clause is on the list for one of the literals that causes it to be true. Initially, when no variables are set, any literal in a clause causes it to be true. When a variable is set or changed, one literal becomes false. The clauses on the list for that literal are examined. If a clause contains other literals that cause it to be true then the clause is moved to the list for that literal. If all the literals in a clause are false, then the predicate is false, and it is time to backtrack. After backtracking the clause that caused the predicate to become false is again true, so it is not necessary to move it (or any other clauses remaining on its list) to a new list. Also, there is no need to move any other clauses when backtracking. Since a clause can be left on the list of <u>any</u> literal that makes it true, there is no need to restore other clauses to their former lists. This method of evaluating intermediate predicates allows us to evaluate large predicates rapidly. For $v = 256$, $t = 4096$, and $s = 3$, we estimate that only 35 microseconds are required for a typical evaluation.

We also performed other operations on the predicate, such as removing tautological clauses and repeated literals within clauses. Although our methods examine the detailed structure of the predicate to facilitate rapid evaluation, they do so in a way that is compatible with our aim of measuring the performance of basic backtracking algorithms.

We plan future studies of predicate analysis methods.

6. Results

Tables 1-5 give the results of the measurements. Table 1 shows the performance of the zero-level algorithm as measured and as calculated from the formulas in [2]. The theory for zero-level backtracking is quite complete; the measurements were done to provide a test for the statistical methods that were used on the other cases. For zero-level backtracking the number of nodes and the number of predicate evaluations are identical.

Tables 2, 3, and 4 give the measured performance of the Fast, Analyzed, and Simple algorithms. For the first two algorithms the number of binary nodes, unary nodes, and predicate evaluations were measured. For the Simple algorithm "total" nodes were measured, where total nodes are defined to be 2*(binary nodes + unary nodes) + 1. This definition corresponds to the one used in [18].

The number of predicate evaluations gives a good indication of the relative speed of the various methods. Even with our very rapid predicate evaluation methods, over 90% of the time was spent evaluating predicates. The measurements indicate that the Fast algorithm is indeed the fastest of the three. The differences, however, are not large, and they increase only slowly with problem size. For large problems there are many more unary nodes than binary nodes, and many more evaluations than nodes. For $s = 3$ and problems of 16 variables or more, the one-level methods are much faster than zero-level. With $s = 4$ the same holds true for problems with 25 or more variables.

Table 5 gives preliminary measurements on the two-level algorithm published in [18]. It appears to be possible to speed up this algorithm by at least a factor of two. We plan to publish more extensive measurements after we have investigated this possibility, and after we have developed a theoretical analysis for two-level algorithms. Even without improvement the two level algorithm is preferred for very large problems.

Our theoretical work [2,19] suggests that the number of binary nodes for all the algorithms we measured (except perhaps the two-level) is asymptotically of the form $a_1 v^{a_2} \exp(a_3 v^{a_4})$. For one-level backtracking the number of unary nodes must be between zero and v times the number of binary nodes, and the number of evaluations must be between one and v times the number of nodes (betweeen one and v^2 times the number of binary nodes). All four constants, a_1, a_2, a_3 and a_4, are needed to reliably predict the actual number of nodes for large values of v. We attempted to determine these constants by doing a least squares fit to our data using STEPIT [3]. We found that it is difficult to do extrapolations with data of the type we have gathered. It is presently impractical to obtain much data for problems larger than the ones we studied. The asymptotic formula is not valid for small problems. Statistical fluctuations make it difficult to achieve high accuracy for any data point. With a limited range of v and with inaccurate data many sets of values for a_1, a_2, a_3 and a_4 give nearly equally good fits: the collinearity problems are severe. For $s = 3$ our data is sufficient to determine two parameters when the other two are fixed. In a few cases it may be adequate for three, but in no case can all four be determined. For $s = 4$ we have not extended our measurements to large enough v to obtain reliable asymptotic fits.

The upper part of Table 6 shows the results of fitting a_3 and a_4

when a_1 and a_2 are held fixed. (They are set to the natural values of one and zero, respectively). For any of the algorithms the values of a_3 and a_4 should be the same whether binary nodes or evaluations are being measured, because these quantities are related polynomially. The variation in the fitted values gives one indication of the inaccuracy of the fitting process. For zero-level backtracking the theoretical value of a_4 is 0.75, so the fitted value is low by 0.15. For the Analyzed algorithm the theoretical value of a_4 is 0.50, so the fits are low by 0.11 to 0.14. Thus, although the fits to our quite extensive measurements give a rough indication of the upper exponent, the error is large compared to the range of values (zero to one) permitted by a naive analysis.

