Using Randomness to Characterize the Complexity of Computation

Lane A. Hemachandra and Gerd Wechsung

Technical Report 286
May 1989
Using Randomness to Characterize the Complexity of Computation

Lane A. Hemachandra
Department of Computer Science
University of Rochester
Rochester, NY 14627 USA

Gerd Wechsung
Department of Mathematics
Friedrich Schiller University
Jena, 6900 GDR

May 1, 1989

He’s making a list, and checking it twice.
Gonna find out who’s naughty or nice.
—Holiday Song

1This report revises and extends [Hem86], and will be presented at the IFIP 11th World Computer Conference [HW].

2This research was supported in part by a Fannie and John Hertz Fellowship, a Hewlett-Packard Corporation equipment grant, and the National Science Foundation under grants DCR-8520597, CCR-8809174, CCR-8996198, and Presidential Young Investigator Award CCR-8957604. Some of this work was done while at Cornell and Columbia Universities, and while visiting Gerd Wechsung in Jena.
Abstract

Kolmogorov complexity—the study of the randomness of strings—has developed into a fundamental tool in proving lower bounds in computation and in constructing oracles separating complexity classes. In this paper, we show that Kolmogorov complexity is a central tool in the understanding of deterministic and nondeterministic complexity classes and hierarchies; we show that many collapses of computational complexity classes can be completely characterized in terms of Kolmogorov complexity.

We discuss P, NP, unique polynomial time, the polynomial hierarchy, and the exponential hierarchy. We show that, for many complexity classes $C$, $C$ equals a smaller complexity class unless some language in $C$ is accepted only by machines whose execution creates computational structures with a non-trivial degree of randomness. Our fundamental proof technique is a divide and conquer scheme on the tree of potential computational structures.
1 Introduction

1.1 Kolmogorov Complexity: Definitions and Background

Kolmogorov complexity quantifies the randomness of a string. The Kolmogorov complexity of a string is the size of the shortest program printing the string; $K(x) = \min\{|y| \mid M_U(y) = x\}$, where $M_U$ is a universal Turing machine. We might describe a string for which $K(x) = \log \log |x|$ as highly non-random (highly compressible), and we might describe a string for which $K(x) = |x|$ as quite random.

Kolmogorov complexity was defined in the 1960's independently “by Solomonoff in Cambridge, Massachusetts, Kolmogorov in Moscow, and Chaitin in New York” ([LV88], which discusses the discoveries and surveys the applications that Kolmogorov complexity has found in computer science). In classical Kolmogorov complexity, no time bound is placed on how long the universal machine may run. Thus the Kolmogorov complexity of a string is badly noncomputable. In the 1980’s, Hartmanis [Har83] and Sipser [Sip83] introduced a time-bounded, and thus computable, version of Kolmogorov complexity. This theory is known as generalized Kolmogorov complexity; in this paper, we refer to it simply as Kolmogorov complexity, as we will never use the non-time-bounded classical Kolmogorov complexity. Following [Har83], we define time-bounded Kolmogorov sets.

**Definition 1.1** $K[s(n), t(n)] = \{x \mid (\exists y)[|y| \leq s(|x|) \land M_U(y) \text{ prints } x \text{ within } t(|x|) \text{ steps}]\}.$

Similarly, we define time-bounded Kolmogorov sets relative to some string.

**Definition 1.2** $K[s(n), t(n) \mid z] = \{x \mid (\exists y)[|y| \leq s(|x|) \land M_U(y\#z) \text{ prints } x \text{ within } t(|x|) \text{ steps}]\}.$

Each $K[s(n), t(n)]$ is not a complexity class, but rather a set of strings of some degree of non-randomness. Intuitively, these are the strings $x$ that can be compressed to length $s(|x|)$ yet still are recoverable in time $t(|x|)$ by a universal Turing machine. Of particular interest to us will be the “K-log-poly” family of sets: $K[c \log n, n^c]$ and $K[c \log n, n^c \mid z]$. These strings can be recovered, in time polynomial in their lengths,
from exponentially shorter strings.\footnote{To avoid trivial problems when \( n \leq 1 \), we consider these shorthand for, e.g., \( K[c + c \log n, c + n^c] \).}

Sipser uses time-bounded Kolmogorov complexity to study probabilistic classes, and places bounded probabilistic polynomial time in the second level of the polynomial hierarchy ([Sip83], see also Lautemann [Lau83]). Kolmogorov complexity is a key tool in proving lower bounds on computation times (see Section 5 of [LV88] for a survey of such results). Hartmanis [Har83] suggests and studies the relationship between Kolmogorov sets and feasible complexity classes such as P and NP. It is in this spirit of understanding the structure of P and NP via Kolmogorov complexity that this paper is written.

