Enumerative Counting is Hard

Jin-yi Cai and Lane A. Hemachandra

YALEU/DCS/TR-585
January 1988

Cai - Research supported by a Sage Fellowship and NSF grant DCR-83301766 while at Cornell's Computer Science Department, and by NSF grant CCR-8709818.
Hemachandra - Research supported by a Fannie and John Hertz Foundation Fellowship, NSF grants DCR-8301766 and DCR-85220597, and a Hewlett-Packard Corporation equipment grant.

Abstract

An n-variable Boolean formula may have anywhere from 0 to 2^n satisfying assignments. Can a polynomial-time machine, given such a formula, reduce this exponential number of possibilities to a small number of possibilites?

We call such a machine an enumerator and prove that if there is a good polynomial-time enumerator for #P (i.e., one where the small set has at most $O(|f|^{1-\epsilon})$ numbers), then P = NP = $P^{\#P}$ and probabilistic polynomial time equals polynomial time. Furthermore, we show that #P and enumerating #P are polynomial-time Turing equivalent.

1 Introduction

#P is the class of functions that count the accepting paths of some NP machine [Valb,Ang,Sto, Wag]. Valiant proved that these functions can count the number of cliques of a given size, compute the permanent of a matrix, and solve many other counting versions of NP and P problems [Valb, Vala]. $P^{\#P}$ is the class of languages computable by polynomial-time machines given an oracle from #P.

#P questions seem computationally hard. Though they can be answered by brute force in PSPACE, it is not known if they are in the polynomial hierarchy. Currently known structural relations are that $P^{\#P} \supseteq \Delta_2^p$, and, in a certain relativized world A, $P^{\#P^A}$ properly contains $\Sigma_2^{p,A} \cup \Pi_2^{p,A}$ [Ang].

The complexity of the class #P is usually studied by considering #SAT. Cook's reduction [Coo,HU] from NP machines to formulas can be made parsimonious [GJ, page 169][Simb, Valb]. Thus #SAT, the function mapping from Boolean formulas f to their numbers of solutions $||f||$ (e.g., #SAT($x_1 \vee x_2$) = 3), is a canonical hardest #P function. We speak interchangeably of computing #P and #SAT, as #SAT can be computed if and only if #P can.

Stockmeyer [Sto] has analyzed the complexity of $r(\cdot)$-approximating #SAT in the sense of approximating the value within a multiplicative factor—finding a function g such that:

$$\frac{||f||}{r(|f|)} \leq g(f) \leq ||f||\, r(|f|),$$

where $|f|$ stands for the size of f. He shows that in Δ_3^p we can $(1+\epsilon|f|^{-d})$-approximate #SAT. This is a good bound when the number of solutions $||f||$ is relatively small. However for formulas with *many* solutions, the size of the set of possible values that the approximation admits may be exponentially large in terms of $|f|$.

In his paper [Sto], Stockmeyer also shows that there is a relativized world where for no constant k can #SAT be k-approximated even with a Δ_2^p function. Though this does not prove that #SAT is hard to approximate, the result can be taken as evidence that we lack the mathematical tools needed to determine the complexity of approximating #SAT.

In this paper, we consider a different approach to approximately solving #P problems. In *an enumeration scheme* one tries to reduce the number of possible values of $||f||$ by giving a small list of possibilities for $||f||$. We show that the existence of a good enumerator that reduces the number of values substantially would imply that $\mathrm{P} = \mathrm{NP} = \mathrm{P}^{\#\mathrm{P}}$, and probabilistic polynomial time equals polynomial time. Thus we believe '#SAT *is* hard to enumerate' with greater certainty than we believe '$\mathrm{P} \neq \mathrm{NP}$.'

