How to Search Efficiently

by

Cynthia A. Brown and Paul Walton Purdom, Jr.

Computer Science Department

Indiana University

Bloomington, Indiana 47405

TECHNICAL REPORT No. 105

How to Search Efficiently

Cynthia A. Brown and Paul Walton Purdom, Jr.

March 1981

This material is based upon work supported by the National Science Foundation under Grant NSF MCS 7906110

How to Search Efficiently

by

Cynthia A. Brown

and

Paul Walton Purdom, Jr.

Abstract: The only technique available for solving many important problems is searching. Since searching can be extremely costly, it is important to identify features that improve the efficiency of search algorithms. We compute the efficiency of simple backtracking, Goldberg's simplification of the Putnam-Davis procedure, the combination of simple backtracking with Goldberg's method, and search rearrangement backtracking over two sets of random problems. A correct analysis of Goldberg's method shows that it requires exponential average time. The performances of the algorithms are compared and features that lead to efficient algorithms are identified.

This research was supported in part by the National Science Foundation under grant number MCS 7906110.

1. Introduction

Some problems can be solved by direct calculation in an efficient, straightforward way, but for many important classes of problems the best known method is a controlled search for solutions. Such searches, unfortunately, can consume extremely (exponentially) large amounts of time. Efforts to study and improve search methods are therefore of considerable practical importance.

By carefully analyzing a particular set of problems it may be possible to find problem-specific information that can be used to control the search. This can be an excellent approach; in some cases it has led to algorithms that avoid searching altogether. Often, however, after all problem-specific information has been used, excessive search time is still required.

Another approach to the problem is to study general search algorithms and to identify features that lead to an efficient search. The two approaches are complementary; the best algorithm for a particular problem is often obtained by combining problem-specific techniques with the best general search method.

Here we report the initial results of a systematic study of search methods. All methods have about the same worst case time (exponentially large). Also, techniques that lead to an improvement in average performance often result in a minor degradation of worst-case behavior, so a study of worst-case behavior can be misleading. The average time performance of these algorithms can be much better than the worst-case performance. For these reasons we have concentrated on average performance. Some of the methods we study lead to an exponential improvement in average search time.

All the search algorithms we study are special versions of a

generalized searching method. By specifing the details of the generalized method in various ways we obtain simple backtracking, search rearrangement backtracking [1,13], Goldberg's simplified Putnam-Davis procedure [7], Goldberg's modification combined with simple backtracking, the full Putnam-Davis procedure [4], and many other interesting algorithms. We present the results of analyzing the first four of these algorithms. Our analysis shows that Goldberg's algorithm requires exponential time (contrary to his claim), that ordinary backtracking is much more efficient than Goldberg's algorithm, and that search rearrangement backtracking can be much more efficient than ordinary backtracking.

We hope to analyze the full Putnam-Davis procedure. The points that remain to be analyzed are the effect of stopping the search when the first solution is found and the effect of the pure literal rule. Stopping at the first solution is clearly important for problem sets that have many solutions. Careful analysis is needed to determine the importance of the pure literal rule.

In the next section we describe the generalized search procedure and some of its variations. In Section 3 we give the model problem sets used in our average-time analyses. In Section 4 we present our results and discuss their signficance, and in Section 5 we consider some open problems.

2. Search Procedures

Searching is used to solve problems that can be expressed in the form

$$P \equiv \bigwedge_{1 \le i \le m} R_i(v_{a_{i1}}, \ldots, v_{a_{ij_i}}) \qquad (1)$$

where each R_i, $1 \le i \le m$, is a relation (a function whose value is true or false) over a small number of variables, and the variables $v_{a_{ik}}$ are taken from a set $\{v_i \mid 1 \le i \le n\}$ of variables, each of which is restricted to a finite set of values. A solution of the problem is an assignment of values to the variables that makes all of the relations true.

Any problem in the class NP can be expressed as a predicate in the form of Eq.(1). Many examples of such problems are given in [5]. We illustrate the encoding of problems in this form with the game of generalized instant insanity [14]. The game is played with m cubes. Each face of each cube is painted with some color. The object of the game is to form a stack of n cubes with each cube oriented so that each face of the stack consists of cube faces that have distinct colors. Each cube has twenty-four possible orientations. Since the order of the cubes in the stack is irrelevant, the problem is equivalent to the predicate

$$\bigwedge_{\substack{1 \le i,j \le m \\ i \ne j}} R_{ij}(o_i, o_j) \qquad (2)$$

where o_k is the orientation of cube k and $R_{ij}(o_i, o_j)$ is true if and only if, when cube i has orientation o_i and cube j has orientation o_j, the pair of cubes forms a legal stack of height two (all faces of the stack are made up of distinct colors).

