# Average Time for the Full Pure Literal Rule

Paul Walton Purdom, Jr., Indiana University

**Abstract:**

The simplified pure literal algorithm solves satisfiability problems by choosing variables in a fixed order and then generating subproblems for various values of the chosen variable. If some value satisfies every relation that depends on the chosen variable, then only the subproblem for that preferred value is generated. Otherwise, a subproblem is generated for every value of the variable. The full pure literal algorithm chooses variables that have a preferred value before choosing those that do not. A recurrence equation is found for the average time used by the full pure literal rule algorithm when solving random conjunctive normal form satisfiability problems. The random problems are characterized by the number of variables ($v$), the number of clauses ($t$), and the probability that a literal is in a clause ($p$). An asymptotic lower bound analysis shows that running time is more than polynomial in $v$ when $t$ increases more rapidly than $(\ln v)^2$ (when $p$ is set to maximize the running time). A numerical study indicates that the results of the lower bound analysis are close to the true results. Thus, the full pure literal rule is faster than the simple pure literal rule (where the polynomial time boundary occurs at $t = \Theta(\ln v)$) but slower than Franco's infrequent variable algorithm (where the time is polynomial for all $p$ when $t = O(v^{1/6})$).

## 1 Introduction

The CNF satisfiability problem [6] is the first problem that was proved to be NP-complete is. For this problem set, a problem instance is a predicate. A *predicate* is the logical *and* of a series of clauses. A *clause* is the logical *or* of a series of literals. A *literal* is a variable or its negation. For a given problem instance, one wishes to know whether or not there is an assignment of values to the variables so that the predicate evaluates to *true*. The difficulty in the problem arises because a given variable can appear in several clauses. To satisfy the predicate must have at least one of its literals evaluate to *true*.

If all occurrences of a variable are without negation (*positive*) or with negation (*negative*) then the variable is *pure*. If a variable is pure, then setting the variable to the value that satisfies all of its clauses simplifies the problem without changing whether or not the problem has solutions. On the other hand, if a variable is not pure, it is not immediately clear whether it is better to set the variable to *true* or to *false*.

A *search algorithm* solves satisfiability problems by selecting some variable and then generating subproblems by setting the variable to each of its possible values. If any of the subproblems have a solution, then the original problem also has a solution. Searching leads to a number of subproblems but each subproblem is usually smaller than the original problem. A *resolution algorithm* selects a variable and then replaces the set of clauses that depend on the selected variable by an equivalent set of clauses that does not contain the selected variable. Resolution leads to a single subproblem, but it is usually larger than the original problem.

Both searching and resolution methods can be used on any problem that is formed by the logical *and* of relations. Previous research on random CNF problems has indicated that search methods are fast when solutions are rare [2, 15, 18] or common [3]. Resolution methods are fast when the number of clauses is small [4, 5, 11, 12, 13, 17]. Both types of methods are fast when solutions are extremely rare or extremely common, or when the number of clauses is extremely small. The performances of several simple satisfiability algorithms are compared in [16].

Many of the ideas for rapidly solving satisfiability problems are contained in the Davis-Putnam procedure. See [8] for the resolution version and [7] for the search version. One technique contained in the Davis-Putnam procedure is the pure literal rule: if all occurrences of a variable are positive (never negation) or negative (always negated), then only one subproblem needs to be generated; the variable can be set so that all of its clauses are satisfied. The performance of searching combined with the pure literal rule has been carefully studied [4, 11, 12, 13, 17]. These studies all assume a fixed order of assigning values to the variables.

The current paper considers the effect selecting pure literals before nonpure literals. (The Davis-Putnam procedure uses such a dynamic search order along with many additional techniques for improving performance.) This algorithm is called the *full pure literal rule algorithm*. It is shown that the full version is indeed much faster than the simple version (which uses a fixed search order) when the number of clauses is not too large.

Franco [10] considered an algorithm whose first phase uses the pure literal rule, the unit clause rule, and a limited form of resolution. Resolution is done on those variables that have exactly one positive occurrence and exactly one negative occurrence. The second phase uses searching on those variables not eliminated in the first phase. His analysis shows that the algorithm is fast when the number of clauses is not too large. Since the analysis was an upper bound analysis, one might wonder whether some simplier combination of techniques would also be fast. Previous analyses show that neither the pure literal rule with fixed search order [4, 11, 12, 13, 17] nor the unit clause rule [2, 15] with dynamic search order have all of the good running time properties that he found. There is, however, no previous study of the full pure literal rule algorithm. This paper shows that although the full pure literal rule algorithm is fast when the number of clauses is not large, Franco's algorithm can be much faster. His limited use of resolution is essential to the good performance of his algorithm.

## 2 Algorithm

The Full Pure Literal Rule Algorithms determines whether the predicate $P$ has any solutions by recursive application of the following two steps:

Step 1. If any literal in $P$ is pure, then generate the problem $P'$ by removing all clauses containing pure literals and solve $P'$ with a recursive call. The original problem is satisfiable if and only if $P'$ is.

Step 2. Otherwise, select the first remaining unset variable, and generate problems $P_t$ and $P_f$ by setting the selected variable to *true* and *false* respectively. Simplify $P_t$ and $P_f$ (by removing all clauses with true literals and removing all false literals from clauses). Solve both $P_t$ and $P_f$ with recursive calls. The original problem is satisfiable if either $P_t$ or $P_f$ is. (Both problems are solved even when the first one has a solution.)

Note that Step 1 can be considered as either a search step where just one subproblem is generated or as a resolution step where no new set of clauses is needed. The effect is the same either way. Step 2, on the other hand, is clearly a search step.

The number of function calls is used as a measure of the running time. This is equivalent to assuming that the two steps can be done in constant time. In determining whether the average time is polynomial or exponential, the time per step has little importance. (It is clear that for a problem with $t$ clauses and $v$ variables, the two steps can be done in at most $O(tv)$ time.)

The algorithm is faster if tautological clauses (those that contain a variable and its negation) are ignored, but the effect is not large unless such clauses are common ($p$ is large). Both forms of the algorithm are analyzed below.

## 3 Probability model

In the analysis random predicates are generated by the *random clause model*. This model generates random CNF satisfiability problems using a set of $v$ variables. A random clause is formed by independently selecting each of the $2v$ literals with probability $p$. A random predicate is formed by independently forming $t$ random clauses.

In this model some variables may not occur in a particular problem. Some clauses may be empty. Some clauses may be tautologies. Some clauses may be duplicates of others. Each of these features may be considered to be a defect in the model, since problems initially given to a satisfiability algorithm are not likely to have them. However, these defects are not as important as it may first appear because most of the features do occur in the subproblems that are produced by searching algorithms. The model has the advantage that the distribution of clauses does not change much when setting a variable. This leads to a relatively easy analysis. As a result more satisfiability algorithms have been analyzed with this model than with any other. The problems generated by this model can be either easy or hard (for the algorithms that have been analyzed so far) depending on how the parameters are set [3, 4, 9, 10, 14, 15, 16, 17, 18]. The model is useful for comparing strengths and weaknesses of various satisfiability algorithms.

