On Ranking

Lane A. Hemachandra*

TR 86-794

December 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

* This work was supported by a Fannie and John Hertz Fellowship and NSF Research Grant DCR-8520597.

On Ranking

*Lane A. Hemachandra**
Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract

This paper structurally characterizes the complexity of ranking.

A set is P-rankable if there is a polynomial time computable function f so that for all x, $f(x)$ computes the number of elements of A that are lexicographically $\leq x$, i.e., the *rank* of x with respect to A. We'll say a class $\mathcal{C}$ is P-rankable if all sets in $\mathcal{C}$ are P-rankable. Our main results show that with the same certainty with which we believe counting to be complex, and thus with at least the certainty with which we believe $P \neq NP$, we may believe that P has no ranking functions of any type—uniform, strong, weak, or approximate.

We show that:

- P and NP are equally likely to be P-rankable.
- P is P-rankable if and only if $P = P^{\#P}$. This extends important work of Blum and Sipser [Sip85].
- Even weak variations of P-ranking are hard if $P \neq P^{\#P}$.
- PSPACE is P-rankable if and only if P = PSPACE.

*This work was supported by a Fannie and John Hertz Fellowship and NSF Research Grant DCR-8520597.

- If P has small ranking circuits, then it has small ranking circuits of relatively low complexity.
- If P has small ranking circuits then the power of counting falls into the polynomial hierarchy (i.e., $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$).
- P/poly, the class of sets with small circuits, is not P-rankable.
- P/poly has small ranking circuits if and only if $\mathrm{P}^{\#\mathrm{P}}/\mathrm{poly} = \mathrm{P}^{\#\mathrm{P}/\mathrm{poly}} = \mathrm{P}/\mathrm{poly}$.
- If P is rankable, then P/poly has small ranking circuits. This links the ranking complexity of uniform and nonuniform classes.
- The ranks of some strings in easy sets are of high relative time-bounded Kolmogorov complexity unless $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. It follows that even approximate ranking is hard unless $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. This partially resolves a question posed by Sipser [Sip85, pp. 447–448].

1 Introduction

This paper characterizes the complexity of ranking sets from standard complexity classes such as P, NP, PSPACE, and P/poly. Over the past decade, much effort has been spent studying the complexity of the *membership problem* for languages from NP and PSPACE [GJ79]. At the same time, Valiant [Val79a,Val79b], Angluin [Ang80], Stockmeyer [Sto85], and others have studied the complexity of the *counting problem* for NP: how hard is it to determine the number of satisfying assignments a Boolean formula has?

The counting problem is formalized by Valiant's counting class #P [Val79a], the class of functions that count the accepting paths of a nondeterministic polynomial time (NP) Turing machine. $f \in \#\mathrm{P}$ if and only if there is an NP machine N so that for every x, $f(x)$ equals the number of accepting paths of $N(x)$. $\mathrm{P}^{\#\mathrm{P}}$ is the *language* class of sets accepted by some polynomial time machine with a #P oracle.

This paper studies the *ranking problem*: given a fixed set, how hard is it to determine the rank of elements within that set. Loosely put, the ranking problem seems harder than the membership problem and closely related to the counting problem.

Studying the ranking problem gives a new perspective on the determinism-vs-nondeterminism question. It is quite likely that the membership problem for NP is harder than the membership problem for P. Nonetheless, the ranking problems of NP and P stand or fall together: either both are easy or both are hard.

Another reason that ranking is of interest is that it is a natural generalization to non-sparse sets of P-printability.[1] A set S is P-printable if there is a polynomial time function that on input 1^n prints all strings in S of length $\leq n$ [HY84]. Recent work by Allender [All86], Balcazar and Book [BB86], and others [HH86b] has shown P-printability to be intimately related to Kolmogorov complexity and to structure of feasible computations. [BB86] and [HH86b] show that in worlds where $\mathrm{P} = \mathrm{NP}$, the sparse sets S that separate NP^S from P^S are exactly those that are P^S-printable. It is well known that a sparse set is strongly P-rankable (defined below) if and only if it is P-printable.

Thus rankability provides one natural generalization to dense sets of P-printability, which applies only to sparse sets. Other generalizations of printability to dense sets are self-encodability and self-P-producibility, essentially equivalent notions, which have been explored by Ko, Long, and Selman (see [Ko85, page 228]), Balcazar and Book [BB86], and Hartmanis and Hemachandra [HH86b].

Sipser [Sip85] proves that 1-way Logspace sets have easy ranking problems and Allender [All85] proves that certain classes of automata accept only sets with easy ranking functions. Nonetheless, Blum and Sipser [Sip85] show that some simple sets may have hard ranking problems: if a certain set in P can be strongly P-ranked then #P functions can be computed by polynomial time machines. We extend

[1] Dr. Osamu Watanabe introduced the author to this view of rankability.

there results into a complete characterization, extend our results to weaker notions of ranking, and study some novel facets of the ranking problem.

Definition 1.1 1. A set A is *strongly P-rankable* if there is a polynomial time function f so

$$(\forall x)\,[f(x) = \|A^{\leq x}\|],$$

where $A^{\leq x}$ indicates all strings in A that are lexicographically $\leq x$. A strong ranking function tells us the number of elements in A up to a given string.

