Enumerations of the Kolmogorov Function

Richard Beigel* Harry Buhrman§ Lance Fortnow¶ Luc Longpré||
Leen Torenvliet**

Abstract

We consider the hardness of enumerating k possible values for the Kolmogorov complexity function $C(x)$ so that one of them is correct. We show several results including

- Any computable enumerator for $C(x)$ must enumerate $\Omega(n)$ possibilities.
- If a k-enumerator (fixed k) for C is reducible to an r.e. set A then A is Turing-equivalent to the halting problem.
- Every nonrecursive set is not weak-truth-table reducible to some 2-enumerator for $C(x)$ or any other recursively-bounded function.
- The time-bounded enumeration question gives a new characterization of the class symP defined by Russell and Sundaram.
- Enumerating $O(\log n)$ values of the space-bounded Kolmogorov function remains hard for PSPACE.

1 Introduction

The Kolmogorov complexity of a binary string x, $C(x)$, is the size of the smallest program that outputs x. Kolmogorov complexity has its roots in the study of randomness: it is one way to measure randomness in a string. It has had a vast area of applications, including information theory, combinatorics, analysis of algorithms, distributed computing, statistical properties of long finite and of infinite sequences, learning theory, and even quantum information processing. It proved to be an invaluable tool in proving or simplifying the proofs of a large number of lower bounds.

The Kolmogorov complexity is not computable [Sol64, Kol65, Cha66], it is even hard for r.e. When a set or function is not computable or intractable, one often turns to the complexity of approximations. For example, would it be possible to approximate the Kolmogorov function to within

*URL: http://www.cis.temple.edu/∼beigel. E-mail: beigel@joda.cis.temple.edu. Supported in part by the National Science Foundation under grants CCR-9996021 and CCR-0049019. Temple University and NEC Research Institute. Address: Temple University, Dept. of Computer and Information Sciences, 1805 N. Broad St. Room 303, Philadelphia, PA 19122-6094, USA.

§URL: http://www.cwi.nl/cwi/people/Harry.Buhrman.html. E-mail: buhrman@cwi.nl. Partially supported by the EU through the 5th framework program FET. CWI, Kruislaan 413, 1098SJ Amsterdam, The Netherlands.

¶URL: http://www.neci.nj.nec.com/homepages/fortnow. E-mail: fortnow@research.nj.nec.com. NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA.

||URL: http://www.cs.utep.edu/longpre/longpre.html. E-mail: longpre@cs.utep.edu. Computer Science Dept., UTEP, El Paso, TX 79968, USA.

**URL: http://www.turing.wins.uva.nl/~leen. E-mail:leen@wins.uva.nl. University of Amsterdam, Plantage Muidergracht 24, 1018TV Amsterdam, The Netherlands.

reasonable bounds? It has been shown that (attributed to Kolmogorov in [ZL70]) Kolmogorov complexity can be approximated from above: there is a total recursive, monotonic decreasing function $f(t,x)$ such that $\lim_{t\to\infty} f(t,x) = C(x)$.

A different approach to approximation is that if we cannot compute the result of the function exactly, perhaps it would be possible to output several candidates for the result of the function, one of which is the actual result. Traditional approximations are a special case of this, in which the set of candidates is the set of numbers in a given range. This kind of approximation has been called enumeration complexity (see [BGGO93, ABG90, ABG$^+$00]). It is hence a natural question as to whether we can approximate the Kolmogorov function in this sense, or more precisely, how many values does a function need to output before it is guaranteed that one of the values is $C(x)$.

This is the main topic of this paper. By a simple table-lookup argument, $C(x)$ can be computably $(n-a)$-enumerated for every a, where $n = |x|$. There are even programming systems such that for some constant a, depending on the programming system, $C(x)$ can be computably n/a-enumerated (see the remark after Theorem 2). However, we show that for every programming system there is another constant c such that $C(x)$ cannot be computably n/c-enumerated.