Much work has already been done on the connection between the K-log-poly Kolmogorov classes and the structure of complexity classes. [Adl79] shows a connection between \( P = NP \) and the possibility that NP sets have certificates from \( K[c \log n, n^c] \). [BB86] and [HH86] show a connection between the \( P = NP \) question and the \( K[c \log n, n^c] \) sets as oracles; in relativized worlds where \( P = NP \), the sparse oracles that separate P from NP are exactly those that escape the K-log-poly classes.

**Theorem 1.3** [HH86, HH88b] If \( P = NP \) and \( S \) is sparse then

\[
P^S \neq NP^S \iff (\forall c)[S \not\subseteq K^{c \log n, n^c}]\]

This theorem gives a characterization of the \( P = NP \) question in terms of Kolmogorov complexity and relativization: \( P = NP \iff (\forall S)[(\exists c)[S \subseteq K^{c \log n, n^c}]] \Rightarrow P^S = NP^S] \). The theorem also broadens previous positive relativization results [Boo74, SMRB83, BBL+84].

Balcázar and Book [BB86] study the connections between K-log-poly and Schöning's low hierarchy [Sch83]. Allender [All86b, AR88] notes that the K-log-poly sets, and the equivalent notion of P-printability, are useful in studying ranking function of sparse sets.

This paper presents direct connections between Kolmogorov complexity and the structure of feasible complexity classes. Indeed, we'll see that Kolmogorov complexity to a large extent characterizes the structure of feasible computations.
1.2 Definitions

Without loss of generality, we assume that all our machines are of standard form and use their full time bound on each input.

Definition 1.4
1. Let \{P_j\} (respectively \{N_j\}) be a standard enumeration of polynomial time deterministic (nondeterministic) Turing machines such that machine \(P_j\) on input \(x\) runs for exactly \(|x|^j + j\) steps, and each computation path of \(N_j(x)\) is exactly \(|x|^j + j\) steps long. W.l.o.g., each state of machine \(N_j\) has exactly two possible successor states.

2. The certificates of machine \(N_j\) on input \(x\), Certificates\(_{N_j}(x)\), are just the accepting paths (if any) of \(N_j(x)\). Each will be of size \(|x|^j + j\).

3. \(\Delta^p_2 = \text{def} \ P^{\text{NP}} \ [GJ79]\). A \(\Delta^p_2\) machine is a P machine with an NP oracle. At each step the answer tape from the oracle will contain the string 0, unless the oracle has just given the reply 1 (=“yes”) to a query, in which case the tape will contain a 1.

4. The pronouncement of a \(\Delta^p_2\) machine \(P^N_k(x)\) on input \(x\), Pronouncement\(_{P^N_k}(x)\), is the vector whose \(i\)'th component is the value on the answer tape at step \(i\) of the run of \(P^N_k(x)\). Throughout this paper, we use \(M^N\) as a shorthand for \(M^{L(N)}\), where \(M\) and \(N\) are machines.

5. \(\Theta^p_2 = \text{def} \ P^{\text{NP}[\log]} = \{L : \text{ for some } k, j, \text{ and } c, L = L(P^N_k) \text{ and for every input } x, \text{ during the run of } P^N_k(x) \text{ the oracle is queried at most } c \log |x| \text{ times}\} \ [PZ83, \text{Wag88}]\).

That is, \(P^{\text{NP}[\log]}\) is \(P^{\text{NP}}\) with the base machine restricted to \(O(\log n)\) oracle queries. The pronouncement of a \(\Delta^p_2\) machine on input \(x\) is essentially the list of answers that

---

2The assumption, made throughout this paper, that all machines are standard machines that run for exactly their clocked time bound, is not an empty one as used in this paper; since Kolmogorov complexity is defined in terms of the certificates (and later in the paper in terms of the pronouncements) and not in terms of the size of the input, nonstandard machines whose certificates are extremely small can have, in a trivial way, Kolmogorov complex certificates. To avoid this fluke, one would have to refine the definition of Kolmogorov complexity so that the computation time is not sensitive only to the size of the uncompressed string.
the oracle give the base machine (plus some placeholder 0's for steps during which no queries were made).

$\Theta^P_2$ has emerged as a class of great interest in recent years. It is the class to which the polynomial hierarchy collapses if NP has sparse Turing-complete sets [Kad87], it has natural complete sets [Kre86, Kad87, Wag88], and it is equal to the class of sets that truth-table reduce to some NP set [Hem87b, Wag88, BH88].

