Can P and NP Manufacture Randomness?

Lane A. Hemachandra*

TR 86-795

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.

Can P and NP Manufacture Randomness?

*Lane A. Hemachandra**

Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract

This paper studies how Kolmogorov complexity dictates the structure of standard deterministic and nondeterministic classes. We completely characterize, in Kolmogorov terms, when $P^{NP[\log]} = P^{NP}$, where [log] indicates that $O(\log n)$ oracle calls are made. We give a Kolmogorov characterization of P=NP that links the work of Adleman [Adl79] and Krentel [Kre86].

Briefly stated, *complexity classes collapse unless they can manufacture randomness.*

A Δ_2^P machine is a P machine with an NP oracle. The series of replies the NP oracle makes is called the *pronouncement.* We show that

> $P^{NP[\log]} = P^{NP}$ if and only if each Δ_2^P language is accepted by some Δ_2^P machine with Kolmogorov simple pronouncements[1]

(i.e., $(\forall P_i)\ (\exists c)\ (\forall x)\ [Pronouncements_{P_i^{SAT}}(x) \in K[c \log n, n^c \mid x]]$).

Turning to functions, we show that:

> $P^{NP[\log]}F = P^{NP}F$ if and only if all Δ_2^P machines have Kolmogorov simple pronouncements.[2]

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

[1]Whenever we say Kolmogorov simple, we refer to Kolmogorov simple relative to the input.

[2]CF is the class of functions computable by machines from class C.

Since $P^{NP[log]}F = P^{NP}F \iff P = NP$ [Kre86], the above gives an alternate Kolmogorov characterization of the P = NP question, which complements Adleman's classic characterization.

Our key technique is an oracle-based divide and conquer over a tree of potential pronouncements. The results generalize to many other classes, including truth-table classes and counting classes.

Thus, Kolmogorov complexity dictates the structure of the classes that are at the center of our understanding of feasible computation.

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 by Kolmogorov [Kol65], Chaitin [Cha66], and Solomonoff [Sol64]. No time bound is placed on how long the universal machine may run. Thus the Kolmogorov complexity of a string is badly noncomputable [Cha74]. In the 1980's, Hartmanis [Har83] and Sipser [Sip83] introduced time-bounded, and thus computable, versions of Kolmogorov complexity. Following [Har83], we define time-bounded Kolmogorov sets.

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

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

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

Each $K[s(n), t(n)]$ is not a complexity class, but rather a set of strings of some degree of non-randomness. 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.[3]

Sipser uses time-bounded complexity to study probabilistic classes, and places bounded probabilistic polynomial time in the second level of the polynomial hierarchy [Sip83]. 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. Hartmanis and Hemachandra [HH86b] show a connection between the P = NP question and the $K[c \log n, n^c]$ sets: 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 [HH86b] If P = NP and S is sparse then

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

This gives a characterization of the P = NP question in terms of Kolmogorov complexity and relativization: $\mathrm{P} = \mathrm{NP} \iff (\forall S \ni (\exists c)[S \subseteq K^S[c \log n, n^c]])[\mathrm{P}^S = \mathrm{NP}^S]$.

Balcazar and Book [BB86] study the connections between K-log-poly and Schoening's low hierarchy [Sch83]. Allender [All86] 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, unrelativized connections between Kolmogorov complexity and the structure of feasible complexity classes. Indeed, we'll see that Kolmogorov complexity dictates 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.

[3] To avoid trivial problems when $n \leq 1$, we consider these shorthand for, e.g., $K[c + c \log n, c + n^c \mid z]$.

Definition 1.4 1. Let $\{P_j\}$ (respectively $\{N_j\}$) be a standard enumeration of polynomial time deterministic (nondeterministic) Turing machines. W.l.o.g. 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. $\Delta_2^{\mathrm{P}} = \mathrm{P}^{\mathrm{NP}}$ [GJ79]. A Δ_2^{P} 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 Δ_2^{P} machine $P_k^{N_j}(x)$ on input x, $Pronouncement_{P_k^{N_j}}(x)$, is the vector whose i'th component is the value on the answer tape at step i of the run of $P_k^{N_j}(x)$.

5. $\mathrm{P}^{\mathrm{NP[log]}} = \{L \mid$ for some k, j, and c, $L = L(P_k^{N_j})$ and for every input x, during the run of $P_k^{N_j}(x)$ the oracle is queried at most $c \log |x|$ times$\}$.

That is, $\mathrm{P}^{\mathrm{NP[log]}}$ is P^{NP} with the base machine restricted to $O(\log n)$ oracle queries. The pronouncement of a Δ_2^{P} machine on input x is essentially the list of answers that the oracle give the base machine (plus some placeholder 0's for steps during which no queries were made).

