Electronic Colloquium on Computational Complexity, Report No. 17 (2015)

Linear list-approximation for short programs (or the power of a few random bits)

Bruno Bauwens[1] and Marius Zimand*[2]

[1]Université de Lorraine
[2]Towson University

January 20, 2015

Abstract

A c-short program for a string x is a description of x of length at most $C(x) + c$, where $C(x)$ is the Kolmogorov complexity of x. We show that there exists a randomized algorithm that constructs a list of n elements that contains a $O(\log n)$-short program for x. We also show a polynomial-time randomized construction that achieves the same list size for $O(\log^2 n)$-short programs. These results beat the lower bounds shown by Bauwens et al. [BMVZ13] for deterministic constructions of such lists. We also prove tight lower bounds for the main parameters of our result. The constructions use only $O(\log n)$ ($O(\log^2 n)$ for the polynomial-time result) random bits. Thus using only few random bits it is possible to do tasks that cannot be done by any deterministic algorithm regardless of its running time.

1 Introduction

The Kolmogorov complexity of a string x, denoted $C(x)$, is the length of a shortest description of x relative to a fixed universal Turing machine U. In many applications, it is desirable to represent information x in a succinct form, i.e., to find a string p such that $U(p) = x$ (such a p is called a *program* for x) with length $|p| \approx C(x)$. Unfortunately this is not possible: Not only is $C(x)$ uncomputable, but it cannot be even approximated in a useful way. Indeed, while the

*A preliminary version has been presented at the 29-th IEEE Conference on Computational Complexity, June 11-13, 2014, Vancouver, British Columbia, Canada.

Bruno Bauwens is grateful to Mathieu Hoyrup and Université de Lorraine for financial support. Marius Zimand was partially supported by NSF grant CCF 1016158.

Author's addresses: Bruno Bauwens; Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Krijgslaan 281, S9, 9000 Gent, Belgium; http://www.bcomp.be; and Marius Zimand, Department of Computer and Information Sciences, Towson University, Baltimore, MD.; http://triton.towson.edu/~mzimand.

ISSN 1433-8092

upper bound $C(x) < |x| + O(1)$ is immediate, Zvonkin and Levin [ZL70] have shown that no unbounded computable function can lower bound $C(x)$. Beigel et al. [BBF$^+$06] have investigated the *list-approximability* of $C(x)$, i.e., the possibility of constructing a short list of numbers guaranteed to contain $C(x)$. They show that there exists a constant a (which depends on the universal machine U) such that any computable list containing $C(x)$ has size n/a (where n is the length of x). Since it is trivial to obtain a list of size $n+O(1)$ that contains $C(x)$, the result of [BBF$^+$06] implies that no list-approximation is possible with lists significantly shorter than the trivial one.

In view of these strongly negative facts, the recent results of Bauwens, Mahklin, Vereshchagin and Zimand [BMVZ13] and Teutsch [Teu14] are surprising. They show that it is possible to effectively construct a short list guaranteed to contain a close-to-optimal program for x. Even more, in fact the short list can be computed in polynomial time. More precisely, [BMVZ13] showed that one can compute lists of quadratic size guaranteed to contain a program of x whose length is $C(x) + O(1)$ and that one can compute in polynomial-time a list guaranteed to contain a program whose length is additively within $C(x) + O(\log n)$. [Teu14] improved the latter result by reducing the $O(\log n)$ term to $O(1)$ (see also [Zim14] for a simpler proof).

In this paper, we investigate how short a computable list that contains a succinct program for x can be. The size of the list in [BMVZ13] is *quadratic* in n and in fact in the same paper it is shown that this is optimal because any effectively computed list that contains a program that is additively c close to optimal length must have size $\Omega(n^2/(c+1)^2)$ (for any c). The size of the list in the polynomial-time construction from [Teu14] is $n^{7+\epsilon}$ and [Zim14] improves it to $O(n^{6+\epsilon})$. We show here that the size of the list can be *linear*, thus beating the above quadratic lower bound, if we allow *probabilistic computation*, in fact even *polynomial-time probabilistic computation*. Namely, we show that there exists a probabilistic algorithm that on input x of length n produces a list of n elements, that, with high probability, contains a program of x which is additively within $O(\log n)$ from optimal. We also show the existence of a polynomial-time algorithm with the same property but for $O(\log^2 n)$ closeness to optimality. The lower bound mentioned above shows that such a list cannot be deterministically computed regardless of the running time. Furthermore, the first algorithm uses only $O(\log n)$ random bits, and the polynomial-time algorithm uses only $O(\log^2 n)$ random bits. The relevance of these facts will be discussed shortly. These results are shown in Section 3. In Section 5, we prove tight lower bounds which show that our results are essentially optimal. More precisely, we consider the parameters c, T, r in our main result which are defined as follows: (1) c is the closeness to $C(x)$ of the length of the desired succinct program, (2) T is the size of the list guaranteed to contain such a succinct program, (3) r is the number of random bits used in the probabilistic construction of the list. In the main result we obtain $c = O(\log n)$, $T = n$, and $r = O(\log n)$. We show that essentially none of these parameters can be improved while keeping the other two the same.

Discussion: The power of randomized computation. Can we solve us-

ing randomness tasks that cannot be solved without? This is a foundational question, and its exploration has an old history [dLMSS56, ZL70] and a recent history [BP14, Bie13, RS13].

There are fields where randomness plays an essential role as a conceptual tool, i.e., as an element introduced in the model. Pre-eminent examples are Game Theory (the utilization of mixed strategy) and Cryptography (the utilization of secret keys). The answer to the above general question is less clear if we restrict to computational tasks. There is a common perception that randomized computation is not fundamentally more powerful than deterministic computation, in the sense that whenever a randomized process solves a task, there also exists a deterministic solution (albeit, often, a slower one). This perception is caused by the simple observation that a probabilistic algorithm can be simulated deterministically after which one can take a majority vote. A similar argument works for tasks computing an infinite object and the classical theorem of de Leeuw, Moore, Shannon and Shapiro [dLMSS56] states that if a function can be computed by a probabilistic algorithm, then it can be computed deterministically. However, these considerations only apply to tasks admitting a unique solution. For tasks admitting multiple solutions, randomness could potentially be helpful.