The reason we have chosen our standard model machines to always run exactly for their time bound is to avoid problems with the fact that the time bounds of common definitions of generalized Kolmogorov complexity [Har83, Sip83] are stated with respect to output size, and thus very short certificates and pronouncements may seem artificially complex. Our standard model is an alternative to the approach of [Wat86], where the definition of Kolmogorov complexity explicitly includes both input and output sizes.

1.3 Overview

NP is the class of sets with succinct certificates. A result of Adleman [Adl79, Theorem 3], rephrased here in the terminology of Kolmogorov complexity, shows that:

$$P = NP \text{ if and only if all NP sets have certificates that are Kolmogorov simple relative to the input string.}$$

That is, $(\forall N_i) (\exists c) (\forall x \text{ accepted by } N_i) [Certificates_{N_i}(x) \cap K[c \log n, n^c \mid x] \neq \emptyset]$. Thus NP differs from P exactly when NP machines can “manufacture randomness”: when they accept infinitely many input strings only via certificates Kolmogorov-far from the input string. This result has been used, interpreted, and expanded by many researchers, in particular Allender [All86a] and Watanabe [Wat86].

Following Adleman's lead, Section 2.1 characterizes the $P^{NP} = P^{NP[log]}$ question in Kolmogorov terms. Recall that the pronunciation of a $\Delta^P_2$ machine (i.e., a P machine with an NP oracle) running on some input string is essentially the sequence of answers given by the oracle at each step. We show that
\( \mathbb{P}^{\mathbb{NP}} = \mathbb{P}^{\mathbb{NP}[\log]} \) if and only if all \( \Delta_2^p \) sets are accepted by some \( \Delta_2^p \) machine with pronouncements that are Kolmogorov simple relative to the input string.

That is, \( (\forall L \in \Delta_2^p)(\exists P_i, N_j, c)(\forall x)[\text{Pronouncement}_{P_i}^{N_j}(x) \in K[c \log n, n^c | x]] \). Thus \( \mathbb{P}^{\mathbb{NP}} \) differs from \( \mathbb{P}^{\mathbb{NP}[\log]} \) exactly if some \( \Delta_2^p \) languages have only machines that have pronouncements Kolmogorov-far from infinitely many input strings.

Section 2.2 further applies these techniques. We derive Kolmogorov results about the structure of truth-table classes, counting classes, unique polynomial time, the polynomial hierarchy, and the exponential hierarchy.

In each case, we must find the key to a class—the computational object that will display the class’s connection to randomness. For \( \mathbb{NP} \), the key is certificates. For \( \mathbb{P}^{\mathbb{NP}} \) and truth-table classes, the key is pronouncements. For \( \#P \), the key is the number of accepting paths.

The technique used to prove these results is divide and conquer coupled with efficient oracle usage and the properties of Kolmogorov complexity and pronouncements. The theorems show that Kolmogorov complexity to a great extent characterizes the structure of the world of feasible computation.

2 Results

2.1 A Kolmogorov Characterization of \( \mathbb{P}^{\mathbb{NP}[\log]} = \mathbb{P}^{\mathbb{NP}} \)

Adleman [Adl79] (see also [Wat86]) shows, with different terminology, that:

**Theorem 2.1** [Adl79] \( \mathbb{P} = \mathbb{NP} \iff \mathbb{NP} \) has Kolmogorov simple certificates relative to the input.

More precisely, he shows that \( \mathbb{P} = \mathbb{NP} \) if and only if \( (\forall N_i, c)(\exists x \in L(N_i))[\text{Certificates}_{N_i}(x) \cap K[c \log n, n^c | x] \neq \emptyset] \).

The proof is direct. If \( \mathbb{P} = \mathbb{NP} \) certificates are trivialized: by using the self-reducibility of \( \text{SAT} \) [Sch86] we can easily find, e.g., the lexicographically smallest certificate. Going the other way, there are only polynomially many simple strings
relative to an input and they can be easily obtained. We need only check if one of them is a valid certificate.

Though the proof is simple, the impact of Adleman's result is great. He shows that the structure of computational classes is tied to the theory of Kolmogorov complexity.