The middle part of Table 6 gives fits for a_2 and a_3 with a_4 set to its theoretical value and a_1 set to one. (For the Fast and Simple algorithms we use the theoretical value from the Analyzed algorithm). For zero-level backtracking the theoretical value of a_3 is 0.730, so the fitted value is low by 0.20. For the Analyzed algorithm theory gives $0.182 \leq a_3 \leq 0.320$ (for $\alpha > \frac{s}{2}$). All fits for the various one-level algorithms were in this range, although the values for the various cases spanned most of the range. The measurements were consistent with $a_3 = 0.320$, but not with $a_3 = 0.182$ (except for the Simple algorithm, for which we don't have much data).

We suspect that the true value of a_3 for one-level backtracking is equal to the upper limit. The lower part of Table 6 shows the results of fitting a_1 and a_2 with a_3 and a_4 set according to this conjecture. The resulting formulas are probably the most reliable for predicting the performance for large v. For zero-level backtracking the theoretical value of a_2 is 3/8, so the fit is low by 0.12.

The various fits for one-level algorithms suggest that the number of unary nodes is proportional to $v^{1/2}$* (binary nodes). Tables 2 and 3 indicate clearly that the Fast algorithm is better than the Analyzed, but Table 6 gives little indication of the functional form of the improvement. It is likely that some slowly growing function (such as log v) is involved.

Figure 4 is a graph comparing the performance of the zero-level, Fast, and two-level algorithms. Curves for the other one-level algorithms would be too close to the curve for the Fast for convenient display. The scales were chosen so that the function $a_1 v^{a_2} \exp(a_3 v^{a_4})$ would approach a straight line for large v.

7. Conclusions

The experimental studies reported in this paper supplement our theoretical results. We have established an upper and lower limit for the size problem for which the one-level search rearrangement algorithms are preferred over zero-level and two-level methods. We have shown that our theoretical results are compatible with the measured performance of the various one-level algorithms, including the Fast algorithm. We have shown that the Fast algorithm is faster than the Analyzed algorithm. Our measurements indicate that the number of unary nodes is roughly $v^{1/2}$ times the number of binary nodes and the number of evaluations is roughly v times the number of binary nodes. Most of these results would have been extremely difficult to obtain using theoretical techniques.

On the other hand, these experimental studies are quite limited in their ability to establish the asymptotic behavior of any method. When choosing or developing an algorithm for a practical problem, both the theoretical asymptotic studies and the results of measurements on problems of reasonable size should be considered. The results in this paper combined with those in [2,19] should be of help in selecting the most appropriate basic backtracking algorithm for a particular problem. The methods in [16,17] can then be used to estimate how long the chosen method will take to solve the problem.

In the future, we hope to perform theoretical analyses and practical measurements on two-level backtracking and on various predicate analysis techniques. Together with our present results, these will provide a rational basis for selecting an algorithm to solve a large problem.

3 Literals Per Term

Var.	Terms	Cal. Nodes	$4.72v^{3/8}\exp(.628v^{3/4})$	Measured Nodes	Cal. Solutions
1	1	3.00	9.	3.00	1.750
4	8	24.36	47.	24.29	5.498
9	27	227.08	281.	233.	13.915
16	64	1954.80	2029.	2131.	12.735
25	125	17457.30	17679.	17557.	1.891
36	216	183062.69	184482.	175871.	2.046×10^{-2}
49	343	2275493.18	2285200.	2075872.	7.232×10^{-6}
64	512	3.326×10^{7}	3.331×10^{7}	3.0×10^{7}	3.750×10^{-11}
81	729	5.675×10^{8}	5.669×10^{8}		1.280×10^{-18}
100	1000	1.122×10^{10}	1.119×10^{10}		1.291×10^{-28}
121	1331	2.555×10^{11}	2.542×10^{11}		1.727×10^{-41}
144	1728	6.665×10^{12}	6.619×10^{12}		1.375×10^{-57}
169	2197	1.981×10^{14}	1.964×10^{14}		2.923×10^{-77}
196	2744	6.683×10^{15}	6.613×10^{15}		7.447×10^{-101}
225	3375	2.547×10^{17}	2.515×10^{17}		1.021×10^{-128}
256	4096	1.092×10^{19}	1.077×10^{19}		3.378×10^{-161}

4 Literals Per Term

Var.	Terms	Cal. Nodes	$4.12v^{5/12}\exp(.730v^{5/6})$	Measured Nodes	Cal. Solutions
1	1	3.00	8.	3.00	1.875
4	8	28.38	75.	28.24	9.548
9	27	518.47	980.	523.	89.638
16	64	15168.57	20536.	15495.	1053.517
25	125	607176.45	680632.	617119.	10523.481
36	216	3.297×10^{7}	3.51×10^{7}	3.26×10^{7}	60656.248
49	343	2.608×10^{9}	2.76×10^{9}		136967.520
64	512	3.083×10^{11}	3.26×10^{11}		82264.657