A function A is said to *$s(\cdot)$-enumerate* #SAT if $A(f)$ is a list of at most $s(|f|)$ integers between 0 and $2^{|f|}$ in which $\|f\|$ appears. If A can be computed in polynomial time, we call it an *$s(\cdot)$-P-enumerator* for #SAT. For example, a 4–P–enumerator given $x_1 \vee x_2 \vee x_3$ might reply $\{0, 1, 4, 7\}$; the enumerator states that the formula has either 0, 1, 4 or 7 satisfying assignments (indeed, it has 7).

Though enumerators reduce the number of possible solutions to a small set, the values in this set my vary over a great range. Thus enumeration is neither strictly broader nor strictly weaker than the type of approximation studied by Stockmeyer.

Section 2 introduces our proof techniques in a simple setting. If #SAT has a k–P–enumerator then $\mathrm{P} = \mathrm{NP}$. Section 3 extends this result to show that if *Enum* is a function n^{α}–enumerating #SAT, $\alpha < 1$, then $\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{Enum}$. Thus enumerative counting and exact counting are polynomial-time Turing equivalent. In particular, if #SAT has an n^{α}–P–enumerator ($\alpha < 1$), then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

These results demonstrate that efficiently enumerating #P implies $\mathrm{P} = \mathrm{NP}$. Thus we have structural evidence that #P can not be easily enumerated.

Our proof uses an arithmetic of formulas that extends the work of Papadimitriou and Zachos [PZ]. Section 4 presents an immediate consequence of this: $\mathrm{NP}^{\#\mathrm{P}} = \mathrm{NP}^{\#\mathrm{P}[1]}$, where [1] indicates that each computation path of the NP machine makes at most one oracle call. We invite comparison between this n^k-for-one result over NP machines, and the recently developed theory of polynomial terseness [AG,GJY,BGGO].

2 If #SAT can be k–enumerated then $\mathrm{P} = \mathrm{NP}$

The proofs of this section and Section 3 have the same architecture. We develop a novel technique to repeatedly expand and prune a formula tree. In this section we keep the tree constantly thin. Section 3 allows trees that are polynomially bushy.

This section shows that if #SAT can be k–enumerated then $\mathrm{P} = \mathrm{NP}$. First, we state a lemma which says that we can easily combine many small formulas into a single larger formula. This lemma generalizes a technique developed by Papadimitriou and Zachos in their proof that $\mathrm{P}^{\mathrm{NP}[\log]} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$ [PZ]. The single large formula has the property that, given its number of solutions, we can quickly determine the number of solutions of each of the small formulas.

Lemma 2.1 *There are polynomial-time functions combiner and decoder such that for any Boolean formulas f and g, combiner(f, g) is a Boolean formula and decoder$(\|$combiner$(f, g)\|)$ prints $\|f\|, \|g\|$.*

Proof Let $f = f(x_1, \ldots, x_n)$ and $g = g(y_1, \ldots, y_m)$, where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are distinct. Let z and z' be two new Boolean variables. Then

$$h = (f \wedge z) \vee (\bar{z} \wedge x_1 \wedge \cdots \wedge x_n \wedge g \wedge z') \tag{1}$$

is the desired combination, since $\|h\| = \|f\| 2^{m+1} + \|g\|$ and $\|g\| \leq 2^m$. **QED**

We can easily extend this technique to combine three, four, or even polynomially many formulas. (Using Cook's connection between formulas and machines, one can equivalently view this result as about counting and combining NP machines and accepting paths.) For example, to combine three formulas $f(x_1, \ldots, x_l)$, $g(y_1, \ldots, y_m)$, $h(z_1, \ldots, z_n)$, our combining formula would be,

$$h = (f \wedge z) \vee \{\bar{z} \wedge x_1 \wedge \cdots \wedge x_l \wedge [(g \wedge z') \vee (\bar{z}' \wedge y_1 \wedge \cdots \wedge y_m \wedge h \wedge z'')]\}. \tag{2}$$

Now we show that if #SAT has a k-P-enumerator then P = NP.

Theorem 2.2 *If* #SAT *can be* k-P-*enumerated then* P = NP.