The main result of this paper extends this approach, and studies the connection between randomness and the $P^{NP[\log]} = P^{NP}$ question. $P^{NP[\log]}$ is a class that, in a recent flurry of activity, is coming into its own. A well-known extension of the Karp-Lipton "small circuits" theorem says that if there is a sparse oracle $S \in NP$ such that $NP \subseteq P^S$, then the polynomial hierarchy is contained in $P^{NP} =_{def} \Delta^p_0$ (Mahaney [Mah80], also see [Lon82] for a further extension). Research of Kadin [Kad87] shows that the conclusion can be strengthened to "the polynomial hierarchy is contained in $P^{NP[\log]}$," and that this is essentially optimal in some oracle worlds. It is thus possible that $P^{NP[\log]}$ describes the complexity of the polynomial hierarchy.

Another recent indication of the centrality of $P^{NP[\log]}$ comes from the study of complete languages. Complete languages provide a cornerstone for our understanding and manipulations of a complexity class. Without SAT and other standard NP-complete languages as stepping stones, our understanding of the omnipresence of potential intractability would be sharply reduced [GJ79]. Indeed, classes such as UP, NP∩coNP, R, and BPP that lack known complete languages (Sipser [Sip82], Hartmanis and Immerman [HI85], [HH88a]) require special and often complex proof techniques [HH88a, Hem88]. Thus complete problems not only show us that some problem embodies the full complexity of a class, but also gives us access to an ever-expanding array of elegant and powerful proof techniques.

Until recently no complete problems were known for $P^{NP[\log]}$. During the last few years, many such problems have been discovered (Kadin [Kad87]—Uniq-Opt-Clause-SAT, [Kre86], [Wag88]).

Yet another motivation to study the $P^{NP[\log]} = P^{NP}$ question comes from the work of Krentel [Kre86]. Krentel distinguishes between various optimization functions based on the amount they use an NP oracle. Relatedly, he notes that little use

---

3Get them by using the effective equivalence between Kolmogorov complexity and P-printability noted in [All86b, BB86, HH86]. That is, run the universal machine with all logarithmically small strings as auxiliary input.
is equivalent to much use of the NP oracle only if $P$ equals $NP$.

$$P^{NP^{\log}} = P^{NP} \iff P = NP$$ [Kre86, Thm 8], where $CF$ is the class of functions computable by canonical machines for class $C$.

Since Krentel has characterized the function version of the $P^{NP^{\log}}$-vs-$P^{NP}$ question, it is natural to wonder about the language version. The following theorem completely characterizes the language version in terms of Kolmogorov complexity. For the harder direction, we present two proofs. The first is based on a tree pruning scheme, the second is a somewhat elaborate use of the census function techniques developed to collapse complexity hierarchies [Hem87b, Hem87a, Kad87, LJK87, Tod87, SW88].

**Definition 2.2** We say that a $\Delta^P_2$ machine $P^{NJ}_i$ has Kolmogorov simple pronouncements if $$(\exists c) (\forall x) [\text{Pronouncement}_{P^{NJ}_i}(x) \in K[c \log n, n^c | x]].$$

**Theorem 2.3** $P^{NP^{\log}} = P^{NP}$ if and only if each $\Delta^P_2$ language is accepted by some $\Delta^P_2$ machine with Kolmogorov simple pronouncements.

**Proof of Theorem 2.3**

$\implies$ If $P^{NP^{\log}} = P^{NP}$, then each $\Delta^P_2$ language $L$ trivially has Kolmogorov simple pronouncements. The $P^{NP^{\log}}$ machine accepting $L$ on input $x$ only has $|x|^{O(1)}$ plausible candidate pronouncements (corresponding to all possible answers to the $O(\log n)$ oracle queries) that each thus can be given a short name.

$\iff$ Simply summarized, put the plausible candidate pronouncements in a tree and repeatedly prune the tree, using an NP oracle very sparingly.

More precisely, let $L \in P^{NP}$. We wish to show that $L \in P^{SAT^{\log}}$. Let $P^{NJ}_i$ be the $\Delta^P_2$ machine that accepts $L$ and has simple pronouncements.

Suppose we wish to know if $x \in L$. For the $c$ mentioned in the theorem, we can (by the assumption of simple pronouncements) run the universal machine (Definition 1.2) on all strings of length $\leq c \log(|x|^i + i)$ to find at most $2(|x|^i + i)^c$ candidate pronouncements.

We must be careful here. To simply check each of the candidate pronouncements individually would be too expensive. There are a polynomial number of candidates and we wish to use only logarithmic access to our oracle.
To achieve just logarithmic access, we exhibit the virtues of divide and conquer on the tree of candidate pronouncements. Form the candidate pronouncements (Figure 1a) into an ordered binary tree (Figure 1b). The tree has at most $2(|x|^i + i)$ leaves.