Table 1. The calculated and measured performance of 0-Level Backtracking. For 0-Level Backtracking the number of nodes is equal to the number of predicate evaluations. The measurements consist of 110 runs for each case except the one variable case. There were 100 runs for the one variable case. The first column gives the number of variables in the predicates being measured, and the second column gives the number of terms. Column three gives the number of nodes predicted by the formula in [2] (calculated nodes). Column four shows the calculated result using the leading term of an asymptotic formula for the number of nodes [2]. Column five shows the average number of nodes obtained in the experimental measurements, and column six shows the average number of solutions per problem predicted by the formulas in [2].

3 Literals Per Term

Var.	Terms	Runs	Binary Nodes	Unary Nodes	Evaluations	Solutions
1	1	65536	0.7492±.0017	0.2508±.0017	4.4984±.0034	1.7492±.0017
4	8	76536	5.0480±.0081	4.1454±.0053	45.446±.046	5.5087±.0087
9	27	76536	16.697±.039	27.778±.041	273.70±.42	13.864±.039
16	64	76536	23.386±.082	81.30±.16	846.0±1.6	12.738±.074
25	125	76536	19.143±.056	140.76±.79	1800.4±3.5	1.900±.033
36	216	76536	23.849±.055	236.71±.49	3666.0±7.1	0.0237±.0041
49	343	76536	33.266±.076	392.7±.84	7125.±14.	0
64	512	76536	46.94±.11	641.3±1.4	13285.±27.	0
81	729	76536	66.17±.15	1028.0±2.3	23869.±49.	0
100	1000	76536	93.13±.21	1623.2±3.6	41634.±86.	0
121	1331	11000	132.35±.80	2558.±15.	71691.±395.	0
144	1728	11000	185.4±1.1	3937.±24.	119644.±675.	0
169	2197	11000	260.7±1.6	6047.±36.	197455.±1109.	0
196	2744	1000	365.1±8.7	9179.±217.	319591.±6943.	0
225	3375	1000	526.±10.	14240.±282.	527996.±9860.	0
256	4096	1000	708.±15.	20570.±438.	810792.±16297.	0

4 Literals Per Term

Var.	Terms	Runs	Binary Nodes	Unary Nodes	Evaluations	Solutions
1	1	1000	0.880±.010	0.120±.010	4.760±.021	1.880±.010
4	8	1000	8.754±.079	3.617±.048	62.96±.33	9.463±.082
9	27	1000	95.97±.99	69.08±.57	946.4±7.2	90.54±1.00
16	64	1000	1164.±16.	1242.±12.	14904.±152.	1050.±16.
25	125	1000	12464.±220.	17892.±222.	207279.±2681.	10821.±209.
36	216	1000	78108.±2021.	173078.±2570.	2043044.±31322.	61795.±1870.
49	343	1000	251045.±6325.	1252922.±17575.	16243331.±220762.	136193.±5321.

Table 2. The measured performance of the Fast 1-Level Backtracking Algorithm. The order of variables is controlled by a stack. A circular search is used to find the unary nodes.

3 Literals Per Term

Var.	Terms	Runs	Binary Nodes	Unary Nodes	Evaluations	Solutions
1	1	10000	0.750±.004	0.250±.004	4.500±.009	1.750±.004
4	8	11000	5.015±.022	4.148±.014	45.37±.12	5.495±.023
9	27	11000	16.59±.10	27.87±.11	278.5±1.1	13.73±.10
16	64	11000	23.15±.22	81.38±.44	876.6±4.5	12.59±.19
25	125	11000	19.09±.15	141.34±.81	1894.±10.	1.853±.082
36	216	11000	24.06±.15	236.8±1.3	3949.±21.	0.026±.013
49	343	11000	33.38±.21	390.1±2.3	7755.±44.	0
64	512	11000	47.51±.32	640.8±4.0	14730.±88.	0
81	729	11000	67.63±.47	1034.1±6.8	26953.±166.	0
100	1000	11000	94.27±.64	1618.±11.	46999.±287.	0
121	1331	11000	134.93±.95	2559.±17.	81931.±523.	0
144	1728	11000	189.5±1.5	3946.±29.	137759.±948.	0
169	2197	11000	267.8±2.0	6083.±44.	229475.±1554.	0
196	2744	1000	369.9±9.5	9098.±228.	367960.±8324.	0
225	3375	1000	518.±12.	13763.±324.	595712.±13221.	0
256	4096	1000	710.±18.	20172.±495.	923832.±21182.	0