Next we turn our attention to the question of whether any k-approximation of $C(x)$ is hard for r.e., as is $C(x)$ itself. We do not solve this question completely, but have some partial results. For every $k > 1$ we show that there exist k-enumerators for C that are not hard for r.e. under weak truth table reductions. This contrasts the $k = 1$ case, $C(x)$ itself, which is hard for r.e. under even normal truth table reductions [Kum96]. We leave the case for Turing reductions as an open problem. However, we show that if a k-enumeration f of C reduces to an r.e. set A, then A is Turing-complete for the r.e. sets. This implies that an enumeration that is not Turing hard is also necessarily not r.e.

Our results have some nice counterparts for polynomial-time reductions and polynomial space bounded versions of C. In Section 5 we show that the class symP [RS98] can be characterized by polynomial time reductions and 2-enumerators for C. Finally we show, contrasting the r.e. case, that every $(O \log n)$ enumerator of the polynomial space bounded Kolmogorov function is hard for PSPACE under nondeterministic polynomial time reductions.

2 Preliminaries

We assume the reader is familiar with basic notions in computational complexity theory. Fix a universal Turing machine U. Except as explicitly stated, all our results are independent of the particular choice of the universal machine. Strings are denoted as lower case letters $x, y, u, v, \ldots$ and are all elements of Σ^*, where $\Sigma = \{0,1\}$. We use the 1-1 correspondence between strings and binary numbers and have numbers sometimes appear as arguments to functions on strings and vice versa. Also we let functions defined on strings sometimes act on numbers where the length of the number is the logarithm of its value. We define the Kolmogorov complexity function $C(x|y)$ (see [LV97]) by $C(x|y) = \min\{|p| : U(p,y) = x\}$. We define unconditional Kolmogorov complexity by $C(x) = C(x|\lambda)$. Hartmanis [Har83] defined a time bounded version of Kolmogorov complexity, but resource bounded versions of Kolmogorov complexity date back as far as [Bar68]. (See also [LV97].) So C^s and C^t maybe defined as space and time bounded versions of C in the usual way. We define an *enumerator* for a function (e.g., C) as follows.

Definition 1 *Let* $k : \Sigma^* \mapsto \mathbb{N}$, $f : \Sigma^* \mapsto 2^{\mathrm{N}}$ *and* $h : \Sigma^* \mapsto \mathbb{N}$ *be functions such that* $f(x) = \{f_1^x, f_2^x, \ldots f_{k(x)}^x\}$. *We call* f *a* $k(x)$-enumerator *of* h *if* $h(x) \in f(x)$.

Often, $k(x)$ is a step function on lengths, i.e., $(\forall x, y)[|x| = |y| \implies k(x) = k(y)]$. Then, we will also use $k(n)$ instead of $k(x)$, meaning $k(|x|)$, where $|x| = n$. Also, $k(n)$ is often just a constant in this paper. Then, we will speak of 2-enumerators, 3-enumerators, k-enumerators etc. Without loss of generality we will assume in the sequel that if f is a $k(n)$-enumerator for g, say $f(x) = \{f_1^x, \ldots, f_{k(|x|)}^x\}$ then $f_i^x < f_j^x$ whenever $i < j$. We will call $f_1^x, \ldots, f_{k(|x|)}^x$, the *coordinates* of x.

A set A *weak-truth-table* (wtt) reduces to set B if A reduces to B via a Turing reduction M and there is recursive function g such that the reduction $M^B(x)$ never asks queries to B of length greater than $g(x)$.

3 Lower bounds

The fact that the Kolmogorov function itself is hard for r.e. is a folk theorem with a relatively easy proof. With C as an oracle we can find a function that majorizes $\min\{t \mid \{e\}_t(e)\downarrow\}$ whenever $\{e\}(e)$ converges. (In fact Kummer [Kum96] has shown that the halting problem, K, is tt-reducible to C.) Conversely, with K as an oracle we can easily decide the Kolmogorov complexity of any string. So, loosely speaking, C and K have the same hardness.

There also exists [LV97] a function $m(n)$ that is monotone nondecreasing and unbounded such that for all n, $m(n) \le \min\{C(x) \mid |x| = n\}$. Almost the same proof (in fact exactly the same proof as in [ZL70]) gives that $m(n)$ is also hard for r.e.: $\min\{n|m(n) > 2e\}$ must be larger than $\min\{t \mid \{e\}_t(e)\downarrow\}$ if $e \in K$. For if $e \in K$ and $\{e\}_{t'}(e)\downarrow$ only for $t' > \min\{n|m(n) > 2e\}$, then $C(0^{\min\{t' | \{e\}_{t'}(e)\downarrow\}}) > 2e$. Yet it is described by e. This can only be true for finitely many e.

