List Approximation for Increasing Kolmogorov Complexity

Marius Zimand

Dept. of Computer and Information Sciences, Towson University, Towson, MD, USA
mzimand@towson.edu

Abstract

It is impossible to effectively modify a string in order to increase its Kolmogorov complexity. But is it possible to construct a few strings, not longer than the input string, so that most of them have larger complexity? We show that the answer is yes. We present an algorithm that on input a string x of length n returns a list with $O(n^2)$ many strings, all of length n, such that 99% of them are more complex than x, provided the complexity of x is less than n. We obtain similar results for other parameters, including a polynomial-time construction.

1998 ACM Subject Classification F.1.1 Models of computation, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Kolmogorov complexity, list approximation, randomness extractor

Digital Object Identifier 10.4230/LIPIcs.STACS.2017.58

1 Introduction

The Kolmogorov complexity of a binary string x, denoted $C(x)$, is the minimal description length of x, i.e., it is the length of a shortest program (in a fixed universal programming system) that prints x. We analyze the possibility of modifying a string in an effective way in order to obtain a string with higher complexity, without increasing its length. Strings with high complexity exhibit good randomness properties and are potentially useful because they can be employed in lieu of random bits in probabilistic algorithms. It is common to define the randomness deficiency of x as the difference $|x| - C(x)$ (where $|x|$ is the length of x), and to say that the smaller the randomness deficiency is, the more random is the string. In this sense, we want to modify a string so that it becomes "more" random. As stated, the above task is impossible because clearly any effective modification cannot increase Kolmogorov complexity (at least not by more than a constant): If f is a computable function, $C(f(x)) \leq C(x) + O(1)$, for every x. Consequently we have to settle for a weaker solution, and the one we consider is that of list-approximation. List approximation consists in the construction of a list of objects guaranteed to contain at least one element having the desired property. Actually, we try to obtain a stronger type of list approximation, in which, not just one, but *most* of the elements in the list have the desired property. More precisely, we study the following question:

> *Question.* Is there a computable function which takes as input a string x and outputs a short list of strings, which are not longer than x, such that most of the list's elements have complexity greater than $C(x)$?

Without the restriction that the length is not increased, the problem is easy to solve by appending a random string (see the discussion in Section 2). The restriction not only makes

© Marius Zimand;
licensed under Creative Commons License CC-BY

the problem interesting, but also amenable to applications in which the input string and the modified strings need to be in a given finite set. The solution that we give can be readily adjusted to handle this case.

The problem of increasing Kolmogorov complexity has been studied before by Buhrman, Fortnow, Newman, and Vereshchagin [3]. They show that there exists a polynomial-time computable f that on input x of length n returns a list of strings, all having length n, such that if $C(x) < n$, then there exists y in the list with $C(y) > C(x)$ (this is Theorem 14 in [3]). In the case of complexity conditioned by the string length, they show that it is even possible to compute in polynomial time a list of constant size. That is $f(x)$ is a list with $O(1)$-many strings of length n and if $C(x \mid n) < n$, then it contains a string y with $C(y \mid n) > C(x \mid n)$ (this is Theorem 11 in [3]).

As indicated above we are after a stronger type of list approximation: We want on input x and $\delta > 0$ to construct a short list of strings not longer than x with the property that a fraction of $(1-\delta)$ of its elements have complexity larger than that of x. There are several parameters to consider. The first one is the size of the list. The shorter is the list, the better is the approximation. Next, the increasing-complexity procedure that we seek will not work for all strings x. Recall that $C(x) \leq |x| + O(1)$ and if x is a string of maximal complexity at its length, then there simply is no string of larger complexity at its length. In general, for strings x that have complexity close to $|x|$, it is difficult to increase their complexity. Thus, a second parameter is the bound on the complexity of x for which the increasing-complexity procedure succeeds. The closer this bound is to $|x|$, the better is the procedure. The third parameter is the complexity of the procedure. The procedure is required to be computable, but it is preferable if it is computable in polynomial time.

We show the following three results, each one beating the other two with respect to one of these three parameters. The first result exhibits a computable list-approximation for increasing Kolmogorov complexity that works for any x with complexity $C(x) < |x|$.

▶ **Theorem 1** (Computable list of polynomial size for increasing Kolmogorov complexity). *There exists a computable function f that on input $x \in \{0,1\}^*$ and a rational number $\delta > 0$, returns a list of strings of length $|x|$ with the following properties:*
1. *The size of the list is $O(|x|^2)\mathrm{poly}(1/\delta)$,*
2. *If $C(x) < |x|$, then $(1-\delta)$ fraction of the elements in the list $f(x)$ have Kolmogorov complexity larger than $C(x)$.*

In the next result, we improve the list size, making it linear in $|x|$ (for constant δ). The price is that the procedure works only for strings x with a slightly lower complexity.

▶ **Theorem 2** (Computable list of linear size for increasing Kolmogorov complexity). *There exists a computable function f that on input $x \in \{0,1\}^*$ and a rational number $\delta > 0$, returns a list of strings of length $|x|$ with the following properties:*
1. *The size of the list is $O(|x|)\mathrm{poly}(1/\delta)$,*
2. *If $C(x) < |x| - \log|x| - \log\log|x|$, then $(1-\delta)$ fraction of the elements in the list $f(x)$ have Kolmogorov complexity larger than $C(x)$.*

Further reducing the list size remains an interesting open question. We could not establish a lower bound, and, as far as we currently know, it is possible that even constant list size may be achievable.