4 Literals Per Term

Var.	Terms	Runs	Binary Nodes	Unary Nodes	Evaluations	Solutions
1	1	1000	0.880±.010	0.120±.010	4.760±.021	1.880±.011
4	8	1000	8.751±.079	3.624±.048	63.06±.33	9.463±.082
9	27	1000	95.81±.99	69.12±.59	965.4±7.4	90.54±1.00
16	64	1000	1166.±16.	1239.±13.	15611.±164.	1050.±16.
25	125	1000	12493.±221.	18026.±241.	222722.±2979.	10821.±209.
36	216	1000	78834.±2046.	178528.±3056.	2259275.±38904.	61795.±1870.
49	343	1000	255661.±6531.	1288497.±21337.	18727862.±297338.	136193.±5321.

Table 3. The measured performance of the Analyzed 1-Level Backtracking Algorithm. The variables have a fixed order. A circular search is used to find the unary nodes. For s = 3 the following is the calculated number of binary nodes: 1 variable, 0.75 binary nodes; 4 variables, 5.021 binary nodes; 9 variables, 16.75 binary nodes. This method was analyzed in [19].

3 Literals Per Term

Var.	Terms	Runs	Total Nodes	Evaluations	Solutions
1	1	100	3.00±0.	4.48±.09	1.74±.04
4	8	100	18.58±.47	43.3±1.2	5.28±.23
9	27	100	87.6±3.5	268.±10.	13.44±.79
16	64	100	179.±11.	780.±40.	8.76±2.7
25	125	100	344.±21.	2075.±113.	1.54±.52
36	216	100	637.±37.	4979.±253.	0.01±.01
49	343	100	927.±54.	9008.±476.	0
64	512	100	1615.±87.	18501.±899	0
81	729	100	2520.±139.	34088.±1712.	0
100	1000	100	3945.±262.	61052.±3715.	0
121	1331	100	6021.±415.	105477.±6632.	0
144	1728	100	9025.±486.	177123.±8641.	0
169	2197	100	12568.±683.	274116.±13853.	0
196	2744	100	22794.±2550.	523414.±51058.	0
225	3375	100	33549.±1898.	851603.±44483.	0
256	4096	100	44533.±3236.	1232146.±82147.	0

4 Literals Per Term

Var.	Terms	Runs	Total Nodes	Evaluations	Solutions
1	1	100	3.00±0	4.72±.08	1.87±.03
4	8	100	25.24±.37	61.50±.99	9.21±.26
9	27	100	328.9±8.4	941.±23.	88.8±3.1
16	64	100	4794.±181.	15036.±527.	1022.±56.
25	125	100	60007.±2320.	210916.±22339.	10442.±592.
36	216	100	513737.±27591.	2212728.±100828.	62927.±5764
49	343	100	3089249.±151617.	18415912.±796432.	148034.±19859.

Table 4. The measured performance of the Simple Level 1 Backtracking Algorithm. The order of the variables is controlled by a stack. A linear search is used to find the unary nodes. This is the method published in [18].

3 Literals Per Term

Var.	Terms	Runs	Total Nodes	Evaluations	Solutions
1	1	100	3.00±0	4.48±.09	1.74±.04
4	8	100	17.16±.05	150.4±6.0	5.28±.23
9	27	100	61.1±3.1	1418.5±7.4	13.44±.79
16	64	100	67.5±7.2	3435.±34.	8.76±2.7
25	125	100	51.5±3.8	6107.±54.	1.54±.52
36	216	100	57.3±3.1	11803.±76.	0.01±.01
49	343	100	70.9±3.2	21217.±117.	0
64	512	100	89.4±4.3	42143.±224.	0
81	729	100	115.1±5.1	63582.±350.	0
100	1000	100	128.1±5.9	70672.±474.	0
121	1331	100	161.6±5.4	120407.±597.	0
144	1728	100	182.4±7.0	166134.±842.	0
169	2197	100	234.0±8.3	314029.±1265.	0
196	2744	100	256.±10.	428160.±1816.	0
225	3375	100	297.±11.	567570.±2194.	0
256	4096	100	343.±11.	772527.±2660.	0

4 Literals Per Term

Var.	Terms	Runs	Total Nodes	Evaluations	Solutions
1	1	100	3.00±0	4.74±.007	1.87±.03
4	8	100	24.52±.04	219.5±4.2	9.21±.26
9	27	100	286.1±7.8	5580.±128.	88.8±3.1
16	64	100	3393.±151.	65270.±3476.	1022.±56.
25	125	100	34329.±1655.	728498.±30133.	10442.±592.
36	216	100	204709.±15998.	5927770.±304356.	62927.±5764.
49	343	100	516182.±55303.	27663834.±1509539.	148034.±19859.