The most obvious method of searching for solutions to a predicate in this form is to try all combinations of values of the variables. This sort of exhaustive search is prohibitively slow for large problems.

Fortunately, the special form of Eq. 1 permits three types of improvements. First, since each relation is defined over a small subset of the variables, a relation may become false as soon as all of its variables have been assigned values. In this case no extension of the current partial assignment of values to variables can be a solution. Such extensions need not be investigated. This is the idea behind backtracking.

A second improvement consists of looking for variables that should be assigned values early in the search. It is particularly helpful to find a variable all of whose values make a clause false (under the current partial assignment), or which has only one value which does not make a clause false. This is the basic idea of search rearrangement backtracking [1,13], and of the unit clause rule in the Putnam-Davis procedure [4].

A third approach involves looking for the values of a variable that are most likely to lead to a solution. In some cases there is a value that makes all relations which depend on the variable true. In that case only that one value of the variable requires consideration. This is the basis of the pure literal rule in the Putnam-Davis procedure.

These ideas have been used extensively to improve the average running time of search algorithms, although they can make the worst-case time somewhat worse. We are studying algorithms that use various combinations of the ideas. Since there are many possible search algorithms we first give a common general procedure and then present the details that distinguish the various methods. The procedure uses a stack to keep track of which variables have been set.

Generalized Search Procedure

1. [Initialize.] Set each variable to undefined. Set Stack to empty.
2. [Select variable.] Select as the current variable a variable that needs to be tested. If there is no such variable, go to Step 5. Push the current variable onto Stack and mark all of its values as untested.
3. [Select value.] For the current variable select an untested value that requires testing. If there is no such value, go to Step 6. Otherwise set the variable to that value and mark the value as tested.
4. [Test.] If some clause of the predicate is false under the present partial assignment of values to variables, go to Step 3. If all solutions are desired, go to Step 2.
5. [Solution.] The current assignment of values to variables is a solution (any remaining unassigned variables may take on any of their values). If only one solution is desired, stop. If all solutions are desired, go to step 3.
6. [Back up.] Set the current variable to undefined. Pop Stack. The new current variable is at the top of Stack. If Stack is empty, stop. Otherwise go to Step 3.

Many search algorithms simplify the predicate as they search. For example, in the Putnam-Davis procedure clauses that become true are dropped. The dropped clauses are restored when the algorithm backs up.

The algorithms that we have analyzed search for all solutions; they never stop at Step 5. In the future, unless otherwise stated, we will assume that Step 5 always goes to Step 3. For these algorithms the order in which values are tested at Step 3 is immaterial, since all values must

eventually be tested.

Simple backtracking is obtained from the generalized search procedure by selecting the variables in a fixed order at Step 2 and the values in a fixed order at Step 3. Every variable and every value requires testing. In search rearrangement backtracking [1] a simple test is used to select a variable at Step 2. Each value of each variable is tested (using the same test used in Step 4), and the variable for which the fewest values pass the test is selected. More sophisticated search rearrangement algorithms [13] test combinations of values.

In the Putnam-Davis procedure variables are selected by examining the predicate directly rather than by testing the values of variables. This method of selection is more powerful, but it is also more difficult to program and requires more knowledge of the internal structure of the clauses.

The original Putnam-Davis procedure was designed for predicates in conjunctive normal form. To describe a more general procedure we first give some (nonstandard) definitions. A relation R is a unit clause if, under the current partial assignment of values to variables, there is an unset variable v such that R is false for every value but one of v. We call v the associated variable of the unit clause. A variable v is relevant if there is a relation R whose value under the current partial assignment depends on v. A variable v is associated with a pure literal if under the current partial assignment there is some value x of v such that when v is assigned value x every R that depends on v is true. The value x is called a safe value.