2. A set A is *P-rankable* if there is a polynomial time function f so

   (a) $(\forall x \in A)\,[f(x) = \|A^{\leq x}\|]$, and

   (b) $(\forall x \notin A)\,[f(x)$ prints "not in A"$]$.

   Thus a P-ranking function ranks elements in the set but merely detects non-membership.

3. A set A is *weakly P-rankable* if there is a polynomial time function f so

$$(\forall x \in A)\,[f(x) = \|A^{\leq x}\|].$$

   A weak P-ranking function may deviously output false values on inputs not in A.

4. We use P-rankable, weak-P-rankable, and strong-P-rankable to denote the classes of languages having these properties.

5. We say a language class $\mathcal{C}$ is P-rankable if all languages in $\mathcal{C}$ are P-rankable (similarly for weakly P-rankable and strongly P-rankable).

Section 2 proves our first ranking result, which

- characterizes the complexity of ranking in structural terms,

- shows that even weak types of ranking are hard unless complexity classes collapse, and

- surprisingly, shows that ranking problems for P and NP are equally likely to be rankable.

Theorem 2.1 The following are equivalent:

1. $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.
2. P = P-rankable.
3. P = strong-P-rankable.
4. P $\subseteq$ weak-P-rankable.
5. P $\subsetneq$ weak-P-rankable
6. NP = P-rankable.
7. NP = strong-P-rankable.
8. NP $\subseteq$ weak-P-rankable.
9. NP $\subsetneq$ weak-P-rankable

Corollary 2.3 P is P-rankable if and only if NP is P-rankable.
The author has been informed that Steven Rudich has, independently, noted the equivalence of (1) and (2) [Rud].

Given our belief that sets in P are hard to rank, it is natural to ask if sets in P have *small ranking circuits*. That is, does each P set A have a family of circuits c_i, so (1) $|c_i|$ grows polynomially in i and (2) c_i, on any x of length i, computes the rank of x in A? (c_i will be a circuit of logical gates [Sav72], with a number of output wires that will carry the binary value of the rank.)

Section 3 shows that if P has small ranking circuits then the power of counting is dominated by the polynomial hierarchy.
Theorem 3.5 If P has small ranking circuits, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$.

Our proof has two steps.

1. We show that if a set has small (but possibly complex) ranking circuits, then it has small ranking circuits that are relatively easily computed, i.e., they are

printed by a Δ_4^p (see [Sto77]) machine.[2] This technique of pulling circuits and sets down into the polynomial hierarchy has been mined by Hartmanis and by Balcazar, Book, Long, Selman, and Schoening [BBL*84] in their work on sparse oracles. Our application of their technique is surprising. Given our assumption of small circuits, we show that a Boolean predicate with shallow quantifier nesting can verify if a circuit is a valid ranking circuit.

2. We combine our result with the useful small circuit theorem of Karp and Lipton [KL80] to obtain Theorem 3.5.

Section 4 turns from studying the ranking problem of uniform classes like P and NP to studying the ranking problem of the nonuniform complexity class P/poly (see [Sch86] for a discussion of results about P/poly). P/poly, the class of sets with small circuits, is an important class. Sets in P/poly are near P; with a sparse amount of additional information at each length a P machine can accept any P/poly set. The outstanding result on P/poly is that if P/poly$\supseteq$NP, then the polynomial hierarchy collapses to its second level [KL80].

The classes we discuss each contain a nondenumerable number of sets; thus many standard techniques no longer apply. For example, these classes lack many-one complete languages. Thus we must modify the proof techniques of previous sections to eliminate their reliance on complete languages.[3] Primitive proof techniques succeed.

By simply counting we show that P/poly is not P-rankable. As a nonuniform analogue of the results of Section 2, we show that P/poly has small ranking circuits

[2] As a corollary of Theorem 3.5, we'll see that the circuits are much simpler than this (Corollary 3.6).

[3] A study of other classes that are suspected (for quite different reasons) not to have many-one complete languages and the techniques that apply to these classes appears in recent papers of Hartmanis and Immerman [HI85] and Hartmanis and Hemachandra [HH86a]. They note that many probabilistic, intersection, and counting classes may not have complete languages (and do not, in some relativized worlds), and that if these classes do have complete languages they are of a particularly simple form.

if and only if P/poly = $\mathrm{P}^{\#\mathrm{P}}$/poly. A connection between the rankability of P and P/poly follows from this. If the uniform class P can be uniformly ranked (i.e., P-ranked), then the nonuniform class P/poly can be nonuniformly ranked.

Theorem 4.3 P/poly has small ranking circuits if and only if P/poly = $\mathrm{P}^{\#\mathrm{P}}$/poly.

Corollary 4.4 If P is P-rankable then P/poly has small ranking circuits.

Finally, having in the previous sections shown that even weak or nonuniform ranking of P causes P = $\mathrm{P}^{\#\mathrm{P}}$, it is natural to ask if rank functions of P sets can easily be approximated. This is an open question posed by Sipser [Sip85].