## 4 Probabilistic Analysis

In this section, a recurrence equation for the average running time of the Full Pure Literal Algorithm is developed. There are three cases to consider: (1) the predicate has an empty clause, (2) the predicate has no empty clauses and no pure literals, or (3) the predicate has one or more new pure literals and no empty clauses. In the first case, the problem has no solution. In the second case, a nonpure literal is set. This will result in zero or more new pure literals. In the third case, all pure literals present are set at once. This removes all of the old pure literals, but zero or more pure literals are produced. In last two cases, the newly generated pure literals are a result of the algorithm simplifying the predicate by discarding true clauses.

When all of the existing pure literals in a predicate are set and the the predicate simplified, it is no longer a random predicate. Specifically, it has a different probability distribution where it is much less likely to contain pure literals. Thus in order to calculate the probability of a literal being pure it is necessary to keep track of some of the history leading to the current predicate. On the other hand, since all existing pure literals are set at once, only one level of history needs to be retained. (The details are given below.)

The main result of this section is a derivation of the following equations. The average number of nodes is given by $T(\infty, t, v)$, where

$$T(t,u,a) = [N(t)]^a + 2\sum_{w \le u-1} Q(u,u-w)T(u,w,a-1) + \sum_{w}\sum_{b\ge 1}\binom{a}{b}A(t,u,w,b)T(u,w,a-b), \quad (1)$$

$$T(t,0,a) = [N(t)]^a, \qquad T(t,u,0) = 1. \tag{2}$$

The functions $N$, $Q$ and $A$ are defined below.

In eq. (1) $t$ is the number of clauses in the previous round, $u$ is the number of clauses on the current round, and $a$ is the number of unset variables. The derivation below shows that it is appropriate to consider the "previous round" for the initial predicate to correspond to a predicate with an infinite number of clauses. Each $T$ is the product of a normizing factor times the average number of nodes in associated set of search trees. The normalizing factor is 1 for $t = \infty$, and it is selected so that the recurrences will have polynomial coefficients. See below for details. The $[N(t)]^a$ term of eq. (1) counts the root of the backtrack tree.

The middle term of eq. (1) is for the case when a nonpure literal is selected. The factor of two is the number of subproblems that are generated. The $w$ index is the number of terms that remain after one nonpure literal is set. The factor of $Q(u, u-w)$ is the probability that setting one variable in a random predicate with no pure literals results in $u-w$ of the $u$ clauses being satisfied. This is independent of $t$ and $a$. The factor $T(u, w, a-1)$ is the number of nodes in the resulting subproblem, which has $w$ clauses, has $a-1$ variables, and came from a problem with $u$ variables.

The last term of eq. (1) is for the case that a batch of pure literals are set. The $b$ index is the number of pure literals that are set, and the $w$ index is the number of clauses that remain after the pure literals are set. The binomial coefficient gives the number of ways that one can choose which $b$ of the $a$ variables are the pure literals. The $A$ factor gives the probability that setting $b$ of $a$ variables on a random that had $t$ clauses on the previous round of variable setting and has $u$ clauses currently will have $w$ clauses after the $b$ variables are set. The $T$ factor gives the average number of nodes for the resulting subproblem.

### 4.1 Calculating probabilities

In a random predicate, the probability that a variable appears positively in $i$ of $t$ clauses and negatively in 0 of $t$ clauses is

$$\binom{t}{i} p^i (1-p)^{2t-i}. \tag{3}$$

Thus, the probability that a variable is a pure literal and it appears in exactly $i$ of $t$ clauses is

$$P(t,i) = (2 - \delta_{i0}) \binom{t}{i} p^i (1-p)^{2t-i}. \tag{4}$$

The factor of 2 results from the fact that the variable may be either positive or negative provided it has at least one occurrence.

Given two disjoint subsets of clauses $X$ and $Y$, where $x = |X|$, and $y = |Y|$, the probability that a variable appears positively in $i$ of $x$ clauses, negatively in 0 of the $x$ clauses, positively in $j$ of $y$ clauses, and negatively in $k$ of the $y$ clauses is

$$\binom{x}{i} p^i (1-p)^{2x-i} \binom{y}{j} \binom{y}{k} p^{j+k} (1-p)^{2y-j-k} = \binom{x}{i} \binom{y}{j} \binom{y}{k} p^{i+j+k} (1-p)^{2x+2y-i-j-k}. \tag{5}$$

Consider a particular nonpure variable in a predicate with $t$ clauses. Now randomly remove $t-u$ of the clauses so that only $u$ clauses remain. Let $P(t,u,i)$ be the probability that the particular variable is pure in the set of $u$ clauses and it appears in $i$ of the $u$ clauses.

$$\begin{aligned} P(t,u,i) &= 2 \sum_{\substack{j \\ k \geq 1}} \binom{u}{i} \binom{t-u}{j} \binom{t-u}{k} p^{i+j+k} (1-p)^{2t-i-j-k} \\ &= 2 \binom{u}{i} p^i [(1-p)^{2u-i} - (1-p)^{t+u-i}] \end{aligned} \tag{6}$$

when $i \geq 1$. The factor of 2 allows for the fact that the particular variable may be positive or negative. The $k$ index must be at least 1 because the particular variable must occur with the opposite polarity among the

$t-u$ clauses. The $j$ index allows for the fact that the particular variable may also appear with the same polarity among the $t-u$ clauses. Also,

$$P(t,u,0)=\sum_{\substack{j\geq 1\\ k\geq 1}}\binom{t-u}{j}\binom{t-u}{k}p^{j+k}(1-p)^{2t-j-k}=[(1-p)^u-(1-p)^t]^2. \tag{7}$$

Comparing this result with the previous paragraph shows that it is natural to use notation $t=\infty$ for the initial problem since

$$P(\infty,u,i)=P(u,i). \tag{8}$$

Now consider the possibility that the particular variable will remain impure after the number of clauses is reduced. In this case there are no history effects. Let $Q(u,i,j)$ be that the particular variable appears positively in $i$ of the $u$ clauses and negatively in $j$ of the $u$ clauses.

$$Q(u,i,j)=\binom{u}{i}\binom{u}{j}p^{i+j}(1-p)^{2u-i-j}. \tag{9}$$

Consider a variable that is nonpure when $t$ clauses are present, but which has the indicated characteristic when only $u$ of the $t$ clauses remain. Define $B(t,u)$ to be the probability that the variable has become pure. Define $Q(u,i)$ to be the probability that it is not pure and appears positively in $i$ clauses. (The probability is the same that the variable is not pure and appears negatively in $i$ clauses). Define $N(u)$ to be the probability that the variable is not pure. Then

$$B(t,u)=\sum_i P(t,u,i)=[(1-p)^u-(1-p)^t][2-(1-p)^u-(1-p)^t], \tag{10}$$

$$Q(u,i)=\sum_{j\geq 1}Q(u,i,j)=\binom{u}{i}p^i(1-p)^{u-i}[1-(1-p)^u], \tag{11}$$

and

$$N(u)=\sum_{i\geq 1}Q(u,i)=[1-(1-p)^u]^2. \tag{12}$$

Note that $B(t,u)=N(t)-N(u)$, $N(\infty)=1$, and $N(0)=0$.