In the next result, the complexity-increasing procedure runs in polynomial time in the following sense. The size of the list is only quasi-polynomial, but each string in the list is computed in polynomial time.

▶ **Theorem 3** (Polynomial-time computable list for increasing Kolmogorov complexity). *There exists a function f that on input $x \in \{0,1\}^*$ and a constant rational number $\delta > 0$, returns a list of strings of length $|x|$ with the following properties:*

1. *The size of the list is bounded by $2^{O(\log^3 |x|)}$,*
2. *If $C(x) < |x| - O(\log^3 |x|)$, then $(1-\delta)$ fraction of the elements in the list $f(x)$ have Kolmogorov complexity larger than $C(x)$, and*
3. *The function f is computable in polynomial time in the following sense: there is a polynomial time algorithm that on input x, i computes the i-th element in the list $f(x)$.*

Note that the procedure in Theorem 3 can be readily converted into a polynomial-time probabilistic algorithm, which uses $O(\log^3 |x|)$ random bits to pick at random which element from the list to return.

This paper is inspired by recent list approximation results regarding another problem in Kolmogorov complexity, namely the construction of short programs (or descriptions) for strings. We recall the standard setup for Kolmogorov complexity, which we also use here. We fix an universal Turing machine U. The universality of U means that for any Turing machine M, there exists a computable "translator" function t, such that for all strings p, $M(p) = U(t(p))$ and $|t(p)| \leq |p| + O(1)$. For the polynomial-time constructions we also require that t is polynomial-time computable. If $U(p) = x$, we say that p is a *program* (or *description*) for x. The Kolmogorov complexity of the string x is $C(x) = \min\{|p| \mid p \text{ is a program for } x\}$. If p is a program for x and $|p| \leq C(x) + c$, we say that p is a c-short program for x. Using a Berry paradox argument, it is easy to see that it is impossible to effectively construct a shortest program for x (or, even a, say, $n/2$-short program for x). Remarkably, Bauwens et al. [1] show that effective list approximation for short programs is possible: There is an algorithm that, for some constant c, on input x, returns a list with $O(|x|^2)$ many strings guaranteed to contain a c-short program for x. They also show a lower bound: The quadratic size of the list is minimal up to constant factors. Teutsch [7] presents a polynomial-time algorithm with similar parameters, except that the list size is larger than quadratic, but still polynomial. The currently shortest list size for a polynomial time list approximation is given by Zimand [10]. Closer to the stronger type of list approximation in this paper, are the probabilistic list approximation results for short programs from Bauwens and Zimand [2] and Zimand [11]. A polynomial-time probabilistic algorithm from [2], on input (x,k) returns a string p of length bounded by $k + O(\log^2 n)$ such that, if the promise $k = C(x)$ holds, then, with 0.99 probability, p is a program for x. In [11], it is shown that the promise can be relaxed to $k \geq C(x)$. The survey paper [8] presents most of these results. In this paper, we build on the techniques in [2, 11].

2 Techniques and proof overview

We start by explaining why an approach that probably first comes to mind cannot lead to a result with good parameters, such as those obtained in Theorem 1 with a more complicated argument.

Given that we want to modify a string x so that it becomes more complex, which in a sense means more random, a simple idea is to just append a random string z to x. Indeed, if we consider strings z of length c, then $C(xz) > C(x) + c/2$, for most strings z, provided c is large enough. Let us see why this is true. Let $k = C(x)$ and let z be a string that satisfies the opposite inequality, that is

$$C(xz) \leq C(x) + c/2\,. \tag{1}$$

Given a shortest program for xz and a self-delimited representation of the integer c, which is $2 \log c$ bits long, we obtain a description of x with at most $k + c/2 + 2 \log c$ bits. Note that from different z's satisfying (1), we obtain in this way distinct $(c/2 + 2 \log c)$-short programs for x. By a theorem of Chaitin [4] (also presented as Lemma 3.4.2 in [5]), for any d, the number of d-short programs for x is bounded by $O(2^d)$. Thus the number of strings z satisfying (1) is bounded by $O(2^{c/2+2\log c})$. Since for large c, $O(2^{c/2+2\log c})$ is much smaller than 2^c, it follows that most strings z of length c satisfy the claimed inequality (the opposite of (1)). Therefore, in this way we can obtain a list with a constant number of strings and most of them have complexity larger than $C(x)$. The problem with appending a random z to x, is that this operation not only increases complexity (which is something we want) but also increases length (which is something we don't want). The natural way to get around this problem is to first compress x to close to minimal description length using the probabilistic algorithms from [2, 11] described in the Introduction, and then to append z. However, the algorithms from [2, 11] compress x to length $C(x) + O(\log n)$, where n is the length of x. After appending a random z of length c, we obtain a string of length $C(x) + O(\log n) + c$, and for this to be n (so that length is not increased), we need $C(x) \leq n - O(\log n) - c$. Thus, in this way we cannot obtain a procedure that works for all x with $C(x) < n$, such as the one from Theorem 1.