It follows directly from the fact that ranks of P sets are #P functions (see "(1) $\Rightarrow$ (7)" of the proof of Theorem 2.1, page 8) and Stockmeyer's result that approximate counting can be done in Δ_3^p, that Δ_3^p machines can compute "approximate ranks" for P sets. The notion here is that the "approximate rank" will be related to the true rank by a multiplicative factor that is around $1 + \mathcal{O}(n^{-d})$, d arbitrary but fixed.

Section 5 studies a different notion of "approximate ranking." Cai and Hemachandra [CH86] introduce a notion of approximate counting in which they ask if one can quickly compute a small set of candidate solutions to a #P problem, one of which is guaranteed to be correct. They show that their approximate counting is Turing equivalent to exact counting. Thus this type of approximate counting is as hard as exact counting.

Let's adopt a similar approach to approximation of ranks.

Definition 1.2 We say a set A is *k-approximate-rankable* if there is a polynomial time computation function f so for every x, $f(x)$ prints a set of k numbers, one of which is the rank of x with respect to A.

Similarly we can define *$g(n)$-approximate-rankable.* As a corollary of the above-mentioned result of [CH86], we show that approximate ranking is impossible unless P = $\mathrm{P}^{\#\mathrm{P}}$.

Theorem 5.1 Each set A in P for some k_A has a k_A-approximate-ranker if and only if $P = P^{\#P}$.

With some care, we can show that $P = P^{\#P}$ if and only if each set A in P for some ϵ_A is $n^{1/2-\epsilon_A}$-approximate-rankable.

Section 6 concludes and presents related open problems. Simply summarized, this paper structurally characterizes the complexity of uniform, weak, nonuniform, and approximate ranking. It follows from the characterizations that if our intuition that complexity classes separate holds, then ranking of any type is beyond the grasp of polynomial time.

2 When Can Uniform Complexity Classes Be Ranked?

This section answers the questions:

- Is NP harder to rank than P?
- When can P be ranked?
- When can NP be ranked?
- When can PSPACE be ranked?

Theorem 2.1 The following are equivalent:

1. $P = P^{\#P}$.
2. P = P-rankable.
3. P = strong-P-rankable.
4. P $\subseteq$ weak-P-rankable.
5. P $\subsetneq$ weak-P-rankable
6. NP = P-rankable.
7. NP = strong-P-rankable.
8. NP $\subseteq$ weak-P-rankable.
9. NP $\subsetneq$ weak-P-rankable

Sipser and Blum [Sip85] show that if a certain P set is strongly P-rankable then P = #P. Their method is to construct a set[4] combining formulas and their assignments, so that all the variable assignments to a formula are lexicographically adjacent. Thus computing a #P question becomes the subtraction of two ranks, i.e., computing how many variable assignments are satisfying (and thus in the set).

This method fails for weak P-rankers; the extreme assignments may be out of the sets, and thus the ranker's answers about them may be lies. We make an easy modification of the argument of Blum and Sipser to show that if all P sets are even weakly P-rankable, then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

A weak P-ranker may give false ranks for elements out of the set, but is honest for elements in the set. We take advantage of the ranker's honesty by inserting easily located *beacons* into the set. The weak P-ranker of course is compelled to be truthful on these beacons, and we force it to solve a #P query (this is done in the proof below that (4) $\Rightarrow$ (1)).

Going the other way, (1) $\Rightarrow$ (7) below shows how to strongly rank NP sets when $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. Simply put, when $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$, we can bring down from the #P oracle, one bit at a time, the correct ranks of strings from our NP language.

Notation 2.2 $Rank_A(x) =_{def} |A^{\leq x}|$, i.e., the number of strings in A that are lexicographically $\leq x$.

Proof of Theorem 2.1 Obvious implications that we use are (3) $\Rightarrow$ (2) $\Rightarrow$ (4), (7) $\Rightarrow$ (6) $\Rightarrow$ (8), (8) $\Rightarrow$ (4), and (7) $\Rightarrow$ (3). We prove that (4) $\Rightarrow$ (1), (1) $\Rightarrow$ (7), (4)$\equiv$(5), and (8)$\equiv$(9). Thus (1) $\Rightarrow$ (7) $\Rightarrow$ (6) $\Rightarrow$ (8)[$\equiv$(9)] $\Rightarrow$ (4)[$\equiv$5] $\Rightarrow$ (1) and (1) $\Rightarrow$ (7) $\Rightarrow$ (3) $\Rightarrow$ (2) $\Rightarrow$ (4) $\Rightarrow$ (1). Thus (1) through (9) are all equivalent.

[(4) $\Rightarrow$ (1)]:

To show that $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$ it suffices to show that #SAT, computing the number of

[4]Essentially, $L = \{f\#assign \mid |assign| = |f|$ and $|assign|$ is a vector of assignments to all variables of f that satisfies $f\}$.

solutions to a Boolean formula f, can be done by a polynomial time machine. Let

$$L_B = Satisfiers \bigcup Beacons,$$

where $Beacons = \{y\#00\#1^{3|y|} \mid y \in \Sigma^*\} \cup \{y\#11\#1^{3|y|} \mid y \in \Sigma^*\}$ and $Satisfiers = \{f\#01\#z \mid |z| = 3|f|$ and z is a vector that assigns all variables of Boolean formula f (followed by padding ones) and f is satisfied by the assignment coded by $z\}$.