Proof Say we are given a formula $F(x_1, \ldots, x_n)$ and we would like to know if $F \in$ SAT. We substitute variables one at a time so that we always have a set S of at most k partial assignments satisfying:

($\star$) $F \in \text{SAT} \iff$ there is a satisfying assignment extending a partial assignment in S.

Each stage assigns a new variable and has three steps. Initially, S consists of the empty assignment.

Stage i:

1. EXPAND TREE: For each partial assignment in S, assign the variable x_i both true and false (Figure 1A). Applying these assignments to F, we have at most $2k$ formulas.

2. COMBINE FORMULAS and RUN ENUMERATOR: Combine the $2k$ formulas into a single super-formula as described in Lemma 2.1. Run our k-P-enumerator on that super-formula. The enumerator prints k guesses for the number of solutions of the super-formula, which is translated immediately to k vectors of guesses of the number of solutions of the $2k$ formulas. (For example, in Figure 1B (where $k = 3$) the first guess says that the four little formulas in our super-formula have, respectively, $7, 0, 3$, and 3 solutions.)

3. PRUNE THE TREE: Note that if $F \in$ SAT, then at least one of the $2k$ formulas is satisfiable, by the inductive hypothesis ($\star$). Thus, if these k guesses are all zero vectors then the formula is unsatisfiable. Suppose they are not all zero vectors, we can choose a set T of at most k columns so that each nonzero row of our guess matrix (Figure 1B) has a nonzero in a column in T. For example, we let $T = \{p \mid$ a row of the guess matrix has its first nonzero entry in column $p\}$.

   Now we prune the tree by setting S to the partial assignments corresponding to the columns in T (there are at most k). Suppose $F \in$ SAT. Then by the inductive hypothesis, the true guess is a nonzero vector, thus by our choice of T the true guess has a nonzero in some column of T. We have assigned another variable and maintained invariant ($\star$).

End of Stage i.

At the final stage all variables are assigned and we just have to look at our set of k complete assignments to F and see if any of them satisfies F. We know by $(\star)$ that F is satisfiable if and only if one of these assignments satisfies F. **QED**

3 Main Result

This section demonstrates that enumerate and exact counting are Turing equivalent. This strengthens the result of the previous section.

Theorem 3.1 *If Enum is an n^α-enumerator for* #SAT, $\alpha < 1$, *then*

$$\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{Enum}.$$

Corollary 3.2 *If* #SAT *can be n^α*-P-*enumerated,* $\alpha < 1$, *then* $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Since $\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{PP}}$ [Gil,Sima], where PP is probabilistic polynomial time, the existence of good enumerators for #SAT also implies that probabilistic and deterministic polynomial time are equivalent.

Corollary 3.3 *If* #SAT *can be n^α*-P-*enumerated,* $\alpha < 1$, *then* $\mathrm{P} = \mathrm{PP}$.

Theorem 3.1 differs from Theorem 2.2 in two important ways. One is that we are satisfied with an n^α–enumerator, $\alpha < 1$. The more interesting point is that we conclude $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. We now discuss each of these improvements.

3.1 How to Count Solutions

The first major change is that we find out not only if a formula is satisfiable, but also how many satisfying assignments it has. We do this with a more rigorous analysis of the guess matrix and a refined pruning strategy.

Lemma 3.4 *If* #SAT *can be k*-P-*enumerated then* $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Proof Consider the tree pruning procedure in Theorem 2.2. Here we want to keep a set S of p $(1 \le p \le k)$ leaves in the partially grown tree, such that $\langle \|f_1\|, \|f_2\|, \ldots, \|f_p\|\rangle$ uniquely determine $\|f\|$, where f_j is the formula obtained from f by the partial assignment associated with the jth leave in S. (We will speak interchangeably of the jth leave in S and f_j.)