Our solution is based on a more elaborate construction. The centerpiece is a type of bipartite graph with a low congestion property. Once we have the graph, we view x as a left node, and the list $f(x)$ consists of some of the nodes at distance 2 in the graph from x. (A side remark: Buhrman et al. [3] use graphs as well, namely constant-degree expanders, and they obtain the lists also as the set of neighbors at some given distance.) In our graph, the left side is $L = \{0,1\}^n$, the set of n-bit strings, the right side is $R = \{0,1\}^m$, the set of m-bit strings, and each left node has degree D. The graphs also depend on three parameters ϵ, Δ, and t, and for our discussion it is convenient to also use $\delta = \epsilon^{1/2}$ and $s = \delta \cdot \Delta$. The graphs that we need have two properties. The first one is a low congestion requirement which demands that for every subset B of left nodes of size at most 2^t, $(1 - \delta)$ fraction of nodes in B share $(1 - \delta)$ fraction of their right neighbors with at most s other nodes in B.[1] The second property is that each right node has at least Δ neighbors.

Let us now see how to use such graphs to increase Kolmogorov complexity in the list-approximation sense. Suppose we have a graph G with the above properties for the parameters n, δ, Δ, D, s, and t. We claim that for each x of length n and with complexity $C(x) < t$, we can obtain a list with $D \cdot \Delta$ many strings, all having length n, such that at least a fraction of $(1 - 2\delta)$ of the strings in the list have complexity larger than $C(x)$. Indeed, let x be a string of length n with $C(x) = k < t$. Consider the set $B = \{x' \in \{0,1\}^n \mid C(x') \leq k\}$. Note that the size of B is bounded by 2^t. A node that does not have the low-congestion property is said to be δ-BAD(B). By the low-congestion of G, there are at most $\delta|B|$ elements in B that are δ-BAD(B). It can be shown that x is not δ-BAD(B). The reason is, essentially, that the strings that are δ-BAD(B) can be enumerated and they make a small fraction of B and therefore can be described with less than k bits. Now, to construct the list, we view x as a left node in G and we "go-right-then-go-left." This means that we first "go-right," i.e., we take all the D neighbors of x, and for each such neighbor y we "go-left," i.e., we take Δ of the y's neighbors and put them in the list. Since x is not δ-BAD(B), $(1 - \delta)D$ of its neighbors have at most $s = \delta \cdot \Delta$ elements in B. Overall, only $2\delta \cdot D \cdot \Delta$ of the strings

[1] More formally, for all $B \subseteq L$ with $|B| \leq 2^t$, for all $x \in B$, except at most $\delta|B|$ elements, all neighbors y of x, except at most δD, have $\deg_B(y) \leq s$, where $\deg_B(y)$ is the number of y's neighbors that are in B.

in the list can be in B, and so at least a fraction of $(1-2\delta)$ of the strings in the list have complexity larger than $k = C(x)$. Our claim is proved.

For our main results (Theorem 1, Theorem 2, and Theorem 3), we need graphs with the above properties for different settings of parameters. Such graphs can be obtained from randomness extractors, which have been extensively studied in the theory of pseudo-randomness (for example, see Vadhan's monograph [9]). The graphs required by Theorem 1 and Theorem 2 are constructed using the probabilistic method in Lemma 5, and the graph required by Theorem 3 is obtained in Lemma 6 from a randomness extractor of Raz, Reingold, and Vadhan [6].

3 Balanced graphs

We define here formally the type of graphs that we need. We work with families of graphs $G_n = (L, R, E \subseteq L \times R)$, indexed by n, which have the following structure:

1. Vertices are labeled with binary strings: $L = \{0,1\}^n$, $R = \{0,1\}^{n-a}$, where we view L as the set of left nodes, and R as the set of right nodes. The parameter a can be positive or negative, and in absolute value is typically small (less than $\text{poly}(\log n)$).
2. All left nodes have the same degree D, $D = 2^d$ is a power of two, and the edges outgoing from a left node x are labeled with binary strings of length d.
3. We allow multiple edges between two nodes. For a node x, we write $N(x)$ for the *multiset* of x's neighbors, each element being taken with the multiplicity equal to the number of edges from x landing into it.

A bipartite graph of this type can be viewed as a function $\text{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{n-a}$, where $\text{EXT}(x,y) = z$ iff there is an edge between x and z labeled y. We want EXT to yield a (k,ϵ) randomness extractor whenever we consider the modified function EXT_k which on input (x,y) returns $\text{EXT}(x,y)$ from which we cut the last $n-k$ bits. Note that the output of EXT_k has $k-a$ bits.

From the function EXT_k, we go back to the graph representation, and we obtain the "prefix" bipartite graph $G_{n,k} = (L = \{0,1\}^n, R_k = \{0,1\}^{k-a}, E_k \subseteq L \times R_k)$, where in $G_{n,k}$ we merge the right nodes of G_n that have the same prefix of length $k-a$. Since we allow multiple edges between nodes, the left degrees in the prefix graph do not change. However, right degrees may change, and as m_k gets smaller, right degrees typically get larger due to merging.