This paper is not about the Kolmogorov function itself, but about enumerations of the Kolmogorov function. We ask how hard are these enumerations? Clearly, the complexity depends on the number of outputs of such a function. The Kolmogorov function itself is a 1-enumerator that is hard for r.e. On the other hand, for any constant a, we can build a recursive $n-a$-enumerator for C, essentially a function that outputs all values $c + a < v \le n + c$ on inputs of length n except on the finite number of strings that have $C(x) \le c + a$, where $C(x)$ is output explicitly. So the complexity somehow depends on the number of values that are output by the enumerator. We show first that no recursive function can do better than essentially enumerate all possible values of C.

Theorem 2 *Let f be a computable $k(n)$-enumerator for C. Then there is a constant a such that $k(n) > n/a$ for almost all n.*

Proof: Let c be the length of a description of f. Fix some length n and let $d = 2(c + C(n))$. The proof consists of two parts. We first prove that $k(n) \ge n/d$ for all but finitely many n. Then, using this result, we prove that there exists some constant a such that $k(n) \ge n/a$ for all but finitely many n.

For $x \in \Sigma^n$, let $f_1^x < \cdots < f_{k(n)}^x$ denote the coordinates of $f(x)$. We prove by induction on j that there can be at most 2^{jd} strings $x \in \Sigma^n$ such that the value $C(x)$ is among the first j coordinates of $f(x)$. Then, as there are 2^n strings $x \in \Sigma^n$, the assumption that f $k(n)$-enumerates C leads to $2^{k(n)d} \ge 2^n$ or $k(n) \ge n/d$.

Base case: Suppose there are more than 2^d strings $x \in \Sigma^n$ such that $f_1^x = C(x)$. In this case there exists $x \in \Sigma^n$ such that $f_1^x > d$. Consider the following program P_1: **begin** enumerate all strings x of length n until the first x such that $f_1^x > d$; output this string. **end**. for the string

x output by this program $C(x) \in f(x)$, so $C(x) \geq f_1^x > d$. Since $|P_1|$ is at most $c + C(n)$ plus a constant, $|P_1| \leq d$ for large enough n. However P_1 outputs a string x such that $C(x) > d$, a contradiction.

Induction: Suppose now that there are more than 2^{jd} strings $x \in \Sigma^n$ such that $C(x)$ is among the first j coordinates of $f(x)$. It follows that there must be a string $x \in \Sigma^n$ such that $f_j^x > jd$. By the induction hypothesis, there are at most $2^{(j-1)d}$ strings $x \in \Sigma^n$ such that $C(x)$ is among the first $j-1$ coordinates of $f(x)$. Let b be the number of strings $x \in \Sigma^n$ for which $C(x)$ is among the first $j-1$ coordinates of $f(x)$. Consider the following program P_2: **begin** By dovetailing all programs, find a set B consisting of b strings $x \in \Sigma^n$ such that each x is output by a program whose length is at most f_{j-1}^x. Output the lexicographically least string y in $\Sigma^n - B$ such that $f_j^y > jd$. **end**.

Now for some fixed constant p,

$$|P_2| \leq |b| + c + C(n) + p \leq (j-1)d + 1 + \frac{1}{2}d + p = jd - \frac{1}{2}d + 1 + p \leq jd - C(n) + 1 + p,$$

but P_2 outputs a string of complexity greater than jd, a contradiction for sufficiently large n.

Next we get rid of the $C(n)$ in d. Suppose that there exists a computable $k(n)$-enumerator f for C such that for every constant a there are infinitely many n such that $k(n) < n/a$. We know that $k(n) > n/(2(C(n) + c))$ for almost all n from the first part of the proof. Now fix some m such that for all $n \geq m$, $k(n) > n/(2(C(n) + c))$. By assumption, for every p there is an $n > m$ such that $k(n) < n/(2(C(m) + c + p))$. But $k(n) > n/(2(C(n) + c))$, so it must be the case that $C(n) > C(m) + p$. However, to describe n we just need f and m, which is a contradiction if p is chosen large enough. Hence there must exist some constant a such that $k(n) > n/a$ for almost all n. □

The result above is, in a sense, tight because there are programming systems, i.e., particular choices of the universal Turing machine, in which every program's length is divisible by a fixed natural number, so $C(x)$ is n/a-enumerable for some fixed a.