The task of computing a string with high Kolmogorov complexity is usually given as an example to illustrate the power of randomized computation (see for example [ZL70, RS13]): The task cannot be solved deterministically, but an algorithm that tosses a coin does it easily by just printing the coin flips. However this example is trivial and not very convincing because the noncomputable output of the above procedure is exactly the noncomputable part introduced in the procedure. More precisely, if f is the probabilistic algorithm with input x and random bits r, then $f(x, r) = r$. Let us consider another task: On input x, find an extension of it called y such that y has larger complexity than x. This second task can be solved in the same trivial way by obtaining via coin tosses a string r and then taking $y = xr$. Note that this time we can have $|r| \ll |y|$. For infinite objects, better examples are known. N.V. Petri (see [RS13]) showed that with positive probability, one can enumerate a graph of a total function f that exceeds all computable functions. The procedure utilizes a polynomial number of random bits to generate $f(x)$. Obviously, the time to generate $f(x)$ from x is not bounded by any computable function. The reader can find other similar examples in [BP14].

So the interesting question is whether there are *non-trivial* computational tasks involving finite objects that can be solved probabilistically (perhaps even in polynomial time) but not deterministically. Furthermore, if the answer is positive, can the amount of random bits necessary to solve such a task be very low? In other words, is it the case that even very few random bits can solve a non-trivial task, which is deterministically unsolvable? Our main results give positive answers to these questions.

Any definition of *trivial task* is inherently debatable. For our discussion it is sufficient to use an informal formulation whose requirements are so minimal that it is unlikely to raise controversy. Intuitively, a task is trivial if a solution for it can be "read" almost directly from the pair consisting of the input and

a random string. For concreteness, we interpret "read" as the composition of a projection and a permutation, or we can take a more general stance and interpret it as the computation of a uniform boolean or arithmetic NC^0 circuit.

Now consider the following task (which depends on a parameter c): Given x of length n, find y, a list of n strings such that at least one of them is a program for x of length bounded by $C(x) + c$.

For $c = o(\sqrt{n})$, by the lower bound from [BMVZ13], we know that it cannot be solved by deterministic algorithms. The results in this paper show that, for $c = O(\log n)$, the task is solvable by a randomized algorithm, which remarkably uses only $O(\log n)$ random bits. Furthermore, for $c = O(\log^2 n)$, the task is solvable by a polynomial-time algorithm that uses only $O(\log^2 n)$ random bits. The task appears to be non-trivial, at least we are not aware of any NC^0 solution.

In Section 6, we elaborate the above example and present an even more natural non-trivial task that (1) can be solved in polynomial time by a randomized algorithm, and (2) cannot be solved by any deterministic algorithm that runs in computably bounded time. This task asks that on every input (x, ℓ), if $\ell = C(x)$, a string z should be constructed such that z is a short program for x. This is a promise problem, because in case $\ell \neq C(x)$, the algorithm is not even required to halt, or, in case it halts, z can be an arbitrary string. Note that on input (x, ℓ), if $\ell = C(x)$, one can simply by exhaustive search find a shortest program for x. However, the exhaustive search does not halt if $C(x) > \ell$. Actually, we show that there is no deterministic algorithm that, in case $\ell = C(x)$, runs in computably bounded time and constructs a $o(n)$-short program for x. On the other hand, relying on the techniques used in the proof of our main results, we present a randomized algorithm that runs in polynomial time and constructs with probability $(1 - \delta)$ a $O(\log^2(n/\delta))$-short program for x, conditioned that the promise $\ell = C(x)$ holds. Furthermore, the randomized algorithm uses only $O(\log^2 n/\delta)$ random bits. In fact, using relativized versions of this task, we notice that polynomial-time randomized computation can be more powerful than deterministic computation that lies arbitrarily high in the arithmetic hierarchy.

It remains to investigate if there exist non-trivial tasks that cannot be solved deterministically but that can be solved using even fewer random bits (e.g., $o(\log n)$ or even $O(1)$ random bits). In the Appendix, we give an example that can be solved with $O(1)$ random bits, but it is still borderline trivial, because the solution is obtained simply by dividing the input length by a constant.

2 Preliminaries

We fix a universal Turing machine U that is *standard* (meaning that for every machine V there is a polynomial-time computable function t such that, for all p, $U(t(p)) = V(p)$ and $|t(p)| = |p| + O(1)$.)

C stands for the plain Kolmogorov complexity relative to U. Thus, for any string x, $C(x) = \min\{|p| \mid U(p) = x\}$. If $U(p) = x$, we say that p is a program for x. If in addition, $|p| \leq C(x) + c$, then we say that p is a c-short program for

x.

We use bipartite graphs $G = (L, R, E \subseteq L \times R)$ with $L = \{0,1\}^n$ (or, a few times, $L \subset \{0,1\}^n$), $R = \{0,1\}^m$ and which are left-regular, i.e., all the nodes in L have the same degree, which we denote 2^d. We denote $N = 2^n, M = 2^m, D = 2^d$.

As it is typically the case, we actually work with a family of graphs indexed on n and such a family of graphs is *constructible* if there is an algorithm that on input (x, y), where $x \in \{0,1\}^n = L$ and $y \in \{0,1\}^d$, outputs the y-th neighbor of x. Some of the graphs also depend on a rational $0 < \delta < 1$. A constructible family of graphs is *explicit* if the above algorithm runs in time poly$(n, 1/\delta)$.

A (k, ϵ) extractor is a function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that for any distribution X on $\{0,1\}^n$ with min-entropy $H_\infty(X) \geq k$, $E(X, U_d)$ is ϵ-close to U_m, where U_d (U_m) is the uniform distribution on $\{0,1\}^d$ (respectively, $\{0,1\}^m$), i.e., for every $A \subseteq R$,

$$\left| \mathrm{Prob}[\mathrm{E}(\mathrm{X}, \mathrm{U_d}) \in \mathrm{A}] - \frac{|\mathrm{A}|}{\mathrm{M}} \right| < \epsilon. \tag{1}$$

It is known that it is enough to require that the condition holds for all distributions X that are *flat* [CG88]. The value $k + d - m$ is called the entropy loss of the extractor. We remind the reader that $H_\infty(X) \geq k$ means that for any $x \in \{0,1\}^n$, $\mathrm{Prob_X(x)} \leq 2^{-\mathrm{k}}$ and that a distribution is flat if it assigns equal probability mass to each element in its support.

To an extractor E, we associate the bipartite graph G_E with $L = \{0,1\}^n, R = \{0,1\}^m$, such that for every $x \in L$, its neighbors are $N(x) = \{E(x, y) \mid y \in \{0,1\}^d\}$.

In this paper we need extractors for which $k + d - m = O(\log(n/\epsilon))$, i.e., the entropy loss is at most logarithmic. Using standard probabilistic methods the following extractor can be shown to exist, which has even smaller entropy loss (see [RTS00]).