Asked the #SAT question about Boolean formula f, we use our weak P-ranker to compute in polynomial time:

$$\{\ Rank_{L_B}(f\#11\#1^{3|f|}) - \ Rank_{L_B}(f\#00\#1^{3|f|})\ \} -1.$$

This equals the number of solutions of f, as the beacons are in L_B and get honest responses from the weak P-ranker. A difference of one between the above ranks is due to the beacons' rank difference, and any additional difference comes from satisfying assignments to f that are lexicographically sandwiched between the beacons. Thus with a weak P-ranker we compute #SAT in polynomial time so $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

[(1) ⇒ (7)]:

Clearly every strongly P-rankable language is in NP and indeed in P. Assuming $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$, we must show that an arbitrary language $L \in \mathrm{NP}$ can be strongly P-ranked. Since $\mathrm{NP} \subseteq \mathrm{P}^{\#\mathrm{P}} = \mathrm{P}$, we know $\mathrm{P} = \mathrm{NP}$ and L is accepted by a polynomial time machine M. Thus there is an NP machine N so on input x, N nondeterministically creates a path for each string y lexicographically $\leq x$, and the path associated with y accepts if and only if $M(x)$ accepts. Since N is an NP machine, its counting function c_N ($c_N(x)$ = the number of accepting paths of $N(x)$) is in #P. Clearly, by our choice of N, we have $Rank_L(x) = c_N(x)$. Thus the language $L_{bits} = \{x\#i\#b \mid c_N(x)$ has $\geq i$ bits and the ith bit is $b\}$ is in P. Repeatedly using L_{bits} to get one more bit of $c_N(x)$, we compute in polynomial time $c_N(x) = Rank_L(x)$, and we have strongly P-ranked language L.

[(4) ≡ (5)], [(8) ≡ (9)]:

(5) ⇒ (4) and (9) ⇒ (8) immediately. We can easily, by diagonalization or using

Kolmogorov complexity, construct a non-recursive language L that has exactly one string at each length ≥ 1. L is weakly P-ranked by the machine that on input x outputs rank $|x|$, yet $L \notin$ P and $L \notin$ NP. So (4) $\Rightarrow$ (5) and (8) $\Rightarrow$ (9).

♠

Aside from showing that ranking is complex, Theorem 2.1 reveals a striking contrast between the determinism-vs-nondeterminism questions for membership and counting. We believe that P $\neq$ NP and thus the membership problems of NP and P are of *different complexities*, respectively hard and easy. Corollary 2.3 says that either both or neither of the ranking problems of NP and P is easy.

Corollary 2.3 P is P-rankable if and only if NP is P-rankable.

Though at first shocking, this is not hard to understand. The ranking problems of P and NP are so complex that if either is easy it causes an earth-shaking collapse of complexity classes, which in turn destroys any difference between P and NP.

Only a powerful complexity class can avoid being dragged into the rubble if $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. PSPACE may be such a class.

Theorem 2.4 PSPACE is P-rankable if and only if P = PSPACE.

It is possible that the ranking problem of PSPACE may be hard even if the ranking problems of P and NP are easy. However, no oracle is known to certify this possibility ($\mathrm{P} = \mathrm{NP} \neq \mathrm{P}^{\#\mathrm{P}}$), and any such oracle would resolve the major open relativization question about the polynomial hierarchy left in the aftermath of Yao's [Yao85] breakthrough: Is there an oracle for which the polynomial hierarchy *collapses* yet differs from PSPACE?

3 Small Ranking Circuits and P

In this section, we answer the questions:

- What does it imply if P has small strong-ranking circuits?

- What does it imply if P has small ranking circuits?
- What does it imply if P has small weak-ranking circuits?

Having small circuits for a problem is the next best thing to having a polynomial time algorithm. If a problem has small circuits, there are relatively small machines that solve the problem for all non-enormous instances.

The previous section revealed the futility of seeking to rank P sets with P machines. A natural weaker goal is to find small circuits for the ranking problem. For example, PRIMALITY has small circuits ([Rab76,Adl78], see [Sch86, pp. 33,40]), but intense efforts have failed to place the problem in P [APL80,GK86].

Again, we prove ranking hard. If ranking can be done with small circuits, then the power of counting falls into the polynomial hierarchy.

We first show that if P has small strong-ranking circuits, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$. Then we sketch how to modify the proof to obtain the more powerful result that if P has small ranking circuits, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$.

Indeed, we can show that if P has small weak-ranking circuits, the same surprising conclusion holds: $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$.

3.1 If P has Small Strong-Ranking Circuits then $\mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{\Sigma}_2^{\mathbf{P}}$

The crucial step in our proof is to show that if P has small strong-ranking circuits (possibly complex ones), then it has small strong-ranking circuits that are relatively simple, i.e., printable by a Δ_3^p machine.