The generalized Putnam-Davis procedure is our general search procedure with the following modifications. In Step 2, if possible,

select a variable associated with a unit clause. Otherwise, if possible, select a variable associated with a pure literal. Only one safe value is tested. If there are no variables associated with pure literals or unit clauses, then select any relevant variable. If there are no relevant variables, go to Step 5. The Putnam-Davis procedure stops when the first solution is found.

Goldberg [7] considered an extremely simplified version of the Putnam-Davis procedure. He did not advocate its use, but merely developed it to obtain an upper bound on the performance of the original procedure. In his version variables are selected in fixed order at Step 2, as long as some variable is relevant (the selected variable is not necessarily relevant). All values are tested at Step 3, and no tests are done at Step 4 (i.e. the test gives the result true) unless all relevant variables have been assigned values. The procedure searches for all solutions. The procedure is like backtracking except that it backs up when all the R_i are true instead of when one of them is false. Goldberg claimed that his algorithm ran in polynomial time. Unfortunately there is an error in his analysis [12], and his algorithm is inefficient. His approach to the analysis, however, is quite interesting.

3. Random Problems

To do an average time analysis it is necessary to select a set of representative problems and a probability distribution over the set. For backtracking it is not obvious what a "typical" problem is. In this paper we consider two types of random problem sets. Both are instances of conjunctive normal form predicates: that is, each relation is a disjunction (logical or) of a set of literals, where a literal is a variable or its negation. The variables can have the values true and

false. For each type of problem set we give a method of forming a random clause; a random predicate is then the conjunction of t random clauses selected independently (thus, a random predicate may happen to contain two copies of the same clause).

In the first method of constructing random clauses each clause has s literals for some fixed s. The s literals are independently selected from the $2v$ possible literals. This method was used in our earlier analyses of backtracking algorithms [2,11].

In the second method each literal has probablity p of being in the clause for some fixed p. This method was used by Goldberg [7]. The two models are roughly equivalent when the parameters are set so that $p \approx \frac{s}{2v}$.

We follow the original papers in computing "running time" in the two models. In our model we assume that the "running time" is equal to the number of binary nodes in the search tree. The actual running time increases more rapidly (by a factor of approximately v [3]) but this error is unimportant compared to the exponential differences in the average running times of the various search algorithms.

In Goldberg's model we assume that the time to process a node is equal to avt, where a is a constant, v is the number of unset variables, and t is the number of terms that are still being considered. Both unary and binary nodes are counted.

4. Results

Table 1 summarizes the exact results for the average "running time" of various search algorithms. A sketch of the analyses that led to these results is given in Appendix 1. Most of the exact results are not in

closed form, so it is difficult to understand their significance. An asymptotic analysis makes these results easier to interpret. To obtain an interesting asymptotic analysis, however, careful consideration must be given to how s, p, and t should vary as v increases. We believe that keeping s fixed, letting $p = a/v$ for fixed a, and letting $t = bv$ for fixed b gives results similar to those for many interesting realistic problems. This keeps the individual terms small while letting the number of terms increase with v. We also consider $t = v^{\alpha}$ for fixed $\alpha>1$. The first choice for t produces problems where the number of constraints increases proportionately to the number of variables. The second choice produces problems where the number of constraints increases more rapidly.

The first step in most of the analyses is to determine which level in the backtrack tree (value of i) has the most nodes in it. The approximate value of i for each case is given in Table 2. The level with the most nodes is always between zero and v. When the entry in Table 2 is less than zero the true value is zero and when the entry is greater than v the true value is v. The following analyses are valid only for values of the parameters that cause the entry in Table 2 to have a value between zero and v.

Table 3 gives the approximate value of the logarithm of the average "running time" for each algorithm. Sketches of the derivations are given in Appendix 3. The two models generate problems with the same number of solutions when $a = (\ln 2)\, s$. Usually the form of the answers is the same for the two models, but Goldberg's model generates problems that are much easier to solve by backtracking when $t = v^{\alpha}$ (for $1<\alpha<s$). The results show that Goldberg's modification of the Putnam-Davis procedure is not helpful when $p \to 0$. A little thought suggests that

Goldberg's method is less helpful than stopping the search when the first solution is found, because both approaches need a solution before they can save any time. Comparing the results for simple backtracking with those for search rearrangement on our model with $t = v^a$ shows that search rearrangement saves about as much time as reducing the size of the clauses by one literal. This exponential improvement indicates that search rearrangement can be much more effective than simple backtracking. Further analysis is needed to determine how search rearrangement behaves for $t = bv$.