Theorem 2.1. *For all n, $k \leq n$, and $\epsilon > 0$, there exists a (k, ϵ) extractor with $m = k + d - 2\log(1/\epsilon) - O(1)$ and $d = \log(n - k) + 2\log(1/\epsilon) + O(1)$.*

The above result is existential, but using the fact that the number of flat distributions with min-entropy k is finite, one can check effectively whether a function is an extractor, and therefore one can effectively construct an extractor with the parameters in Theorem 2.1. Such an exhaustive search can be done in space $2^{O(n)}$.

Moving to explicit (i.e., polynomial-time computable) extractors, the currently best result for extractors with $O(\log 1/\epsilon)$ entropy loss is due to Guruswami, Umans, and Vadhan [GUV09]:

Theorem 2.2. *For all n, $k \leq n$, and $\epsilon > 0$, there exists an explicit (k, ϵ) extractor with $m = k + d - 2\log(1/\epsilon) - O(1)$ and $d = \log(n) + O(\log k \cdot \log(k/\epsilon))$.*

Note. In case $k = \Omega(n)$ (and constant ϵ), the extractor in Theorem 2.2 has $d = O(\log^2 n)$. This is the source of the $O(\log^2 n)$ terms in our main result Theorem 3.2. It is a major open problem to obtain explicit extractors with $O(\log 1/\epsilon)$ entropy loss that have $d = O(\log n)$. Such extractors would reduce the overhead in our result to $O(\log n)$.

3 The upper bounds

The following two theorems are the main results.

Theorem 3.1. *There exists a probabilistic algorithm that on input $x \in \{0,1\}^n$ and rational $0 < \delta < 1$, outputs a list with n elements which with probability at least $(1-\delta)$ contains a $O(\log(n/\delta))$-short program for x.*

Moreover, the algorithm uses $O(\log(n/\delta))$ random bits and can be executed in space $2^{O(n)}$.

Theorem 3.2. *There exists a probabilistic polynomial-time algorithm that on input $x \in \{0,1\}^n$ and rational $0 < \delta < 1$, outputs a list with n elements which with probability at least $(1-\delta)$ contains a $O(\log^2(n/\delta))$-short program for x.*

Moreover, the algorithm uses $O(\log^2(n/\delta))$ random bits.

The proofs of these two results have a common structure. The key part is building a bipartite graph with the "rich owner" property, roughly meaning that no matter how we restrict the left side to a subset of a certain size, then, in the restricted graph, most left nodes "own" most of their neighbors (in the sense that these neighbors are not shared with any other node).

We start by defining precisely the "rich owner" property in a bipartite graph G. Let B be a subset of left nodes. We say that a right node y is *shared* in B if it has at least two neighbors in B. For any $\delta > 0$, a left node x is *δ-rich* in B if $x \in B$, it has at least one right neighbor, and at most a fraction δ of its neighbors are shared in B (so it "owns" at least a fraction $1-\delta$ of its neighbors). Later, we also use a refined version of these concepts. We say that a right node is *s-shared* in B if it has at least s left neighbors in B. A left node x is *(s,δ)-rich* in B if $x \in B$, it has at least one neighbor, and if at most a fraction δ of its neighbors are s-shared in B. If the set B is omitted in these definitions, $B = L$ is assumed.

Definition 3.3. *A bipartite graph $G = (L, R, E)$ has the* rich owner *property for parameters (ℓ, c, δ) if for any left subset B of size at most 2^ℓ, all but at most $2^{\ell-c}$ of its elements are δ-rich in B.*

For us the key parameters are the left degree $D = 2^d$ and $m = \log|R|$, because d corresponds to the number of random bits used in the main results and $m-\ell$ essentially gives the "quality" of the short program (i.e., the distance between its length and $C(x)$). The following theorem, whose proof we defer for Section 4, shows the existence of this type of graphs with parameters that will allow us to establish the main results.[1]

Theorem 3.4. *(1) (Constructible graphs with the rich owner property.) For all $n, \ell, c, \delta > 0$ there exists a family of graphs $G_{n,\ell} = (L = \{0,1\}^n, R = \{0,1\}^m, E)$*

which have the rich owner property for (ℓ, c, δ), with

$$\begin{aligned} m &= \ell + O(c + \log(n/\delta)) \text{ and} \\ d &= O(c + \log(n/\delta)) \end{aligned}$$

[1] Such graphs can be obtained from the extractor-condenser pairs from [RR99], Theorem 5.1. We give here a similar but slightly more efficient construction, with a proof tailored for our purposes.

The family is uniformly computable in n, ℓ, c, δ.

(2) (Explicit graphs with the rich owner property.) For all $n, \ell, c, \delta > 0$ there exists a family of graphs $G_{n,\ell} = (L = \{0,1\}^n, R = \{0,1\}^m, E)$ which have the rich owner property for (ℓ, c, δ), with

$$\begin{aligned} m &= \ell + O(c + \log^2(n/\delta)) \text{ and} \\ d &= O(c + \log^2(n/\delta)). \end{aligned}$$

The y-th neighbor $E(x, y)$ of x (in the corresponding graph in the family) is computable in time poly$(n, c, 1/\delta)$.

Equipped with graphs that have the rich owner property, we can prove our main results.

of Theorem 3.1. We start by showing a weaker claim:

Claim 3.5. *There exists a probabilistic algorithm that on input $x, \ell, c, \delta > 0$ always terminates and outputs a program of length $\ell + O(c + \log(n/\delta))$, such that for all ℓ, c and n, for all but at most $2^{\ell-c}$ strings x of length n with $C(x) < \ell$, with probability $1 - \delta$ the algorithm outputs a program that computes x. Moreover, the probabilistic algoritm uses $O(\log(c + n/\delta))$ random bits.*

If the algorithm was only required to terminate if $C(x) < \ell$, the claim would be easy: run all programs of length less than ℓ in parallel, wait until a program for x appears, and output this program; but this procedure never terminates if $C(x) \geq \ell$.

To show the claim, fix some ℓ, c, δ, and n. Let $B_{n,\ell}$ be the set of all n-bit strings x with $C(x) < \ell$. Note that $B_{n,\ell}$ can be enumerated uniformly in n and ℓ and that $|B_{n,\ell}| < 2^\ell$. Let $G_{n,\ell}$ be the graph satisfying the conditions of Theorem 3.4(1). Thus all but at most $2^{\ell-c}$ nodes are δ-rich in $B_{n,\ell}$.

