Electronic Colloquium on Computational Complexity, Report No. 41 (2021)

An Efficient Coding Theorem via Probabilistic Representations and its Applications

Zhenjian Lu*
Department of Computer Science
University of Warwick

Igor C. Oliveira†
Department of Computer Science
University of Warwick

March 15, 2021

Abstract

A *probabilistic representation* of a string $x \in \{0,1\}^n$ is given by the code of a randomized algorithm that outputs x with high probability [Oli19]. We employ probabilistic representations to establish the first unconditional Coding Theorem in *time-bounded* Kolmogorov complexity. More precisely, we show that if a distribution ensemble $\mathcal{D}_m$ can be uniformly sampled in time $T(m)$ and generates a string $x \in \{0,1\}^*$ with probability at least δ, then x admits a time-bounded probabilistic representation of complexity $O(\log(1/\delta) + \log(T) + \log(m))$. Under mild assumptions, a representation of this form can be computed from x and the code of the sampler in time polynomial in $n = |x|$.

We derive consequences of this result relevant to the study of data compression, pseudodeterministic algorithms, time hierarchies for sampling distributions, and complexity lower bounds. In particular, we describe an *instance-based* search-to-decision reduction for Levin's Kt complexity [Lev84] and its probabilistic analogue rKt [Oli19]. As a consequence, if a string x admits a succinct time-bounded representation, then a near-optimal representation can be generated from x with high probability in polynomial time. This partially addresses in a time-bounded setting a question from [Lev84] on the efficiency of computing an optimal encoding of a string.

*Email: zhen.j.lu@warwick.ac.uk
†Email: igor.oliveira@warwick.ac.uk

ISSN 1433-8092

Contents

1 Introduction

Shannon's information theory provides a foundation for the study of data transmission and data compression. However, it inherently considers *probability distributions* and *random variables*, and for this reason it does not apply to an *individual* object. The theory of Kolmogorov complexity on the other hand captures the information or computational content of an individual string or message. Despite significant conceptual differences between the two theories, results obtained in one setting sometimes admit an analogue in the other (see e.g. the textbooks [CT06, LV08, SUV17]).

A fundamental result connecting the distributional framework of Shannon and the information of an individual object x is the *Coding Theorem* in Kolmogorov complexity [Lev74].[1] This theorem states that if a randomized machine A generates a string x with probability δ, then its Kolmogorov complexity $\mathsf{K}(x)$ is at most $\log(1/\delta) + O_A(1)$. The result is part of a deep and beautiful theory that we will not be able to survey here. For a complexity-theoretic perspective that is closer to our work, we refer to [Lee06], where the coding theorem is referred to as one of the four pillars of Kolmogorov complexity.

An issue with this result and with Kolmogorov complexity more broadly is that many aspects of the theory are *nonconstructive*. For instance, computing or even estimating $\mathsf{K}(x)$ of an input string x is known to be undecidable. This limits the applicability of Kolmogorov complexity and of the aforementioned coding theorem in algorithms and complexity theory.

In order to import methods of Kolmogorov complexity from computability to complexity theory, a number of works have introduced *time-bounded* variants of Kolmogorov complexity (cf. [All92, All01, For04, All17] for a survey of results). In other words, one considers the minimum description length of a string x with respect to machines that operate *under a time constraint*. Among many applications, this idea has led Sipser [Sip83] to a proof that BPP is contained in the polynomial hierarchy, and Levin [Lev84, Section 1.3] to further develop universal search and optimal search algorithms.

Naturally, many authors have investigated time-bounded variants of the main results in Kolmogorov complexity.[2] Unfortunately, under standard hardness assumptions some of its most powerful theorems do not survive in time-bounded settings. Two notable examples are the language compression theorem (see [BLvM05] and references therein) and the principle of symmetry of information (see [LM93, LW95]).