Recall Arslanov's completeness criterion [Ars81]: If F has r.e. degree and $h \leq_T F$ is such that $W_x \neq W_{h(x)}$ for almost all x, then F is complete.

Theorem 3 *Let k be a constant. Let f be a k-enumerator for C that is of r.e. degree, then f is Turing-complete for r.e.*

Proof: Let j be minimal such that f_j^x is unbounded. Let m be such that for all strings y of length m or larger, $C(y) > f_i^x$ for all $i < j$ and $x \in \Sigma^*$.

Our algorithm with j and m hard-coded works as follows: On input x, find y such that $|y| \geq m$ and $f_j^y > 2|x|$. Now let $h(x)$ be a code for $\{y\}$. Note $C(y) \geq f_j^y > 2|x|$. We have that for almost all x, $W_x \neq \{y\}$ or $C(y) \leq |x| + O(1) < 2|x|$. For these x we have $W_{h(x)} \neq W_x$ so by Arslanov's completeness criterion, f is complete. □

4 Incompleteness

We have seen in the previous section that essentially no enumerator of C can be computable and that many enumerators for C are complete. Since C is as hard as the halting problem, it is natural to ask whether every 2-enumerator for C is as hard as the halting problem. We will show that there exists a 2-enumerator for C that is not as hard as the halting problem, at least under weak-truth-table reductions. In fact, given any nonrecursive set A, there exists a 2-enumerator for C that is not as hard as A.

Theorem 4 *Let A and e be such that for every 2-enumerator f of C, e is a 1-tt reduction from A to f. Then A is recursive.*

Proof: Let y be the query generated by e on input x. Let $e^f(x)$ denote the outcome of the computation e on input x using oracle f, and let $e^{\{n_1,n_2\}}(x)$ denote the outcome of the program e run on input x, but with the numbers n_1, n_2 substituted for the answer to the query y. We determine whether $x \in A$ as follows: Compute $e^{\{n_1,n_2\}}(x)$ for all possible $\{n_1, n_2\}$ where $n_1 \leq n_2 \leq |y|$. Clearly if there are an n_1 and an $i \in \{0, 1\}$ such that $(\forall n \leq |y|)[e^{\{n_1,n\}}(x) = i]$, then also $e^f(x) = i$ since $C(y)$ is one of these values. It remains to argue that such an n_1 exists. However $n_1 = C(y)$ meets this condition. □

This theorem easily generalizes in two directions: to more general reductions and to more general functions.

Corollary 5 *Let g be any recursively bounded function. Suppose that there is a fixed reduction e that tt-reduces a set A to all possible 2-enumerators for g. Then A is recursive.*

Proof sketch: The proof is almost the same as above. Instead of simulating e with all possible pairs of oracle answers, we simulate e with all possible pairs of tuples of oracle answers. □

Alternate proof: Alternatively, let $g'(x) = \langle g(q_1), \ldots, g(q_m)\rangle$, where $q_1, \ldots, q_m$ are the queries made by the reduction e on input x. The reduction e must compute $A(x)$ correctly if, for each q_i, it is given given a pair of answers (a_i, b_i) such that $g(q_i) \in \{a_i, b_i\}$. *A fortiori*, e must compute $A(x)$ correctly if it is given a pair of sequences $(\langle a_1, \ldots, a_m\rangle, \langle b_1, \ldots, b_m\rangle)$ such that $g'(x) \in \{\langle a_1, \ldots, a_m\rangle, \langle b_1, \ldots, b_m\rangle\}$. Thus there is a fixed reduction e' from A to all possible 2-enumerators for g'. Since g' is recursively bounded, A is recursive by Theorem 4. □

The condition that g is recursively bounded is necessary. If g grows faster than any recursive function then so does f_2^x for any 2-enumerator f of g, so a nonrecursive function is 1-tt reducible to any 2-enumerator for g. Furthermore, if we let $g(x) = \min s|\{x\}_s(x)\downarrow$ if $x \in K$, 0 otherwise, then K is reducible to any 2-enumerator for g.