The requirement that $G_{n,k}$ is a (k,ϵ) randomness extractor means that for every subset $B \subseteq L$ of size $|B| \geq 2^k$, for every $A \subseteq R_k$,

$$\left| \frac{|E_k(B,A)|}{|B| \times D} - \frac{|A|}{|R_k|} \right| \leq \epsilon, \tag{2}$$

where $E_k(B,A)$ is the set of edges between B and A in $G_{n,k}$.

We also want to have the guarantee that each right node in $G_{n,t}$ has degree at least Δ, where Δ and t are parameters.

Accordingly, we have the following definition.

▶ **Definition 4.** A graph $G_n = (L, R, E \subseteq L \times R)$ as above is (ϵ, Δ, t)-balanced if the following requierments hold:

1. For every $k \in \{1, \dots, n\}$, let $G_{n,k}$ be the graph corresponding to EXT_k described above. We require that, for every $k \in \{1, \dots, n\}$, $G_{n,k}$ is a (k,ϵ) extractor, i.e., $G_{n,k}$ has the property in Equation (2).
2. In the graph $G_{n,t}$, every right node with non-zero degree has degree at least Δ.

In our applications, we need balanced graphs in which the neighbors of a given node can be found effectively, or even in time that is polynomial in n. As usual, we consider families of graphs $(G_n)_{n\geq 1}$, and we say that such a family is *computable* if there is an algorithm that on input (x, y), where x is a left node, and y is the label of an edge outgoing from x, outputs z, where z is the right node where the edge y lands. If the algorithm runs in time polynomial in n, we say that the family $(G_n)_{n\geq 1}$ is *explicit*. For polynomial-time list approximation, we actually need a stronger property which essentially states that going from right to left can also be done in polynomial time (see the "Moreover..." part in Lemma 6).

The following two lemmas provide the balanced graphs that are used in the proofs of the main result as explained in the proof overview in Section 2.

▶ **Lemma 5.**

(a) *For every sufficiently large positive integer n, every rational $\epsilon > 0$, and every positive integer constant Δ, there is a computable (ϵ, Δ, t)-balanced graph $G_n = (L = \{0,1\}^n, R = \{0,1\}^m, E \subseteq L \times R)$, with left degree $D = 2^d = O(n^2 \cdot (1/\epsilon)^2)$, $m = n + 2\log n$, and $t = n$.*

(b) *There exists a constant c, such that for every sufficiently large positive integer n, every rational $\epsilon > 0$, and every positive integer constant Δ, there is a computable (ϵ, Δ, t)-balanced graph $G_n = (L = \{0,1\}^n, R = \{0,1\}^m, E \subseteq L \times R)$, with left degree $D = 2^d = O(n \cdot (1/\epsilon)^2)$, $m = n + d - 2\log(1/\epsilon) - c$, and $t = n - \log n - \log\log n$.*

The proof of Lemma 5 is by the standard probabilistic method, and is presented in Section 5.1.

▶ **Lemma 6.** *There exists a constant c such that for every positive integer n, every rational $\epsilon > 0$, and every positive integer $\Delta \leq 2^n$, there is an explicit (ϵ, Δ, t)-balanced graph $G_n = (L = \{0,1\}^n, R = \{0,1\}^m, E \subseteq L \times R)$, with left degree $D = 2^d$, for $d = O(\log^3(n)\log^2(1/\epsilon))$, $m = n - c \cdot d$, and $t = n - \max(0, \log\Delta - c \cdot d)$.*

Moreover, there is an algorithm that on input (z, y) (and n), where $z \in R = \{0,1\}^m$ and $y \in \{0,1\}^d$ computes a list of Δ left neighbors of z reachable from z by edges labeled y, or NIL if there are less than Δ such neighbors. This algorithm computes the list implicitly, in the sense that given an index i, it returns the i-th element in the list in time polynomial in n and $\log i$.

The proof of Lemma 6 is based on a randomness extractor of Raz, Reingold, and Vadhan [6] and is presented in Section 5.2.

Let us now proceed to the proofs of Theorem 1, Theorem 2, and Theorem 3.

4 Proofs of Theorem 1, Theorem 2, and Theorem 3

The theorems have essentially identical proofs, except that balanced graphs with different parameters are used. The following lemma shows a generic transformation of a balanced graph into a function that on input x produces a list so that most of its elements have complexity larger than $C(x)$.

▶ **Lemma 7.** *Suppose that for every constant $\delta > 0$, there is $t = t(n)$, $a = a(n)$, and a computable (respectively, explicit and satisfying the property stated in the "moreover" part of Lemma 6) $(\delta^2, (1/\delta)\Delta, t)$- balanced graph $G_n = (L_n = \{0,1\}^n, R_n = \{0,1\}^{n-a}, E_n \subseteq L_n \times R_n)$, with $\Delta = 2(1/\delta^2) \cdot D \cdot 2^a$, where D is the left degree.*

Then there exists a computable (respectively, polynomial-time computable) function f that on input a string x and a rational number $\delta > 0$ returns a list containing strings of length $|x|$ and

1. *The size of the list is* $2(1/\delta)^3 D^2 2^a$,
2. *If* $C(x) \leq t$, *then* $(1-2\delta)$ *of the elements in the list have complexity larger than* $C(x)$.