Balcazar, Book, Long, Schoening, and Selman [BBL*84] and Hartmanis used this paradigm of dragging down complexities in their work on sparse oracles. Our application to ranking of "dragging complexities down into the polynomial hierarchy" is a bit subtle. Our goal is to show that if small strong-ranking circuits exists, we can *compute* then using just a few polynomially bounded quantifiers.

Simply put, we guess a small circuit and make sure that the function that it computes:

1. starts at zero,
2. increases by one at each (small enough) string in L, and
3. stays the same at each (small enough) string not in L.

If so, it is a small strong-ranking circuit.

This is done in Lemma 3.2 below. Theorem 3.3 follows after a quick appeal to a fundamental result of Karp and Lipton on the difficulties of NP having small circuits.

Definition 3.1 We say a set L has *small strong-ranking circuits* if there is a family $\{c_i\}$ of circuits so

1. $|c_i|$ is polynomial in i, and
2. $(\forall x \ni |x| \leq i)\,[c_i(x)$ computes $Rank_L(x)]$. When we say $c_i(x)$, we of course assume that x is prefixed by enough leading special padding characters (#) to make its length exactly i. (Equivalently our circuits could have two wires for each input, one to say if there was a digit in that place and another to give the digit's value. Alternatively, we could use $\log i$ extra inputs to specify the length.)

Comment on Definition 3.1: Typical definitions of a family of small circuits [Sch86] have circuit c_i perform correctly on all length i strings. Our definition (which will shorten the proofs) says it performs correctly on all length $\leq i$ strings. It is clear that a set L has small strong-ranking circuits under the first definition if and only if it has small strong-ranking circuits under the second definition (one direction is trivial, in the other direction our circuit first determines the length of the (unpadded) input and uses the appropriate circuit).

Lemma 3.2 If P has small strong-ranking circuits, then every P set has a family of small strong-ranking circuits that is printable by a $\Delta_3^p\,(= \mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}})$ machine.

Theorem 3.3 If P has small strong-ranking circuits, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$.

Proof of Lemma 3.2 Let $L \in \mathrm{P}$ have small strong-ranking circuits. W.l.o.g. $|c_i| \leq i^k$.

Note that it suffices to show that the bottom NP machine of an $\mathrm{NP}^{\mathrm{NP}}$ machine can find (if one exists) a small strong-ranking circuit consistent with some prefix, for L up to a given length. This is because then a $\mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}} (= \Delta_3^p)$ machine with this $\mathrm{NP}^{\mathrm{NP}}$ oracle can print a small circuit. It does so by getting one bit of the circuit at a time from the oracle with queries of the form "Is there a strong-ranking circuit for L up to length i whose representation starts with bits 001101?"

Here is the $\mathrm{NP}^{\mathrm{NP}}$ circuit checking procedure. x_- indicates the string that lexicographically precedes x. Let σ be the prefix we are extending and $c_i \sqsupseteq \sigma$ indicate that (the representation of circuit) c_i starts with the bits σ. Recall that the expression below simply says we start at zero, increase by one at elements of L, and stay the same at nonmembers of L.

$$\Big(\exists c_i \sqsupseteq \sigma \ni \Big\{(|c_i| \leq i^k) \wedge \ \{(\ \epsilon \in L \ \wedge \ c_i(\epsilon) = 0 \ ) \ \vee \ (\ \epsilon \notin L \ \wedge \ c_i(\epsilon) = 1 \ )\}\Big\} \Big)$$
$$(\forall x \ni |x| \leq i)$$
$$\Big[\Big((\ x \in L \ \wedge \ c_i(x) - c_i(x_-) = 1 \ ) \ \vee \ (\ x \notin L \ \wedge \ c_i(x) - c_i(x_-) = 0 \ )\Big)\Big]$$

This depth-two-polynomially-bounded-quantifier language can be accepted by an $\mathrm{NP}^{\mathrm{NP}}$ machine (the base machine does the $\exists$ and the NP oracle does the $\forall$, see Stockmeyer [Sto77]). This, as noted at the start of this proof, suffices to prove our lemma.

♠

Proof of Theorem 3.3 By Lemma 3.2, every P set has small strong-ranking circuits computable in Δ_3^p. Thus we can solve #SAT, as in the proof of Theorem 2.1, with two calls to this Δ_3^p-computable object, and thus place $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{P}^{\Delta_3^p} = \Delta_3^p$. But it is easy to see that if P has small strong-ranking circuits then NP has small membership circuits (via the language L_B of page 10). So by the Karp-Lipton result [KL80], $\Sigma_2^p = \Pi_2^p = \mathrm{PH}$. Thus

$$\mathrm{P}^{\#\mathrm{P}} \subseteq \Delta_3^p \subseteq \mathrm{PH} \subseteq \Sigma_2^p.$$

♠

3.2 If P has Small Ranking Circuits then $\mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{\Sigma}_2^{\mathbf{p}}$

This follows the same argument as the previous section, except we must account for the change from strong-ranking circuits to ranking circuits. The potential problem is that a ranking circuit, on input $x \notin L$, merely says "this is not in L" (unlike a strong-ranker, which ranks nonmembers).