Table 4 shows the results of setting $s = 3$, $a = 3 \ln 2 \approx 2.08$, and $b = (\ln 2)(\ln (1-2^{-s}))^{-1} \approx 5.19$ in the formulas from Table 3. These values for the parameters lead to an interesting set of difficult problems where the typical problem has about one solution regardless of the size of the problem, as is often the case for realistic difficult problems. The values in the table demonstrate clearly the exponential improvement that can result from using simple backtracking.

5. Open Problems

The most straightforward direction for future work is completion of the analysis of search rearrangement backtracking. The blank entries in Table 3 for level one backtracking can be filled in by completing calculations similar to those which led to the first entry. The multi-level backtracking algorithms appear to be even more efficient for large problems [3], but analyzing their performance is difficult.

Backtracking algorithms are easy to use: once the general search algorithm has been coded the user need only provide the routine to test partial solutions. Methods like the Putnam-Davis procedure that

manipulate the predicate require more programming effort, but they may also be much more powerful. Analyses are needed to determine whether this is the case.

The Putnam-Davis procedure can be viewed as a combination of 1) backtracking, 2) unit clause selection (one level search rearrangement), 3) noticing when there are no terms (the Goldberg rule), 4) pure literal selection, and 5) stopping at the first solution. The techniques of [11,12] and of this paper are adequate to analyze an algorithm with the first three parts. To analyze the first four parts appears to be much more difficult. Our preliminary attempts suggest that the pure literal rule is not very important (for much the same reasons that the Goldberg rule is not), but without a precise analysis it is hard to be sure. Stopping at the first solution is only important with problems that have many solutions; for them it is very important. Analysis of the effect of this rule is difficult because solutions tend to occur in clusters.

There may be algorithms that are both simpler to analyze and more powerful than the Putnam-Davis procedure. One weakness of the Putnam-Davis procedure is that it does not have any guidance concerning which variable to select in cases where the pure literal rule and the unit clause rule do not apply. A good technique is to select a variable from the shortest remaing clause. Other interesting techniques for modifying backtracking have been proposed [6,8,9,10].

Another area where progress is possible is in the use of rules to manipulate the predicate. Subsumption can be combined with the Putnam-Davis procedure. Substitution (the key to the powerful Gaussian elimination method for solving equations) can also be used.

6. Conclusions

Many of the techniques used in this paper were developed in earlier work [2,7,11]. Each of the original papers analyzes one algorithm on one model. Here we apply the techniques to a variety of algorithms and models to determine how important various features are to efficient searching. A large number of ideas have been suggested for improving the efficiency of searching. If they were all combined the result would be a large, complex program, containing many parts that made little or no contribution to its efficiency. Analysis of average running time is a powerful technique for determining the value of proposed improvements.

Appendix 1

Brief derivations for the results in Table 1 are given here. The referenced papers give a fuller explanation of the techniques.

1. Number of solutions, our model: See [2].

2. Number of solutions, Goldberg's model: Follow the method of [2] using probabilities instead of counts. Use $E_p = 1$, $F_p(i) = (1-p)^{2v-i}$, and $Q(E-F_p(i),t) = (E-F_p(i))^t$.

3. Simple backtracking, our model: See [2].

4. Simple backtracking, Goldberg's model: Follow [2] as in part 2 above. Introduce the factor $a(v-i)t$ to allow for the time to process a node.

5. Goldberg Putnam-Davis, our model: Follow [2] using probabilities, with $E_p=1$, $F_s(i) = 1 - (1 - \frac{i}{2v})^s$, and $Q(E-F_s(i),t) = (E-F_s(i))^t$.

6. Goldberg Putnam-Davis, Goldberg's model: See [12].

7. Simple backtracking with Goldberg Putnam-Davis, our model: The probability that no term on level i has every literal false is $(1-(\frac{i}{2v})^s)^t$. The probability that all terms are true on level i is $(1-(1 - \frac{i}{2v})^s)^t$. Since we never have both conditions at once, the probability that a node on level i requires expanding is $(1-(\frac{i}{2v})^s)^t - (1-(1 - \frac{i}{2v})^s)^t$. Proceed as in [2].