Consider a machine that given an encoding of ℓ, c, δ, n and a right node z of $G_{n,\ell}$ does the following: it enumerates all $x \in B_{n,\ell}$ and when the first such neighbor x of z in $G_{n,\ell}$ appears, it outputs x and halts. All but at most $2^{\ell-c}$ such nodes x are δ-rich in $B_{n,\ell}$, and for such x a fraction $(1 - \delta)$ of neighbors are associated to programs for x as described above. The associated programs can be assumed to have length $\ell + O(c + \log(n/\delta))$. On input x of length n, and ℓ, c, δ, the algorithm of the claim, using $O(\log n/\delta)$ random bits, randomly chooses a right neighbor z of x and outputs its associated program. (Note that there is always at least one neighbor.) It remains to convert a program on this special machine to a program for our standard reference machine U. This is possible using the function t in the definition of standard machines; it increases the length by $O(1)$ and its computation time by a polynomial function. The claim is proven. □

We now proceed to the proof of Theorem 3.1. Let e be a constant such that $C(x) < |x| + e$ for all x. For some x and c, we could apply the algorithm of the claim for $\ell = e+1, e+2, \ldots, |x|+e$ with the same choice of random bits at each iteration (so that the number of random bits remains $O(\log n/\delta)$). In this way we obtain a list of $|x|$ programs such that, with probability $1 - \delta$, one of them

computes x. This has almost the desired effect, the only problem being that the construction may fail on a 2^{-c} fraction of strings x of some length (namely, on the strings that are not rich owners).

To handle this, we modify the definition of $B_{n,\ell}$ above. The idea is that now $B_{n,\ell}$ should not only contain strings of small complexity, but also the few strings that are not δ-rich in $G_{n,\ell+1}$. More precisely, fix some n and apply Theorem 3.4(1) with $c = 2$. For $\ell = e + 2, \ldots, n + e + 1$ let $B_{n,\ell}$ be the union of the set of all n-bit x with $C(x) < \ell - 1$ (first type) and those strings that are not δ-rich in $G_{n,\ell+1}$ (second type). By downward induction we show that $B_{n,\ell}$ can be enumerated from n and ℓ, and that the size of $B_{n,\ell}$ is bounded by 2^ℓ. Indeed, $B_{n,n+e+1}$ only contains strings of the first type and thus it satisfies the conditions. Now assume the conditions hold for some $\ell \leq n+e+1$. Both types of strings can be enumerated. Moreover, the number of strings in $B_{n,\ell-1}$ of the first type is bounded by $2^{\ell-2}$. By the induction hypothesis, $B_{n,\ell}$ has size at most 2^ℓ, thus the number of strings that are not rich in $G_{n,\ell}$ is at most $2^{\ell-c} = 2^{\ell-2}$. Hence, the size of $B_{n,\ell-1}$ is at most $2^{\ell-2} + 2^{\ell-2} = 2^{\ell-1}$. This modification only changes the programs associated to right nodes by some fixed instructions, and this does not affect their length by more than a $O(1)$ constant (and the time to generate them by more than a polynomial factor). □

of Theorem 3.2. It is the same proof as above, except that we use Theorem 3.4 (2). For later reference, we state explicitly the polynomial-time version of claim 3.5.

Claim 3.6. *There exists a probabilistic algorithm that on input $x, \ell, c, \delta > 0$ outputs in polynomial time a program of length $\ell + O(c + \log^2(n/\delta))$, such that for all ℓ, c and n, for all but at most $2^{\ell-c}$ strings x of length n with $C(x) < \ell$, with probability $1 - \delta$ the algorithm outputs a program that computes x. Moreover, the probabilistic algoritm uses $O(c + \log^2(n/\delta))$ random bits.* □

□

4 Construction of graphs with the rich owner property

All that is left is to prove Theorem 3.4, i.e., to show the construction of graphs with the rich owner property. We use the concept of a (s, δ)-*rich* node in B, introduced earlier (also recall our convention that in case the set B of nodes is omitted, it is assumed to be L, the set of left nodes).

Our proof has four steps which we describe roughly. First we show that most of the left nodes in an extractor graph share their right neighbors with a small number of other left nodes; i.e., most left nodes are (s, δ)-rich for appropriate s and δ (Lemma 4.1). In the second step of the proof we split right nodes (and edges) in a graph such that right nodes share less left neighbors. We do this in a way such that any (s, δ)-rich node in some left subset becomes a 2δ-rich node (Lemma 4.3). In the third step, we combine the previous results to show that within a small computational cost, extractors can be converted to

graphs with the rich owner property (Proposition 4.4). Finally, the theorem is proven using extractors from Theorems 2.1 and 2.2.

Lemma 4.1. *Let $0 < \varepsilon < 1$. All but at most 2^k left nodes of a (k, ε) extractor with average right degree at most a are $(a/\varepsilon, 2\varepsilon)$-rich.*

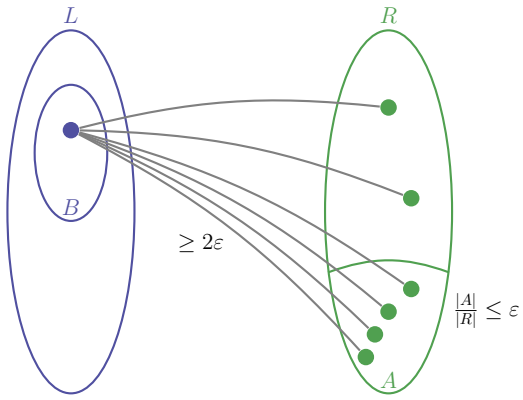

Figure 1: Sets B and A in the proof of Lemma 4.1.

Proof. It suffices to show the lemma for a equal to the average right degree, because for larger a more left nodes are $(a/\varepsilon, 2\varepsilon)$ rich. Let A be the set of right nodes with degree at least a/ε. Note that $|A|/|R|$ is at most ε. Let B be the set of left nodes that are not $(a/\varepsilon, 2\varepsilon)$ rich, i.e., have more than a fraction 2ε of neighbors in A. Consider a flat distribution over all edges leaving from B. Inequality (1) in the definition of a (k, ε) extractor is violated:

$$\left|\mathrm{Prob}[\mathrm{E}(\mathrm{X}, \mathrm{U_d}) \in \mathrm{A}] - \frac{|\mathrm{A}|}{\mathrm{M}}\right| > |2\varepsilon - \varepsilon| = \varepsilon\,.$$

This implies that B has less than 2^k elements, and this implies the lemma. □

In the second step, we split edges and right nodes of a graph. This splitting satisfies the following property for all subsets B of left nodes: if a right node has few neighbors in B, then most of the corresponding splitted nodes have no or a unique neighbor in B. We do this using a technique from [BFL01] (and also employed in [BMVZ13]). We hash the right nodes using congruences modulo a small set of prime numbers. For this technique we need the following lemma.