Given that small ranking circuits exists, as before we guess a small circuit and make sure that the function it computes meets conditions that guarantee that it really is a ranking circuit of L. Simply put, our conditions are (compare them to the simpler conditions that checked strong-ranking circuits on page 12) that the function starts at zero, is correct about membership and nonmembership, and increases its rank exactly one between adjacent members of L.

That is, we guess a circuit and check that the function it computes:

1. starts at zero,
2. at each (small enough) string out of L it proclaims nonmembership,
3. at each (small enough) string in L it proclaims membership by announcing (its version of) the string's rank.
4. for every pair of (small enough) strings x, y (w.l.o.g. y is lexicographically greater than x), for which
   (a) the circuit thinks that x and y are in L and thus gives them ranks, and
   (b) there exists no string z lexicographically between x and y in L,

the rank the circuit assigns y is exactly one greater than the rank the circuit assigns x.

Looking at the quantifier structure implicit in the above conditions, they can be written as a Σ_3^p predicate; so we can *print* a small ranking circuit with a Δ_4^p machine. Theorem 3.5 now follows as in the previous section.

Interestingly, as a corollary of the theorem, we conclude that the circuits can be printed by a Δ_3^p machine.[5]

Lemma 3.4 If P has small ranking circuits, then every P set has a family of small ranking circuits that is printable by a Δ_4^p machine.

Theorem 3.5 If P has small ranking circuits, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$.

Corollary 3.6 If P has small ranking circuits, then each P set has a family of small ranking circuits that is printable by a Δ_3^p machine.

By slightly modifying the four conditions above[6] we can strengthen our results to apply to small *weak*-ranking circuits.

Theorem 3.7 If P has small weak-ranking circuits, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \Sigma_2^p = \mathrm{PH}$.

4 Ranking P/poly: Primitive Proof Techniques Suit Uncountable Classes

This section answers the questions:

- Can P/poly be P-ranked?
- When can P/poly have small ranking circuits?

[5]Even though we conclude that all *languages* in the polynomial hierarchy are in Σ_2^p, some *functions* computable by polynomial hierarchy machines may need Δ_3^p machines.

[6]Eliminate condition 2. Change condition 1 so that it checks that the first string in the set is indeed assigned rank one.

- What techniques can be used in proofs about uncountable classes, which lack many-one complete sets?

The classes P/poly and $\mathrm{P}^{\#\mathrm{P}}$/poly contain uncountable numbers of languages. Because of this, they are immune to many techniques we use on standard complexity classes such as P and NP. In particular, they have no many-one complete languages, at least with respect to uniform many-one reductions, so the technique of Theorem 2.1, which used the fact that trivializing #SAT trivializes all of #P, does not apply.

Surprisingly, we prove an analogue of Theorem 2.1 *without* using any complete language. Thus complete sets, usually a crucial tool, seem to have been thrown aside. However, when we look carefully, we see what has happened. Before, our proof centered on a complete set and the set dragged its class with it. Now, our proofs must take a generic member of the nondenumerable class and operate on it. Thus instead of working with a simple set like #SAT, we work directly with an arbitrary #P/poly set by formalizing (in the standard way [Coo71,Har78]) its computations.

Also, note that by simple counting, P/poly is not P-rankable; P/poly contains sets spanning an uncountable number of ranking functions, and there are only a countable number of P-rankers.

Theorem 4.1 P/poly is not P-rankable.

Definition 4.2 A language L is in $\mathrm{P}^{\#\mathrm{P}}$/poly if there is a polynomial time Turing machine M, a #P function f, and an advice function g so

1. $(\exists k)(\forall x)[|g(x)| \leq n^k + k]$, and
2. $(\forall x)\,[M^f(x\#g(1^{|x|})$ accepts if and only if $x \in L]$.

L is in $\mathrm{P}^{\#\mathrm{P}}$/poly if a $\mathrm{P}^{\#\mathrm{P}}$ machine, given a bit of free advice that depends only on the length of the input, can compute L. It is easy to prove that

$$\mathrm{P}^{\#\mathrm{P}}/\mathrm{poly} = \mathrm{P}^{\#\mathrm{P}/\mathrm{poly}},$$

where #P/poly counts the accepting paths of NP machines that are given a bit of advice.

Theorem 4.3 P/poly has small ranking circuits if and only if

$$\mathrm{P/poly} = \mathrm{P}^{\#\mathrm{P}}/\mathrm{poly} = \mathrm{P}^{\#\mathrm{P/poly}}.$$

Proof of Theorem 4.3

$\Longleftarrow$: Given $L \in$ P/poly, we know that L has small circuits (we go back to the standard definition of circuits; each circuit accepts all strings of the set of some single length). We wish to show that L has small *ranking* circuits.

Let NP machine N, on input $x\#c_0\#\ldots\#c_{|x|}$, make a nondeterministic path for each string y lexicographically $\leq x$. The path associated with y will accept if and only if circuit $c_{|y|}$ accepts y. Clearly, if the circuits given to N are correct circuits for L, then $Rank_L(x)$ equals the number of accepting paths of $N(x)$.