Our focus in this work is on the coding theorem and its applications. Interestingly, there is no barrier to proving certain versions of the coding theorem in a time-bounded setting. For instance, Fortnow and Antunes [AF09] have implicitly established a form of this result, *under a strong computational assumption*. While their results are conditional and currently beyond reach without an assumption, they indicate that a useful time-bounded version of the coding theorem might be true.

Before proceeding with our discussion, we recall Levin's time-bounded Kolmogorov complexity. Let M be a deterministic machine and $|M|$ be its description length. For a string $x \in \{0,1\}^*$,

$$\mathsf{Kt}(x) \stackrel{\text{def}}{=} \min_{\text{TM } M,\, t \geq 1} \{|M| + \log t \mid M(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps}\},$$

[1] Some authors also refer to it as the Source Compression Theorem or Source Coding Theorem.

[2] Troy Lee's PhD thesis [Lee06] contains a particularly nice exposition of the contrast between Kolmogorov complexity and its time-bounded variants, including pointers to relevant references.

where $M(\varepsilon)$ denotes the computation of M over the empty string.[3] An important advantage of this definition over Kolmogorov complexity $\mathsf{K}(x)$ is that an encoding of minimal Kt complexity can be computed in exponential time via exhaustive search. Moreover, given an encoding w of x of Kt complexity at most k, we can recover x from w in time at most 2^k. Beyond its established applications in topics such as optimal search algorithms and pseudorandomness, Levin's Kt complexity plays an important role in a few other areas. Recent examples include hardness magnification (e.g. [OPS19, CJW19]) and the investigation of non-disjoint promise problems [Hir20].

A *probabilistic version* of Levin's Kt complexity has been recently investigated in [Oli19]. In this definition, we minimize over *randomized* machines that output x with probability at least $2/3$ after computing for at most t steps. In other words,

$$\mathsf{rKt}(x) \stackrel{\text{def}}{=} \min_{\text{RTM } M,\, t\geq 1} \{|M| + \log t \mid M(\varepsilon) \text{ outputs } x \text{ in } t \text{ steps with probability} \geq 2/3\}.$$

Consequently, an rKt description w of a string x provides a *probabilistic representation* of x, in the sense that x can be recovered with high probability from w. (Note that the description itself is a deterministic object.)

Under a mild derandomization assumption, $\mathsf{Kt}(x) = \Theta(\mathsf{rKt}(x))$ for every string x [Oli19, Theorem 5]. If so, this would show that any object that can be succinctly described in a probabilistic way can also be succinctly described in a deterministic way, and that results about rKt can be transferred to Kt. However, we appear to be far from establishing this relation, and it is consistent with our knowledge that there is an infinite sequence $\{x_n\}_{n\geq 1}$ where each x_n is an n-bit string satisfying $\mathsf{rKt}(x) = O(\log n)$ and $\mathsf{Kt}(x) = \Omega(n)$.

We do know without assumptions that various natural objects such as certain n-bit prime numbers can have rKt complexity $n^{o(1)}$ [Oli19, OS17], while establishing even an upper bound of $o(n)$ on the Kt complexity of a sequence of n-bit primes would lead to a breakthrough in the deterministic generation of primes [TCH12]. It seems therefore that probabilistic representations are useful to represent data, given our current knowledge of algorithms and complexity.

Another interesting aspect of rKt is that, despite its conjectured equivalence to Kt, we are able to settle basic questions about rKt that remain longstanding conjectures in the case of Kt. For instance, a natural computational problem is to approximate, given a string x, the value $\mathsf{rKt}(x)$. In other words, we would like to decide if a string has a short probabilistic representation. [Oli19] proved *unconditionally* that this problem cannot be solved in probabilistic polynomial time. In contrast, showing that computing Kt cannot be done in deterministic polynomial time (i.e., proving $\mathsf{MKtP} \notin \mathsf{P}$) is an important problem in time-bounded Kolmogorov complexity (cf. [ABK+06]).

We are motivated by these intriguing recent results and by the possibility of studying algorithmic information theory from the vantage point of probabilistic descriptions. The following questions are particularly relevant in this context.