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 if and only if all NP sets have certificates that are *Kolmogorov simple* relative to the input string.

That is, $(\forall N_i)\ (\exists c)\ (\forall x$ 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 [All] and Watanabe [Wat86].

Following Adleman's lead, Section 2.1 characterizes the $P^{NP} = P^{NP[log]}$ question in Kolmogorov terms. Recall that the *pronouncement* of a Δ_2^P 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

> $P^{NP} = P^{NP[log]}$ if and only if all Δ_2^P sets are accepted by some Δ_2^P machine with pronouncements that are *Kolmogorov simple* relative to the input string.

That is, $(\forall L \in \Delta_2^P)\ (\exists P_i,\ N_j\ c)\ (\forall x)\ [Pronouncement_{P_i^{N_j}}(x) \in K[c \log n, n^c \mid x]]$. *Thus* P^{NP} *differs from* $P^{NP[log]}$ *exactly when* P *and* NP, *working together, can manufacture randomness: some* Δ_2^P *languages have only machines that have pronouncements Kolmogorov-far from infinitely many input strings.*

Section 2.2 further applies these techniques. Merging the work of Krentel [Kre86] with our methods, we prove an alternate Kolmogorov characterization of the P = NP question:

> P = NP if and only if all Δ_2^P machines have pronouncements that are Kolmogorov simple relative to their input strings.

Also, we derive Kolmogorov results about the structure of truth-table classes and counting classes.

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

The techniques used to prove these results are direct and fundamental: divide and conquer coupled with efficient oracle usage and an understanding of Kolmogorov

complexity and pronouncements. Thus the proofs will be easily understood and internalized. Nonetheless, their intuitive sweep is great; they show that Kolmogorov complexity dictates the structure of the world of feasible computation.

2 Results

2.1 A Kolmogorov Characterization of $P^{NP[log]} = P^{NP}$

Adleman [Adl79] shows (with different terminology) that

$$P = NP \iff \text{NP has Kolmogorov simple certificates relative to the input.}$$

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

The proof is direct. If P = NP certificates are trivialized: by using the self-reducibility of 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. [4] We need only check if one of then is a true certificate.

Though the proof is simple, the impact if Adleman's result is great. He is suggesting that we tie the structure of computational classes to the theory of Kolmogorov complexity.

The main result of this paper follows this path 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 $S \in NP$ so $NP \subseteq P^S$, then $PH \subseteq P^{NP} =_{def} \Delta_2^P$ (Mahaney [Mah80], also see [Lon82] for a further extension). New research of Kadin [Kad86] shows that the conclusion can be strengthened to $PH \subseteq P^{NP[log]}$. 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

[4] Get them by using the effective equivalence between Kolmogorov complexity and P-printability noted in [All86,BB86,HH86b]. That is, run the universal machine with all logarithmically small strings as auxiliary input.

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 $\bigcap$ coNP, R, and BPP that lack known complete languages (Sipser [Sip82], Hartmanis and Immerman [HI85], [HH86a]) require special and often complex proof techniques [HH86a]. Thus complete problems not only show us that *some* problem embodies the full complexity of a class, but also gives us access to a vast array of simple, elegant, powerful proof techniques.

Until recently no complete problems were known for $\mathrm{P}^{\mathrm{NP[log]}}$. During the last year such problems have been discovered (Kadin [Kad86]—Uniq-Opt-Clause-SAT).

Yet another motivation to study the $\mathrm{P}^{\mathrm{NP[log]}} = \mathrm{P}^{\mathrm{NP}}$ question comes from the well-received 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 is equivalent to much use of the NP oracle only if P equals NP.

$$\mathrm{P}^{\mathrm{NP[log]}}\mathrm{F} = \mathrm{P}^{\mathrm{NP}}\mathrm{F} \Rightarrow \mathrm{P} = \mathrm{NP}\ \text{[Kre86, Thm 8]}.$$

Indeed, it is clear that the other direction is easy so:

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

Since Krentel has characterized the *function* version of the $\mathrm{P}^{\mathrm{NP[log]}}$-vs-$\mathrm{P}^{\mathrm{NP}}$ question, it is natural to wonder about the language version. We completely characterized the language version in terms of Kolmogorov complexity.

Theorem 2.1 $\mathrm{P}^{\mathrm{NP[log]}} = \mathrm{P}^{\mathrm{NP}}$ if and only if each Δ_2^{p} language is accepted by some Δ_2^{p} machine with Kolmogorov simple pronouncements (i.e., $(\forall P_i)(\exists c)(\forall x)\,[\mathit{Pronouncements}_{P_i^{\mathrm{SAT}}}(x) \in K[c \log n,\, n^c \mid x]]$).