### 4.2 Setting pure literals in batches

To simplify the analysis the algorithm was designed to set all pure literals present in one step.

Consider a random predicate with $t$ clauses at one time and with $u$ clauses after some randomly selected clauses have been removed. Define $A(t,u,w,i)$ to be the probability that $i$ particular variables are nonpure in the set of $t$ clauses, are pure in the set of $u$ clauses, and the $i$ variables occur in $u-w$ of the $u$ clauses. When this is the case, setting these variables to their preferred value will reduce the number of clauses to $w$.

Clearly,

$$A(t,u,w,1)=P(t,u,u-w). \tag{13}$$

To calculate $A(t,u,w,i)$ for larger $i$, suppose that there are $i$ pure literals and that $i-1$ of the pure literals occur in a total of $y$ clauses and the remaining pure literal occurs in $x$ clauses. Then the probability that the $i$ literals occur a total of $z$ of the $u$ clauses is

$$\binom{y}{z-x}\left(\frac{u-x}{u}\cdots\frac{u-z+1}{u-z+x+1}\right)\left(\frac{x}{u-z+x}\cdots\frac{z-y+1}{u-y+1}\right)=\binom{y}{z-x}\binom{u-y}{u-z}\Big/\binom{u}{x} \tag{14}$$

This gives the recurrence

$$A(t,u,u-z,i)=\sum_{x,y}A(t,u,u-x,1)A(t,u,u-y,i-1)\binom{y}{z-x}\binom{u-y}{u-z}\Big/\binom{u}{x}\,, \tag{15}$$

which can be written as

$$A(t,u,w,i) = \sum_{x,y} A(t,u,x,1)A(t,u,y,i-1)\binom{u-y}{x-w}\binom{y}{w}\Big/\binom{u}{x} \tag{16}$$

Similar considerations lead to

$$A(t,u,w,i) = \sum_{x,y} A(t,u,x,j)A(t,u,y,i-j)\binom{u-y}{x-w}\binom{y}{w}\Big/\binom{u}{x} \tag{17}$$

for all $j$ in the range $1 \le j \le i-1$. This more general result is not used in the paper.

Plugging in the value for $A(t,u,x,1)$ into eq. (16) and summing over $x$ gives

$$A(t,u,w,i) = \sum_{y} \binom{y}{w} p^{y-w}\{2(1-p)^w - [(1-p)^u + (1-p)^t]\delta_{wy}\}[(1-p)^u - (1-p)^t]A(t,u,y,i-1). \tag{18}$$

Professor Bugrara has shown that

$$A(t,u,w,i) = [(1-p)^u - (1-p)^t]^i \binom{u}{w} \sum_{j} \binom{u-w}{j}(-1)^j[2(1-p)^{w+j} - (1-p)^u - (1-p)^t]^i. \tag{19}$$

This result also is not used in the paper.

Define $M(t,u,i)$ to be the probability that $i$ particular variables are nonpure in the set of $t$ clauses are pure in the set of $u$ clauses.

$$M(t,u,i) = \sum_{w} A(t,u,w,i). \tag{20}$$

Summing the recurrence equation for $A(t,u,w,i)$ over $w$ and factoring gives

$$M(t,u,i) = M(t,u,1)M(t,u,i-1) = [B(t,u)]^i. \tag{21}$$

### 4.3 Recurrence equation

Let $T(t,u,a,b)$ be the number of nodes in the search tree for a predicate that had $t$ clauses before the last batch of variables were set and now has $u$ clauses. Also the predicate has $a$ pure literals, and $b$ nonpure literals. Let $T(\infty,u,a,b)$ be number of nodes for the initial predicate. Predicates with no clauses and predicates with no variables each have one node, so

$$T(t,0,a,b) = 1 \qquad \text{and} \qquad T(t,u,0,0) = 1. \tag{22}$$

If the predicate has no pure literals, then one nonpure variable is set. The search tree has a root and two subtrees. Let $j$ be the number of clauses that disappear because they contain the selected variable, the and let $k$ be number of clauses that become shorter. (In the $j$ clauses, the literal associated with the variable is set to *true* while in the $k$ clauses it is set to *false*.) Let $c$ be the number of pure literals that arise from setting the selected variable. (In general, the value of $j$, $k$ and $c$ will be different for the two subproblems, but in the formula all values of these variables are summed over, so this difference has no significance.) The probability that the selected variable occurs positively in $j$ clauses and negatively in $k$ is $Q(u,j,k)/N(u)$. (This is also the probability that it has $j$ negative occurrences and $k$ positive ones.) Given that the selected variable occurs in $j$ clause that disappear when it is set, the probability that a particular unset variable becomes pure is $B(u,u-j)/N(u)$. The probability that $c$ pure literals result is

$$\binom{b-1}{c}\left[\frac{B(u,u-j)}{N(u)}\right]^c\left[1-\frac{B(u,u-j)}{N(u)}\right]^{b-c-1}. \tag{23}$$

Thus,

$$T(t,u,0,b) = 1 + 2\sum_{c}\sum_{\substack{j\geq 1\\ k\geq 1}} \binom{b-1}{c}\left[\frac{B(u,u-j)}{N(u)}\right]^{c}\left[1-\frac{B(u,u-j)}{N(u)}\right]^{b-c-1} \times \frac{Q(u,j,k)}{N(u)} T(u,u-j,c,b-c-1). \quad (24)$$

If the predicate has $a > 0$ pure literals, then the $a$ literals are set. The tree has a root and one subtree. Let $w$ be the number of clauses that remain after the $a$ variables are set, and let $c$ be the number of pure literals that result. The probability that $w$ clause will result from setting the $a$ pure literals is $A(t,u,w,a)/M(t,u,a)$. Given that $w$ clauses remain, the probability that $c$ pure literals are produced is

$$\binom{b}{c}\left[\frac{B(u,w)}{N(u)}\right]^{c}\left[1-\frac{B(u,w)}{N(u)}\right]^{b-c}. \quad (25)$$

Thus,

$$T(t,u,a,b) = 1 + \sum_{c}\sum_{w}\binom{b}{c}\left[\frac{B(u,w)}{N(u)}\right]^{c}\left[1-\frac{B(u,w)}{N(u)}\right]^{b-c}\frac{A(t,u,w,a)}{M(t,u,a)}T(u,w,c,b-c). \quad (26)$$

To obtain the average number of nodes for the initial problem the $T(\infty,t,a,b)$ must be averaged, allowing for the probability that a problem with $v$ variables has $c$ pure literals. The average number of nodes is

$$\sum_{c}\binom{v}{c}[B(\infty,t)]^{c}[1-B(\infty,t)]^{v-c}T(\infty,t,c,v-c). \quad (27)$$