1. Is it possible to employ rKt to establish new results in time-bounded Kolmogorov complexity? Specifically, can we establish an *unconditional* version of the coding theorem discussed above?

2. We have natural examples where probabilistic representations might be helpful, but no positive algorithmic results about *finding* such descriptions. Is it possible to compute a probabilistic representation of a string in some non-trivial way?

[3] To obtain precise results about the encoding lengths, it is necessary to fix a universal machine and to consider short inputs for this machine that produce x, as done by Levin. Since our techniques are not sensitive to encoding choices and our results incur a constant factor overhead in the encoding length, this is not relevant for our discussion.

3. Can we further develop the theory of probabilistic representations and show stronger results for natural objects such as prime numbers?

1.1 Contributions

We make progress in the context of these questions, obtaining results that suggest new research directions connected to data compression, time-bounded Kolmogorov complexity, search-to-decision reductions, and related areas. In particular, our results highlight the usefulness of *probabilistic representations* in algorithms and complexity. We describe these contributions in detail next.

1.1.1 Main results

For an algorithm or machine A, recall that we use $|A|$ to denote its description length under a fixed encoding scheme. One of our main contributions in this work is to show that an important result in Kolmogorov complexity survives in the time-bounded setting.

Theorem 1 (Efficient Coding Theorem for rKt)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^m)$ runs in time $T(m)$ and outputs a string $x \in \{0,1\}^*$ of length $n \leq T(m)$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) \;=\; O\big(\log(1/\delta) + \log(T(m)) + \log(m)\big),$$

where the constant behind the $O(\cdot)$ depends on $|A|$ and is independent of the remaining parameters. Moreover, given x, m, the code of A, and δ, it is possible to compute with probability ≥ 0.99 some rKt encoding of x of at most this complexity in time $\mathsf{poly}(|A|, \log(m), |x|, \log(1/\delta))$.

A few comments are in order. It is not hard to prove an effective coding theorem for a distribution whose *cumulative probability function* is computable in polynomial time (cf. [LV08]). Other works have established incomparable results under the assumption that deciding if a string is in the support of the distribution is easy (see e.g. [TVZ05]). Crucially, the theorem above only assumes that the distribution is *samplable*, which makes it more broadly applicable. To our knowledge, Theorem 1 is the first result that provides a general approach to constructing a probabilistic encoding of a string.

It follows from Shannon's Coding Theorem that the expected encoding length in Theorem 1 is essentially optimal up to a constant factor. An interesting feature of the result is that the algorithm computing the rKt encoding runs in polynomial time regardless of the time complexity of the sampler $A(1^m)$. Furthermore, producing the encoding of a string x only requires knowledge of a lower bound on its probability weight, in contrast to encoding algorithms such as Huffman coding where knowledge of the probabilities of all elements in the support of the distribution is necessary.

Using an argument of Levin (see [GS91] and [Lee06, Section 5.3] or Appendix A), under the existence of one-way functions there is a polynomial-time samplable source $\mathcal{D}_n$ supported over $\{0,1\}^n$ such that every $x \in \mathsf{Support}(\mathcal{D}_n)$ has probability weight $\mathcal{D}_n(x) \geq 2^{-n^\varepsilon}$, but $\mathcal{D}_n$ does not admit a pair $(\mathsf{Enc}_n, \mathsf{Dec}_n)$ of efficient (probabilistic) encoding and decoding algorithms such that each $x \in \mathsf{Support}(\mathcal{D}_n)$ is assigned a description of length $\leq n-3$. In contrast, Theorem 1 is able to sidestep this limitation by using an efficient encoding algorithm whose associated decoding procedure (provided by the rKt representation itself) is not necessarily efficient. There are applications

where this trade-off might be acceptable, i.e., where it is crucial to achieve high compression rates and fast (or low energy) encoding, but for which decoding is allowed to take more resources.[4]

We refer the reader to Section 3.3 for additional remarks on Theorem 1 and its consequences. In Appendix A, we contrast the result to the coding theorem in (unrestricted) Kolmogorov complexity. In Appendix B, we optimise the space complexity of the encoding and decoding algorithms employed in the proof of Theorem 1, under the assumption that the sampler runs in bounded space.

