Search Rearrangement Backtracking and Polynomial Average Time

by

Paul Walton Purdom, Jr.

Computer Science Department

Indiana University

Bloomington, Indiana 47405

TECHNICAL REPORT NO. 123

SEARCH REARRANGEMENT BACKTRACKING AND POLYNOMIAL AVERAGE TIME

Paul Walton Purdom, Jr.

Revised: October 1982

Research reported herein was supported in part by the National Science Foundation under grant number MCS 79-06110.

Abstract

The average time required for simple search rearrangement backtracking is compared with that for ordinary backtracking when each algorithm is used to find all solutions for random conjunctive normal form predicates. The sets of random predicates are characterized by v: the number of variables, $t(v)$: the number of clauses, and $p(v)$: the probability that a literal appears in clause. For large v if $vp(v) \leq \ln 2$ both backtracking methods require exponential time if and only if the average number of solutions per problem is exponential. Any backtracking which finds all solutions to problems has this limitation. For $vp(v) > \ln 2$, there is a difficult region where the average number of solutions per problem is exponentially small, but backtracking requires an exponentially large time. The difficult region for search rearrangement backtracking is only slightly smaller than the difficult region for ordinary backtracking. It is conjectured that search rearrangement backtracking is exponentially faster than ordinary backtracking for nearly all of the difficult region. It is proved that there is no major advantage in using search rearrangement backtracking outside of the difficult region.

1 Introduction

Backtracking is a method of solving problems by assigning values to variables, one variable at a time. After each variable is set, an intermediate test is done to determine whether there is any hope of a solution. So long as there is a hope, additional variables are assigned values. If there is no hope, the next value is tried for the most recently set variable and testing continues. In this way all possible assignments of values to variables are considered implicitly. The versions of backtracking considered in this paper search for all solutions to a problem, so the search continues even after the first solution is found. By using intermediate tests backtracking can solve problems much more rapidly than they can be solved by explicit enumeration.

Ordinary backtracking uses a fixed order for selecting variables. Each time it is necessary to assign a value to an unset variable, the first unset variable is selected. Bitner and Reingold [1] studied a backtracking method that uses a dynamic search order. In their algorithm, which I call simple *search rearrangement backtracking*, when it is necessary to select a variable, all unset variables have each of their values tested. The variable with the fewest values that pass the test is selected. Both measurements [1, 3] and analytical studies [5] show that this method of dynamically selecting variables can greatly improve the speed of backtracking.

In this paper the average performance of simple search rearrangement backtracking is compared with ordinary backtracking for the same wide range of conditions that were used in [6,7,8]. The problem set is random conjunctive normal form predicates, characterized by v: the number of variables, $t(v)$: the number of clauses, and $p(v)$: the probability that a literal appears in a clause. The results of [6,7,8] are summarized in Fig. 1. For each possible asymptotic function for $p(v)$, there is a critical asymptotic function $t(v)$ that separates polynominal time from exponential time. For most of the region shown in Fig. 1, backtracking does as well as one could hope; it requires exponential time if and only if the average number of solutions per problem is exponential. The shaded part of Fig. 1, however, shows the *difficult region*, where the average number of solutions per problem is exponentially small but the time required by backtracking is exponentially large.

Fig. 2 shows an enlargement of the most interesting part of Fig. 1, where $\lim_{v\to\infty} vp(v)$ and $\lim_{v\to\infty} t(v)/v$ are constant. It also shows the results of this paper. In Fig. 2 the difficult region for ordinary backtracking is between the curves marked *Level 0* and *Solutions*. (The names *Level 0* and *Level 1* are terms from [9] used to indicate how extensive a search is used to select the next variable.) The performance of search rearrangement backtracking is shown by the shaded region (only limits on the performance are obtained). The difficult region of search rearrangement backtracking is between the curve marked *Solutions* and a curve somewhere inside the shaded region.

This paper contains three results. First, there is a significant class of problems where search rearrangement backtracking uses polynomial average time while ordinary backtracking uses exponential average time. Second, for most of the difficult region for ordinary backtracking, search rearrangement backtracking also takes exponential time; search rearrangement converts only a small fraction of the region from exponential to polynomial time. Third, for most places outside of the difficult region there is no reason to use search rearrangement backtracking (indeed, because it uses more time per node, one should avoid using it). The difficult region, however, contains the problems most often solved by backtracking.

The paper also has an important conjecture. The lower limit analysis shows that search rearrangement backtracking may save exponential time throughout the difficult region (even though the resulting time may still be exponential). Unfortunately the upper limit analysis gives a result that is slightly too large for proving the conjecture.

2 The Statistical Model

To do backtracking one needs a problem, which is a predicate P defined over a set of variables $S = \{x_i\}_{1\le i\le v}$, where each x_i has finitely many possible values. Also one must have some intermediate predicate P_A defined over sets $A \subseteq S$. It is convenient to require that $P_S \equiv P$. The intermediate predicate must have the value *true* for any assignment of values to the variables in A that can be extended to a solution of P. Thus $P_A = \mathit{false}$ indicates that there is no hope of extending the partial setting of the variables in A to a solution. An intermediate predicate is powerful if it is *false* for most other assignments of values. Powerful intermediate predicates that can be calculated efficiently eliminate false starts quickly, making backtracking feasible for large problems.

For this study the predicate P is a conjunctive normal form formula. For each $A \subseteq S$, P_A is the conjunction of those clauses from P that use only variables from A. The existence of this simple natural set of intermediate predicates is the primary reason for using conjunctive normal form predicates for investigating the performance of backtracking algorithms.

Random predicates are defined the same way as in [6]. The set S contains v variables. Each variable can be assigned the value *true* or *false*. For each variable there are two literals: the variable and its negation. A random clause is formed by independently selecting each of the $2v$ literals with probability $p(v)$. A random predicate, P, is formed by independently selecting $t(v)$ random clauses. Clauses that contain no literals are *false* while predicates that contain no clauses are *true*. This model is the same as the model used in [6]. A similar model is used in [4], but there clauses that contain both a variable and its negation are never formed.