Table 5. The measured performance of Level 2 Backtracking. This is the method published in [18]. Refinements can probably reduce the number of predicate evaluations by a factor of two.

Algorithm	Binary Nodes	Unary Nodes	Evaluations	Range
level zero			$\exp(1.41v^{.601})$ $\chi^2 = 2.16$	25-64
fast	$\exp(.763v^{.387})$ $\chi^2 = 10.5$	$\exp(1.75v^{.313})$ $\chi^2 = 5.40$	$\exp(3.20v^{.261})$ $\chi^2 = 5.23$	100-256
analyzed	$\exp(.748v^{.392})$ $\chi^2 = 0.907$	$\exp(1.75v^{.313})$ $\chi^2 = 0.873$	$\exp(3.24v^{.261})$ $\chi^2 = 1.70$	100-255
simple	$\exp(2.24v^{.282})$ $\chi^2 = 6.98$		$\exp(3.29v^{.262})$ $\chi^2 = 5.56$	100-256
level two	$\exp(2.29v^{.168})$ $\chi^2 = 0.985$		$\exp(5.43v^{.165})$ $\chi^2 = 0.220$	169-256
level zero			$v^{1.22}\exp(.528v^{3/4})$ $\chi^2 = 6.81$	4-64
fast	$v^{.397}\exp(2.71v^{1/2})$ $\chi^2 = 8.14$	$v^{1.05}\exp(.257v^{1/2})$ $\chi^2 = 6.88$	$v^{1.90}\exp(1.89v^{1/2})$ $\chi^2 = 23.2$	100-256
analyzed	$v^{.383}\exp(.278v^{1/2})$ $\chi^2 = 1.39$	$v^{1.05}\exp(.258v^{1/2})$ $\chi^2 = 0.836$	$v^{1.91}\exp(.195v^{1/2})$ $\chi^2 = 2.87$	100-256
simple	$v^{1.33}\exp(.209v^{1/2})$ $\chi^2 = 7.45$		$v^{1.92}\exp(.211v^{1/2})$ $\chi^2 = 6.42$	100-256
level zero			$6.88v^{.256}\exp(.628v^{3/4})$ $\chi^2 = 0.0435$	16-64
fast	$2.27v^{.113}\exp(.320v^{1/2})$ $\chi^2 = 15.5$	$3.29v^{.651}\exp(.320v^{1/2})$ $\chi^2 = 14.0$	$9.32v^{1.130}\exp(.320v^{1/2})$ $\chi^2 = 41.2$	49-256
analyzed	$2.19v^{.124}\exp(.320v^{1/2})$ $\chi^2 = 5.90$	$3.16v^{.661}\exp(.320v^{1/2})$ $\chi^2 = 5.13$	$8.52v^{1.176}\exp(.320v^{1/2})$ $\chi^2 = 12.1$	49-256
simple	$10.1v^{.594}\exp(.320v^{1/2})$ $\chi^2 = 7.04$		$8.26v^{1.233}\exp(.320v^{1/2})$ $\chi^2 = 6.45$	49-256

Table 6. Various least square fits to the measured number of binary nodes, unary nodes, and evaluations for various backtracking algorithms on problems with 3 literals per term. For the simple and level two algorithm the fits are for total nodes and evaluations. The first group of fits is of the form $\exp(a_1 v^{a_2})$. The second group is of the form $v^{a_1}\exp(a_2 v^{\theta})$ where $\theta = 3/4$ for level zero and $\theta = 1/2$ for level one. The third group is of the form $a_1 v^{a_2} \exp(\gamma v^{\theta})$ where $\gamma = 0.730$, $\theta = 3/4$ for level zero and $\gamma = 0.320103$, $\theta = 1/2$ for level one. Good fits for level one were not obtained with the form $a_1 v^{a_2} \exp(0.182 v^{1/2})$. The "range" column shows the range of number of variables (range of v) over which the fits were done. Small values of v were omitted because the asymptotic behavior does not have much influence on the function value at these points. The range was chosen to be as large as possible subject to keeping χ^2 reasonably small while using the same range for the various level one algorithms. For most cases the statistical accuracy of the data was not adequate for stable fits with 3 parameters. For $s = 4$ the range of the data did not permit reliable fits to asymptotic formulas.