A $\mathrm{P}^{\#\mathrm{P}}$/poly machine whose #P oracle is the counting function of N and whose advice function is the circuit family of L can compute the ranking function and, in particular, can accept the languages L_k, for each fixed $0 \leq k \leq |x|$.

$$L_k = \{x\#b \mid \text{ the } k\text{'th bit of } Rank_L(x) \text{ is } b\}.$$

By assumption, $\mathrm{P}^{\#\mathrm{P}}/\mathrm{poly} = \mathrm{P/poly}$, so we have small circuits for $L_1, L_2, \ldots, L_{|x|}$, i.e., each circuit computes one bit of the ranking function. We can easily fan out the inputs and combine these circuits, thus obtaining a small ranking circuit for our arbitrary P/poly language L.

$\Longrightarrow$: Let L be an arbitrary language from $\mathrm{P}^{\#\mathrm{P}}$/poly. We show that, if P/poly has small ranking circuits, then L is in P/poly. Let the #P oracle of the $\mathrm{P}^{\#\mathrm{P}}$/poly machine accepting L w.l.o.g. be the counting function (computing the number of accepting paths) of machine N. W.l.o.g. N runs in NTIME$[n^k]$. Let

$$L' = \left\{ x\#01\#path\#1^{pad} \mid |path\#1^{pad}| = |x|^{k+1} \text{ and } N(x) \text{ has } path \text{ as an accepting path} \right\} \cup$$

$$\left\{x\#00\#1^{|x|^{k+1}} \mid x \in \Sigma^*\right\} \cup$$
$$\left\{x\#11\#1^{|x|^{k+1}} \mid x \in \Sigma^*\right\}.$$

$L' \in \mathrm{P}$, so from our assumption L' has small ranking circuits.

Here is how to accept L with a P/poly machine. Our advice will consist of the advice of L plus the small ranking circuits of L'. On input x, the P/poly machine simulates the run of the $\mathrm{P}^{\#\mathrm{P}}$/poly machine accepting L (recall, we have its advice function) up to its first call to its #P oracle; say the call is y. We know that the answer will be (see page 10):

$$\{\ Rank_{L'}(y\#11\#1^{|y|^{k+1}}) - Rank_{L'}(y\#00\#1^{|y|^{k+1}})\ \} - 1.$$

But from our advice function, *we have small circuits computing the rank function of L'*, so our P/poly machine can easily perform the above subtraction, and thus answer the first query made in the run of $\mathrm{P}^{\#\mathrm{P}}$/poly.

Our P/poly machine continues to simulate the $\mathrm{P}^{\#\mathrm{P}}$/poly machine accepting L, answering subsequent queries in the same way. Thus we have accepted an arbitrary language $L \in \mathrm{P}^{\#\mathrm{P}}$/poly via a P/poly machine.

♠

It is easy to see that

$$\mathrm{P} = \mathrm{P}^{\#\mathrm{P}} \Rightarrow \mathrm{P/poly} = \mathrm{P}^{\#\mathrm{P}}/\mathrm{poly}.$$

From Theorems 2.1 and 4.3, it follows that if uniform (P) sets can be uniformly ranked, then nonuniform sets (P/poly) can be nonuniformly ranked. Thus uniform and nonuniform ranking problems are structurally related; Yap [Yap83] has related uniform and nonuniform membership and hierarchy questions.

Corollary 4.4 If P is P-rankable, then P/poly has small ranking circuits.

5 Approximate Rankings

In this section we answer the question:

- Can all sets in P be approximately ranked?

Our notion of an approximate ranker is novel: a given string x has more than $2^{|x|-1}$ reasonable ranks. An approximate ranker (recall Definition 1.2) reduces this huge set of potential ranks to a small set of plausible ranks. It follows that if all P sets have approximate rankers, then ranks of strings in P sets are of low relative Kolmogorov complexity. The *small name* of a rank, relative to the string it ranks, is its *index* in the small set of plausible ranks.

This approach to approximation is modeled after the work of Cai and Hemachandra [CH86] on approximate counting, and our results follow as corollaries of their main theorem: [CH86] If for some ϵ we can in polynomial time, for every Boolean formula f, compute a set of $|f|^{1-\epsilon}$ values of which one is the number of satisfying assignments to f, then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Our key observation is that if we have a k-approximate-ranker (Definition 1.2), we can k^2-approximate #SAT. Why? With a k-approximate-ranker, when we do the subtraction of ranks that yields our #SAT answer (as in the Proof of Theorem 2.1), we now have k possibilities for each of the two participating ranks in the subtraction so we have k^2 overall possibilities. The following theorem follows from this observation, the result of [CH86], and an easy reverse direction that is a corollary of Theorem 2.1. Theorem 5.1 below says that it is overwhelmingly unlikely that we can approximately rank P sets. Indeed, approximate ranking is so hard that it is no more likely that we can approximately rank than that we can exactly rank.

Theorem 5.1 Each set A in P for some k has a k-approximate-ranker[7] if and only if $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

Indeed, since we just square the amount of ranking approximation to get the amount of #SAT approximation, and since the [CH86] result is valid up to $\mathcal{O}(n^{1-\epsilon})$-approximate-counting, we can even conclude that:

[7] More precisely, $(\forall A \in \mathrm{P})(\exists k)(\exists f$ computable in P$)[f$ is a k-approximate-ranker for $A]$.