The probability, with the present model, that a clause contains both a variable and its negation is $1-(1-p(v)^2)^v$, which is small compared to one if and only if $vp(v)^2$

is small. The tautological clauses that contain both a variable and its negation do not effect the operation of backtracking algorithms. Including them only inflates the count of how many clauses the predicate contains, but a backtrack algorithm (unlike some other kinds of searching algorithms) will generate the same search tree whether or not these clauses are included. The inflation is small when $vp(v)^2$ is small. (Also it has been shown that including [8] or not including [4] such clauses has little effect on the average time of the pure literal rule algorithm so long as $p(v) \ll \frac{1}{2}$.)

In [2, 5] a model is used where $p(v)$ is omitted and instead random clauses with exactly s literals per clause are formed. The value $s = vp(v)$ is appropriate to use when comparing the model in [2] with the current model. The present model may produce less realistic problem sets, but it leads to easier analyses, and more algorithms have been analyzed using it. The advantages are important for this study.

The simple search rearrangement algorithm is as follows [5]. 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. We denote the value of predicate variable w by $Value[w]$. The set S'' is maintained as a stack.

Simple 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 is immaterial for our purposes.) For both $Value[w] \leftarrow false$ and $Value[w] \leftarrow true$, compute $P_{S'' \cup \{w\}}$. If the result is *false* in both cases, exit the loop, and go to Step 6. If it is *false* in one and *true* in the other, remember the value of w that gives *true*, exit the loop, and go to Step 5. If the result is *true* in both cases, continue with testing the next value of w.

Step 4. [*Binary node*]: Let w be the smallest element of S'. Set $S'' \leftarrow S'' \bigcup \{w\}$ and $S' \leftarrow S' - \{w\}$. Set $Value[w] \leftarrow false$, mark w as binary, and go to Step 2.

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

Step 6. [*Next value*]: If S'' is *empty*, stop. Otherwise, set $w \leftarrow top(S'')$. If w is marked as binary and $Value[w] = false$, set $Value[w] \leftarrow true$ and go to Step 2.

Step 7. [*Backtrack*]: Set $S'' \leftarrow S'' - \{w\}$, $S' \leftarrow S' \bigcup \{w\}$, and go to Step 6.

By way of example, Figure 3 shows the backtrack tree that results when the simple search rearrangement algorithm is run on a particular predicate.

3 Basic Formulas

In this paper the average number of *binary* nodes in a backtrack tree is calculated. Ordinary backtracking (when used on problems with binary variables) produces trees where all nodes have degree zero or two, so the total tree size is one plus twice the number of binary nodes. Search rearrangement backtracking produces trees with nodes of degrees zero, one, and two, so the total number of nodes is no more than one plus v times the number of binary nodes. For ordinary backtracking the number of intermediate

predicate evaluations is equal to the number of nodes. For search rearrangement backtracking with binary variables the number of evaluations is no more than $2v$ times the number of nodes. Thus the number of evaluations is no more than $2v$ plus $2v^2$ times the number of binary nodes.

In this paper I assume that running time is equal to the number of predicate evaluations. It might be more realistic to assume that running time is equal to the number of predicate evaluations times the predicate size. On the other hand there are clever methods of repeated predicate evaluation (see [3] for example) that are faster than this. Also it is a common practice in theoretical computer science to compute running time as a function of problem size. The average problem size, $(1 + 2vp(v))t(v)$, can be used to indicate the effect of converting the number of binary nodes as a function of the number of variables (which is what is calculated) to the actual running time as a function of the problem size. Qualitatively the effect is unimportant when $t(v)$ is polynomial, but the average actual running time is always a polynomial function of the average problem size when $t(v)$ is exponential.

To calculate the average number of binary nodes, consider each node in a complete binary tree of height v, and for each node compute the probability that the corresponding node occurs on the backtrack tree for a random predicate. The method of derivation combines those used in [6] and in [5]. The following abbreviations will be used:

p for $p(v)$,
t for $t(v)$, and
lim for $\lim_{v\to\infty}$.

The function $x(v)$ is *exponential* when $x(v) = \exp(\Theta(v))$, i.e. for some positive constants c_1 and c_2 one has $\exp(c_1 v) \leq x(v) \leq \exp(c_2 v)$ for all $v > v_0$ where v_0 is some positive constant. Likewise $x(v)$ is *polynomial* when $x(v) < v^n$ for some n and all $v > v_0$ for some v_0. In all cases exponential and polynomial are used to imply exponential or polynomial with respect to v.

The probability that a clause is *false* after q variables are set is $(1-p)^{2v-q}$. The probability, when q variables are set, that setting one more variable will cause a clause to become *false* is

$$Q = (1-p)^{2v-q-1} - (1-p)^{2v-q} = p(1-p)^{2v-q-1}. \tag{1}$$

Notice that Q is the probability that a variable is *forced* to not have a particular value. If p_c is the probability that a clause is *false* then $(1-p_c)^t$ is the probability that a predicate with t clauses is *not false*.

For ordinary backtracking the average number of binary nodes is

$$N = \sum_{0\leq q<v} 2^q[1-(1-p)^{2v-q}]^t. \tag{2}$$

(A similar formula is given in [6] for the *total* number of nodes, here the nodes of degree zero are not counted.) The average number of solutions (regardless of the method) is

$$S = 2^v[1-(1-p)^v]^t. \tag{3}$$

For search rearrangement backtracking no simple exact formula is known for the number of binary nodes. (The exact formula in [5] has $\exp(\Theta(v))$ terms.) To produce

a formula like (2) one needs the probability that when the first q variables are set, variable $q+1$ does not have its value forced. One needs to consider both how the first q variables can directly force variable $q+1$ and how they can indirectly force it by first forcing other variables. A lower limit on the number of binary nodes is obtained by ignoring indirect forcing. This gives the lower limit for search rearrangement of

$$L = \sum_{0 \le q < v} 2^q [1 - (1-p)^{2v-q} - 2(v-q)Q]^t. \tag{4}$$

Two upper limits are needed. First, the intermediate predicate can not be *false* at a binary node, so formula (2) is also an upper limit on the number of binary nodes for search rearrangement backtracking. The second upper limit is based on the fact that at a binary node the predicate is not *false* and no unset variables are forced. A simple formula is obtained by ignoring the requirement that variables can not spontaneously force each other; forcing can be done only by variables that already have values. This upper limit on the number of binary nodes for search rearrangement backtracking is

$$U = \sum_{0 \le q < v} 2^q v \binom{v-1}{q} [1 - (1-p)^{2v-q} - 2(v-q)Q]^t, \tag{5}$$

where the factor v allows for the number of ways to select the binary node and $\binom{v-1}{q}$ allows for the number of ways to select which q variables have values.