Again we substitute variables one at a time. Inductively, for the formulas $f_1, \ldots, f_p$ in S, we wish to maintain at most k vectors u_i $(i = 1, \ldots, q,\ q \le k)$ of dimension p, and integers $s_1, \ldots, s_q$, such that the following conditions hold.

1° $(\forall i)[u_i \ne 0]$,

2° $(\forall i \ne j)[u_i \ne u_j]$, and

3° $f \in \mathrm{SAT} \Longrightarrow (\exists i)[u_i = \langle \|f_1\|, \ldots, \|f_p\|\rangle \;\&\; s_i = \|f\|]$.

Notice that when the tree has fully grown, for the formulas $f_1, \ldots, f_p$ in S it can be easily checked whether some $u_i = \langle \|f_1\|, \ldots, \|f_p\| \rangle$. If no such u_i exists, then f is unsatisfiable, by 3°. If such u_i exists, we know from 1° that f is satisfiable, and the u_i must be unique, by condition 2°. And thus by condition 3° we can output $\|f\| = s_i$.

The proof is a double induction. Each stage has the same general structure as before in the proof of Theorem 2.2.

Initially S consists of f (or, the empty assignment) and we apply our enumerator on f. If the enumerator guesses all zeros, then output $\|f\| = 0$. Otherwise, let $u_1, \ldots, u_q$ be all the distinct nonzero guesses $(1 \leq q \leq k)$, and $s_i = u_i$ trivially.

We inductively maintain 1°, 2° and 3° as we go along, and at each stage we use a second induction for the tree pruning process.

Stage l:

1. EXPAND TREE: For each partial assignment in S, assign the variable x_l both true and false. We have r new leaves, where $2 \leq r = 2|S| \leq 2k$.

2. COMBINE FORMULAS and RUN ENUMERATOR: Combine f with the formulas $f_1, \ldots, f_r$ associated with these new leaves. Let G be the resulting "super-formula." Run our k-enumerator of G. We obtain at most k guesses for the number of solutions of G. Using our decoder (see Section 3.2) we get up to k distinct vectors, say $\widehat{v_1}, \ldots, \widehat{v_{q'}}$, $1 \leq q' \leq k$, where $\widehat{v_i} = \langle v_{i\,0}, v_{i\,1}, \ldots, v_{i\,r} \rangle$ is a guess for $\langle \|f\|, \|f_1\|, \ldots, \|f_r\| \rangle$.

3. PRUNE THE TREE: Let $v_i = \langle v_{i\,1}, \ldots, v_{i\,r} \rangle$.

   If some $v_i = 0$ we may discard $\widehat{v_i}$. In fact, if $f \in$ SAT then one of the formulas in S is satisfiable, by inductive hypotheses 3° and 1°. Thus one of the new leaves is satisfiable. Since $v_i = 0$ can't be a true guess if $f \in$ SAT, deleting it causes no harm, in terms of maintaining 1°, 2° and 3°.

   Secondly, for any pair $\widehat{v_i}$, $\widehat{v_j}$, if $v_i = v_j$, we can effectively delete at least one of them. This is because $\widehat{v_i}, \neq \widehat{v_j}$ and $v_i = v_j$ imply $v_{i\,0} \neq v_{j\,0}$. Now $\langle v_{i\,1} + v_{i\,2}, \ldots, v_{i\,r-1} + v_{i\,r} \rangle$ must equal one of the u's (call it u_t) associated with the formulas in S (otherwise $\widehat{v_i}$ as well as $\widehat{v_j}$ is clearly false by 3°). This u_t must be unique by 2°; furthermore, either $v_{i\,0} \neq s_t$ or $v_{j\,0} \neq s_t$.

   If $v_{i\,0} \neq s_t$, clearly $\widehat{v_i}$ is false; we may delete $\widehat{v_i}$. The same argument applies to $\widehat{v_j}$.