Proof of Theorem 2.1

$\Longrightarrow$ If $\mathrm{P}^{\mathrm{NP[log]}} = \mathrm{P}^{\mathrm{NP}}$, then each Δ_2^{p} language L trivially has Kolmogorov simple pronouncements. The $\mathrm{P}^{\mathrm{NP[log]}}$ 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.

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

More precisely, let $L \in \mathrm{P}^{\mathrm{NP}}$. We wish to show that $L \in \mathrm{P}^{\mathrm{SAT[log]}}$. Let $P_i^{N_j}$ be the Δ_2^{P} 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 $2\left(|x|^i + i\right)^c$ candidate pronouncements.

We must be careful here. To simple 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 a ordered binary tree (Figure 1b). The tree has at most $2\left(|x|^i + i\right)^c$ 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.[5] 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 detect 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

[5] For 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.

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 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 $P_i^{N_j}(x)$ 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 polynomial 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**

We have proven our Kolmogorov characterization of the $\mathrm{P}^{\mathrm{NP}[\log]}$-vs-$\mathrm{P}^{\mathrm{NP}}$ question.

2.2 Related Results

2.2.1 Another Kolmogorov Characterization of P=NP and Some Comments on Functions versus Languages

The same techniques we use to prove Theorem 2.1 also show the following result. The corollary gives an alternate Kolmogorov characterization of the P = NP question.

Theorem 2.2 $\mathrm{P}^{\mathrm{NP}[\log]}\mathrm{F} = \mathrm{P}^{\mathrm{NP}}\mathrm{F}$ if and only if all Δ_2^{p} machines have Kolmogorov simple pronouncements.

Corollary 2.3 P = NP if and only if all Δ_2^{p} machines have Kolmogorov simple pronouncements.

Combining this with Adleman's result and the easy observation that Δ_2^{p} machines can find the smallest certificate of an NP machine, and the fact that if P = NP then

Δ_2^P machines will have horribly simple pronouncements, we have

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

But this is (a slightly strengthened version of) Krentel's theorem (page 7)! This is not surprising; once we understand the relationship between certificates and pronouncements, and the above theorem, it is clear that Adleman's (page 6) 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, of course, notes and classifies many interesting complete sets, and thus ties $P^{NP[log]}F$ to concrete problems, such as the optimization problem of clique.

Krentel also notes that his results indicate that it may be harder to collapse function classes than language classes; in the [BGS75] world where $P \neq NP = coNP$, 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 Δ_2^P language is accepted by some Δ_2^P machine with Kolmogorov simple pronouncements.
- $P^{NP[log]}F = P^{NP}F \iff$ every Δ_2^P 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 2.4 $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}} =_{def} \{L \leq_{\mathrm{tt}}^{\mathrm{P}} \mathrm{NP}\}$.

That is, $L \in \mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$ (L truth-table reduces to NP [Yes83]) if there is a polynomial machine $P_{i,\mathrm{tt}}^{\mathrm{NP}}$ 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.

Fact 2.5 $\mathrm{P}^{\mathrm{NP[log]}} \subseteq \mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}}$.

Theorem 2.6

1. $\mathrm{P}^{\mathrm{NP[log]}} = \mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$ if and only if every $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$ language is accepted by some $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$ machine with answer-vectors Kolmogorov simple relative to the input.
2. $\mathrm{P}^{\mathrm{NP[log]}}\mathrm{F} = \mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}\mathrm{F}$ if and only if every $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$ 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[6] [CH86], that:

Theorem 2.7 $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}} \iff$ for every Boolean formula f, the string $< f\#$number of satisfying assignments of $f >$ is Kolmogorov simple relative to x.

This is closely related to the work on ranking found in [Hem86]. We should note that the "Kolmogorov simple" here is more delicate that than in the previous results. Due

[6]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 (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.

to limitations in our knowledge about approximate counting, we must explicitly choose the space constant in the K-log-poly class to be so small that for formulas of size n, we have at most $O(n^{1-\epsilon})$ possible ranks.

3 Conclusions and Open Problems

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:

- if NP can manufacture random certificates,
- if P and NP working together (as Δ_2^{P}) can manufacture random pronouncements,
- if P and NP working together (as $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$) can manufacture random answer-vectors, and
- if #P can manufacture random counts of accepting paths.

In each case, the question is completely characterized by the collapse of complexity classes. Thus *complexity classes collapse unless they can manufacture randomness—the structure of complexity classes is dictated by the accessibility to machines of random computational objects.*

Acknowledgements

Professor Juris Hartmanis contributed numerous insights into the meaning of randomness and on the functions-vs-languages question. I thank Professor Eric Allender for kindly bringing to my attention important literature in this area, and for some fascinating discussions. I am grateful to Professor Jin-yi Cai for many helpful conversations on trees, truth-tables, and typesetting.