Proof of Lemma 7. We can assume without loss of generality that $1/\delta$ is sufficiently large (for the following arguments to be valid) and also that it is a power of 2. Let $\epsilon = \delta^2$. Thus, $\Delta = 2(1/\epsilon) \cdot D \cdot 2^a$. Let x be a binary string of length n, with complexity $C(x) = k$. We assume that $k \leq t$. We explain how to compute the list $f(x)$, with the property stipulated in the theorem's statement.

We take G_n to be the $(\epsilon, (1/\delta) \cdot \Delta, t)$-balanced graph with left nodes of length n promised by the hypothesis. Let $G_{n,t}$ be the "prefix" graph obtained from G_n by cutting the last $n-t$ bits in the labels of right nodes (thus preserving the prefix of length $t-a$ in the labels).

The list $f(x)$ is computed in two steps:

1. First, we view x as a left node in $G_{n,t}$ and take $N(x)$, the multiset of all neighbors of x in $G_{n,t}$.
2. Secondly, for each p in $N(x)$, we take A_p to be a set of $(1/\delta)\Delta$ neighbors of p in $G_{n,t}$ (say, the first $(1/\delta)\Delta$ ones in some canonical order). We set $f(x) = \bigcup_{p \in N(x)} A_p$ (if p appears n_p times in $N(x)$, we take A_p in the union also n_p times; note that $f(x)$ is a multiset).

Note that all the elements in the list have length n, and the size of the list is $|f(x)| = (1/\delta)\Delta \cdot D = (1/\delta)^3 D^2 2^a$.

The rest of the proof is dedicated to showing that the list $f(x)$ satisfies the second item in the statement. Let

$$B_{n.k} = \{x' \in \{0,1\}^n \mid C(x') \leq k\},$$

and let $S_{n,k} = \lfloor \log |B_{n,k}| \rfloor$. Thus, $2^{S_{n,k}} \leq |B_{n,k}| < 2^{S_{n,k}+1}$. Later we will use the fact that

$$S_{n,k} \leq k \leq t. \tag{3}$$

We want to use the properties of extractors for sources with min-entropy $S_{n,k}$ and therefore we consider the graph $G_{n,S_{n,k}}$, which is obtained, as we have explained above, from G_n by taking the prefixes of right nodes of length $S_{n,k} - a$. To simplify notation, we use G instead of $G_{n,S_{n,k}}$. The set of left nodes in G is $L = \{0,1\}^n$ and the set of right nodes in G is $R = \{0,1\}^m$, for $m = S_{n,k} - a$.

We view $B_{n,k}$ as a subset of the left nodes in G. Let us introduce some helpful terminology. In the following all the graph concepts (left node, right node, edge, neighbor) refer to the graph G. We say that a right node z in G is $(1/\epsilon)$-light if it has at most $(1/\epsilon) \cdot \frac{|B_{n,k}| \cdot D}{|R|}$ neighbors in $B_{n,k}$. A node that is not $(1/\epsilon)$-light is said to be $(1/\epsilon)$-heavy. Note that

$$(1/\epsilon) \cdot \frac{|B_{n,k}| \cdot D}{|R|} \leq (1/\epsilon) \frac{2^{S_{n,k}+1} \cdot D}{2^{S_{n.k}} \cdot 2^{-a}} = \Delta,$$

and thus an $(1/\epsilon)$-light node has at most Δ many neighbors in $B_{n,k}$.

We also say that a left node in $B_{n,k}$ is δ-BAD with respect to $B_{n,k}$ if at least a δ fraction of the D edges outgoing from it land in right neighbors that are $(1/\epsilon)$-heavy. Let δ-BAD$(B_{n,k})$ be the set of nodes that are δ-BAD with respect to $B_{n,k}$.

We show the following claim.

▶ **Claim 8.** *At most a* 2δ *fraction of the nodes in* $B_{n,k}$ *are* δ*-BAD with respect to* $B_{n,k}$.

(In other words: for every x' *in* $B_{n,k}$*, except at most a* 2δ *fraction, at least a* $(1-\delta)$ *fraction of the edges going out from* x' *in* G *land in right nodes that have at most* Δ *neighbors with complexity at most* k*.)*

We defer for later the proof of Claim 8, and continue the proof of the theorem.