Lemma 4.2. *Let $x_1, x_2 \ldots, x_s$ be distinct n-bit strings, which we view in some canonical way as integers $< 2^{n+1}$. Let p_i be the i-th prime number and let $L = \{p_1, \ldots, p_t\}$, where $t = (1/\delta) \cdot s \cdot n$.*

For every $i \leq s$, for less than a fraction δ of p in L, the value of $x_i \bmod p$ appears more than once in the sequence $(x_1 \bmod p, x_2 \bmod p, \ldots, x_s \bmod p)$.

Proof. By the Chinese Remainder Theorem and taking into account that the x_i values are bounded by 2^{n+1}, we notice that, for every $x_j \neq x_i$, "$x_i = x_j \bmod p$" holds for at most n prime numbers p. Therefore, "$\exists x_j \neq x_i (x_i = x_j \bmod p)$"

holds for at most $(s-1)n$ prime numbers p. Since L contains $(1/\delta) \cdot s \cdot n$ prime numbers, it follows that

$$\mathrm{Prob_{p\in L}}[x_i \mod p \text{ is not unique}] \leq \frac{(\mathrm{s}-1)\mathrm{n}}{(1/\delta)\cdot \mathrm{s}\cdot \mathrm{n}} < \delta.$$

□

Lemma 4.3. *For any s, $\delta > 0$ and (left-regular) graph $G = (L = \{0,1\}^n, R, E)$, let $t = sn/\delta$. There exists a (left-regular) graph $H = (L, R \times S, E')$ such that*

1. *The left degree of H is exactly t times the left degree of G,*
2. *The set S has size at most $O\left(t^3\right)$,*
3. *If in G the i-th right node of a left node n can be constructed in $time(n)$, the same operation in H takes $time(n) + poly(t)$, and*
4. *If a left node x is (s,δ)-rich in some B for G, then it is 2δ-rich in B for H.*

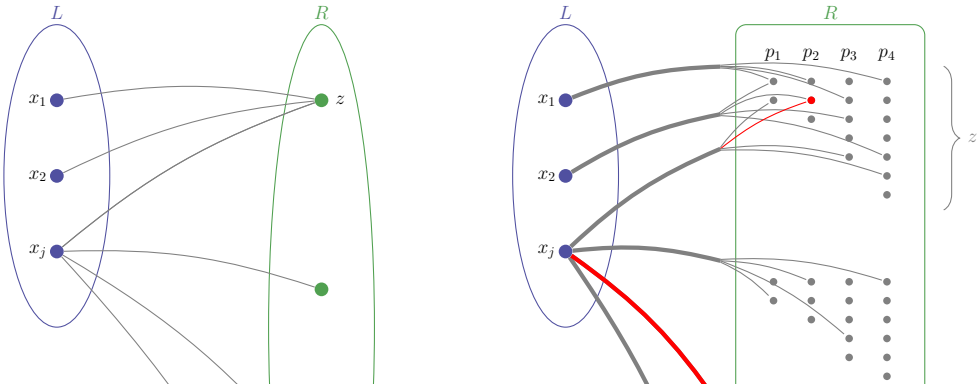

Figure 2: Splitting of right nodes in the proof of Lemma 4.3.

Proof. Let $G = (L, R, E)$ and let $p_1, p_2, \ldots, p_t$ be the first t prime numbers. The right set of H is given by

$$R \times \{p_1, \ldots, p_t\} \times \{0, 1, \ldots, p_t - 1\}.$$

The edges of H are obtained by adding for each edge (x, z) in G the edges

$$(x, (z, p_1, x \bmod p_1)), (x, (z, p_2, x \bmod p_2)), \ldots, (x, (z, p_t, x \bmod p_t))$$

in H (one can think that each edge (x, z) in G is split into t edges in H, see Figure 2).

This operation increases the left degree by a factor of t, which implies (1). The size of $S = \{p_1, \ldots, p_t\} \times \{0, 1, \ldots, p_t - 1\}$ is bounded by ${p_t}^2$. Because $p_t \leq t \ln t + t \ln \ln t$ for $t \geq 6$, this size is bounded by $O(t^3)$, which implies

(2). Enumerating the first t prime numbers is possible in time poly(t), which implies (3).

It remains to show why each (s, δ)-rich left node in some B in graph G, is 2δ-rich in B in graph H. The proof for general B follows the proof for $B = L$ which is presented here. Suppose a right node z in G is not s-shared, i.e., it has s' neighbors $x_1, \ldots, x_{s'}$ with $s' \leq s$.

For some $j \leq s'$, how many nodes $(z, p_i, x_j \bmod p_i)$ with $i = 1, \ldots, t$ are shared? If for some p_i the value $x_j \bmod p_i$ appears more than once in $x_1 \bmod p_i, \ldots, x_{s'} \bmod p_i$, then $(z, p_i, x_j \bmod p_i)$ is shared. By Lemma 4.2, at most a fraction δ of nodes $(z, p_i, x_j \bmod p_i)$ in H are shared (red element in figure 2).

Let x_j be a left node in G that is (s, δ)-rich.

Then at most a fraction δ of its right nodes in G are s-shared (denoted by thick red edges in figure 2). By the previous paragraph, of the remaining $1 - \delta$ fraction of nodes, at most a fraction δ are shared in H. Thus the total fraction of shared nodes is at most $\delta + \delta(1 - \delta) \leq 2\delta$; i.e., x_j is 2δ-rich for H. □

Now we combine previous results to show that we can convert extractors to graphs in which most left nodes in a sufficiently small set B are ε-rich for some ε.

Proposition 4.4. *Let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k, ε) extractor, and let G be the graph associated to E. For any natural number a, there exist a set S and a non-empty (left-regular) graph $H = (L, R \times S, E')$ such that*

1. *the left degree is at most $O(2^d an/\varepsilon^2)$ and $|S| \leq O\left((an/\varepsilon^2)^3\right)$,*
2. *from $n, k, \varepsilon, (x, y)$, one can compute a list of all s such that $(x, (y, s)) \in E'$ in time poly$(n, a, 1/\varepsilon)$,*
3. *if $B \subseteq \{0,1\}^n$ is such that the average number of edges from B arriving in a right node from G is at most a, then all but at most 2^k left elements in B are 4ε-rich in B relative to H.*

Proof. Note that after deleting some left nodes from a (k, ε) extractor graph, what is left is still a (k, ε) extractor graph. More precisely, for any graph $G = (L, R, E)$, a subset $L' \subseteq L$ defines a subgraph $G' = (L', R, E' \subseteq E)$ where E' are all edges in E leaving from L'. If G is an extractor graph, then also G' is an extractor graph: indeed, we must verify (1) for all $B \subseteq L$ of size at least 2^k in G' and $A \subseteq R$, but every element in B has the same edges in G' and hence defines the same probabilities.