With a bit more care, the proof of Theorem 4 even extends to T-reductions.

Corollary 6 *Let g be any recursively bounded function. Suppose that there is a fixed reduction e that T-reduces a set A to all possible 2-enumerators for g. Then A is recursive.*

Proof: Consider the first query q_1 generated by e on input x. If we continue the computation with both possible answers $n_1, n_2 \leq g(x)$, then one of the coordinates n_1 (where $n_1 = g(q_1)$) will lead to queries that are also asked as the second query by $e^f(x)$, where f is some 2-enumerator for g. Then, the extensions of these $g(x)$ paths have the same property (for queries q_3 etc.). Also, all paths along which for each query q the first element of the simulated answer is $g(q)$ *must* end with the true value of $A(x)$, since there is a 2-enumerator for g that returns exactly these values, and e reduces A to such an enumerator. So to find $A(x)$ we simulate g until all the right paths have been produced. □

The following contrasts sharply with Theorem 3 for r.e. 2-enumerators, as well as Kummer's theorem [Kum96] that K is wtt-reducible to C.

Corollary 7 *For any nonrecursive set A and any recursively bounded function g there exists a 2-enumerator f of g such that $A \not\leq_{\mathrm{wtt}} f$.*

Proof: Use a finite extension argument on Corollary 6. Since A is nonrecursive we can find for any candidate wtt-reduction e_i a 2-enumerator f_i such that e_i fails. Since e_i is a wtt-reduction, moreover e_i fails on a recursively bounded initial segment. These f_i can be combined to form a single 2-enumerator on which all e_i fail. $\square$

Corollary 7 does not extend to Turing reductions as the following counter example shows.

Theorem 8 *There exists a recursively-bounded function g such that every 2-enumerator for g is Turing-hard for the r.e. sets.*

Proof: Let $s_x = \max\{s | K_s^{\leq x} - K_{s-1}^{\leq x} \neq \emptyset\}$ and $g(x) = \min\{y \mid y \in K_{s_x} - K_{s_x - 1}\}$. That is, $g(x)$ is the last string to be enumerated in K among all strings $\leq x$. Then $g(x)$ is defined for almost all x, and $g \equiv_T K$. Suppose that f is a 2-enumerator for g. Either

1. For almost all x only one output of $f(x)$ halts, then we can just use the value of this output (i.e., this is $g(x)$).

2. For infinitely many x both outputs halt. Then we can just for some y find an x larger than y for which both outputs of $f(x)$ halt, and find $g(x)$ by running both outputs.

In both cases we can with just f find for any x an upper bound for $\min\{t | \{x\}(x)_t \downarrow\}$ if such a t exists or find that it does not exist, i.e. $f \equiv_T K$. $\square$

5 Resource Bounded Reductions

Our negative results have some nice counterparts for polynomial-time reductions. An interesting subset of Σ_2^p known as symP was defined by [RS98].

Definition 9 *A language L is in* symP *if and only if there exists a polynomial time computable 3-ary predicate Q such that $s \in L$ if and only if $(\exists^p x)(\forall^p y)Q(s,x,y)$ if and only if $(\forall^p y)(\exists^p x)Q(s,x,y)$. (The superscript p designates a polynomial-bounded quantifier.)*

Note that for general languages in $\Sigma_2^p \cap \Pi_2^p$ the second occurrence of Q could be replaced by an arbitrary polynomial time predicate Q'. It is not known whether symP $= \Sigma_2^P \cap \Pi_2^p$. Now the following holds.

Theorem 10 *Let A and e be such that for every 2-enumerator f of C, e is a polynomial-time 1-tt reduction from A to f. Then A is in* symP*.*

Proof: Proceed as in the proof of Theorem 4. Let $Q(x, n_1, n_2)$ be the predicate $e^{\{n_1,n_2\}}(x) = 1$. Then $x \in A \Leftrightarrow (\exists n_1)(\forall n_2)Q(x,n_1,n_2)$ and $x \notin A \Leftrightarrow (\exists n_1)(\forall n_2)[\neg Q(x,n_1,n_2)]$. Since Q is symmetric, we're done. $\square$

Corollaries 5–7 generalize similarly, except that in Corollary 7 we get T-reductions.