For any positive integer k, let

$$B_k = \{x' \mid C(x') \le k \text{ and } k \le t(|x'|)\}.$$

Let $I_k = \{n \mid k \le t(n)\}$. Note that $|B_k| = \sum_{n \in I_k} |B_{n,k}|$. Let $x' \in B_k$, and let $n' = |x'|$. We say that x' is δ-BAD with respect to B_k if in $G_{n'}$, x' is δ-BAD with respect to $B_{n',k}$. We denote δ-BAD(B_k) the set of nodes that are δ-BAD with respect to B_k. We upper bound the size of δ-BAD(B_k):

$$\begin{aligned} |\delta\text{-BAD}(B_k)| &= \textstyle\sum_{n' \in I_k} |\delta\text{-BAD}(B_{n',k})| \\ &\le \textstyle\sum_{n' \in I_k} 2\delta \cdot |B_{n',k}| \qquad \text{(by Claim (8))} \\ &= 2\delta \textstyle\sum_{n \in I_k} |B_{n',k}| \\ &= 2\delta |B_k| \\ &\le 2\delta \cdot 2^{k+1}. \end{aligned}$$

Note that the set δ-BAD(B_k) can be enumerated given k and δ. Therefore a node x' that is δ-BAD with respect to B_k can be described by k, δ and its ordinal in the enumeration of the set δ-BAD(B_k). We write the ordinal on exactly $k + 2 - \log(1/\delta)$ bits and δ in a self-delimited way on $2 \log\log(1/\delta)$ bits (recall that $1/\delta$ is a power of 2), so that k can be inferred from the ordinal and δ. It follows that if x' is δ-BAD with respect to B_k, then, provided $1/\delta$ is sufficiently large,

$$C(x') \le k + 2 - \log(1/\delta) + 2\log\log(1/\delta) + O(1) < k. \tag{4}$$

Now, recall our string $x \in \{0,1\}^n$ which has complexity $C(x) = k$. The inequality (4) implies that x cannot be δ-BAD with respect to B_k, which means that $(1-\delta)$ of the edges going out from x land in neighbors in G having at most Δ neighbors in B_k. The same is true if we replace G by $G_{n,t}$, because, by the inequality (3), right nodes in G are prefixes of right nodes in $G_{n,t}$.

Now suppose we pick at random a neighbor p of x in $G_{n,t}$, and then find a set A_p of $(1/\delta) \cdot \Delta$ neighbors of p in $G_{n,t}$. Then with probability $1-\delta$, only a fraction of δ of the elements of A_p can be in B_k. Recall that we have defined the list $f(x)$ to be

$$f(x) = \bigcup_{p \text{ neighbor of } x \text{ in } G_{n,t}} A_p.$$

It follows that $(1-2\delta)$ of elements in $f(x)$ have complexity larger than $C(x)$ and this ends the proof. ◀

It remains to prove Claim 8.

Proof of Claim 8. Let A be the set of right nodes that are $(1/\epsilon)$-heavy. Then

$$|A| \le \epsilon |R|.$$

Indeed the number of edges between $B_{n,k}$ and A is at least $|A| \cdot (1/\epsilon) \cdot \frac{|B_{n,k}| \cdot D}{|R|}$ (by the definition of $(1/\epsilon)$-heavy), but at the same time the total number of edges between $B_{n,k}$ and R is $|B_{n,k}| \cdot D$ (because each left node has degree D).

Next we show that

$$|\delta\text{-BAD}(B_{n.k})| \le 2\delta |B_{n,k}|. \tag{5}$$

For this, note that G is a $(S_{n,k}, \epsilon)$ randomness extractor and $B_{n,k}$ has size at least $2^{S_{n,k}}$. Therefore by the property (2) of extractors,

$$\frac{|E(B_{n,k}, A)|}{|B_{n,k}| \cdot D} \leq \frac{|A|}{|R|} + \epsilon \leq 2\epsilon.$$

On the other hand the number of edges linking $B_{n,k}$ and A is at least the number of edges linking δ-BAD$(B_{n,k})$ and A and this number is at least $|\delta\text{-BAD}(B_{n,k}| \cdot \delta D$. Thus,

$$|E(B_{n,k}, A)| \geq |\delta\text{-BAD}(B_{n,k})| \cdot \delta D.$$

Combining the last two inequalities, we obtain

$$\frac{|\delta\text{-BAD}(B_{n,k})|}{|B_{n,k}|} \leq 2\epsilon \cdot \frac{1}{\delta} = 2\delta.$$

End of the proofs of Claim 8 and of Lemma 7. ◀

Theorem 1, Theorem 2 , and Theorem 3 are obtained by plugging into the above lemma the balanced graphs from Lemma 5 and Lemma 6. More precisely, Theorem 1 is obtained by using Lemma 7 with the balanced graph promised by Lemma 5(a), with parameters $\epsilon = \delta^2$, and $\Delta = 2(1/\delta^2) \cdot D \cdot 2^{-2\log n} = O(1)$. Theorem 2 is obtained by using Lemma 7 with the balanced graph promised by Lemma 5(b), with parameters $\epsilon = \delta^2$, and $\Delta = 2(1/\delta^2) \cdot D \cdot 2^{-(d-2\log(1/\epsilon)-c)} = O(1)$. Finally, Theorem 3 is obtained by using Lemma 7 with the balanced graph promised by Lemma 6, with parameters $\epsilon = \delta^2$, and $\Delta = 2(1/\delta^2) \cdot D \cdot 2^{cd} = 2^{O(\log^3 n)}$.

5 Construction of balanced graphs

5.1 Proof of Lemma 5

We prove part (b), where the relations between parameters are somewhat tighter. The proof of part (a) is similar. We use the probabilistic method. For some constant c that will be fixed later, we consider a random function $\text{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{n+d-2\log(1/\epsilon)-c}$. We show the following two claims, which imply that a random function has the desired properties with positive probability. Since the properties can be checked effectively, we can find a graph as stipulated in part (b) by exhaustive search. We use the notation from Definition 4 and from the paragraph preceding it.