8. Simple backtracking with Goldberg Putnam-Davis, Goldberg's model: Following [12], let $A(t,v)$ be the running time. Then

$$A(t,v) = avt + 2(1-(1-p)^{2v})^t \sum_i \binom{t}{i} p^i (1-p)^{t-i} A(t-i,v-1) \qquad (3)$$

Define the generating function

$$G_v(z) = \sum_{t\geq 0} A(t,v)\frac{z^t}{t!} \qquad (4)$$

Multiplying (3) by $z^t/t!$ and summing over z leads to

$$G_v(z) = avze^z + 2e^{(1-(1-p)^{2v}))pz}\, G_{v-1}((1-(1-p)^{2v})(1-p)z) \qquad (5)$$

with $G_0(z) = 0$.

Solving gives

$$G_v(z) = avze^z + \sum_{1\le i\le v} 2^i a(v-i)H_i e^{(1-(1-p)^{2v})z} \qquad (6)$$

with $$H_i = \prod_{1\le j<i} (1-(1-p)^{2(v-j)})(1-p)^i. \qquad (7)$$

Expanding $G_v(z)$ in a power series leads to the answer in Table 1.

A simpler approximate analysis can be obtained by ignoring the time required to process predicates that contain a false term. This approach gives

$$A(t,v) = avt(1-(1-p)^{2v})^t + 2\sum_i \binom{t}{i} p^i (1-p)^{t-i} A(t-i,v-1) \qquad (8)$$

The equation for the generating function is

$$G_v(z) = av(1-(1-p)^{2v})ze^{(1-(1-p)^{2v})z} + 2e^{pz}\, G_{v-1}((1-p)z) \qquad (9)$$

The solution for $A(t,v)$ is

$$A(t,v) = at \sum_{0\le i\le v} [2(1-p)]^i\, (v-i)(1-(1-p)^{2v-i})^{t-1} \qquad (10)$$

This approximation is used for our later work. The resulting error is insignificant.

9. Level one backtracking, our model: See [11].

10. Level one backtracking, Goldberg's model: Modify the analysis in [11] in a way similar to that used above to analyze simple backtracking with Goldberg's model.

Appendix 2

Brief derivations of the results in Table 2 are given. The main steps in finding the level with the maximum term are to take the

logarithm of the term being summed over, take its derivative with respect to i, set the derivative to zero, and solve for i.

1. Simple backtracking, our model:

$$i \ln 2 + t \ln \left(1-\left(\frac{i-1}{2v}\right)^s\right) = 0 \tag{11}$$

This gives [2] $i \approx 2v\left(\frac{2v \ln 2}{t(s-1)}\right)^{\frac{1}{s-1}}$ (12)

2. Simple backtracking, Goldberg's model:

$$\ln 2 - \frac{1}{v-i} + t\,\frac{(1-p)^{2v-i+1} \ln (1-p)}{1-(1-p)^{2v-i+1}} = 0 \tag{13}$$

with $p = a/v$, we get

$$i \approx v\left(2 - \frac{1}{a} \ln \left(1 + \frac{at}{v \ln 2}\right)\right) \tag{14}$$

3. Goldberg Putnam-Davis procedure, our model:

For $i \leq v$, $2^i\left[1-\left(1-\left(1-\frac{i-1}{2v}\right)^s\right)\right]^t$ is an increasing function.

4. Goldberg Putnam-Davis procedure, Goldberg's model:

The result is implied by the entry in Table 3, which was obtained directly.

5. Simple backtracking with Goldberg Putnam-Davis procedure, our model:

$$\ln 2 = \frac{st\left(1-\left(\frac{i-1}{2v}\right)^s\right)^{t-1}\left(\frac{i-1}{2v}\right)^{s-1} + st\left(1-\left(1-\frac{i-1}{2v}\right)^s\right)^{t-1}\left(1-\frac{i-1}{2v}\right)^{s-1}}{2v\left[\left(1-\left(\frac{i-1}{2v}\right)^s\right)^t - \left(1-\left(1-\frac{i-1}{2v}\right)^s\right)^t\right]} \tag{15}$$

The solution is approximately the same as with simple backtracking.