   Without loss of generality, we are left with $\langle \widehat{v_1}, \ldots, \widehat{v_q} \rangle$, $q \leq k$. If $q = 0$ then output $\|f\| = 0$. In fact, if $f \in$ SAT then some $\widehat{v_i}$ with $v_i \neq 0$ must represent the truth and must have been kept.

   Let $s_i = v_{i\,0}$, $i = 1, \ldots, q$, $1 \leq q \leq k$. Note that $v_1, \ldots, v_q$ and the s_i's satisfy conditions 1°, 2° and 3°. For condition 3°, since when f is satisfiable, we have only deleted false guesses. Let the guess matrix consist of $v_1, \ldots, v_q$ as row vectors. We will inductively extract at most q columns of the matrix, so that the q row vectors of the submatrix (the projection of $v_1, \ldots, v_q$ onto the chosen dimensions) also satisfy 1°, 2° and 3°.

Since 3° is automatically satisfied with any subset of columns, we need only to maintain 1° and 2°.

To prune, initially let $j_1 = \min\{j \geq 1 \mid v_{1j} \neq 0\}$. Let $w_1 = (v_{1j_1}) \in Z^1$. Inductively, suppose $w_1, \ldots, w_h \in Z^{h'}$ have been constructed, $h' \leq h < q$, satisfying 1° and 2°. Each w_i is the projection of v_i onto the h' chosen columns. Take these h' columns of v_{h+1}; call this vector $\widetilde{w_{h+1}}$.

If $\widetilde{w_{h+1}} = w_i$ for some $1 \leq i \leq h$, then this i is unique, by 2°. Also, $\widetilde{w_{h+1}} \neq 0$ by 1°. Now all we need to do is to distinguish w_{h+1} from w_i. But since $v_{h+1} \neq v_i$, this is easily done by choosing one more column. (Every $w_1, \ldots, w_h$, $\widetilde{w_{h+1}}$ is extended one dimension to get the new $w_1, \ldots, w_h$ and w_{h+1}.)

If $\widetilde{w_{h+1}} \neq w_i$, for all $1 \leq i \leq h$, then we only need to insure $\widetilde{w_{h+1}} \neq 0$. Again this is easily done by extending at most one dimension, since $v_{h+1} \neq 0$.

Finally, $w_1, \ldots, w_q$ are constructed. Set u_i to w_i and p to the dimension of w_i; S consists of those new leaves corresponding to the p selected columns. 1°, 2° and 3° are satisfied.

End of Stage l.
QED

3.2 Dealing With Polynomial Enumerators

In this section we show how to combine many formulas into a super-formula efficiently and prune so that our tree does not blow up in width. This is an extension of the way, in Section 2, we went from combining two to combining three formulas (Equations 1 and 2).

We now discuss how to proceed with an $n^{1-\epsilon}$–enumerator. We maintain a polynomially wide band as we prune down the tree. Suppose we are given m Boolean formulas $f_1, f_2, \ldots, f_m$ on variables $x_1, x_2, \ldots, x_n$. We first make all the variables distinct among different f_i's. This blows up the size of each formula by a factor of at most $1 + \log m$. Second, we choose m new variables $z_1, z_2, \ldots, z_m$ each of size $O(\log m)$, and combine the formulas $f_1, f_2, \ldots, f_m$ via the straightforward generalization of Eq. 2. Let F be the resulting super-formula.

We wish to bound the size of F in terms of the sizes of the f_i's. Let N be a bound on the sizes of each f_i, $|f_i| \leq N$. We conclude, upon examining our combination formula (Eq. 2), that the size of F is bounded above by

$$B(m, N) = 2mN(1 + \log m) + O(m \log m) \tag{3}$$

Note that for large N, $(B(2N^t + 1, N))^\alpha < N^t$, if $\alpha < 1$ and $t > \frac{2\alpha}{1-\alpha}$. Hence, for an $n^{1-\epsilon}$-enumerator, we can maintain a bushy tree of width N^t as we carry out the tree pruning.