### 4.4 Normalizing $T$.

Although the average number of nodes is a polynomial in $p$, the formula for $T(t,u,a,b)$ is a rational function in $p$. Define

$$T_1(t,u,a,b) = [N(u)]^{b}[B(t,u)]^{a}T(t,u,a,b).$$

Plugging into eqs. (22, 24, 26) gives

$$T_1(t,0,a,b) = [B(t,0)]^{a}\delta_{b0}, \quad (28)$$

$$T_1(t,u,0,0) = 1, \quad (29)$$

$$T_1(t,u,0,b) = [N(u)]^{b} + 2\sum_{c}\sum_{j\geq 1}\binom{b-1}{c}Q(u,j)T_1(u,u-j,c,b-c-1), \quad (30)$$

$$T_1(t,u,a,b) = [N(u)]^{b}[B(t,u)]^{a} + \sum_{c}\sum_{w}\binom{b}{c}A(t,u,w,a)T_1(u,w,c,b-c). \quad (31)$$

The average number of nodes is given by

$$\sum_{c}\binom{v}{c}T_1(\infty,t,c,v-c). \quad (32)$$

All the coefficients in these equations are polynomials in $p$ (with integer coefficients). Since the $T$ of highest index has coefficient 1, the solution is also a polynomial in $p$ with integer coefficients.

### 4.5 Removing an index.

All the equations involve a particular combination of the $T$'s. Define

$$T(t,u,a) = \sum_b \binom{a}{b} T_2(t,u,b,a-b). \tag{33}$$

Take linear combinations of the previous equations so that this definition applies to the left sides. The result is

$$T(t,0,a) = [N(t)]^a, \tag{34}$$

$$T(t,u,0) = 1, \tag{35}$$

$$T(t,u,a) = [N(t)]^a + 2\sum_{w\le u-1} Q(u,u-w)T(u,w,a-1) + \sum_w \sum_{b\ge 1} \binom{a}{b} A(t,u,w,b)T(u,w,a-b). \tag{36}$$

(To obtain eq. (36) it is necessary to sum eqs. (30, 31).) The average number of nodes for the original problem is

$$T(\infty,t,v). \tag{37}$$

These are the equations at the beginning of Section 4.

## 5 Ignoring tautological clauses

Now the effect of ignoring tautological clauses is considered. Since the results and the techniques are quite similar those previously considered, only a few details and the final results are given. To distinguish the two cases, all capital letters that refer to tautology ignoring algorithm will be superscripted with a $T$.

For clauses formed at random, the probability that a variable does not occur in a clause is $(1-p)$. The probability that a literal occurs positively and not negatively is $p(1-p)$. The probability that that a literal occurs negatively and not positively is the same. The sum of these three cases is $1-p^2 = (1-p)(1+p)$. If the literal occurs both ways then the clause is a tautology, and it has no relevance for the current algorithm. Variables other than the one under consideration can also affect whether a clause is a tautology or not, but the variables occur independently. Thus, if clauses are formed at random and then the tautological clauses are removed, the probability that a nontautological clause does not contain a particular variable is $(1-p)/(1+p)$, the probability that the variable occurs positively is $p/(1+p)$, and the probability that it occurs negatively is $p/(1+p)$. Define

$$q = \frac{p}{1+p}. \tag{38}$$

Adapting the earlier definitions to the current algorithm, one obtains the following results. (The derivation are line by line the same as before with little changes in the algebra. Basically, $p$ becomes $q$, $(1-p)^2$ becomes $(1-2q)$, and $1-p$ becomes $1-q$. The first two rules are not unique in its application, so one must do the algebra instead of just modifying the final answers.)

$$P^T(t,i) = (2-\delta_{i0})\binom{t}{i} q^i (1-2q)^{t-i}. \tag{39}$$

$$P^T(t,u,i) = 2\binom{u}{i} q^i (1-2q)^{u-i}[1-(1-q)^{t-u}]. \tag{40}$$

$$P^T(t,u,0) = (1-2q)^u[1-2(1-q)^{t-u}+(1-2q)^t]. \tag{41}$$

$$Q^T(u,i,j) = \binom{u}{i}\binom{u}{j} q^{i+j}(1-2q)^{u-i-j}. \tag{42}$$

$$B^T(t,u) = 2(1-q)^u - (1-2q)^u - 2(1-q)^t - (1-2q)^t. \tag{43}$$

$$Q^T(u,i) = \binom{u}{i} q^i [(1-q)^{u-i} - (1-2q)^{u-i}]. \tag{44}$$

$$N^T(u) = 1 - 2(1-q)^u + (1-2q)^u. \tag{45}$$

Note that $B^T(t,u) = N^T(t) - N^T(u)$, $N^T(\infty) = 1$, and $N^T(0) = 0$.

$$A^T(t,u,w,1) = P^T(t,u,u-w). \tag{46}$$

$$A^T(t,u,w,i) = \sum_y \binom{y}{w} \{2q^{y-w}(1-2q)^w[(1-q)^{u-y} - (1-p)^{t-y}] - \delta_{wy}[(1-2q)^u - (1-2q)^t]\} A^T(t,u,y,i-1). \tag{47}$$

$$M^T(t,u,i) = [B^T(t,u)]^i. \tag{48}$$

After normalizing and removing an index, the recurrence becomes

$$T^T(t,0,a) = [N^T(t)]^a, \tag{49}$$

$$T^T(t,u,0) = 1, \tag{50}$$

$$\begin{aligned} T^T(t,u,a) = [N^T(t)]^a &+ 2 \sum_{w \le u-1} Q^T(u,u-w) T^T(u,w,a-1) \\ &+ \sum_w \sum_{b \ge 1} \binom{a}{b} A^T(t,u,w,b) T^T(u,w,a-b). \end{aligned} \tag{51}$$

The probability that a clause in $v$ variables is a tautology is $(1-p^2)^v$, so the probability that a predicate contains $l$ nontautological clauses is

$$\binom{t}{l} (1-p^2)^{lv} [1 - (1-p^2)^v]^{t-l},$$

and the average number of nodes is given by

$$\sum_l \binom{t}{l} (1-p^2)^{lv} [1 - (1-p^2)^v]^{t-l} T^T(\infty, l, v).$$

## 6 Lower bound

All the terms in the recurrence for $T(t,u,v)$ are positive, so dropping terms gives a lower bound on $T$. In particular

$$T(t,u,v) \ge [N(t)]^v + 2 \sum_{w \le u-1} Q(u,u-w) T(u,w,v-1). \tag{52}$$

The same techniques used to study this equation still apply if the $b = 1$ term from eq. (36) is also included, but the algebra becomes longer and the result is not much better under the conditions of this paper. Therefore, the discussion is limited to eq. (52).

Define

$$U(t,u,v) = [N(t)]^v + 2 \sum_{w \le u-1} Q(u,u-w) U(u,w,v-1). \tag{53}$$

Then $T(t,u,v) \ge U(t,u,v)$.