Corollary 11 *For any set A not in symP there exists a 2-enumerator f of C such that $A \not\leq_{\mathrm{T}} f$.*

This result is in some sense optimal, since there exists a function f such that $|f(x)|$ is polynomially bounded and such that every 2-enumerator for f is 1-tt hard for symP. Let L be a language in symP, and let f be defined as follows.

$$f(s) = \begin{cases} (x, 1) & \text{where } x = \min\{z \mid (\forall^p y) Q(z, y) & \text{if } s \in L \\ (y, 0) & \text{where } y = \min\{z \mid (\exists^p x) Q(x, z) & \text{if } s \notin L \end{cases}$$

The following algorithm 1-tt reduces χ_L to every g that 2-enumerates f.
input s
let $\{(u, a), (v, b)\} = g(s)$, with $a \leq b$
if $a = b$ then output a
else output $Q(s, u, v)$

By a VC-dimension argument we can extend the result above to functions f whose *value* is polynomially bounded. The following are equivalent:

(1) $L \in$ symP

(2) There exists a function f from strings to $\{1, 2, 3\}$ such that L is polynomial-time Turing reducible to all 2-enumerators for f.

(3) There exist $\epsilon > 0$, functions $t(n)$ and $k(n)$ on N such that $t(n) \geq 3$ and $t(n)^{k(n)} \geq 2^{n^\epsilon}$, and a function f from strings to $\{1, \dots, t(n)\}$ such that L is $k(n)$-tt reducible to all 2-enumerators for f.

6 Enumerations for Space Bounded Kolmogorov Complexity

Buhrman and Torenvliet [BT00] show that the interactive proof (IP) protocol [LFKN90, Sha92] can be simulated by an NP-oracle machine that has access to a set of space bounded Kolmogorov random strings. The NP machine in the proof guesses a sequence of polynomials and a sequence of numbers of high space bounded complexity, that are used to generate the proof in the IP protocol. The fact that these numbers have high complexity is enough to guarantee correctness, since intersections of known polynomials have low complexity. For the proof of that theorem it suffices to generate numbers that have complexity that exceeds some constant. Below, we show how a $\log n$-enumerator can be used by an NP machine to generate numbers that have complexity that exceeds a given constant. As a consequence enumerating $O(\log n)$ values of the space bounded Kolmogorov function remains hard for PSPACE.

Theorem 12 *Let k and m be such that $k(n)m(n) \in O(\log n)$. Let f be a function that $k(n)$-enumerates $CS^n(x|w)$ for all x and w. Then here is an* NP *oracle machine M such that M^f on input $\langle n, w\rangle$ can guess and verify a string y such that $CS^n(x|w) \geq m(n)$.*

Corollary 13 *Let f be a function that $k(n)$-enumerates $CS^n(x|w)$ for $k(n) \in O(\log n)$. Then* PSPACE $\subseteq$ NP^f.

Proof: Set m of Theorem 12 to some high enough constant and use [BT00]. □

Proof of Theorem 12: Fix n and w, consider strings of length n. Again, let f on input $\langle x, w\rangle$ output $f_1^x, \dots, n_{k(n)}^x$ without loss of generality ordered such that $f_i^x < f_j^x$ whenever $i < j$. The proof consists of $k(n) + 1$ cases, depending on the values of the enumerator on strings of length n. We present case 0, case 1, case c for $1 < c < k(n)$ and case $k(n)$.

case 0: If there is a string x of length n such that $f_1^x > m(n)$ then guess this string, verify $f_1^x > m(n)$ by querying $f(x)$ and output x.