Now, choose a node, the **splitting point**, of the tree that has in the subtree underneath it at least one quarter and at most one half of the leaves of the tree (Figure 1c). It is easy to do this. We wish to know if the path from the root to the splitting point (call it the **splitting path**), represents the actual action of $P_i^{N_j}(x)$. We can do this with two calls to our SAT oracle.

With the first call we insure that the queries represented by 1's (thus, the queries that claim that they received yes answers from $N_j$) in the splitting path really get yes answers. How? Simulate the run of $P_i^{N_j}(x)$ pretending that the splitting path is correct; we find the names of the queries made along the splitting path. We (1) reduce all the queries corresponding to 1's on the splitting path to queries to the NP-complete set SAT, (2) take the logical AND of these formulas (assigning distinct variable names), and (3) ask SAT about this resulting large formula. Clearly, the path's ones are correct if and only if SAT accepts the large formula.

With the second call to our SAT oracle, we do the same for the 0's that represent "no" replies from $N_j$. Recall that some 0's merely represent steps when no query was made, but we can simulate the run of $P_i(x)$, using the splitting path for oracle answers, and easily detect which 0's are of this type, assuming the splitting path is correct. Now, (1) reduce all the queries corresponding to 0's on the splitting path to queries to SAT, (2) take the logical OR of these formulas (assigning distinct variable names), and (3) ask SAT about this resulting large formula. Clearly, the path's zeros are correct if and only if SAT rejects the large formula—i.e., $N_j$ really rejects all the queries the path thinks it rejects.

If we find that both the 0's and the 1's on the splitting path are correct, then the path corresponds to a correct prefix of the pronouncement. Throw out all of the tree except the paths from the root of the tree to the leaves that are children of the splitting point. We throw out over one quarter of the tree's leaves with just two uses.

---

4For example, call the weight of a node the number of leaves in the subtree it is the root of. Start at the root and, until you arrive at a node with between one quarter and one half of the leaves beneath it, repeatedly move to your heavier child.
of SAT.

On the other hand, if we find that either the 0's or the 1's were lies (so the actual
pronouncement of \( P_t^{\mathcal{N}_j}(x) \) does not have the splitting path as a prefix), then throw
out the subtree rooted at the splitting node. Again, we throw out over one quarter
of the tree's leaves with just two uses of SAT.

This approach, divide-and-conquer combined with economically combining queries,
throws out 25% of the tree's leaves with just two queries to SAT. By iterating the
process, after a logarithmic number of rounds, we have used only \( O(\log |x|) \) queries
to SAT, and have pruned the tree to either (1) a single correct pronouncement or (2)
no paths at all. In the first case we accept \( x \) and the second case we reject \( x \). QED

Alternate Proof of \( \equiv \) Direction of Theorem 2.3

We sketch an alternate proof that uses census functions. Use the assumption of
Kolmogorov simple pronouncement and the universal Turing machine to get, in poly­
nomial time, a polynomial-sized set of candidate pronouncements.

We'll call a candidate pronouncement, \( d \), nice for \((x, i, j)\) if, when \( P_t^{(i)}(x) \) is sim­
ulated using \( d \) as the oracle answers, all the queries that would be asked at steps
where \( d \) says "a query in \( L(N_j) \) is asked" are indeed in \( L(N_j) \). Similarly, we'll call a
candidate pronouncement, \( d \), naughty for \((x, i, j)\) if, when \( P_t^{(i)}(x) \) is simulated using
\( d \) as the oracle answers, there is some query that would be asked at some step where
\( d \) says "the query is not in \( L(N_j) \)" such that the query in fact is in \( L(N_j) \).

Consider the NP languages: \( L_1 = \{x\#i\#j\#1|x|^i+i\#1|x|^j+j\#d_1\#d_2\#\ldots\#d_z\#count \mid \text{at least count of the } d_k \text{'s, } 1 \leq k \leq z, \text{ are nice for } (x, i, j) \} \) and
\( L_2 = \{x\#i\#j\#1|x|^i+i\#1|x|^j+j\#d_1\#d_2\#\ldots\#d_z\#count \mid \text{at least count of the } d_k \text{'s, } 1 \leq k \leq z, \text{ are naughty for } (x, i, j) \} \).