Apply Lemma 4.3 with $s = a/\varepsilon$ and $\delta = \varepsilon$; thus $t = sn/\delta = an/\varepsilon^2$ and this guarantees the existence of H such that conditions 1 and 2 are satisfied. Clearly the graph is left regular, and because extractor-graphs are non-empty, this also holds for H. To show item 3, we use Lemma 4.1. As discussed above, the extractor property remains true if the left set is restricted to a subset B, thus we can apply Lemma 4.1. Hence, all but at most 2^k nodes in B are $(a/\varepsilon, 2\varepsilon)$-rich in B, and by the last part of Lemma 4.3, they are 4ε-rich in B relative to H. □

Finally, we are ready to finish the proof.

of Theorem 3.4. We prove (1), the proof for (2) is similar (we use the extractor from Theorem 2.2 instead of Theorem 2.1).

Let $G = (L = \{0,1\}^n, R = \{0,1\}^m, E)$ be an extractor satisfying the conditions of Theorem 2.1 and suppose B is a subset of L of size at most 2^{k+c}. Let us compute an upper bound for the average right degree of the subgraph containing all edges leaving from B:

$$\frac{|B|D}{2^m} \leq 2^c \frac{2^k D}{2^m}$$

i.e., $2^c \times 2^{\text{(the entropy loss)}}$. Since the entropy loss in Theorem 2.1 is $2\log(1/\varepsilon) + O(1)$, the average right degree is $O(2^c/\varepsilon^2)$.

Next we choose δ and ℓ:

$$\delta = 4\varepsilon \qquad \text{and} \qquad \ell = k + c\,.$$

We apply Proposition 4.4 and conclude that all but at most $2^{\ell-c}$ left nodes of H are δ-rich in B. Any left node in a non-empty left-regular graph has at least one neighbor, thus the graph has the rich owner properties for parameters (l, c, δ). It remains to check the claimed values for d and m.

In the extractor, the left degree $D \leq O(n/\varepsilon^2) = \Theta(n/\delta^2)$. The new graph has left degree $O(Dan/\varepsilon^2)$ and

$$D \cdot a \cdot n/\varepsilon^2 = O\left(\frac{n}{\delta^2}\frac{2^c}{\delta^2}\frac{n}{\delta^2}\right).$$

Hence, the logarithm of the left degree is $d = O\left(\log Dan/\varepsilon^2\right) = O\left(c + \log(n/\delta)\right)$.

The logarithm of the size of the set of right vertices is

$$m = (k + \log D - 2\log(1/\varepsilon) + O(1)) + \log|S|\,.$$

Note that $|S| \leq O\left((Dan/\varepsilon^2)^3\right)$, so $\log|S|$ is $O\left(c + \log(n/\delta)\right)$. Similar for $\log D$ and $\log(1/\varepsilon)$, and thus $m = (\ell - c) + O\left(c + \log(n/\delta)\right)$. □

5 Lower bounds

There are three parameters of interest in the main result Theorem 3.1, which on input x constructs with high probability a list containing a short program for x:

T = the size of the list,

r = the number of random bits used in the construction, and

c = the closeness to $C(x)$ of the short program guaranteed to exist in most lists.

In Theorem 3.1, we show $T = n$, $r = O(\log n)$, and $c = O(\log n)$, where n is the length of the string x (for simplicity, we have assumed here constant error probability δ). We show here that Theorem 3.1 is tight, in the sense that, essentially, none of these parameters can be reduced while keeping the other two at the same level.

To prove the lower bounds, we need to specify the model carefully. A *probabilistic algorithm that list-approximates short programs* is given by a Turing machine F that takes an input x, and a sequence ρ (of random bits) and satisfies the following properties:

(a) $|\rho| = r$ depends only on the length of x,

(b) for all x and ρ of length r, $F(x, \rho)$ halts and outputs a finite set of strings $L_{x,\rho}$ (which we typically call a *list*),

(c) the size $|L_{x,\rho}| = T$, depends only on the length of x.

(d) for all x, at least $1/2$ of the sets $\{L_{x,\rho}\}_{|\rho|=r}$ contain a c-short program for x. (Since we seek lower bounds, assuming $\delta = 1/2$, implies at least as strong lower bounds for smaller δ).

After these preparations, we can state the lower bounds.

Theorem 5.1. *For any probabilistic algorithm that list-approximates short programs with parameters T, r and c,*

(1) $r \geq 2\log n - \log T - 2\log c - O(1)$. In particular, if $T = n$, and $c = O(\log n)$, then $r \geq \log n - O(\log\log n)$.

(2) $c \geq \log(n^2/T) - 2\log\log(n^2/T)$. In particular, if $T = O(n)$, then $c \geq \log n - 2\log\log n - O(1)$.

(3) $T = \Omega(n/(c+1))$.

Proof. (1) The union $L = \bigcup_{|\rho|=r} L_{x,\rho}$ is a computable set with $R \cdot T$ elements, where $R = 2^r$, and contains a c-short program for x. It is known from [BMVZ13] that any such set must have size $\Omega(n^2/(c+1)^2)$. Thus for some constant c_1, $R \cdot T \geq c_1 \cdot (n^2/(c+1)^2)$, and thus $r \geq 2\log n - \log T - 2\log c - O(1)$.

(2) Let $\mathcal{P}$ be the set of c-short programs for x. By a result of Chaitin [Cha76] (see also Lemma 3.4.2 in [DH10]), we know that $\ell = |\mathcal{P}|$ satisfies $\ell = O(2^c)$. Since at least half of the $R = 2^r$ lists $L_{x,\rho}$, with ρ ranging in $\{0,1\}^r$, contain an element of $\mathcal{P}$, there is some element of $\mathcal{P}$ that belongs to at least $1/(2\ell)$ fraction of lists. Clearly the length of this element is bounded by $n + d$, where d is a constant that depends on the universal machine. In steps $m = 1, 2, \ldots, n + d$, we select all the strings of length m that appear in at least $1/(2\ell)$ of the lists. As argued above, there exists a c-short program for x among the selected elements. Let us estimate how many strings have been selected. Let s_m be the number of elements selected at the m-th step of the selection procedure. The elements selected at step m occur at least $s_m \cdot \frac{R}{2\ell}$ times in the union $L = \bigcup_{|\rho|=r} L_{x,\rho}$. Since $|L| = R \cdot T$, we obtain

$$R \cdot T \geq s_1 \cdot \frac{R}{2\ell} + s_2 \cdot \frac{R}{2\ell} + \ldots + s_{n+d} \cdot \frac{R}{2\ell}.$$

Thus, $s_1 + s_2 + \ldots + s_{n+d} \leq T \cdot 2\ell$. By the same result from [BMVZ13], the total number of selected elements is at least $c_1 \cdot n^2/(c+1)^2$, for some constant c_1, because some c-short program is selected. Thus,

$$T \cdot 2\ell \geq s_1 + s_2 + \cdots + s_{n+d} \geq c_1 \frac{n^2}{(c+1)^2}.$$

It follows that $2^c \geq c_1' \cdot n^2 \cdot (1/T) \cdot (1/(c+1)^2)$, for some constant c_1'. The conclusion follows after some simple calculations.