4 Analysis: Preliminaries

This paper is primarily concerned with when the running time of a backtracking algorithm is polynomial or exponential. The sums in (2), (4), and (5) are all sums of v positive terms. Therefore each sum differs from its maximum term by a factor that is no larger than v. This factor has no effect on whether the running time is polynomial or exponential. I use n, l, and u for the logarithm of a term from (2), (4), and (5) respectively. Also I use s for the logarithm of S. So

$$n = q \ln 2 + t \ln [1 - (1-p)^{2v-q}] \tag{6}$$

$$s = v \ln 2 + t \ln [1 - (1-p)^{v}] \tag{7}$$

$$l = q \ln 2 + t \ln [1 - (1 - p + 2(v-q)p)(1-p)^{2v-q-1}] \tag{8}$$

$$u = q \ln 2 + \ln v + (v - 1/2)\ln(v-1) - (q + 1/2)\ln q - (v - q - 1/2)\ln(v-q-1) + t \ln [1 - (1 - p + 2(v-q)p)(1-p)^{2v-q-1}] + \Theta(1) \tag{9}$$

This paper is primarily concerned with how these four functions behave for large v and small p. For discussing such limits it is useful to define

$$a = v(-\ln(1-p)) = pv(1 + O(p)) \tag{10}$$

and

$$x = q(-\ln(1-p)) = pq(1 + O(p)). \tag{11}$$

Also

$$(1-p)^{2v-q} = (1-p)^{2v-q-1}(1 + O(p)) = \exp(-2a + x). \tag{12}$$

The definitions use $-\ln(1-p)$ rather than p so that various exponential functions can be expressed exactly.

To analyze the behavior of the number of solutions for large v, rewrite (7) as

$$\frac{s}{v} = \ln 2 + \frac{t}{v}\ln[1 - \exp(-a)]. \tag{13}$$

For constant ϵ with $s/v > \epsilon > 0$ the average number of solutions is exponential while for constant k with $s/v < (\ln v^k)/v$ the average is polynomial. The critical value of t that separates an exponential average number of solutions from a polynomial average is given by setting $s/v = \epsilon$ to obtain

$$\frac{t_*}{v} = \frac{\ln 2 - \epsilon}{-\ln[1 - \exp(-a)]}, \tag{14}$$

and taking the limit as ϵ goes to zero. (Technically the time is exponential for any fixed $\epsilon > 0$ while it is polynominal if $\epsilon(v) \leq n(\ln v)/v$ for some fixed n and large v, but for this paper it will be suitable and much simplier to use the limit as ϵ goes to zero to define critical values.) This is plotted in Fig. 2 (with $\epsilon = 0$) as the curve marked *Solutions*. For large a one can write (14) as

$$\frac{t_*}{v} = (\ln 2 - \epsilon)\exp(a)[1 + O(\exp(-a))]. \tag{15}$$

The behavior of n is analyzed in [6]. To maximize n one needs to consider (6) with $q = 0$, $q = v - 1$, and the value of q such that $dn/dq = 0$. However $q = 0$ never results in exponential time ($n_{(q=0)} < 0$). For $a \leq \ln 2$ the critical value of t is obtained by setting $q = v - 1$. Since

$$n_{(q=v-1)} = s(1 + O(p)) \tag{16}$$

one can use the result for s in this case. For $a \geq \ln 2$ the critical value of t is obtained by setting q to the first root of $dn/dq = 0$. The critical value for t is

$$\frac{t_*}{v} = \frac{d(\ln 2 - \epsilon)}{a}(1 + O(p)) \tag{17}$$

where d is the largest root of

$$\ln(1 + d) + d\ln\left(1 + \frac{1}{d}\right) = 2a. \tag{18}$$

This is plotted in Fig. 2 as the curve marked *Level 0*. For large a one can write

$$d = \exp(2a) + O(1) \tag{19}$$

and

$$\frac{t_*}{v} = \frac{(\ln 2 - \epsilon)}{a}\exp(2a) + O(\tfrac{1}{a}) + O(p) \tag{20}$$

Finally for $a \geq \ln 2$ and for

$$\frac{t}{v} = \ln(\tfrac{2}{a})\exp[(1 + \alpha a], \quad \text{where } \alpha \text{ is any number such that } 0 < \alpha < 1, \tag{21}$$

the maximum value of n is

$$n_* = (\ln 2)(1 - \alpha)v + O(1). \tag{22}$$

These results will be compared with the corresponding results for search rearrangement backtracking.

5 Analysis: Lower Limit for Search Rearrangement Backtracking

The lower limit on the number of binary nodes for search rearrangement backtracking will be considered first. Note that

$$l_{(q=v-1)} = n_{(q=v-1)}(1 + O(p)), \tag{23}$$

and if l is maximized by setting $q = v - 1$ then n is also maximized by setting $q = v - 1$. Thus whenever l is maximized by setting $q = v - 1$, search rearrangement backtracking is no better than ordinary backtracking. (This argument ignores some low degree polynomial factors (up to degree v^2).) Likewise whenever the maximum value of n is zero or less, the running time for each algorithm is determined primarily by the time required to process the root node of the backtrack tree; so search rearrangement also does not lead to any significant improvement in this case.

The lower limit will first be used to identify values of pv and t/v where (in the limit of large v) search rearrangement definitely does not lead to any significant improvement in running time. Then it will be used to identify values where search rearrangement might lead to improvement.