Using, respectively, \( O(\log n) \) queries to \( L_1 \) and \( L_2 \) (or equivalently to SAT, since
each reduces to SAT), we find \( C_{Nice-Yeses} \) and \( C_{Naughty-Noes} \), which are respectively
the number of nice candidate pronouncements and the number of naughty candi­
date pronouncements. Given these two numbers, we can use \( O(\log n) \) calls to the
following NP language, with the numbers being used as the last two components,
to find the index \( k' \) of the candidate pronouncement \( d_{k'} \) (the first index if the
pronouncement appears multiple times) that is the correct pronouncement of \( P_t^{\mathcal{N}_j}(x) \):
\( L_3 = \{x\#i\#j\#1|x|^i+i\#1|x|^j+j\#d_1\#d_2\#\ldots\#d_z\#name\#C_{Nice}\#C_{Naughty} \mid \text{the follow-} \)
ing procedure accepts: guess $C_{Nice} d_k$'s that are provably nice\textsuperscript{5} for $(x, i, j)$ and guess $C_{Naughty} d_k$'s that are provably naughty\textsuperscript{6} for $(x, i, j)$; all paths that fails to guess these structures reject; the paths (if any) that correctly guess these structures then assume that all $d_k$'s other than the $C_{Nice}$ ones guessed are not nice and that all the $d_k$'s other than the $C_{Naughty}$ ones guessed are not naughty; notice which $d_k$, \textit{within our current assumptions}, are both nice and not naughty (and thus—if our assumption is correct, as it will be if $C_{Naughty}$ and $C_{Nice}$ are the true census functions of the naughty and nice strings—have all their yes answers and all their no answers correct), and if any such $d_k$ has $k \leq \text{name}$, then we accept\}. Now that we know the correct pronounce­ment $dk_l$, it is easy to use it, $P_i$, and $x$ to determine—by running $P_i^{(k)}(x)$ using $d_k$ as the oracle answers, whether it accepts. Thus $P_{NP} = P_{NP[^{log}]}$ under our assumptions.

QED

PP, probabilistic polynomial time, is the two-sided error class introduced by Gill [GiI\textsuperscript{77}]. Since $P_{NP[^{log}]} \subseteq PP$ [BHW, Tod\textsuperscript{89}], our Kolmogorov characterization of the $P_{NP[^{log}]}$ vs. $P_{NP}$ question yields:

\textbf{Corollary 2.4} If each $\Delta^P_2$ language is accepted by some $\Delta^P_2$ machine with Kolmogorov simple pronouncements, then $\Delta^P_2 \subseteq PP$.

\section{2.2 Related Results}

\subsection{2.2.1 Another Kolmogorov Characterization of P=NP and Some Comments on Functions versus Languages}

The same techniques we use to prove Theorem 2.3 also show the following result. This yields an alternate Kolmogorov characterization of the $P = NP$ question, and an alternate way of viewing the work of Krentel [Kre\textsuperscript{86}].

\textbf{Theorem 2.5} $P_{NP[^{log}]} F = P_{NP} F$ if and only if all $\Delta^P_2$ machines have Kolmogorov simple pronouncements.

\textsuperscript{5}That is, their “yes”s (viewed as the queries dictated by the pronouncements for the run of $P_i^{(k)}(x)$) are all indeed provably (by guessing certificates) in $L(N_j)$.

\textsuperscript{6}That is, they each have at least one “no” that in fact is provably (by guessing a certificate) in $L(N_j)$.
Corollary 2.6 (Implicit in [Kre86]) \( P = NP \) if and only if all \( \Delta^P_2 \) machines have Kolmogorov simple pronouncements.

Combining this with Adleman's result and the easy observation that \( \Delta^P_2 \) machines can find the smallest certificate of an NP machine, and the fact that if \( P = NP \) then \( \Delta^P_2 \) machines will have horribly simple pronouncements, we have

\[
P = NP \iff P^{NP[\log]} = P^{NP}.\]

But this is Krentel's theorem (see Section 2.1). This is not surprising; once we understand the relationship between certificates and pronouncements, and the above theorem, it is clear that Adleman's (page 5) and Krentel's results are essentially the same—both results exploit the destructive impact of the existence of easy certificates to conclude that \( P = NP \).

Krentel also notes that his results indicate that it may be harder to collapse function classes than language classes; in the [BGS75] world where \( P \neq NP = \text{coNP} \), the language classes \( P^{NP[\log]} \) and \( P^{NP} \) are equal but the function classes \( P^{NP[\log]}F \) and \( P^{NP}F \) are different. Our characterization shows exactly how hard it is to collapse the language classes.

- \( P^{NP[\log]} = P^{NP} \iff \) every \( \Delta^P_2 \) language is accepted by some \( \Delta^P_2 \) machine with Kolmogorov simple pronouncements.
- \( P^{NP[\log]}F = P^{NP}F \iff \) every \( \Delta^P_2 \) machine has Kolmogorov simple pronouncements \( \iff P = NP \).