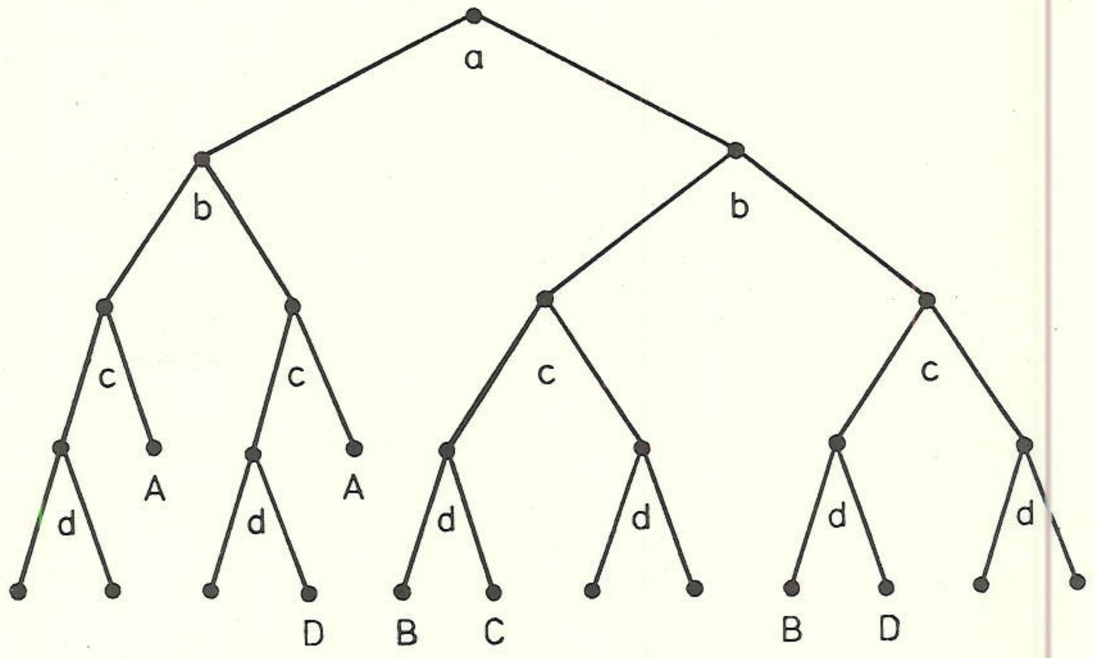


Figure 1. Zero-level backtrack tree for the predicate $A \wedge B \wedge C \wedge D$, where $A = a \vee \neg c$, $B = \neg a \vee c \vee d$, $C = \neg a \vee b \vee c \vee \neg d$, $D = \neg b \vee c \vee \neg d$. Each node where the predicate is false is labelled with a term that is false. Under each node the variable that is introduced into the search at that point is given. In each case, the false branch is to the left.

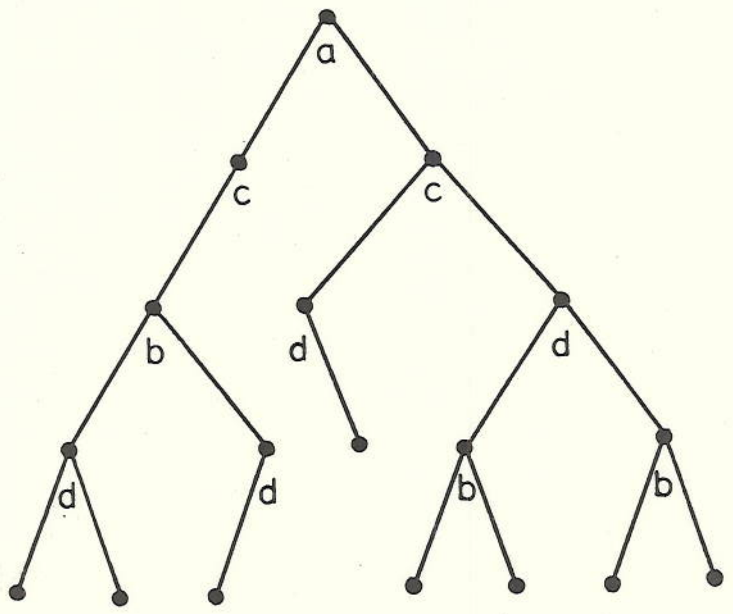


Figure 2. The backtrack tree produced by the fast one-level algorithm for the predicate from Fig. 1. This tree has 7 binary nodes, 3 unary nodes, 1 zero-degree node, and 7 solution nodes. Notice the order in which variables were selected on the right branch. Since c was a good first variable on the left branch, it was also selected for the right branch (all variables result in a binary node at this point). The algorithm also selected d before b on the right branch. The selection of c before b converted one binary node into a unary node and eliminated a zero-degree node. The selection of d before b did not lead to any savings.