To maximize l first compute

$$\frac{dl}{dq} = \ln 2 + \frac{t[-2p + (1 - p + 2(v - q)p)(-\ln(1 - p))](1 - p)^{2v-q-1}}{1 - (1 - p + 2(v - q)p)(1 - p)^{2v-q-1}} \tag{24}$$

Notice that the bottom of the fraction is between zero and one (for $0 < p < 1$, $0 \leq q < v$) because it is a probability. The factor in square brackets is negative when

$$v - q < \frac{1}{-ln(1 - p)} - \frac{1}{2p} + \frac{1}{2} = \frac{1}{2p}(1 - \Theta(p^2)), \tag{25}$$

so for

$$v < \frac{1}{2p}(1 - \Theta(p^2)) \tag{26}$$

the top is negative for all q. Thus for $\lim pv < 1/2$, $dl/dq > 0$ and l is maximized by setting $q = v - 1$. For larger values of pv, one also has $dl/dq > 0$ provided t is small enough. One can show that for $a \geq 1/2$, one has $dl/dq > 0$ for all q provided

$$\frac{t}{v} < \frac{(\ln 2)(1 - \exp(-1/2))\exp(a)}{a}(1 + O(p)) - \approx \frac{0.273exp(a)}{a}. \tag{27}$$

For large a one has $dl/dq > 0$ for all q provided

$$\frac{t}{v} < \frac{\ln 2 \exp(a)}{a}(1 + O(\frac{1}{a}) + O(p)). \tag{28}$$

Also note that

$$l \leq n, \tag{29}$$

so if $n/v \leq \epsilon$ then $l/v \leq \epsilon$.

The results of the previous paragraph imply that, except in the shaded region of Fig. 1, search rearrangement backtracking is no better than ordinary backtracking. Eq.(26) implies that it is no better for $pv \leq 1/2$, eq. (28) implies that it is no better

when $t/v \leq (0.273/a)\exp(a)$, and eq. (29) implies that it is no better for t/v such that $n_* \leq 0$. Thus the only place where there is any hope for improvement from using search rearrangement is the region in Fig. 2 that is to the left of *Level 0* and to the right of *Solution* (Except that for moderate a search rearrangement might also do better slightly to the right of the *Solution* curve; the coefficient in (27) is below ln 2.)

Although most of $[pv, t/v]$ space has been eliminated as a place where one should use search rearrangement, the remaining region contains the problems most like those that are normally solved by backtracking. I will now consider the lower limit in more detail.

The maximum value of l is obtained by setting q so that either $q = 0$, $q = v - 1$, or $dl/dq = 0$. The $q = 0$ case results in $l < 0$, so it never leads to exponential time. If pv and pt are large enough then dl/dq is negative for small q, infinite for $q \approx v - 1/p$ and positive for larger q. The maximum of l occurs for $q = 0$ or $q = v - 1$. A more interesting case occurs when pv is large enough and pt is only moderately large. Then dl/dq is positive for small q, but it becomes zero for moderate q, because $(1 - p)^{2v-q-1}$ in the top of (24) increases rapidly with q. For $q \approx v - 1/p$, dl/dq is infinite. For larger q it is initially negative but it increases. The first zero of dl/dq corresponds to a local maximum for l while the second one (if any) corresponds to a minimum. The global maximum occurs either at the first root of dl/dq or at $q = v - 1$. By setting dl/dq to zero one obtains

$$q_* = 2v - 1 - \frac{1}{-\ln(1-p)} \ln\left[1 - p + 2(v - q_*)p + \frac{t[-2p + (1 - p + 2(v - q_*)p)(-\ln(1-p))]}{\ln 2}\right]. \quad (30)$$

Although (30) contains q_* on both sides, one can obtain a useful approximation by ignoring the q_*'s on the right (except for two special cases: 1) $q_* \approx v$ when pv is large and pt is small and 2) $q_* \approx v - 1/p$ when pt and p^2tv are large). I will come back to (30) later.

First however let us consider the critical value of t that results in the maximum value of l/v equaling $\epsilon > 0$. Notice that eq. (8), the definition of l, is a linear function of t. The two values of q that might maximize l (for the critical t) are the first root of dl/dq and $v - 1$. For each value of q one can calculate the corresponding value of t that results in $l(t, q)/v = \epsilon$. The larger of the two values of t results in $\max_{0 \leq q < v} l(t, q)/v = \epsilon$ (the global maximum is ϵ) and gives the critical value. For $q = v - 1$, $l = s(1 + O(p))$ so the value of t that results in $l_{(q=v-1)}/v = \epsilon$ is given by (15).

To find the values of t and q that result in $dl/dq = 0$ and $l/v = \epsilon$, one can postpone the difficulties associated with the transcendental nature of (24) by solving $dl/dq = 0$ for t to obtain

$$t_* = \frac{(\ln 2)[1 - (1 - p + 2(v - q)p)(1 - p)^{2v-q-1}]}{[-2p + (1 - p + 2(v - q)p)(-\ln(1-p))](1 - p)^{2v-q-1}}. \quad (31)$$

Plugging this value of t into (8) gives

$$l_* = (\ln 2)\left\{q + \frac{[1 - (1 - p + 2(v - q)p)(1 - p)^{2v-q-1}]\ln[1 - (1 - p + 2(v - q)p)(1 - p)^{2v-q-1}]}{[-2p + (1 - p + 2(v - q)p)(-ln(1-p))](1 - p)^{2v-q-1}}\right\}. \quad (32)$$

In the limit of large v and small p, one can rewrite (31) and (32) as

$$\frac{t_*}{v} = (\ln 2)\frac{[1 - (1 + 2(a - x))\exp(-2a + x)]}{a[-1 + 2(a - x)]\exp(-2a + x)} \tag{33}$$

and

$$\frac{l_*}{v} = \frac{\ln 2}{a}\left[x - f\frac{1 + 2(a - x)}{-1 + 2(a - x)}\right], \tag{34}$$

where

$$f = -\frac{(1 - y)\ln(1 - y)}{y} \tag{35}$$

and

$$y = (1 + 2(a - x))\exp(-2a + x). \tag{36}$$

The relative error in these approximations is (1+ O(p)) *provided* $[-1+2(a-x)\exp(-2a+x)] > \epsilon > 0$ for some constant ϵ. For $a \geq 1/2$ and $0 \leq x \leq a$, f is roughly constant ($0.478 \leq f \leq 1$). The value of x that results in $l_*/v = \epsilon$ is given by

$$x_*^2 + (\frac{1}{2} - f - a - \epsilon)x_* + \frac{f}{2} + af - \frac{\epsilon}{2} + a\epsilon = 0. \tag{37}$$