2.2.2 Truth-Table and Counting Classes

The same techniques link the complexity of both truth-table and counting classes to Kolmogorov complexity. Briefly stated,

- the class of sets that truth-table reduce to NP equals \( P^{NP} \) if and only if all truth-table machines have Kolmogorov simple answer-vectors (relative to the input string), and
the power of counting falls into polynomial time \((P = P^p)\) if and only if the number of satisfying assignments to all Boolean formulas is Kolmogorov simple (relative to the formula).

We state these results briefly.

**Definition 2.7** \(P^P_{tt} = \{L \leq^P_{tt} NP\}\).

That is, \(L \in P^P_{tt}\) \((L\) truth-table reduces to \(NP\) \([Yes83]\)) if there is a polynomial machine \(P^P_{i,tt}\) that on input \(x\):

1. without using its oracle, runs for a while and makes a list of questions (w.l.o.g. at least \(|x|\) questions) for \(NP\), then
2. simultaneously has all its questions answered by the \(NP\) oracle, and then
3. runs a bit more (without using its oracle) and determines if \(x \in L\).

We denote such a polynomial time machine with a "tt" (truth-table) subscript. The answer-vector of a truth-table machine is the vector of answers returned in step 2 above. We use the following folk theorem (see [Hem86, Hem87b, KSW87, Wag88, BH88]).

**Fact 2.8** \(P^{NP[\log]} = P^P_{tt} \subseteq P^{NP}\).

**Theorem 2.9**

1. Every \(P^P_{tt}\) language is accepted by some \(P^{NP}\) machine with answer-vectors Kolmogorov simple relative to the input.
2. \(P^{NP[\log]} = P^P_{tt}\) if and only if every \(P^P_{tt}\) machine has answer-sets Kolmogorov simple relative to the input.

Similar results apply to counting. Valiant's counting class \(\#P\) \([Val79a, Val79b]\) counts the accepting paths of \(NP\) Turing machines. It is easy to see, as a corollary to recent work by Cai and Hemachandra\(^7\) \([CH86]\), that:

\(^7\)They show that the exact and "approximate" versions of \(\#SAT\) are Turing equivalent. Their definition of the approximate version is simply that given a formula \(f\) we can P-print a short list
Theorem 2.10 [CH] \( P = \text{P}^\#P \iff \) for every Boolean formula \( f \), the string \(< f \#\text{number of satisfying assignments of } f >\) is Kolmogorov simple relative to \( f \).

The use of the term "Kolmogorov simple" in Theorem 2.10 is more delicate than that in the previous results. Due to limitations in our knowledge about approximate counting, we must explicitly choose the space constant in the \( K\log\text{-}\text{poly} \) class to be so small that for formulas of size \( n \), we have at most \( O(n^{1-\epsilon}) \) possible ranks. Theorem 2.10 is closely related to recent work showing that \( P = \text{P}^\#P \) is exactly the condition under which many complexity classes are rankable [GS85, Huy88, HR].

2.2.3 Unique Polynomial Time, the Polynomial Hierarchy, and the Exponential Hierarchy

We briefly note that similar results can be proven for many classes, using the techniques of this paper. For unique polynomial time, \( \text{UP} \) [Val76, HH88a], the class of languages accepted by categorical machines (nondeterministic polynomial-time Turing machines that on no input have more than one accepting path), the following is an immediate analog of Adleman’s P-vs-NP results (Theorem 2.1); one can also state a similar result for \( \text{FewP} \) [All86b, CH89].

Definition 2.11 [Val76]

1. A categorical machine is a nondeterministic polynomial-time Turing machine \( N_i \) such that for all \( x \), \( N_i(x) \) has at most one accepting path.

2. \( \text{UP} = \{ L(N_i) \mid N_i \text{ is a categorical Turing machine} \} \).

Theorem 2.12 \( P = \text{UP} \) if and only if every categorical machine has Kolmogorov simple certificates relative to its input. That is, \((\forall \text{ categorical } N_i) (\exists c) (\forall x \text{ accepted by } N_i) [\text{Certificates}_{N_i}(x) \cap K[c\log n, n^c \mid x] \neq \emptyset] \).

(of size \( O(|f|^{1-\epsilon}) \)) that contains as a list element the correct number of solutions of the formula. This is essentially a Kolmogorov complexity statement—the number of solutions to a formula can be expressed with only a small number of bits, given the formula. That small set of bits is used to specify which place in the list is occupied by the correct formula.
Using the tree pruning divide and conquer technique of this paper, one can prove versions of our results that apply to many other classes. In general, the technique is suitable for application to classes \( C \) that have the properties that:

- If \( A \in C \) and \( p \) is a polynomial, then \( \{ < n, w_1, \ldots, w_k > | n \in N \text{ and } k \leq p(n) \text{ and } (\forall i : 1 \leq i \leq k)[|w_i| \leq p(n)] \text{ and } (\forall i : 1 \leq i \leq k)[w_i \in A] \} \in C \).