Since

$$U(t,u,v) - [N(t)]^v = 2 \sum_{w \le u-1} Q(u,u-w) U(u,w,v-1) \tag{54}$$

is independent of $t$, define $V(u,v) = U(t,u,v) - [N(t)]^v$. Then

$$V(u,v) = 2[N(u,v)]^v + 2 \sum_{w \le u-1} Q(u, u-w) V(w, v-1). \tag{55}$$

Plugging in the formulas for $N$ and $Q$ gives

$$V(u,v) = 2[1-(1-p)^u]^{2v} + 2[1-(1-p)^u] \sum_{w \le u-1} \binom{u}{w} p^{u-w} (1-p)^w V(w, v-1). \tag{56}$$

The boundary conditions also independent of $t$. They are

$$V(0,a) = 0, \qquad V(u,0) = 0. \tag{57}$$

This recurrence is similar to the recurrence for the simple pure literal rule algorithm [17]. The recursive term is identical, but for the full algorithm, the nonrecursive term, $2[1-(1-p)^u]^{2v}$, is small for certain values of $v$, $p$, and $u$. This lower bound analysis shows that any advantage that the full pure literal literal rule has over the simple pure literal rule comes from the ability of this term to sometimes cause the exponential growth associated with the algorithm to begin from a very low level. (Since the only difference between the full and the simple algorithm is the search order, and since the search order is better for the full algorithm, the full algorithm has a search tree that is at least as small as the simple algorithm. See [3] for a full proof of a similar result.)

If one assumes that $V(u,v)$ is an increasing function of $u$, then $V(u,v) \ge W(u,v)$, where $W$ is the solution of the recurrence

$$W(u,v) = 2[1-(1-p)^u]^{2v} + 2[1-(1-p)^u] \Bigg[ \sum_{u-k \le w \le u-1} \binom{u}{w} p^{u-w} (1-p)^w \Bigg] W(u-k, v-1), \tag{58}$$

which is, in effect, a first order linear recurrence equation for $W$. (This result holds for any $k$.) If $V(u,v)$ is not an increasing function of $v$ then one has a similar equation with $W(u-k, v-1)$ replaced by $W(u_*, v-1)$ where $u_*$ is the index in the range $u-k$ to $u-1$ that results in the smallest $W$ value. By the end of the derivation it will be clear that if the value for $u_*$ is not $u-k$, then an even larger lower limit results, so the assumption that $V(u,v)$ is increasing is not critical to the final lower bound result.

If $k$ is selected so that the sum starts a little before the peak of the binomial distribution (which is at $w = pu$), then the sum is close to $1-(1-p)^u$. Using Chernoff bounds [1] one has for for any small $\beta > 0$ and $k = up(1+\beta)$

$$W(u,v) \ge 2[1-(1-p)^u]^{2v} + 2[1-(1-p)^u]^2 [1 - e^{-\beta^2 pu/3}] W(u(1-(1+\beta)p), v-1), \tag{59}$$

To obtain a simpler lower bound, drop the first term on the right side of the equation for $W$ except at the boundary $u = u_0$, $v = v_0$. At the boundary drop the second term. Let the final value of $u$ be $t$ and the initial value be $xt$. Use a constant step size of $(1+\beta)pt$ for the $u$ index. (This was the largest step size that was used.) It takes $(1-x)/[(1+\beta)p]$ steps to cover the range from $xt$ to $t$. Set all coefficients to their minimum value over the range. Thus,

$$X(u,v) = 2[1 - e^{xt \ln(1-p)}]^2 [1 - e^{-\beta^2 xpt/3}] X(u - (1+\beta)pt, v-1), \tag{60}$$

with the boundary condition

$$X(xt, v - (1-x)/[(1+\beta)p]) = 2[1 - e^{xt \ln(1-p)}]^{2v - (1-x)/[(1+\beta)p]} \tag{61}$$

is a lower bound for $V(u,v)$.

The solution at $u = t$ to this equation is

$$X(t,v) = 2[1 - e^{xt\ln(1-p)}]^{2v}\{2[1 - e^{xt\ln(1-p)}][1 - e^{-\beta^2 xpt/3}]\}^{(1-x)/[(1+\beta)p]}. \tag{62}$$

Now make $t$ as large while $X(t,v)$ is $\Theta(v^n)$. This can be accomplished by setting

$$\beta \text{ to a small positive constant,} \tag{63}$$

$$x = \frac{1}{2}, \tag{64}$$

$$p = \frac{\ln 2}{2(1+\beta)(1+\ln v)n}, \tag{65}$$

$$t = \frac{4(1+\beta)(1+\ln v)[\ln 2v - \ln n - \Theta((\ln v)/v)]n}{\ln 2}. \tag{66}$$

Since this is a lower bound analysis, to verify these claims, one just needs to show that these values lead to $X(t,v) = \Theta(v^n)$. The details are omitted. To derive these values, notice that the $[1 - e^{xt\ln(1-p)}]^{2v}$ term is near one so long as $e^{xt\ln(1-p)}$ is above about $1/v$ (which means that $p$ must not be too small) while the exponent $(1-x)/[1+\beta)p]$ increases as $p$ becomes small. Thus the maximum with respect to $p$ occurs when $e^{xt\ln(1-p)}$ is equal to a small factor times $1/v$. This implies that the last exponential factor has a base that is almost 2. The above results are obtained after some additional algebra.

## 7 Numerical investigations

There are many places where one could make an error in the derivation of eq.(1). To reduce the chances of such an error, the recurrence was checked by computing its solution for $1 \le t \le 6$, $1 \le v \le 6$ using a Maple program. For each value of $t$ and $v$, the solution is a polynomial in $p$ with integer coefficients and degree $2tv$. The full pure literal algorithm was also programmed in Pascal and run on all problems in this range with the added condition that $tv \le 12$. Number of nodes for each problem was multiplied by the polynomial giving the probability of the problem instance. The polynomials generated by the Pascal program were identical to the corresponding polynomials from the Maple program.

To obtain a general idea of the behavior of the full pure literal rule algorithm, eqs. (1, 2) were solved for $v = 50$ and $2 \le t \le 153$. Figure 1 shows the results of these calculations. Figure 2 shows the results of similar calculations for the version of the full pure literal rule that ignores tautological clauses. The reader may wish to compare this figure with the figures in [16], where the same data is given for other algorithms. The simple pure literal algorithm has similarly shaped contours, but the full versions of the algorithm remains fast for much large values of $t$.