References

[Adl79] L. Adleman. *Time, Space, and Randomness.* Technical Report Tech Report MIT/LCS/TM-131, MIT, April 1979.

[All] E. Allender. Private communication.

[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.

[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.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[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).

[Cha66] G. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

[Cha74] G. Chaitin. Information-theoretic limitations of formal systems. *Journal of the ACM*, 21:403–424, 1974.

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

[Har83] J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 439–445, 1983.

[Hem86] L. Hemachandra. On ranking. 1986. Submitted.

[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.

[Kad86] J. Kadin. $P^{NP[\log]}$ and sparse Turing complete sets for NP. 1986. Manuscript.

[Kol65] A. Kolmogorov. Three approaches for defining the concept of information quantity. *Prob. Inform. Trans.*, 1:1–7, 1965.

[Kre86] M. Krentel. The complexity of optimization problems. In *ACM Symposium on Theory of Computing*, pages 69–76, May 1986.

[Lon82] T. J. Long. A note on sparse oracles for NP. *Journal of Computer and System Sciences*, 24:224–232, 1982.

[Mah80] S. Mahaney. Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 54–60, 1980.

[Sch83] U. Schoening. A low and a high hierarchy in NP. *Journal of Comp. and Sys. Sci.*, 27:14–28, 1983.

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

[Sip82] M. Sipser. On relativization and the existence of complete sets. In *Automata, Languages, and Programming (ICALP 1982)*, Springer-Verlag *Lecture Notes in Computer Science #140*, 1982.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *ACM Symposium on Theory of Computing*, pages 330–335, 1983.

[Sol64] R. Solomonoff. A formal theory of inductive inference. *Informantion and Control*, 7:224–254, 1964.

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

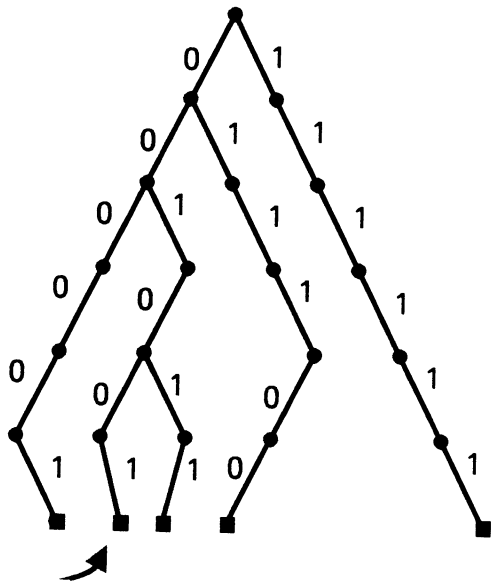
[Val79b] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Wat86] O. Watanabe. Generalized Kolmogorov complexity of computations. 1986. Manuscript.

[Yes83] Y. Yesha. On certain polynomial-time truth-table reducibilities of complete sets to sparse sets. *SIAM Journal on Computing*, 12(3):411–425, 1983.

000001
001001
001010
011100
111111

Figure 1a: Five Candidate Pronouncements



Candidate
001001

Figure 1b: An Ordered Binary Tree of Our Candidates

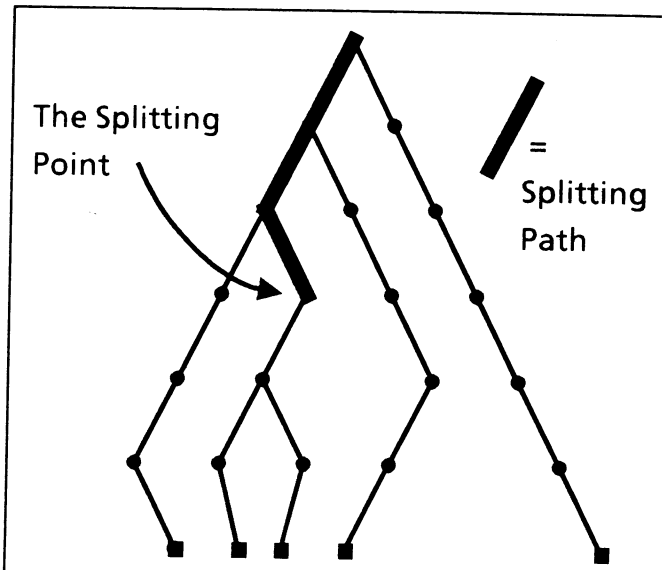


Figure 1c: The Splitting Point and Splitting Path

Figure 1: Examples of the Main Theorem's Construction