Using

$$b = \frac{a}{2} + \frac{f}{2} + \frac{\epsilon}{2} - \frac{1}{4} \tag{38}$$

and

$$c = af + \frac{f}{2} + a\epsilon - \frac{\epsilon}{2} \tag{39}$$

the root of interest is given by

$$x_* = b - \sqrt{b^2 - c}. \tag{40}$$

To obtain t_*, the candidate for the critical value of t associated with $dl/dq = 0$ do the following. Set $\epsilon = 0$ and assign a value for a. Guess a value for x. Compute f with (35) and (36). Use (38), (39), and (40) to obtain a new value for x. Iterate until the limiting value of x is obtained with adequate accuracy. Then use (33) to compute t_*/v. The upper boundary for $pv > 4.575$ of the *Level 1* region in Fig. 2 is a plot of this t_*/v. For $pv < 4.574$, $q = v - 1$ gives the global maximum, so there the *Level 1* region is bounded by the *Solutions* curve. For large a, one can find t_*/v without iteration:

$$f = 1 + O(a\exp(-a)) \tag{41}$$

$$x_* = 1 - \epsilon - \Theta\left(\frac{1}{a}\right), \tag{42}$$

and

$$\frac{t_*}{v} = \frac{\ln 2}{2ea^2}\exp(2a - \epsilon)\left(1 + 0\left(\frac{1}{a}\right)\right). \tag{43}$$

Notice that for large a, the critical t_* for l is much larger than t_* for s (eq. (15)), but t_* for n (eq. (20)) is larger than t_* for l by only the factor $2ea$. So at best search rearrangement

backtracking has only a moderate effect on reducing the size of the region that contains problems where the number of solutions is exponentially small but where backtracking methods require exponential time.

Finally notice that for large v, large pv, small p, and t/v given by eq. (21), that the solution to eq. (30) is given by

$$\frac{q_*}{v} = \left[1 - \alpha - \frac{\ln(2a)}{a} + O\left(\frac{1}{a^2}\right)\right], \tag{44}$$

so

$$\frac{l_*}{v} = \left[1 - \alpha - \frac{\ln(2a)}{a} + O\left(\frac{1}{a^2}\right)\right](\ln 2). \tag{45}$$

Comparing (45) with (22) shows that for the region defined by eq. (21) (the shaded part of Fig. 1) search rearrangement backtracking may be able to save an exponential amount of time, but the coefficient of the exponent, $(\ln a)/a$, decreases rapidly with a.

6 Analysis: Upper Limit for Search Rearrangement Backtracking

Now it is time to investigate the upper limit to show that search rearrangement backtracking actually does work better than ordinary backtracking for at least part of the difficult region. The derivative of eq. (9) gives

$$\frac{du}{dq} = \ln\left(\frac{2(v-q-1)}{q}\right) - \frac{1}{2q} + \frac{1}{2(v-q-1)} - \frac{t[-2p + (1-p+2(v-q)p)(-\ln(1-p))](1-p)^{2v-q-1}}{1-(1-p+2(v-q)p)(1-p)^{2v-q-1}}. \tag{46}$$

Due to the $\Theta(1)$ term in eq. (9) this is not necessarily an accurate value for du/dq, but it is surely satisfactory for maximizing (9) (to within $\Theta(1)$). Although (46) is much more complex than (24), for most values of p, t, v its qualitative behavior is similar except at $q = 0$ and $q = v - 1$. Proceeding as before, one has

$$t_* = \frac{\left[\ln\left[\frac{2(v-q-1)}{q}\right] - \frac{1}{2q} + \frac{1}{2(v-q-1)}\right][1-(1-p+2(v-q)p)(1-p)^{2v-q-1}]}{[-2p+(1-p+2(v-q)p)(-\ln(1-p))](1-p)^{2v-q-1}} \tag{47}$$

and

$$u_* = q\ln\left[\frac{2(v-q-1)}{q}\right] + \frac{1}{2}\ln\left[\frac{v^2(v-q-1)}{(v-1)q}\right] + v\ln\left[\frac{v-1}{v-q-1}\right] + \frac{\left\{\ln\left[\frac{2(v-q-1)}{q}\right] - \frac{1}{2q} + \frac{1}{2(v-q-1)}\right\} y\ln y}{[-2p+(1-p+2(v-q)p)(-\ln(1-p))](1-p)^{2v-q-1}} \tag{48}$$

where $y = 1 - (1-p+2(v-q)p)(1-p)^{2v-q-1}$. For large v and small p one has

$$\frac{t_*}{v} = \frac{\left\{\ln\left[\frac{2(a-x)}{x}\right]\right\}[1-(1+2(a-x))\exp(-2a+x)]}{a[-1+2(a-x)]\exp(-2a+x)} \tag{49}$$

and

$$\frac{u_*}{v} = \frac{x}{a}\ln\left[\frac{2(a-x)}{x}\right] + \ln\left[\frac{a}{a-x}\right] + \frac{\ln\left[\frac{2(a-x)}{x}\right][1-(1+2(a-x))\exp(-2a+x)]\ln[1-(1+2(a-x))\exp(-2a+x)]}{a[-1+2(a-x)]\exp(-2a+x)}. \tag{50}$$

Using (35) for f, one can write

$$\frac{u_*}{v} = \frac{x}{a}\ln\left[\frac{2(a-x)}{x}\right] + \ln\left[\frac{a}{a-x}\right] - f\left\{\ln\left[\frac{2(a-x)}{x}\right]\right\}\frac{1+2(a-x)}{-1+2(a-x)}. \tag{51}$$

The value of x that results in $u_*/v = \epsilon$ is given by

$$x_*^2 + (1/2 - f - a - ag - \epsilon)x_* + \frac{f}{2} + af + \frac{ag}{2} - a^2 g - \frac{\epsilon}{2} + a\epsilon = 0, \tag{52}$$

where

$$\ln g = \frac{\ln\left[\frac{a}{a-x}\right]}{\ln\left[\frac{2(a-x)}{a}\right]}. \tag{53}$$

Using

$$\begin{aligned} b &= \frac{a}{2} + \frac{ag}{2} + \frac{f}{2} - \frac{1}{4} + \frac{\epsilon}{2} \\ c &= af + ag - a^2 g + \frac{f}{2} + a\epsilon - \frac{\epsilon}{2} \end{aligned} \tag{54}$$

the root of interest is given by

$$x_* = b - \sqrt{b^2 - c} \tag{55}$$

One can find x_* by iterating the relevant equations. For $a < 3.7$, however, the iteration has an oscillating divergence, which can be avoided by setting x to the average of the old x and the predicted x. The resulting critical value for t_*/v is plotted in Fig. 2 as the lower boundary (for $a \geq 2.6$) of the *Level 1* region. For $a \leq 2.5$ the *Level 0* boundary, which is also an upper limit on the performance of search rearrangement backtracking, provides a more accurate limit.