case 1: We are not in case 0, so none of the strings x of length n has $f_1^x > m(n)$. Let $S = \{x \mid f_2^x > 2m(n)\}$ and suppose $\|S\| > 2^{m(n)}$. Let z be the concatenation of the first $2^{m(n)} + 1$ strings in S. Note that one of these strings x must have $C(x|w) \geq f_2^x > 2m(n)$. We claim that $C(z|w) \geq m(n)$. Suppose not then let P be a program of at most $m(n) - 1$ bits that prints z, and let s be a number that reveals the position of the first substring x in z with $C(x|w) > 2m(n)$. Note that $|s| \leq m(n) + 1$. Then Ps is a string of at most $2m(n)$ bits that describes a string of complexity greater than $2m(n)$.

case c We are not in case $c-1$, so there are at least $2^n - 2^{(c-1)m(n)} - 1$ strings x with $f_c \leq cm(n)$. Let $S = \{x \mid f_c^x > (c+1)m(n)\}$. Suppose that $\|S\| > 2^{cm(n)}$. Let z be the concatenation of the first $2^{cm(n)} + 1$ strings in S. We claim that $C(z|w) \geq m(n)$. Suppose not, then let P be a program of at most $m(n) - 1$ bits that prints z and let s be a number that reveals the position of the first string x in z with $C(x|w) > (c+1)m(n)$. Note that $|s| \leq cm(n) + 1$. Then Ps is a string of at most $(c+1)m(n)$ bits that describes a string of complexity greater than $(c+1)m(n)$.

case k(n) We are not in case $k(n) - 1$, so there are at least $2^n - 2^{(k(n)-1)m(n)} - 1$ strings x with $f_{k(n)}^x \leq k(n)m(n)$. However there is a constant b such that there are are $2^n/b$ random strings, so this case cannot occur.

The NP machine can now on input x and w given f as an oracle guess a number $c < k(|x|)$ (the case). If $c = 0$ it guesses a y of length n such that $f_1^y > m(n)$. Otherwise it guesses $S' \subseteq S = \{x \mid f_c^x > (c+1)m(n)\}$ such that $\|S'\| = 2^{cm(n)} + 1$ and lets z be the concatenation of all strings in S'. By the first part of the proof $C(z|w) > m(n)$. □

7 Acknowledgments

The idea for the investigation of enumerations of the Kolmogorov function was given to us by Bill Gasarch. We would like to thank him for that. We would like to thank Bas Terwijn for interesting discussions on the subject in earlier stages of the research, and Paul Vitányi for providing reference [ZL70].

References

[ABG90] A. Amir, R. Beigel, and W. I. Gasarch. Some connections between bounded query classes and nonuniform complexity. In *Proceedings of the 5th Annual Conference on Structure in Complexity Theory*, pages 232–243, 1990.

[ABG+00] A. Ambainis, H. Buhrman, W. Gasarch, B. Kalyanasundaram, and L. Torenvliet. The communication complexity of enumeration, elimination and selection. In *Proc. 15th IEEE Conf. on Computational Complexity*, pages 44–53, 2000.

[Ars81] M. M. Arslanov. On some generalizations of the theorem on fixed points. *Soviet Mathematics (Iz. VUZ)*, 25(5):1–1, 1981.

[Bar68] Ja. M. Barzdin. Complexity of programs to detemine whether natural numbers not greater than n belong to a recursively enumerable set. *Soviet Math. Dokl.*, 9:1251–1254, 1968.

[BGGO93] R. Beigel, W. Gasarch, J. Gill, and J. Owings Jr. Terse, superterse and verbose sets. *Information and Computation*, 103:68–85, 1993.

[BT00] H. Buhrman and L. Torenvliet. Randomness is hard. *sicomp*, 2000. to appear.

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

[Har83] J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pages 439–445, 1983.

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

[Kum96] M. Kummer. On the complexity of random strings. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 1996.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proc. 31st Symposium on Foundations of Computer Science*, pages 2–90, New York, 1990. IEEE.

[LV97] Ming Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer-Verlag, second edition, 1997.

[RS98] A. Russel and R. Sundaram. Symmetric alternation capturers bpp. *Computational Complexity*, 7(152-162), 1998.

[Sha92] A. Shamir. IP=PSPACE. *Journal of the ACM*, 4:869–877, October 1992.

[Sol64] R. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22, 224–254, 1964.

[ZL70] A.K. Zvonkin and L.A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russ. Math. Surv.*, 25:83–124, 1970.