▶ **Claim 9.** *For some constant c, with probability $\geq 3/4$, it holds that for every $k \in \{1, \ldots, n\}$, in the bipartite graph $G_{n,k} = \{L, R_k, E_k \subseteq L \times R_k\}$, every $B \subseteq L = \{0,1\}^n$ of size $|B| \geq 2^k$, and every $A \subseteq R_k = \{0,1\}^{k+d-2\log(1/\epsilon)-c}$ satisfy*

$$\left| \frac{|E_k(B, A)|}{|B| \times D} - \frac{|A|}{|R_k|} \right| \leq \epsilon. \tag{6}$$

▶ **Claim 10.** *For every sufficiently large positive integer n, with probability $\geq 3/4$, every right node in the graph $G_{n,n-\log n-\log\log n}$ has at least Δ neighbors in L.*

Proof of Claim 9. First we fix $k \in \{1, \ldots, n\}$ and let $K = 2^k$ and $N = 2^n$. Let us consider $B \subseteq \{0,1\}^n$ of size $|B| \geq K$, and $A \subseteq R_k$. For a fixed $x \in B$ and $y \in \{0,1\}^d$, the probability that $\text{EXT}_k(x, y)$ is in A is $|A|/|R_k|$. By the Chernoff bounds,

$$\text{Prob}\left[\left| \frac{|E_k(B, A)|}{|B| \times D} - \frac{|A|}{|R_k|} \right| > \epsilon\right] \leq 2^{-\Omega(K \cdot D \cdot \epsilon^2)}.$$

The probability that relation (6) fails for a fixed k, some $B \subseteq \{0,1\}^k$ of size $|B| \geq K$ and some $A \subseteq R_k$ is bounded by $2^{K \cdot D \cdot \epsilon^2 \cdot 2^{-c}} \cdot \binom{N}{K} \cdot 2^{-\Omega(K \cdot D \cdot \epsilon^2)}$, because A can be chosen in $2^{K \cdot D \cdot \epsilon^2 \cdot 2^{-c}}$ ways, and we can consider that B has size exactly K and there are $\binom{N}{K}$ possible choices of such B's. If $D = \Omega((n-k)/\epsilon^2)$ and c is sufficiently large, the above probability is much less than $(1/4)2^{-k}$. Therefore the probability that relation (6) fails for some k, some B and some A is less than $1/4$. ◀

Proof of Claim 10. We use a standard "coupon collector" argument. Let $t = n - \log n - \log\log n$. Let $N = 2^n$ and $C = 2^c$, where c is the constant for which Claim 9 holds.. We work in the bipartite graph $G_{n,n-\log n-\log\log n} = (L, R, E \subseteq L \times R)$ in which every left node has degree $D = 2^d$, $L = \{0,1\}^n$, and $R = \{0,1\}^m$, where $m = (n - \log n - \log\log n + d - 2\log(1/\epsilon) - c)$. For a left node x, an edge labeled $y \in \{0,1\}^d$ and a right node z, we say that (x,y) hits z if the y-labeled edge outgoing from x lands in z. We want to show that with high probability each z is hit at least Δ times. Let us order $\{0,1\}^n \times \{0,1\}^d$ in, say, lexicographical order $\{(x,y)_1 < (x,y)_2 < \ldots < (x,y)_{ND}\}$. We define Δ groups of "shooting" at R by taking $(x,y)_1, \ldots (x,y)_r$ in the first group, $(x,y)_{r+1}, \ldots (x,y)_{2r}$ in the second group, and so on with r left nodes in each group, where r will be fixed later. The probability that a fixed z is not hit by some $(x,y)_i$ is $(1 - 1/|R|) \leq e^{-1/|R|}$. The probability that a fixed z is not hit by any element in a given group is at most $e^{-r/|R|}$ and the probability that there exists some $z \in R$ that is not hit by a given group is bounded by $|R|e^{-r/|R|}$. We take $r = |R|(\ln|R| + \ln(4\Delta))$, and the above probability is bounded by $1/(4\Delta)$. Therefore, the probability that some z in R is not hit by some group in the set of Δ groups is at most $1/4$. Note that $r \cdot \Delta \leq ND$, provided n is large enough, and thus all the groups fit into $\{0,1\}^n \times \{0,1\}^d$. ◀

End of the proof of Lemma 5. ◀

5.2 Proof of Lemma 6.

The construction relies on the randomness extractor of Raz, Reingold, and Vadhan [6].