6. Simple backtracking with Goldberg Putnam-Davis procedure, Goldberg's model:

$$0 = \ln 2 - \frac{1}{v-i} + \frac{2(1-p)^{2i} \ln (1-p)}{(1-p)^{2i} - (1-p)^{2v}} - \frac{(1-p)^i \ln (1-p)}{(1-p)^i - (1-p)^{2v}} + t\,\frac{(1-p)^{2v-i} \ln (1-p)}{1-(1-p)^{2v-i}} \tag{16}$$

The solution is approximately the same as with simple backtracking.

7. Search rearrangement backtraking, our model: See [11].

Appendix 3

A brief description of the method of calculation for the entries in Table 3 is given. See [11] for the search rearrangement analysis. The entry for the Goldberg Putnam-Davis algorithm with Goldberg's model is computed in [12]. The "less than $\ln v$" entries are obtained by summing v items, each of which is no bigger than 1.

In all other cases the logarithm of the "running time" is $\ln$ (biggest term) + $O(\ln v)$ since it is the sum of v terms, all of which are between 0 and (biggest term). The value of i from Table 2 is plugged into the summand for the corresponding entry in Table 1. Then p is replaced with a/v and t is replaced with bv or v^{α}. The limit as $v \to \infty$ is computed. We use power series for $\ln(1+x/v)$, $\lim_{v\to\infty} (1-1/v)^{cv} = e^{-c}$, and L'Hospital's rule as needed.

Table 1

Case	Our Model	Goldberg's Model
Solutions	$2^v\,(1 - F(v))^t$ $F(i) = (\frac{i}{2v})^s$	$F(i) = (1 - p)^{2v-i}$
Simple Backtracking (Level 0)	$R(0) + \sum_{1 \le i \le v} 2^i\, R(i)\, (1 - F(i-1))^t$ $R(i) = 1$	$R(i) = a(v-i)t$
Goldberg Putnam-Davis	$1 + \sum_{1 \le i \le v} 2^i\, [1 - (1 - \frac{i-1}{2v})^s)^t]$	$at\frac{[2(1-p)]^{v+1} - 2(1-p) + (2p-1)v}{(2p-1)^2}$
Simple Backtracking with Goldberg Putnam-Davis	$1 + \sum_{1 \le i \le v} 2^i\, [(1 - F(i-1))^t - (1 - (1 - \frac{i-1}{2v})^s)^t]$	$at\left[v + \sum_{1 \le i \le v} \frac{2^i\,(v-i)\,H_i}{(1 - (1-p)^{2v})t}\right]$ where $H_i = \prod_{1 \le j < i} (1-(1-p)^{2(v-j)})(1-p)^i$
Search Rearrangement Backtracking (Level 1)	See [11].	Modify the results in [11].

Table 1. Exact formulas for the number of solutions and for the "running time" of various searching algorithms for two models of random problems.

Table 2

Algorithm	Our Model	Goldberg's Model
Simple Backtracking	$2\,(\frac{\ln 2}{s-1})^{\frac{1}{s-1}}\, v^{\frac{s-\alpha}{s-1}}$	0
	$2\,(\frac{\ln 2}{b(s-1)})^{\frac{1}{s-1}}\, v$	$(2 - \frac{1}{a} \ln (1 + \frac{ab}{\ln 2}))v$
Goldberg Putnam-Davis	v	v
	v	v
Simple Backtracking with Goldberg Putnam-Davis	$2\,(\frac{\ln 2}{s-1})^{\frac{1}{s-1}}\, v^{\frac{s-\alpha}{s-1}}$	0
	$2\,(\frac{\ln 2}{b(s-1)})^{\frac{1}{s-1}}\, v$	$(2 - \frac{1}{a} \ln (1 + \frac{ab}{\ln 2}))v$
Search Rearrangement	$O(v^{\frac{s-\alpha-1}{s-2}})$	

Table 2. The approximate level in the backtrack tree that makes the largest contribution to the running time as $v \to \infty$, with s constant, $p = a/v$, and $t = v^{\alpha}$ (upper box) or $t = bv$ (lower box). Replace values by the closer of 0 and v, when the table entry is outside of the range 0 to v.