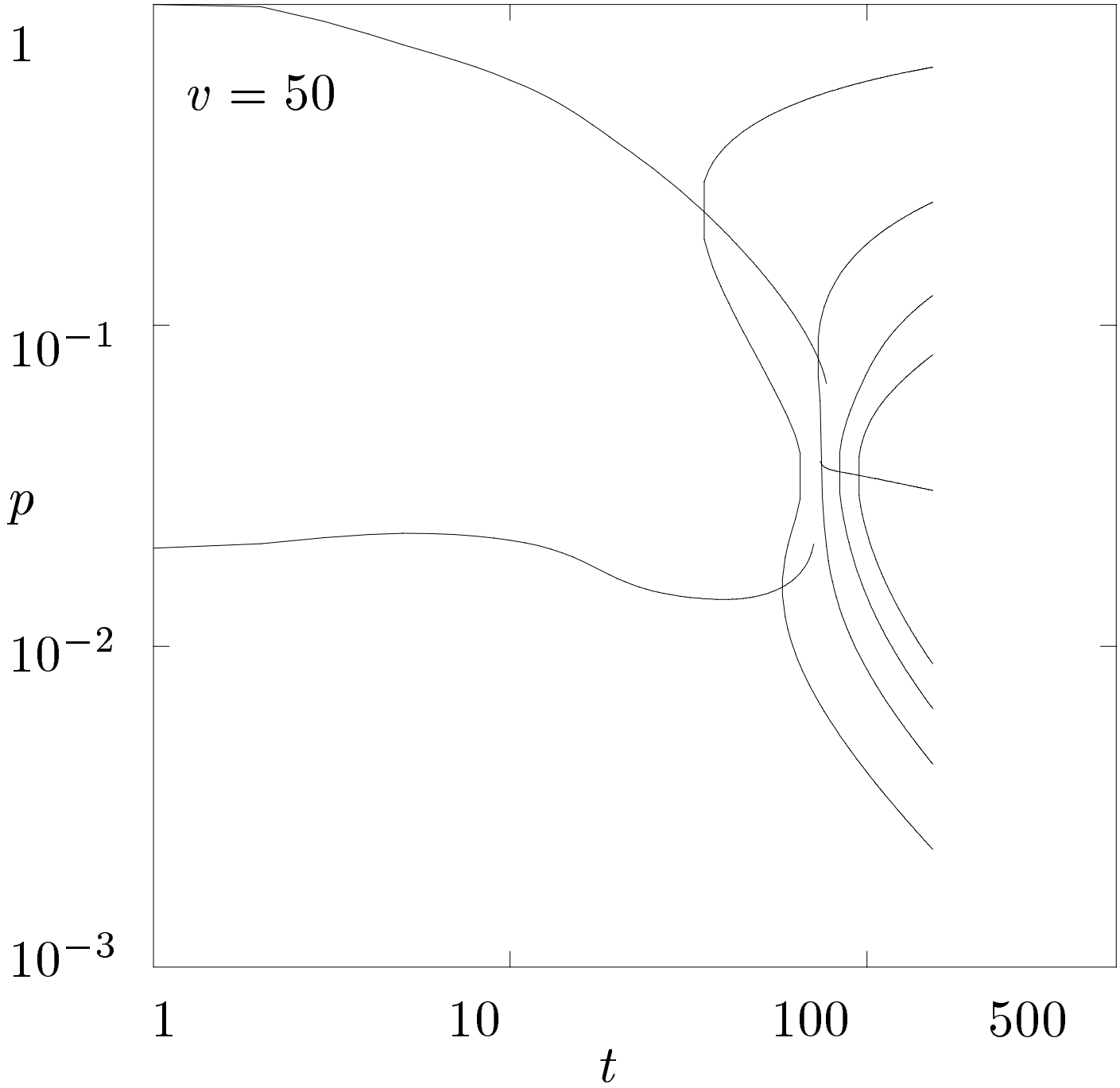


Fig. 1. The figure shows the average running time of the full pure literal rule algorithm as a function of $p$ and $t$. The outer contour shows for each $t$ the value of $p$ that results in an average of 50 nodes. Proceeding inward, contours for an average of $50^2$, $50^3$, and $50^4$ are also shown. The line that starts on the right side and then splits into an upper segment and a lower segment shows for each $t$ which value of $p$ results in the largest average number of nodes. For $t$ below 57, there are two values of $p$ that result in the number of nodes being a local maximum. Consecutive points on a contour are connected by a straight line.

Numerical studies suggest that the results of the lower bound analysis are close to the true results. In particular, Table 1 shows for various value of $v$ the smallest value of $t$ that results in a running time of $v^n$ for small integer $n$. Since the running time is usually not exactly $v^n$ (for the worst $p$ value) for integer $t$, the tabulated values were obtained by using linear interpolation between the two closest integer $t$ values.