▶ **Theorem 11** (Theorem 22, (2) in [6]). *There exists a function* $\mathrm{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$*, computable in time polynomial in n, with the following properties:*
(1) $d = O(\log^3(n)\log^2(1/\epsilon))$,
(2) *$m = n - c \cdot d$, for some constant c,*
(3) *For every $k \leq n$, the function EXT_k obtained by computing EXT and cutting the last $n-k$ bits of the output is a (k,ϵ) extractor,*
(4) *For every $y \in \{0,1\}^{d(n)}$, the function $f_y(x) = \mathrm{EXT}(x,y)$ is a linear function from $(\mathrm{GF}[2])^n$ to $(\mathrm{GF}[2])^m$ (where we view $x \in \{0,1\}^n$ as an element of $(\mathrm{GF}[2])^n$ in the natural way). In other words, $\mathrm{EXT}(x,y) = A_y \cdot x$, where A_y is an m-by-n matrix with entries in $\mathrm{GF}[2]$, computable from y in time polynomial in n.*

Note. Item (4) is not explicitly stated in [6], so we provide here a short explanation. The construction given in [6] of $\mathrm{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, views x as the specification of a function $u_x(\cdot,\cdot)$ of two variables (in a way that we present below), defines some functions $g_1(y), h_1(y), \ldots, g_m(y), h_m(y)$, each one computable in time polynomial in n, and then sets

$$\mathrm{EXT}(x,y) = u_x(g_1(y), h_1(y)), \ldots, u_x(g_m(y), h_m(y)), \tag{7}$$

i.e., the i-th bit is $u_x(g_i(y), h_i(y))$. Thus, it is enough to check that $f_{v,w}(x) = u_x(v,w)$ is linear in x. Let us now describe u_x. The characteristic sequence of u_x is the Reed-Solomon code of x. More precisely, for some s, x is viewed as a polynomial p_x over the field $\mathrm{GF}[2^s]$.

The elements of GF$[2^s]$ are viewed as s-dimensional vectors over GF[2] in the natural way. Note that in this view the evaluation of p_x at point v is a linear transformation of x, i.e., $p_x(v) = A_v x$ for some s-by-n matrix A_v with entries from GF[2]. Finally, $u_x(v, w)$ is defined as the inner product $w \cdot p_x(v)$ and therefore $u_x(v, w) = (wA_v)x$, and thus it is a linear function in x. Now we plug $h_i(y)$ as w and $g_i(y)$ as v, and we build the matrix A_y, by taking its i-th row to be $h_i(y)A_{g_i(y)}$. Using the Equation (7), we obtain item (4) in the theorem.

Now let us proceed to the actual proof of Lemma 6. The function EXT from Theorem 11 defines the explicit bipartite graph G_n. Let $t = n - \max(0, \log \Delta - c \cdot d)$. By removing the last $n-t$ bits in each right node we obtain the graph $G_{n,t}$. We only need to check that in the bipartite graph $G_{n,t} = (L_t = \{0,1\}^n, R_t = \{0,1\}^{m_t}, E_t \subseteq L_t \times R_t)$ (where $m_t \leq n - \log \Delta$), every right node with non-zero degree has degree at least Δ. This follows easily from the linearity of $\text{EXT}_t(x, y)$ defined to be $\text{EXT}(x, y)$ from which we cut the last $n - t$ bits.

Indeed, let z in $\{0,1\}^{m_t}$ be a right node with non-zero degree. This means that there exist x and y such that $\text{EXT}_t(x, y) = z$. Since the function $f_y(x) = \text{EXT}_t(x, y)$ is linear in x, it follows that $\{x' \mid \text{EXT}(x', y) = z\} = \{x' \mid A_y \cdot x' = z\}$ (i.e., the preimage of z) is an affine space over GF[2] with dimension at least $n - m_t \geq \log \Delta$, and therefore z has degree at least Δ. Moreover, given y, we can find Δ preimages of z in time polynomial in n, by solving the linear system. ◀

Acknowledgments. The author is grateful to Bruno Bauwens for his insightful observations.

References

1 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.

2 Bruno Bauwens and Marius Zimand. Linear list-approximation for short programs (or the power of a few random bits). In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 241–247. IEEE, 2014. doi:10.1109/CCC.2014.32.

3 H. Buhrman, L. Fortnow, I. Newman, and N. Vereshchagin. Increasing Kolmogorov complexity. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 412–421, Berlin, 2005. Springer-Verlag *Lecture Notes in Computer Science #3404*.

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

5 R. Downey and D. Hirschfeldt. *Algorithmic randomness and complexity.* Springer Verlag, 2010.

6 Ran Raz, Omer Reingold, and Salil P. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. *J. Comput. Syst. Sci.*, 65(1):97–128, 2002. doi:10.1006/jcss.2002.1824.

7 Jason Teutsch. Short lists for shortest descriptions in short time. *Computational Complexity*, 23(4):565–583, 2014. doi:10.1007/s00037-014-0090-3.

8 Jason Teutsch and Marius Zimand. A brief on short descriptions. *SIGACT News*, 47(1):42–67, March 2016.

9 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012. doi:10.1561/0400000010.

10 Marius Zimand. Short lists with short programs in short time – A short proof. In Arnold Beckmann, Erzsébet Csuhaj-Varjú, and Klaus Meer, editors, *Language, Life, Limits – 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23-27, 2014*.

Proceedings, volume 8493 of *Lecture Notes in Computer Science*, pages 403–408. Springer, 2014. doi:10.1007/978-3-319-08019-2_42.

11 Marius Zimand. Kolmogorov complexity version of Slepian-Wolf coding. *CoRR*, abs/1511.03602, 2015. URL: http://arxiv.org/abs/1511.03602.