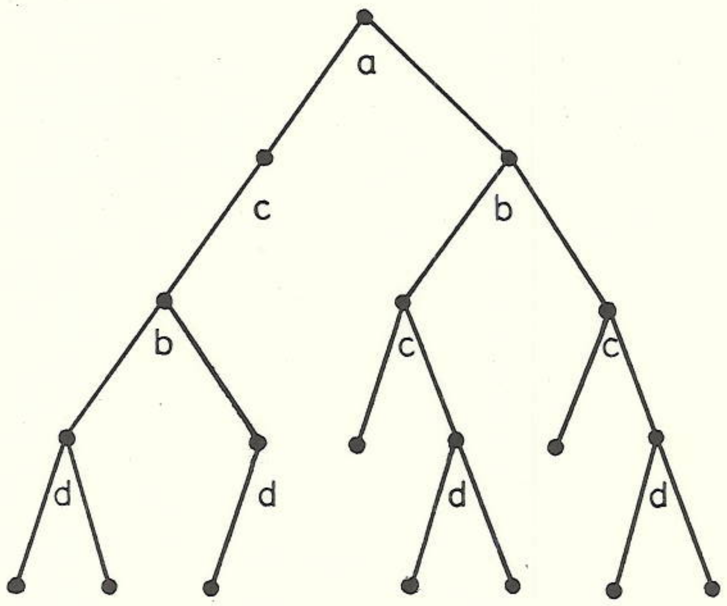
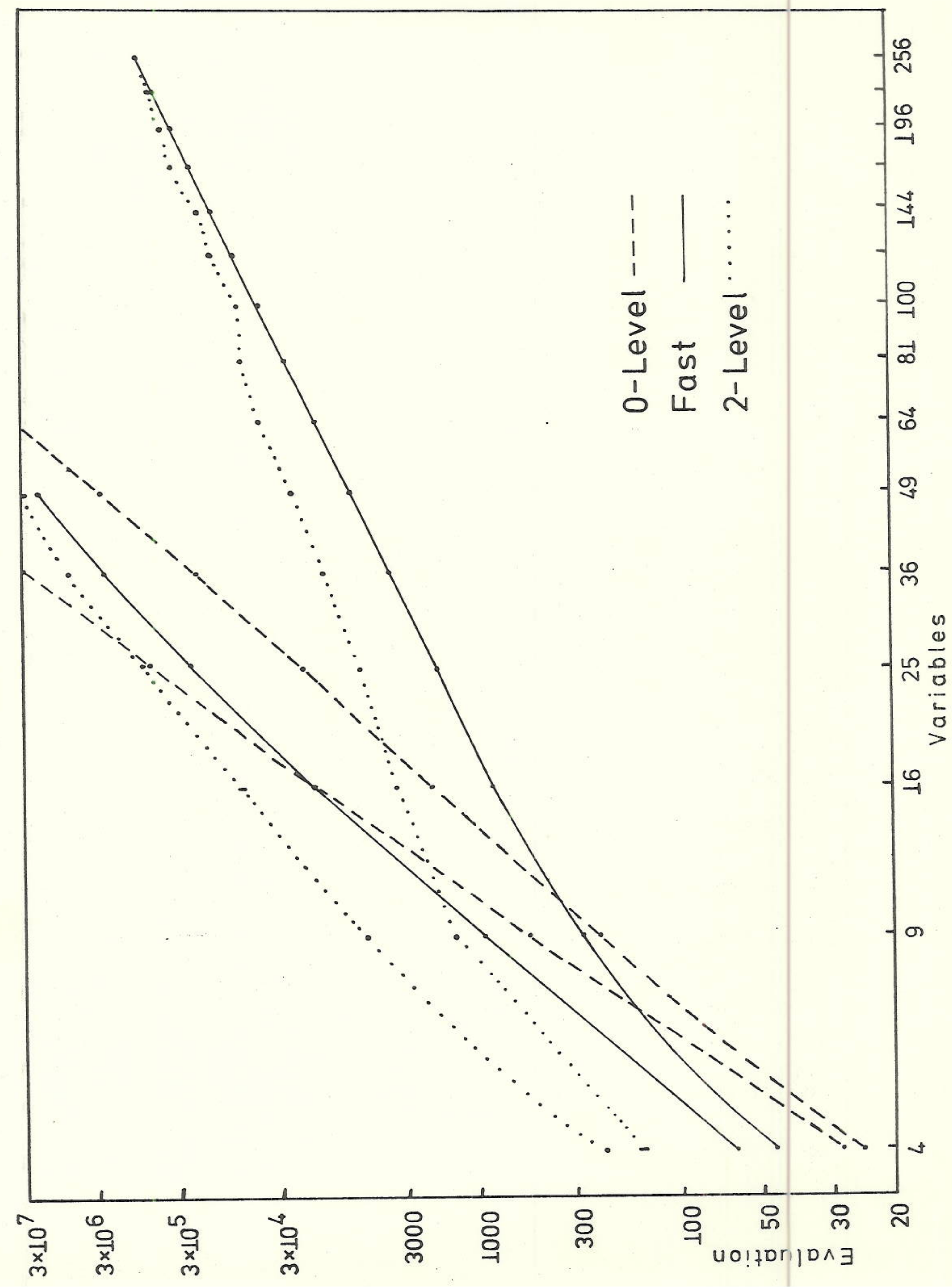