4 An n^k-for-one Result: $\mathrm{NP}^{\#\mathrm{P}} = \mathrm{NP}^{\#\mathrm{P}[1]}$

The recently developed theory of terseness asks if many queries to an oracle are more powerful than one query [AG,GJY,BGGO]. Kadin has proven that if for some k, $\mathrm{P}^{\mathrm{NP}[k]} = \mathrm{P}^{\mathrm{NP}[k+1]}$,

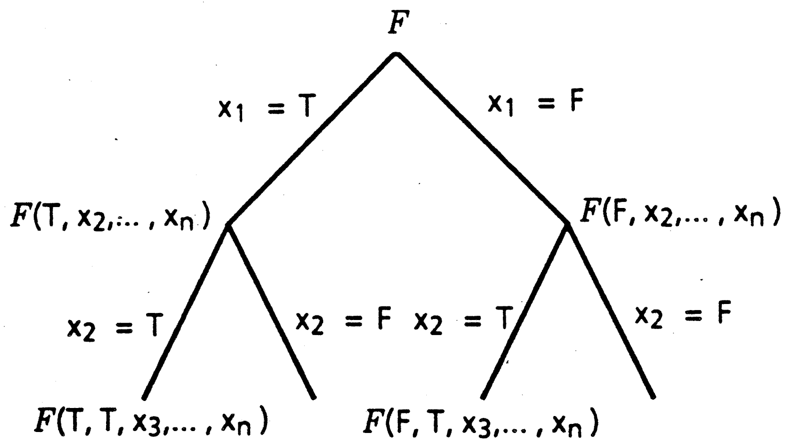
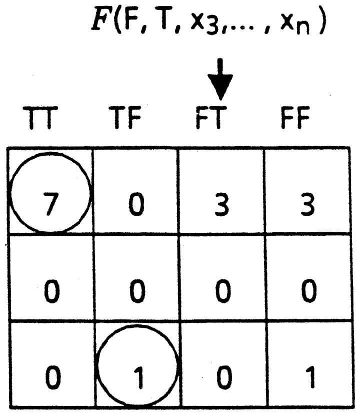


Figure 1A: The Tree


$F(F, T, x_3, \ldots, x_n)$

TT	TF	FT	FF
7	0	3	3
0	0	0	0
0	1	0	1

Figure 1B: The Guess Matrix

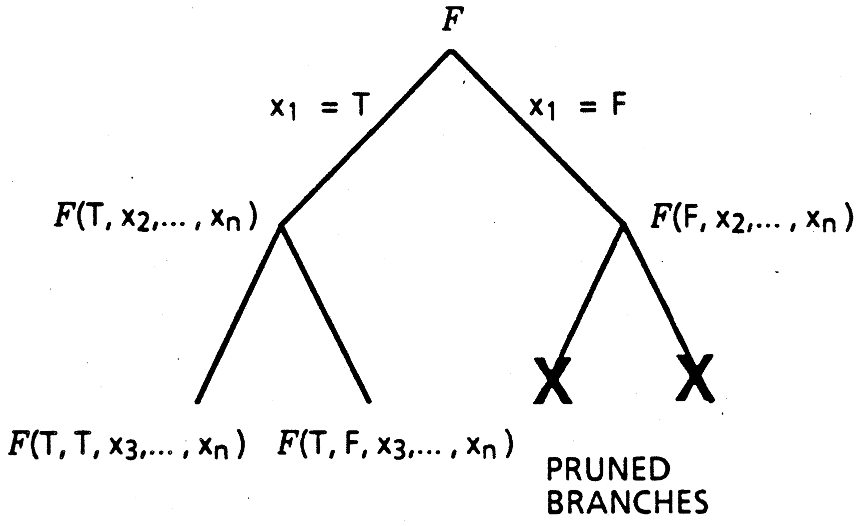


Figure 1C: The Pruned Tree