Table 3

Case	Our Model	Goldberg's Model
Solutions	$\frac{v^{\alpha} \ln (1-2^{-s})}{(\ln 2 + b \ln (1-2^{-s}))\, v}$	$\frac{v^{\alpha} \ln (1-e^{-a})}{(\ln 2 + b \ln (1-e^{-a}))\, v}$
Simple	$(s-1)(\frac{2 \ln 2}{s})^{\frac{s}{s-1}} v^{\frac{s-\alpha}{s-1}}$	less than $\ln v$ when $t = v^{\alpha}$
Backtracking	$(2(\frac{2 \ln 2}{b(s-1)})^{\frac{1}{s-1}} \ln 2 + b \ln (1-(\frac{2 \ln 2}{b(s-1)})^{\frac{s}{s-1}}))\, v$	$[2 \ln 2 - \frac{\ln 2}{a} \ln (1 + \frac{ab}{\ln 2}) - b \ln (1 + \frac{\ln 2}{ab})]\, v$
Goldberg Putnam-Davis	$\frac{v \ln 2}{v \ln 2}$	$\frac{v \ln 2}{v \ln 2}$
Simple	$(s-1)(\frac{2 \ln 2}{s})^{\frac{s}{s-1}} v^{\frac{s-\alpha}{s-1}}$	less than $\ln v$ when $t = v^{\alpha}$
Backtracking with Goldberg Putnam-Davis	$(2(\frac{2 \ln 2}{b(s-1)})^{\frac{1}{s-1}} \ln 2 + b \ln (1-(\frac{2 \ln 2}{b(s-1)})^{\frac{s}{s-1}}))\, v$	$[2 \ln 2 - \frac{\ln 2}{a} \ln (1 + \frac{ab}{\ln 2}) - b \ln (1 + \frac{\ln 2}{ab})]\, v$
Search Rearrangement Backtracking (Level 1)	$O(v^{\frac{s-\alpha-1}{s-2}})$	

Table 3. The approximate value of the logarithm (base e) of the number of solutions and ''running time''. The leading term in an asymptotic expansion is given.

Table 4

Case	Our Model	Goldberg's Model
Solutions	≈ 0	≈ 0
Simple Backtracking	0.247 v	0.456 v
Goldberg Putnam-Davis	0.693 v	0.693 v
Simple Backtracking with Goldberg Putman-Davis	0.247 v	0.456 v

Table 4. The approximate value of the logarithm of the ''running time" for large v when $s = 3$, $p = (3 \ln 2)/v \approx 2.08/v$, $t = (\ln 2)(\ln (1 - 2^{-s}))^{-1} v \approx 5.19 v$.

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, SIAM J. Comp., to appear.

3. Cynthia A. Brown and Paul Walton Purdom, Jr., An Empirical Comparison of Backtracking Algorithms, Indiana University Computer Science Department Technical Report No. 100 (1981).

4. Martin Davis and Hilary Putnam, A Computing Procedure for Quantification Theory, J. ACM 7 (1960), pp. 201-215.

5. Michael R. Garey and David S. Johnson, Computers and Intractability, W. H. Freeman (1979).

6. John Gaschnig, Performance Measure and Analysis of Certain Search Algorithms, Ph.D. Thesis, Carnegie-Mellon University (1979).

7. Allen Goldberg, Average Case Complexity of the Satisfiability Problem, Proceedings of the Fourth Workshop on Automated Deduction (1979), pp. 1-6.

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

9. Robert M. Haralick and Linda G. Shapiro, The Consistent Labelling Problem: Part I, IEEE Transactions on Pattern Analysis and Machine Intelligence 1 (1979), pp. 199-219, and The Consistent Labelling Problem: Part II, IEEE Transactions on Pattern Analysis and Machine Intelligence 2 (1980), pp. 193-203.

10. Alan K. Mackworth, Consistency in Networks of Relations, Artificial Intelligence 8 (1977), pp. 99-118.

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

12. Paul Walton Purdom, Jr. and Cynthia A. Brown, The Goldberg Putnam-Davis Procedure Takes Exponential Average Time, Indiana University Computer Science Department Technical Report No. 101 (1981).

13. Paul Walton Purdom, Jr., Cynthia A. Brown and Edward L. Robertson, Backtracking with Multiple-Level Search Rearrangement, ACTA Informatica 15 (1981), pp. 99-113.

14. Edward Robertson and Ian Munro, NP Completeness, Puzzles and Games, Utilitas Math. 13 (1978), pp. 99-116.