(3) Let $L'_{x,\rho}$ be the set of lengths of strings in $L_{x,\rho}$. We say that an integer m has a *pseudo-presence* in $L'_{x,\rho}$ if at least one of the values $m, m+1, \ldots, m+c$ is in $L'_{x,\rho}$. Clearly, at most $(c+1)T$ integers have a pseudo-presence in $L'_{x,\rho}$. We say that m is *significant* if it has a pseudo-presence in at least half of the sets $L'_{x,\rho}$, $|\rho| = r$. Note that the set of significant integers contains $C(x)$, because the union of all lists contains a c-short program for x. By a result of [BBF$^+$06], any computable set containing $C(x)$ must have size at least n/a for some constant a. Thus, there are at least n/a significant integers. Each significant integers has at least $R/2$ pseudo-presences in the union of all the lists. We obtain that $(n/a) \cdot (R/2) \leq R \cdot (c+1)T$, which implies $T \geq (1/(c+1)) \cdot (n/(2a))$. □

Note. The lower bound in (3) holds even for randomized algorithm that may not halt on some probabilistic branches. This can be proved in the same way, taking advantage of the fact that the lower bound in [BBF$^+$06] holds true even for for algorithms that *enumerate* a list of possible values for $C(x)$.

6 Randomized vs. deterministic computation

We present a non-trivial task that shows the superior power of randomized algorithms versus deterministic algorithms. This task

(1) cannot be solved by any deterministic algorithm that runs in computably bounded time, but

(2) can be solved by a randomized algorithm in polynomial time, and, furthermore, the number of random bits is only polylogarithmic.

Moreover, the task is natural in the sense that it is not artificially constructed to satisfy the conditions. The task is a promise problem, meaning that the input is guaranteed to satisfy a certain property.

TASK T: On input $(x, C(x))$, generate a $c(|x|)$-short program for x.

If the input is of the form (x, ℓ) with $\ell \neq C(x)$ (i.e., the promise is not satisfied), then no output needs to be generated or the output can be an arbitrary string. Note that in case the promise is satisfied, then by exhaustive search we can always find a $c(|x|)$-short program even with $c(|x|) = 0$. However, we will see in Lemma 6.2, that, for some $c(n) = O(n)$, no deterministic algorithm can succeed in computably bounded time.

First we show that for some $c(n) = O(\log^2 n)$ task T can be solved in randomized polynomial time using only a polylogarithmic number of random bits.

Theorem 6.1. *There exists a polynomial-time randomized algorithm that on input $(x, C(x), 1/\delta)$ returns a string z that, with probability $(1-\delta)$, is a $O(\log^2(|x|/\delta))$-short program for x. Furthermore, the algorithm uses $O(\log^2(|x|/\delta))$ random bits.*

Proof. The result is a corollary of Claim 3.6. Let n be the length of x and we apply the claim with $\ell = C(x) + 1$ and $c = 3 \log n$. The algorithm in the claim provides

a $O(\log^2(n/\delta))$-short program for x unless x belongs to some "bad" set of at most $2^{\ell-c}$ strings. We show that for large n and ℓ, strings in this set satisfy $\ell > C(x) + 1$, hence, x can not belong to this set.

Indeed, apply the algorithm of the claim on all strings of length n and all random seeds of sufficient length. Subsequently, search for $2^n - 2^{\ell-c}$ strings x for which at least a fraction $1 - \delta$ of the generated programs print x. The bad strings are contained in the remaining $2^{\ell-c}$ strings. Hence, such strings x satisfy

$$C(x) \leq \ell - c + 2 \log n + 2 \log c + 2 \log(1/\delta) < \ell - 1.$$

□

Next, we show that task T cannot be solved by an algorithm in computably bounded time.

Lemma 6.2. *For each standard machine U there exists an e such that there is no computable function $t(n)$ and no algorithm that on input $(x, C_U(x))$ outputs a $|x|/e$-short program for x in time $t(|x|)$.*

Proof. Suppose the lemma was false, and for some value $2e$ there exists a computable function t that bounds the computation time of such an algorithm computing an $|x|/(2e)$-short program. Then, there is a total computable function f such that for all x, the value $f(x, C(x))$ is a $|x|/(2e)$-short program. We can assume $|f(x, k)| \leq k + |x|/(2e)$, because if the length were larger, we could replace it by any shorter string without affecting the assumption on f.

Fix some large length n and consider for all x the list

$$[f(x, n/(2e) - (e/2 - 1)),\ f(x, n/(2e) - (e/2 - 2)),\ \ldots,\ f(x, n/(2e) + (e/2 - 1))]\,.$$

This list contains at most $e - 1$ strings and the sum of the lengths is at most $n(e-1)/e = n - n/e$. Therefore, there are at most $2^{n-n/e}$ distinct lists. In the sequence of 2^n lists that correspond to strings x of length n, we take L to be the list that appears most often (if there is a tie, we break it in some canonical way). Let S be the set of strings x that generate L. It follows from above that S has size at least $2^{n/e}$.

Note that S can be computed from n and e. We show that for large e the set S contains e strings, $x_1, x_2, \ldots, x_e$ that have complexity between $n/(2e) - (e/2 - 1)$ and $n/(2e) + (e/2 - 1)$. This leads to a contradiction because, by the assumption on f, these e distinct strings must each have a program in L, but L has only at most $e - 1$ members.

Let w be a string of length $n/(2e)$ with $C(w \mid n, e) \geq n/(2e)$ (such a string exists by a counting argument). We interpret w as a natural number (bounded by $2^{n/(2e)}$), and for $i = 1, \ldots, e$, let x_i be the lexicographically $(w+i)$-th element of S.