For large a , one has (using eq. (41))

$$g = \frac{x}{a\ln\left(\frac{2a}{x}\right)}\left[1 + O\left(\frac{x}{a}\right)\right], \tag{56}$$

$$x_* = \left[\frac{1}{1+\ln(2a)} - \epsilon\right]\left[1 + O\left(\frac{x}{a}\right)\right], \tag{57}$$

and

$$\frac{t_*}{v} = \frac{\ln(2a)}{2ea^2}\exp(2a-\epsilon)\left[1 + O\left(\frac{\ln a}{a}\right)\right]. \tag{58}$$

The upper limit for the critical value of t_*/v for search rearrangement backtracking (eq. (58)) is larger than the lower limit (eq. (43)) by only a factor of $\ln a$. It is smaller than the critical value for ordinary backtracking (eq. (20)) by a factor of $(2ea \ln 2)/\ln a$. It is larger than the critical value for solutions (eq. (15)) by a factor of $\frac{(\ln a)(\ln 2)}{2ea^2} \exp(a)$. Thus search rearrangement backtracking can solve a lot more problems in polynomial time, but the improvement eliminates only a small portion of the difficult region.

Finally the value of q that results in $du/dq = 0$ is

$$q_* = 2v - 1 - \frac{1}{-\ln(1-p)} \ln\left[1 - p + 2(v - q_*)p + \frac{t[-2p + (1 - p + 2(v - q_*)p)(-\ln(1-p))]}{\ln\left[\frac{2(v-q-1)}{q}\right] - \frac{1}{2q_*} + \frac{1}{2(v-q_*-1)}}\right]. \tag{59}$$

For large v, large pv, small p, and t/v given by (21), the solution to (59) is given by

$$\frac{q_*}{v} = \left[1 - \alpha - \frac{\ln(2a)}{a \ln\left(\frac{\alpha}{1-\alpha}\right)} + O\left(\frac{\ln a}{a^2}\right)\right], \tag{60}$$

so

$$\frac{u_*}{v} = \left[1 - \alpha - \frac{\ln(2a)}{a \ln\left(\frac{\alpha}{1-\alpha}\right)} + O\left(\frac{\ln a}{a^2}\right)\right] \ln \frac{\alpha}{1-\alpha}. \tag{61}$$

Unfortunately (61) is larger than (22) for fixed α and large a, so u is not an accurate enough approximation to determine whether search rearrangement backtracking saves time over most of the difficult region.

7 Conclusions

Empirical studies [1] have shown that simple search rearrangement backtracking can save a huge amount of time, compared to ordinary backtracking. In this study the performance of each type of backtracking is compared using random problem sets. Considering that the forms of backtracking being studied find all solutions to their problems, their average performance is almost as good as one could hope: they take exponential time if the average number of solutions per problem is exponential and they usually take polynomial time if the average number of solutions is polynomial. There is, however, a difficult region, marked by shading in Fig. 1 and between the *Solution* and *Level 0* curves in Fig. 2, where ordinary backtracking requires exponential time even though the average number of solutions per problem is exponentially small.

Outside of the difficult region there is no advantage to using simple search rearrangement backtracking. Some problems are so easy that both methods are quick, others are so hard that both methods are slow. The problems normally solved by backtracking, however, are similar to the typical problems from the difficult region, so the performance of the algorithms in the difficult region is particularly important.

The difficult region for simple search rearrangement backtracking is smaller than that for ordinary backtracking by a moderately large factor (pv), so the algorithm is of

considerable practical importance. The difficult region for each algorithm extends from $t/v \approx \exp(pv)$ to $t/v \approx \exp(2pv)$ so a reduction by a polynomial factor such as pv does little to eliminate the region.

The lower limit analysis suggests that simple search rearrangement backtracking may save exponential time throughout the difficult region. Other studies (such as [5] suggest that the lower limit is much closer to the true performance than the upper limit. There are better upper limits available (see [5]), and these should be analyzed by anyone wishing to prove that there is an exponential saving throughout the difficult region.