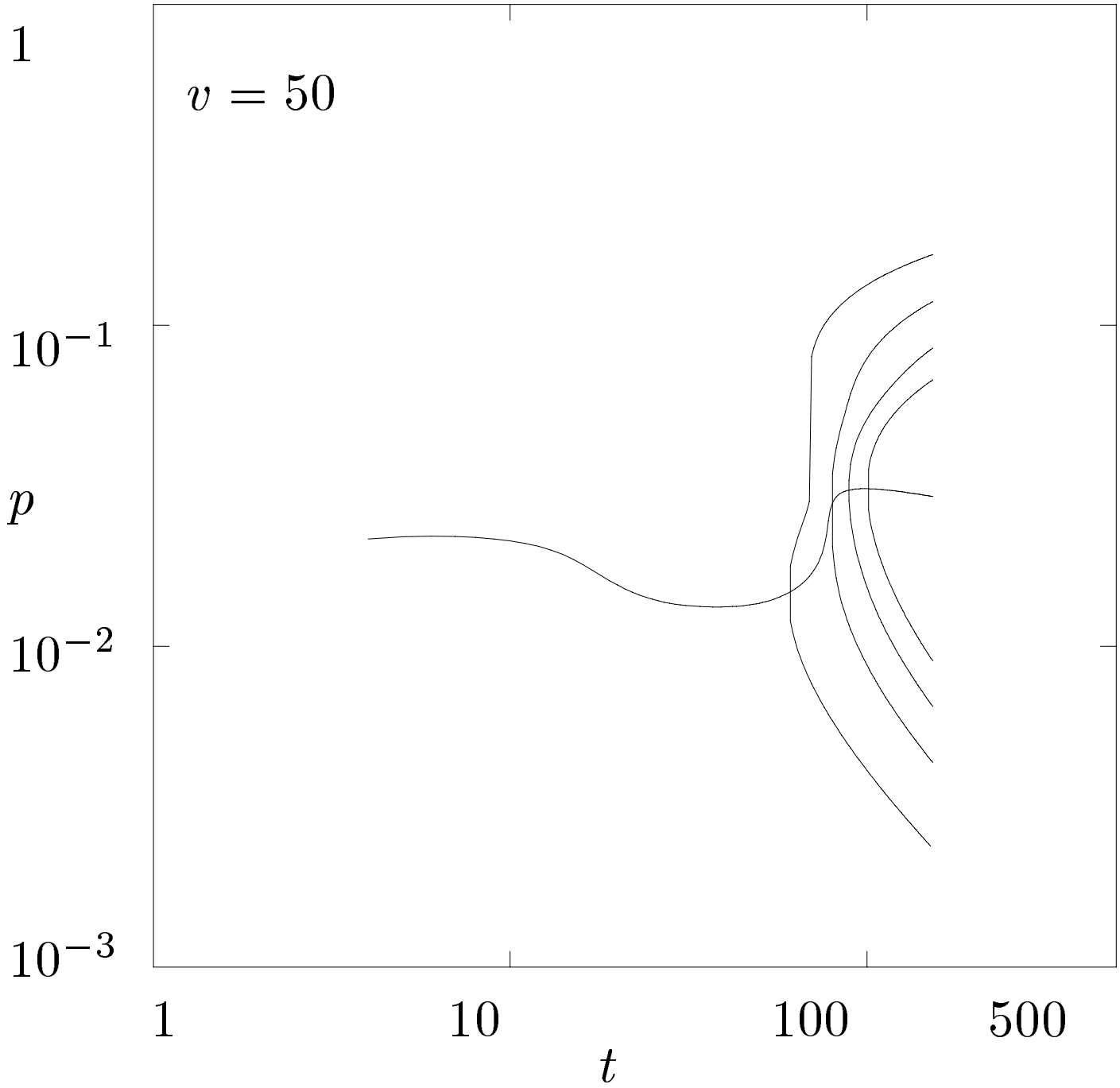


Fig. 2. The figure shows the average running time for the version of the full pure literal rule algorithm that ignores tautological clauses. The outer contour shows for each $t$ the value of $p$ that results in an average of 50 nodes. Proceeding inward, contours for an average of $50^2$, $50^3$, and $50^4$ are also shown. The line that goes from the left side to the right side shows for each $t$ which value of $p$ results in the largest average number of nodes.

| $v$ | $n=1$ | $n=2$ | $n=3$ | $n=4$ |
|---|---|---|---|---|
| 25 | 20.32 | 39.52 | 50.22 | 67.20 |
| 30 | 23.55 | 46.52 | 56.60 | 70.91 |
| 35 | 26.50 | 53.53 | 63.20 | 76.01 |
| 40 | 29.23 | 60.49 | 69.90 | 81.75 |
| 45 | 31.75 | 67.12 | 76.61 | 87.86 |
| 50 | 34.10 | 72.60 | 83.38 | 94.17 |
| 100 | 51.89 | 112.50 | 155.50 | |
| 175 | 69.11 | | | |
| 400 | 99.14 | | | |

Table 1. For the pure literal algorithm and various $v$ the value of $t$ that results in an average runing time of $v^n$ (for the most difficult value of $p$) is given. Linear interpolation is used between the two nearest integer $t$ values.

| $v$ | $n=1$ | $n=2$ | $n=3$ | $n=4$ |
|---|---|---|---|---|
| 25 | 0.2716 | 0.2991 | 0.2806 | 0.3078 |
| 30 | 0.2836 | 0.3152 | 0.2814 | 0.2872 |
| 35 | 0.2935 | 0.3318 | 0.2862 | 0.2792 |
| 40 | 0.3019 | 0.3484 | 0.2930 | 0.2770 |
| 45 | 0.3091 | 0.3632 | 0.3008 | 0.2781 |
| 50 | 0.3154 | 0.3722 | 0.3094 | 0.2810 |
| 100 | 0.3531 | 0.4181 | 0.4128 | |
| 175 | 0.3790 | | | |
| 400 | 0.4110 | | | |

Table 2. The value of $a$ for the formula

$$t = a\frac{4(1+\ln v)(\ln 2v - \ln n)n}{\ln 2},$$

where $t$ is the value given in Table 1.

Table 2 shows the coefficient $a$ that fits the $t$ value in Table 1 to the formula

$$t = a\frac{4(1+\ln v)(\ln 2v - \ln n)n}{\ln 2}, \tag{67}$$

which is eq. (66) with the extra coefficient $a$ and with small terms dropped. For all large values of $v$ the value of $a$ should be below 1 ($v$ needs to be large enough that the $1+\beta$ factor and the $\Theta$ term are not important). If the lower bound analysis gave the actual running time, the values would approach 1 as $v$ became large.

The lower bound analysis shows that for the full pure literal rule algorithm the $t$ value that results in $v^n$ nodes (with the most difficult $p$ value) is $O(n(\ln v)^2)$. The analysis of the simple pure literal rule algorithm combined with the fact that the full algorithm has less nodes shows that the $t$ value is $\Omega(n \ln v)$ [17]. In particular, a $t$ value of $\Theta(n(\ln v)^b)$ for $b$ near 1, would be expected to give coefficients in Table 2 that approached 0. The $v$ values in Tables 1 and 2 are large, but $\ln v$ is the important parameter in the formulas, so some caution is needed in interpreting these results, but the results do suggest that the value for an average running time of $v^n$ (with the worst $p$ value) is $t = \Theta(n(\ln v)^2)$.

Figure 3 shows the average number of nodes that result for several satisfiability algorithms when $p$ is set to the value that results in the largest average. Each algorithm and $t$ value leads to a different $p$ value. The top of the figure shows $2^{51}-1$ nodes, the value appropriate for a naive search algorithm.

All pure literal algorithms are fast when the number of clauses is small. The full pure literal rule remains fast for somewhat larger values of $t$, but once $t$ becomes somewhat larger, its time increases rapidly so that it loses most of its advantage over the simple pure literal rule algorithm.

## 8 Conclusions

All forms of the pure literal rule algorithm are fast for problems where the number of clauses is bounded by a constant times the logarithm of the number of variables. (These are problems with very few clauses indeed.) The full pure literal rule remains fast when the number of clauses increases to a constant times the square of the logarithm of the number of variables. This improvement can be quite important when there are only 50 variables, but when the number of variables becomes large, these problems also have very few clauses.

When the number of variables is large, Franco's improvement to the pure literal rule is much faster. For random problems where the probability is not near the value that results in hard problems and $t$ is large backtracking algorithms are fast [16].

In practice, the user does not need to decide between these various algorithms. One can include backtracking, the full pure literal rule, and Franco's limited resolution all in one algorithm. The resulting