We now prove that for all i, $C(x_i) = n/(2e) \pm O(\log e)$. Because $i \leq e$, we have that $C(w \mid n, e) = C(x_i \mid n, e) + O(\log e)$. The left hand side equals

$n/(2e) \pm O(1)$, hence $C(x_i \mid n/(2e), e) = n/(2e) \pm O(\log e)$. We can eliminate the $n/(2e)$ in the conditioning by appending to the program that prints x_i a self-delimiting string of length $O(\log e)$ that represents the difference between the length of the program and $n/(2e)$ (so that the quantity $n/(2e)$ present in the condition can now be read from the modified program). Thus, $C(x_i \mid e) = n/(2e) \pm O(\log e)$, and therefore, as claimed, $C(x_i) = n/(2e) \pm O(\log e)$.

Choose e large enough such that the value of the $O(\log e)$ term in the above equation is at most $e/2 - 1$. (Note that the constant hidden in $O(\log e)$ is machine dependent, and, consequently, so is e.) This implies, that indeed $|C(x_i) - n/(2e)| \leq (e/2 - 1)$, which, as we have seen, implies a contradiction. □

Note. Our methods show that randomized algorithms running in polynomial time can be more powerful than deterministic machines in any level of the arithmetic hierarchy at solving non-trivial tasks. For any $m \geq 0$, let $\emptyset^{(m)}$ denote as usual the m-th jump of the Halting Problem and let $C^{(m)}(x)$ denote the length of the shortest program that prints the string x relative to a fixed standard oracle universal machine U that uses $\emptyset^{(m)}$ as an oracle. Thus $C^{(0)}(x) = C(x)$, the usual Kolmogorov complexity. We define the notion of *c-short $\emptyset^{(m)}$-program* for a string x similarly to c-short programs except that we allow U to use the oracle $\emptyset^{(m)}$. Let us consider the task $\mathrm{T}^{(m)}$ (defined analogously to the task T):

TASK $\mathrm{T}^{(m)}$: On input $(x, C^{(m)}(x))$, generate a $c(|x|)$-short $\emptyset^{(m)}$-program for x.

Theorem 6.1 and Lemma 6.2 can be relativized in the straightforward way to show that for any $m \geq 0$,

(a) Task $\mathrm{T}^{(m)}$ with parameter $c(n) = O(\log^2 n)$ can be solved by a randomized algorithm (without any oracle) that runs in polynomial time and uses $O(\log^2 n)$ random bits (where $n = |x|$).

(b) Task $\mathrm{T}^{(m)}$, for some parameter $c(n) = O(n)$, cannot be solved by any deterministic algorithm that uses $\emptyset^{(m)}$ as an oracle and runs in time bounded by any $\emptyset^{(m)}$-computable function.

References

[BBF+06] R. Beigel, H.M. Buhrman, P. Fejer, L. Fortnow, P. Grabowski, L. Longpre, A. Muchnik, F. Stephan, and L. Torenvliet. Enumerations of the Kolmogorov function. *Journal of Symbolic Logic*, 7(501):501 – 528, 2006.

[BFL01] Harry Buhrman, Lance Fortnow, and Sophie Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM J. Comput.*, 31(3):887–905, 2001.

[Bie13] Laurent Bienvenu. Probabilistic algorithms in computability theory, 2013. Presentation at CCR 2013, Moscow.

[BMVZ13] B. Bauwens, A. Makhlin, N. Vereshchagin, and M. Zimand. Short lists with short programs in short time. In *Proceedings of 28th IEEE Conference on Computational Complexity, Stanford, California, USA*, 2013.

[BP14] L. Bienvenu and L. Patey. Diagonally non-computable functions and fireworks. *CoRR*, abs/1411.6846, 2014.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17:230–261, 1988.

[Cha76] Gregory J. Chaitin. Information-theoretic characterizations of recursive infinite strings. *Theor. Comput. Sci.*, 2(1):45–48, 1976.

[DH10] R. Downey and D. Hirschfeldt. *Algorithmic randomness and complexity*. Springer Verlag, 2010.

[dLMSS56] K. de Leeuw, E. F. Moore, C. F. Shannon, and N. Shapiro. Computability by probabilistic machines. In *Automata Studies, Annals of Mathematics Studies 34*. Princeton University Press, 1956.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4), 2009.

[RR99] Ran Raz and Omer Reingold. On recycling the randomness of states in space bounded computation. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *STOC*, pages 159–168. ACM, 1999.

[RS13] Andrei Yu. Rumyantsev and Alexander Shen. Probabilistic constructions of computable objects and a computable version of Lovász local lemma. *CoRR*, abs/1305.1535, 2013.

[RTS00] J. Radhakrishnan and A. Ta-Shma. Tight bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, February 2000.

[Teu14] Jason Teutsch. Short lists for shortest descriptions in short time. *Computational Complexity*, 23(4):565–583, 2014.

[Zim14] Marius Zimand. Short lists with short programs in short time—a short proof. In Arnold Beckmann, Erzsébet Csuhaj-Varjú, and Klaus Meer, editors, *Proceedings 10-th CiE, Budapest, Hungary, Language, Life, Limits*, volume 8493 of *Lecture Notes in Computer Science*, pages 403–408. Springer International Publishing, 2014.

[ZL70] A. Zvonkin and L. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83–124, 1970.

A Another non-computable task with trivial randomized solution

The task is to select for any x a value $0 < k \leq |x|$ such that $|C(x) - k| \geq 2\log|x|$. It is probabilistically solvable with probability error at most ϵ by using only a constant amount of random bits: Choose at random an integer r in the range $0 < r \leq 1/\epsilon$, and return $\varepsilon r|x|$. By the following lemma this task is not deterministically solvable.

Lemma A.1. *There exist no computable function f such that $0 < f(x) \leq |x|$ and such that $|f(x) - C(x)| \geq 2\log|x|$ for all x.*

Proof. Suppose such a computable function exists. Choose some large n and consider the set of values $f(x)$ for all x of length n. There is a value i that appears at least $2^n/n$ times and let S be the set of x of length n such that $f(x)$ equals i. Thus, $|S| \geq 2^n/n$. To obtain a contradiction, we consider two cases. Assume $i \geq n - \log n$. Because $|S| \geq 2^n/n$, the set contains a string x with $C(x) \geq n - \log n$. Because $C(x) \leq n + O(1)$, this implies $|i - C(x)| \leq \log n + O(1)$, and because $f(x) = i$ this violates the assumption on f for large n. Now suppose $i < n - \log n$. Thus $2^i \leq 2^n/n \leq |S|$. Consider the set of the lexicographic first 2^i strings in S This set can be computed from n and i (and i can be computed from n using f). Hence, there is an x such that $C(x|n) = i + O(1)$. This implies $|C(x) - i| \leq \log n + O(\log\log n)$ and by construction $f(x) = i$ and this violates the assumption on f, a contradiction. □

ECCC ISSN 1433-8092

http://eccc.hpi-web.de