Each set A in P for some ϵ_A has an $\mathcal{O}(n^{1/2-\epsilon_A})$-approximate-ranker if and only if $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

6 Conclusions and Open Problems

Ranking is fundamental. Aside from the membership question ($x \in A$?), what question could be more natural than asking x's rank in A?

This paper has studied the complexity of the ranking problem and has characterized the complexity of many types of ranking. Now, with the same the certainty with which we believe counting is hard, and thus with at least the certainly with which we believe $\mathrm{P} \neq \mathrm{NP}$, we may believe that ranking of any type—uniform, nonuniform, strong, weak, or approximate—can not done efficiently.

A number of related open questions remain. Section 2, page 11, asks if there is a world in which P and NP have easy membership problems but hard ranking problems. That is, is there a relativized world in which $\mathrm{P} = \mathrm{NP} \neq \mathrm{P}^{\#\mathrm{P}}$? Resolving this question would settle the outstanding open problem about the relativized structure of the polynomial hierarchy that remains in the wake of Yao's [Yao85] results: Is there a relativized world in which the polynomial hierarchy *collapses* yet is not equal to PSPACE?

Any advance beyond the work of [CH86] in our knowledge if the complexity of approximate counting would strengthen the results of Section 5. Can the $\mathcal{O}(n^{1-\epsilon})$ bound of [CH86] be improved to n^k, k arbitrary?

Acknowledgements

I thank Professors Eric Allender, Jin-yi Cai, and Juris Hartmanis and Dr. Joan Feigenbaum for many helpful discussions, comments, and references. I thank Dr. Osamu Watanabe for introducing me to the notion of rankability as a generalization of P-printability.

References

[Adl78] L. Adleman. Two theorems on random polynomial time. In *Proceedings IEEE Symposium on Foundations of Computer Sciece*, pages 75–83, 1978.

[All85] E. Allender. *Invertible Functions.* PhD thesis, Georgia Institute of Technology, 1985.

[All86] E. Allender. The complexity of sparse sets in P. In *Structure in Complexity Theory*, pages 1–11, Springer-Verlag *Lecture Notes in Computer Science #223*, 1986.

[Ang80] D. Angluin. On counting problems and the polynomial-time hierarchy. *Theoretical Computer Science*, 12:161–173, 1980.

[APL80] L. Adleman, C. Pomerance, and Lenstra. On distinguishing prime numbers from composite numbers. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 387–406, 1980.

[BB86] J. Balcázar and R. Book. On generalized Kolmogorov complexity. In *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science*, pages 334–340, 1986.

[BBL*84] J. Balcázar, R. Book, T. Long, U. Schoening, and A. Selman. Sparse oracles and uniform complexity classes. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 308–313, 1984.

[CH86] J. Cai and L. Hemachandra. *Exact Counting is as Easy as Approximate Counting.* Technical Report TR86-761, Cornell Department of Computer Science, June 1986. (Submitted for publication).

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[GK86] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *ACM Symposium on Theory of Computing*, pages 316–329, 1986.

[Har78] J. Hartmanis. *Feasible Computations and Provable Complexity Properties.* SIAM, 1978.

[HH86a] J. Hartmanis and L. Hemachandra. Complexity classes without machines: on complete languages for UP. In *Automata, Languages, and Programming (ICALP 1986)*, Springer-Verlag *Lecture Notes in Computer Science*, 1986.

[HH86b] J. Hartmanis and L. Hemachandra. On sparse oracles separating feasible complexity classes. In *STACS 1986: 3rd Annual Symnposium on Theoretical Aspects of Computer Science*, pages 321–333, Springer-Verlag *Lecture Notes in Computer Science #210*, 1986.

[HI85] J. Hartmanis and N. Immerman. On complete problems for NP$\bigcap$coNP. In *Automata, Languages, and Programming (ICALP 1985)*, pages 250–259, Springer-Verlag *Lecture Notes in Computer Science #194*, 1985.

[HY84] J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34:17–32, 1984.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *ACM Sym. on Theory of Computing*, pages 302–309, 1980.

[Ko85] K. Ko. Continuous optimization problems and a polynomial hierarchy of real functions. *Journal of Complexity*, 1:210–231, 1985.

[Rab76] M. Rabin. Probabilistic algorithms. In *Algorithms and Complexity*, Academic Press, 1976.

[Rud] Steven Rudich. Third-hand rumor.

[Sav72] S. Savage. Computational work and time on finite machines. *Journal of the ACM*, 19:660–674, 1972.

[Sch86] U. Schoening. *Complexity and Structure*. Springer Verlag *Lecture Notes in Computer Science #211*, 1986.

[Sip85] M. Sipser. Compression and ranking. In *ACM Symposium on Theory of Computing*, pages 440–448, 1985.

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Sto85] L. Stockmeyer. On approximation algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, November 1985.

[Val79a] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val79b] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Yao85] A. Yao. Separating the Polynomial-time hierarchy by oracles. In *Proceedings IEEE Symposium on Foundations of Computer Sciece*, pages 1–10, 1985.

[Yap83] C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.