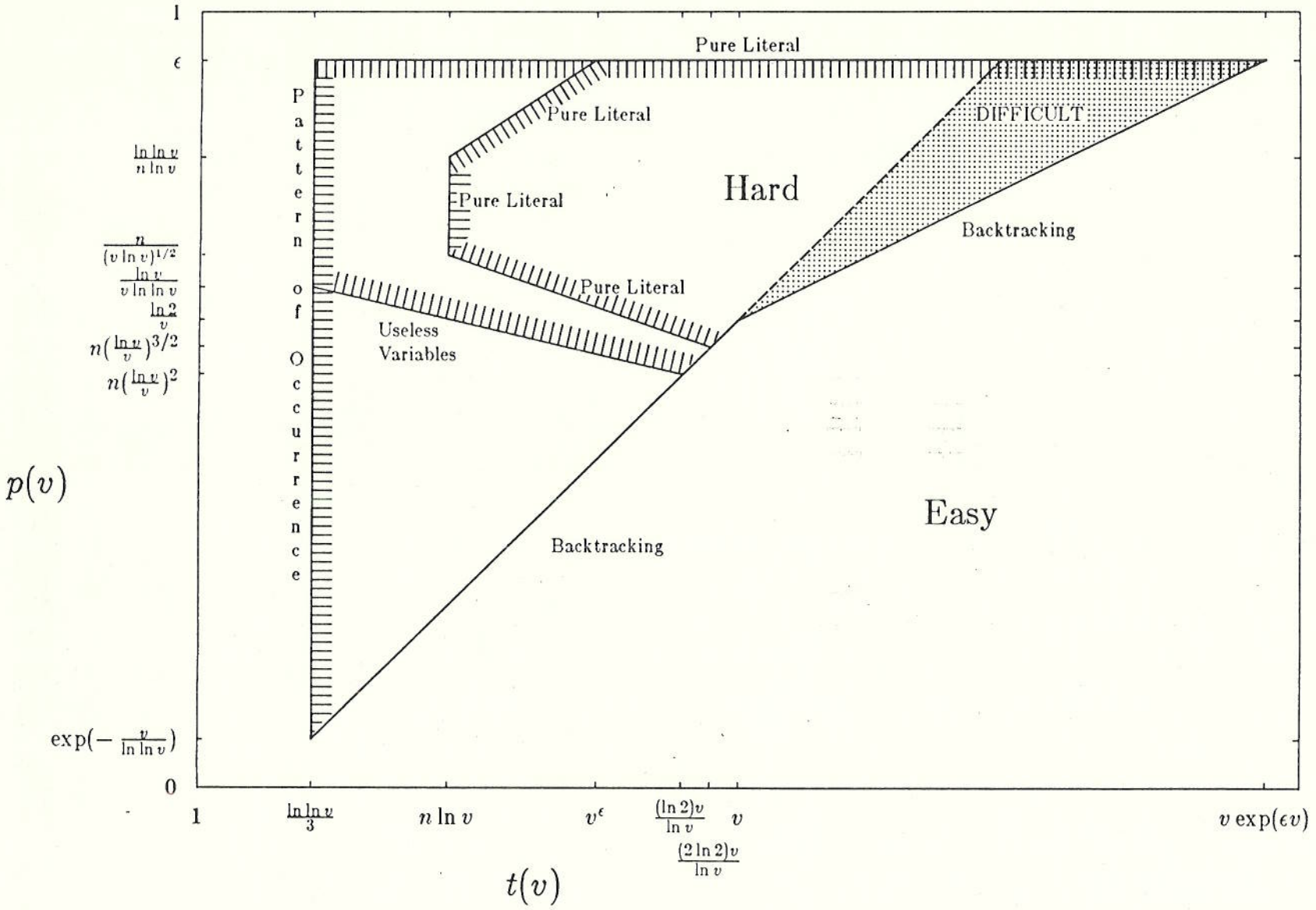
Pure Literal
Pure Literal
Pure Literal
Pure Literal
DIFFICULT
Hard
Backtracking
Useless
Variables
Backtracking
Easy
Pattern of Occurrence
1
ϵ
$\frac{\ln\ln v}{n\ln v}$
$\frac{n}{(v\ln v)^{1/2}}$
$\frac{\ln v}{v\ln\ln v}$
$\frac{\ln 2}{v}$
$n(\frac{\ln v}{v})^{3/2}$
$n(\frac{\ln v}{v})^{2}$
$\exp(-\frac{v}{\ln\ln v})$
0
$p(v)$
1
$\frac{\ln\ln v}{3}$
$n\ln v$
v^{ϵ}
$\frac{(\ln 2)v}{\ln v}$
$\frac{(2\ln 2)v}{\ln v}$
v
$v\exp(\epsilon v)$
$t(v)$

Figure 1. The recent results from analyses of the average time required by various search algorithms (when solving CNF satisfiability problems) are shown. For each analysis the significant portion of the contour that separates polynomial average time from worse time is shown. The contour is marked with hash marks if it is the result of an upper bound analysis. The result of an exact analysis might be anywhere on the hashed side of the line. The vertical axis is for various functions for $p(v)$, the probability that a literal appears in a clause. Small functions (as v goes to infinity) are at the bottom while large functions are at the top. The horizontal axis is for various functions for $t(v)$, the number of clauses, with small functions at the left. Each line in the figure is labelled with the associated algorithm. Problem sets that are in the region marked *hard* (which extends to both sides of the dashed line and which is bounded by the innermost of the contours surrounding the word **Hard**) appear to require exponential average time with current algorithms. Since most of the boundaries of the hard region result from upper bound analyses, some later analysis might shrink the region. Problem sets outside of the hard region are definitely easy; some algorithm is known that requires polynomial average time. The lower backtracking curve and the upper dashed curve separate problems which on the average have a polynomial number of solutions (those to the lower right of the line) from those that have an exponential number of solutions.

The shaded region indicates a *difficult region* where backtracking requires exponential time even though the average number of solutions per problem is exponentially small. Everywhere else in the hard region the average number of solutions is exponentially large. The difficult region is shown in more detail in Figure 2.

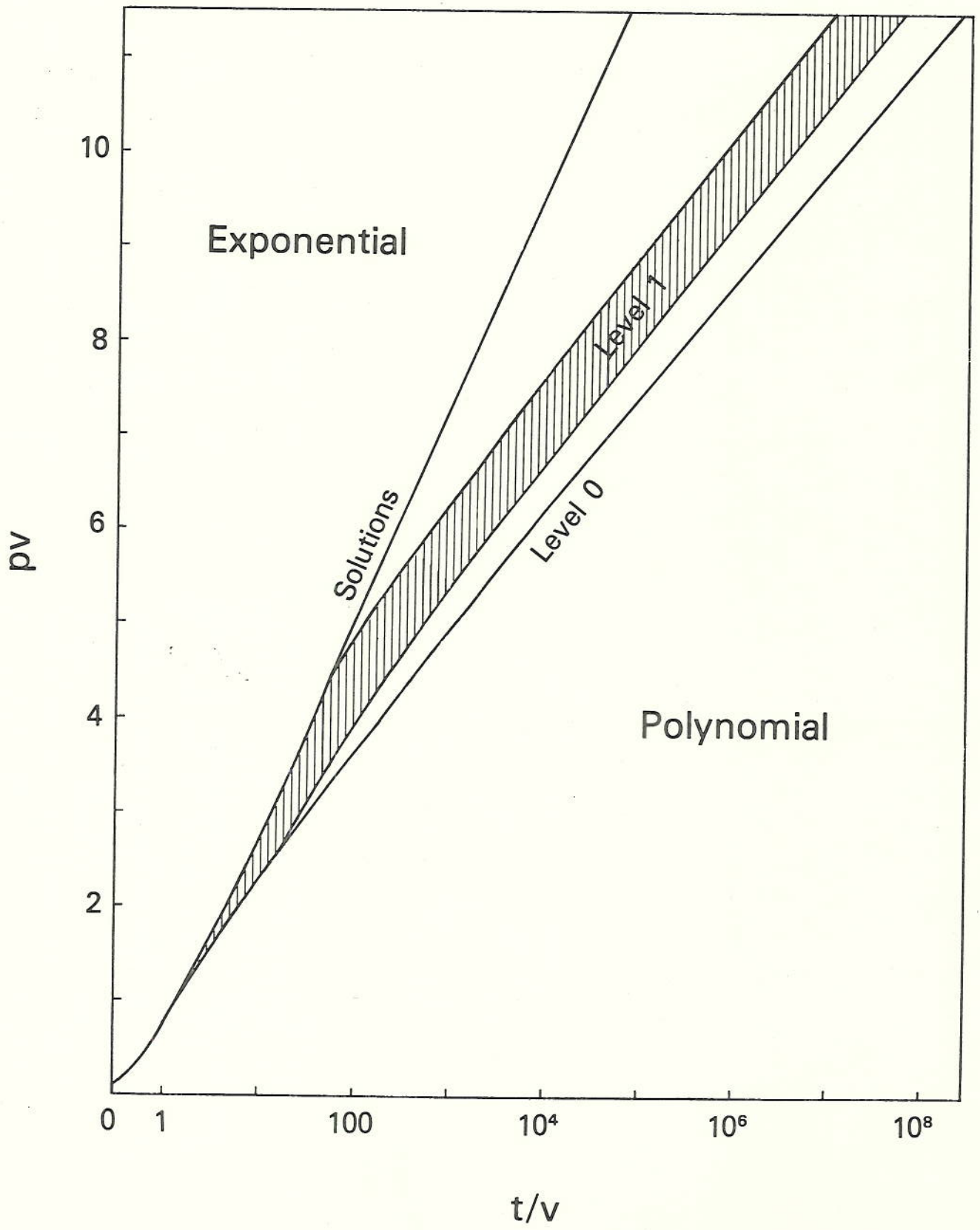
Exponential
Polynomial
Solutions
Level 1
Level 0
pv
t/v
10
8
6
4
2
0
1
100
10⁴
10⁶
10⁸