Figure 3. The backtrack tree produced by the analyzed one-level algorithm for the predicate from Fig. 1. This tree has 8 binary nodes, 2 unary nodes, 2 zero-degree nodes, and 7 solution nodes.



0-Level
Fast
2-Level
Variables
Evaluation
4
9
16
25
36
49
64
81
100
144
196
256
20
30
50
100
300
1000
3000
3×10^4
3×10^5
3×10^6
3×10^7

Figure 4. The average number of intermediate predicate evaluations used by various backtracking algorithms as a function of the number of variables. For each algorithm the lower curve is for problems with 3 literals per term and the upper curve is for 4 literals per term. On the evaluation axis linear distance is proportional to log evaluations. On the variable axis linear distance is proportional to log log variables. These scales result in curves of the form $a_1 v^{a_2} \exp(a_3 v^{a_4})$ approaching a straight line for large v.

References

1. James R. Bitner and Edward M. Reingold, "Backtrack programming techniques", Comm. ACM, 18 (1975) pp. 651-655.

2. Cynthia A. Brown and Paul Walton Purdom, Jr., "An Average Time Analysis of Backtracking", to appear SIAM J. Comput. (1981).

3. J. P. Chandler, "STEPIT", Quantum Chemistry Program Exchange, Department of Chemistry, Indiana University, Bloomington, Indiana 47405.

4. Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer, "Alternation", JACM 28 (1981) pp.114-133.

5. Stephen A. Cook, "The Complexity of Theorem-Proving Procedures", Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York (1971) p. 151-158.

6. Martin Davis and Hilary Putnam, "A Computing Procedure for Quantification Theory", JACM 7 (1960) pp. 201-215.

7. E. W. Dijkstra, "A Note on Two Problems in Connexion With Graphs", Numerische Mathematik 1 (1959), pp. 269-271.

8. Michael R. Garey and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, San Francisco (1979).

9. John Gaschnig, "Performance Measurement and Analysis of Certain Search Algorithms", PhD Thesis, Carnegie-Mellon (1979).

10. Allen Goldberg, Paul Purdom, and Cynthia A. Brown, "Average Time Analyses of Simplified Davis-Putnam Procedures", submitted to Information Processing Letters.

11. Robert M. Haralick, Larry S. Davis, Azriel Rosenfeld, and David L. Milgram, "Reduction Operations for Constraint Satisfaction", Information Sciences 14 (1978) pp. 199-219.

12. Robert M. Haralick and Linda G. Shapiro, "The Consistent Labeling Problem: Part I", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1 (1979) pp. 173-184.

13. Robert M. Haralick and Linda G. Shapiro, "The Consistent Labeling Problem: Part II", IEEE Transactions on Pattern Analysis and Machine Intelligence, 2 (1980) pp. 193-203.

14. Robert M. Haralick and Gordon L. Elliot, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Report from Virginia Polytechnic Institute, 1979.

15. Richard M. Karp, "Reducibility Among Combinatorial Problems", in R. E. Miller and J. W. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, New York, (1972) pp. 85-103.

16. Donald E. Knuth, "Estimating the Efficiency of Backtracking Programs", Math. Comput. 29 (1975) pp. 121-136.

17. Paul W. Purdom, "Tree Size by Partial Backtracking", SIAM J. Comput. 7 (1977) pp. 481-491.

18. Paul Walton Purdom, Jr., Cynthia A. Brown, and Edward L. Robertson, "Backtracking With Multi-Level Search Rearrangement", Acta Informatica 15 (1981) pp. 99-114.

19. Paul Walton Purdom, Jr. and Cynthia A. Brown, "An Analysis of Backtracking With Search Rearrangement", Indiana University Computer Science Technical Report No. 89, (1980).