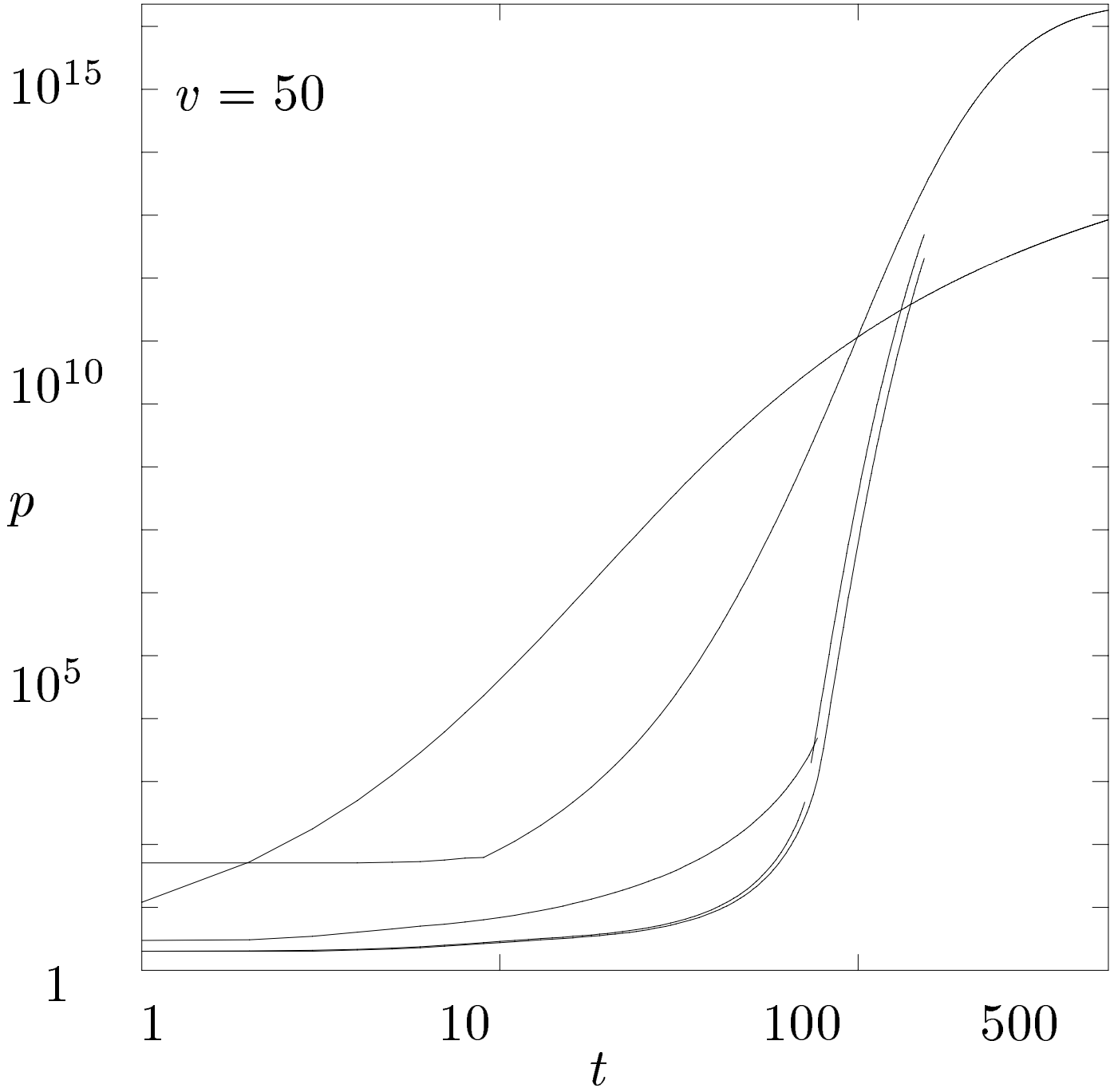


Fig. 3. The average number of nodes for various search algorithms is shown. The number of variables is 50. For each algorithm and each $t$, $p$ has been adjusted to make the average number of nodes as large as possible. The top of the graph is at $2^{51} - 1$, the number of nodes for a naive search algorithm. Results for the following algorithms are shown: simple pure literal rule (upper most curve at $t = 500$), clause order backtracking (lower curve at $t = 500$), full pure literal rule (upper most curve of the two that stop at $t = 153$; for $t \leq 57$ curves for both local maxima of this algorithm are shown), full pure literal rule combined with ignoring of tautological clauses (lower most of the two that stop at $t = 153$.

algorithm is able to process each node almost as fast as a simple algorithm while taking advantage of each technique's ability to reduce the number of nodes in the search tree.

**Acknowledgement.** Extensive assistance of Professor Khaled Bugrara with this research is greatly appreciated. I also wish to thank Northeastern university for the use of computer resources for some of the

calculations in this paper.

**References**


1. D. Angluin and L. G. Valiant, *Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings*, Journal of Comput. and System Sciences **18** (1979) pp 155–193.
2. Cynthia A. Brown and Paul Walton Purdom Jr. *An Average Time Analysis of Backtracking*, SIAM J. Comput. **10** (1981) pp 583–593.
3. Khaled Bugrara and Paul Walton Purdom, Jr., *Average Time Analysis of Clause Order Backtracking* (1991).
4. Khaled Bugrara, Youfang Pan, and Paul Purdom, *Exponential Average Time for the Pure Literal Rule*, SIAM J. Comput. **18** (1988) pp 409–418.
5. Vašek Chvátal and Endre Szemerédi, *Many Hard Examples for Resolution*, JACM **35** (1988) pp 759–770.
6. Stephen A. Cook, *The complexity of Theorem-Proving Procedures*, Proc. 3rd ACM Symp. on Theory of Computing, ACM, New York (1971) pp 151–158.
7. Martin Davis, George Logemann, and Donald Loveland, *A Machine Program for Theorem Proving*, C. ACM **5** (1962), pp. 394–397.
8. Martin Davis and Hilary Putnam, *A Computing Procedure for Quantification Theory*, JACM, **7** (1960) pp 201–215.
9. John Franco, *On the Occurrence of Null Clauses in Random Instances of Satisfiability*, Indiana University Computer Science Tech. Report 291 (1989).
10. John Franco, *Elimination of Infrequent Variables Improves Average Case Performance of Satisfiability algorithms*, SIAM J. Comput. **20** (1991) pp 1119–1127.
11. Allen Goldberg, *On the Complexity of the Satisfiability Problem*, Courant Computer Science Report No. 16, New York University, New York.
12. Allen Goldberg, *Average Case Complexity of the Satisfiability Problem*, Proc. Fourth Workshop on automated Deduction, 1979, pp 1–6.
13. Allen Goldberg, Paul Purdom, Cynthia Brown, *Average Time Analyses of Simplified Davis-Putnam Procedures*, Information Processing Letters, **15** (1982) pp 72–75. Printing errors corrected in **16** (1983) pp 213.
14. Kazuo Iwama, *CNF Satisfiability Test by Counting and Polynomial Average Time*, SIAM J. Comput. **18** (1989) pp 385–391.
15. Paul W. Purdom, *Search Rearrangement Backtracking and Polynomial Average Time*, Artificial Intelligence **21** (1983) pp 117–133.
16. Paul W. Purdom, *A Survey of Average Time Analyses of Satisfiability Algorithms*, Journal of Information Processing **13** (1990) pp 449–455. This originally appeared as *Random Satisfiability Problems*, Proc. of the International Workshop on Discrete Algorithms and Complexity, The Institute of Electronics, Information and Communication Engineers, Tokyo (1989) pp 253–259.
17. Paul Walton Purdom, Jr. and Cynthia A. Brown, *The Pure Literal Rule and Polynomial Average Time*, SIAM J. Comput. **14** (1985) pp 943–953.
18. Paul W. Purdom and Cynthia A. Brown, *Polynomial-Average-Time Satisfiability Problems*, Information Sciences **41** (1987) pp 23–42.