Figure 2. A graph giving more details on the performance of backtracking algorithms. The vertical axis is pv, the average number of literals per clause. The horizontal axis is t/v, the number of clauses per variable. The curve marked *Solutions* separates the region where the average number of solutions per problem is exponential from where it is polynomial. The curve marked *Level 0* separates the region where the average running time of ordinary backtracking is exponential from where it is polynomial. The analysis for simple search rearrangement backtracking produces only limits on its performance. The shaded region marked *Level 1* separates the region where the average running time of simple search rearrangement backtracking is exponential from where it is polynomial.

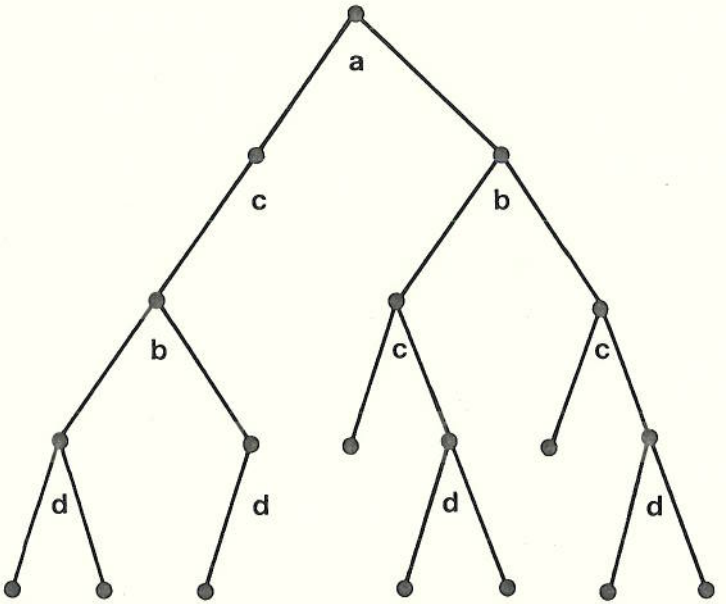
a
c
b
b
c
c
d
d
d
d

Figure 3. The backtrack tree produced by the search rearrangement backtracking for the predicate $A \wedge B \wedge C \wedge D$ where $A = a \vee \neg c$, $B = \neg a \vee c \vee d$, $C = \neg a \vee b \vee c \vee \neg d$, $D = \neg b \vee c \neg d$. 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. The tree has eight binary nodes, two unary nodes, two zero-degree nodes, and seven solution nodes.

References

[1] J. R. Bitner and E. M. Reingold, *Backtrack Programming Techniques*, CACM **18** (1975) pp. 651-655.

[2] Cynthia A. Brown and Paul Walton Purdom Jr., *An Average Time Analysis of Backtracking*, SIAM J. Comp. **10** (1981), pp. 583–593.

[3] Cynthia A. Brown and Paul Walton Purdom Jr., *An Empirical Comparison of Backtracking Algorithms*, IEEETPAMI **4** (1982), pp. 309–316.

[4] Allen Goldberg, Paul Walton Purdom Jr., and Cynthia A. Brown, *Average Time Analysis of Simplified Putnam-Davis Procedures*, Information Processing Letters (to appear). See also Allen Goldberg, *Average Case Complexity of the Satisfiability Problem*, Proc. of the Fourth Workshop on Automated Deduction (1979), pp. 1–6.

[5] Paul Walton Purdom Jr. and Cynthia A. Brown, *An Analysis of Backtracking with Search Rearrangement*, Indiana University Computer Science Technical Report No. 89, (1980) (to appear in SIAM J. Comput.).

[6] Paul Walton Purdom Jr. and Cynthia A. Brown, *Polynomial Average-Time Satisfiability Problems*, Indiana University Computer Science Technical Report No. 118 (1981).

[7] Paul Walton Purdom Jr. and Cynthia A. Brown, *Evaluating Search Methods Analytically*, Proc. National Conference on Artificial Intelligence, (1982) pp. 124–127.

[8] Paul Walton Purdom Jr. and Cynthia A. Brown, *The Pure Literal Rule and Polynomial Average Time*, Indiana University Computer Science Technical Report No. 128 (1982).

[9] Paul Walton Purdom Jr., Cynthia A. Brown, and Edward L. Robertson, *Backtracking with Multi-Level Dynamic Search Rearrangement*, Acta Informatica **15** (1981), pp. 99–113.