- If \( A \in C \) and \( p \) is a polynomial, then \( \{ < n, w_1, \ldots, w_k > | n \in N \text{ and } k \leq p(n) \text{ and } (\forall i : 1 \leq i \leq k)[|w_i| \leq p(n)] \text{ and } (\exists i : 1 \leq i \leq k)[w_i \in A] \} \in C \).

In particular, we state analogs of Theorem 2.3 for the exponential hierarchy [HIS85, Hem87b] and for the polynomial hierarchy [Sto77]. The extension of pronouncements to classes beyond \( \Delta^p_2 \) is the natural one. Note that, as we have throughout the paper, we assume that the base machines are from a nice, clocked enumeration that runs exactly in the clock bound; for example, the E machines in Theorem 2.13 are from an enumeration \( \{ E_i \} \), where \( E_i \) runs for exactly \( 2^{2|x|} \) steps on input \( x \). The relation between \( E = NE \) and \( E = E^{NP} \) has been studied by Sewelson [Sew83] and Allender and Watanabe [AW88], and shown to be deeply connected to Kolmogorov complexity.

**Theorem 2.13** \( E^{NP[O(n)]} = E^{NP} \) if and only if \( (\forall L \in E^{NP})(\exists c)(\exists E^{NP} \text{ machine } M^N), \) such that \( M \)'s pronouncements are in \( K[c \log n, n^c | x] \).

**Definition 2.14** We say that a \( \Delta_{k+1} \) machine \( P_i^{N_j} \) (where \( P_i \) is a deterministic polynomial time transducer and \( N_j \) is a \( \Sigma^p_k \) alternating Turing machine) has Kolmogorov simple pronouncements if \( (\exists c)(\forall x) \text{ [Pronouncement}_{P_i^{N_j}}(x) \in K[c \log n, n^c | x]] \).

**Theorem 2.15** \( P^{\Sigma^p_k[log]} = P^{\Sigma^p_k} \) if and only if each \( \Delta^p_{k+1} \) language is accepted by some \( \Delta^p_{k+1} \) machine with Kolmogorov simple pronouncements.

The divide and conquer technique is not needed to prove results relating \( \Delta^p_k = \Sigma^p_k \) to Kolmogorov complexity. It is easy to see that Theorem 2.1 relativizes. Thus, we have:

**Theorem 2.16** (Relativized Adleman’s Theorem) \( P^A = NP^A \) if and only if every \( NP^A \) language is accepted by an \( NP^A \) machine with Kolmogorov simple certificates.
Corollary 2.17 For \( k \geq 1 \), \( \Sigma_k^P = \Delta_k^P \) if and only if \((\forall \text{ NP oracle machines } N_i \text{ and } L \in \Sigma_k^P) (\exists c)(\forall x \in L(N_i^k)) [\text{Certificates}_{N_i^k}(x) \cap K_{\Sigma_k^P}^P(c \log n, n^c \mid x) \neq \emptyset].\)

3 Conclusions

The theme of this paper is held in the question: can machines manufacture computational objects that are Kolmogorov-far from their input? That is, can they achieve some small amount of randomness relative to the input? This paper asks, for example, when NP machines can have random certificates, and when \( \text{P and NP working together (as } \Delta_2^P \text{)} \) can create random pronouncements, and when \( \text{P and NP working together (as } \text{P}_i^{\text{NP}} \text{) can create random answer-vectors, and when } \#P \text{ can have random counts of accepting paths.} \)

In each case, the question is completely characterized by the collapse of complexity classes. Thus the structure of complexity classes is linked to the ability of deterministic and nondeterministic machines to create random computational objects.

Acknowledgements

Professor Juris Hartmanis contributed numerous insights into the meaning of randomness and on the functions-vs-languages question. We thank Professor Eric Allender for kindly bringing to our attention important literature in this area, and for some fascinating discussions. We are grateful to Professor Jin-yi Cai for conversations on trees, truth-tables, and typesetting, and to Sanjay Jain for help with the alternate proof of Theorem 2.3. We thank anonymous IFIP World Computer Conference referees for many extremely helpful suggestions.

References


S. Toda. On probabilistic computations with a restricted access to NP oracles. manuscript, February 1989.