Similarly to the coding theorem in Kolmogorov complexity, it is possible to derive a variety of consequences from Theorem 1. We cover in more detail one of its most unexpected implications, deferring the discussion of a couple of other results to Section 1.1.2 below.

A *search-to-decision reduction* is an efficient procedure that allows one to find solutions to a problem from the mere ability to decide when a solution exists. These reductions are particularly important in algorithms and complexity. On the one hand, theories of computational complexity are often easier to develop in the context of *decision problems*. On the other hand, in practice solving a *search problem* tends to be the relevant task. A search-to-decision reduction for a given problem shows that the complexities of its search and decision versions are similar.

It is well known that any NP-complete problem admits a search-to-decision reduction. However, there are problems of interest for which such reductions are still unknown. A notable example in the realm of time-bounded Kolmogorov complexity is whether MCSP, the Minimum Circuit Size Problem, admits a search-to-decision reduction (cf. [Ila20] for a discussion and a recent result).

We establish the existence of the following search-to-decision reductions for rKt and Kt complexities.

Theorem 2 (Instance-based search-to-decision reductions)**.** *The following results hold:*

(*i*) *There is a randomized polynomial-time algorithm that, when given an input string $x \in \{0,1\}^n$ and a value $k \geq \mathsf{rKt}(x)$, outputs with probability ≥ 0.99 a valid rKt representation of x of complexity $O(k)$.*

(*ii*) *Similarly, there is a randomized polynomial-time algorithm that, when given an input string $x \in \{0,1\}^n$ and a value $k \geq \mathsf{Kt}(x)$, outputs with probability ≥ 0.99 a valid Kt representation of x of complexity $O(k)$.*

We note that [ABK+06] established that any language in deterministic exponential time E can be non-uniformly reduced via polynomial-size circuits to the problem of deciding Kt complexity (or even of just approximating Kt on a large fraction of inputs). Since the problem of finding a minimal Kt representation of an input string can be encoded as a language in E, it follows from their work that there is a polynomial-size search-to-decision reduction for Kt. Observe that the reduction given by [ABK+06] finds a description of minimum length, while Theorem 2 only provides a description that is within a constant factor of the optimal description length.

There are now a number of search-to-decision reductions in the context of time-bounded Kolmogorov complexity with respect to a variety of string complexity measures (e.g. [CIKK16, Hir18, Ila20, ILO20, LP20]). We are not aware, however, of a previous *instance-based* search-to-decision reduction in the sense of Theorem 2. In other words, Theorem 2 shows that it is possible to produce a near-optimal Kt representation of x from a decision oracle for Kt *that is only queried on x*.[5] In

[4] As a speculative scenario, one can imagine data transmission for deep space exploration.

[5] A binary search is sufficient to compute $\mathsf{rKt}(x) \in \mathbb{N}$ from a decision oracle that checks if $\mathsf{rKt}(x) \leq \gamma$ for a given threshold γ.

contrast, the aforementioned reductions require an oracle to the decision problem that is correct on all or at least on a large fraction of inputs. More broadly, search-to-decision reductions for problems in other domains such as circuit satisfiability and graph theory tend to query inputs that modify the original input x.

As a result of the property described above, we are able to derive consequences from Theorem 2 that do not follow from other reductions. Unsurprisingly, our approach employs significantly different techniques compared to the papers cited above. In particular, it departs from the PRG-based approach of [ABK+06] and of a few other subsequent works.

We elaborate next on some aspects of Theorem 2 that make it particularly interesting, at least to the authors. Note that Kt (similarly for rKt and probabilistic algorithms) is a *universal* complexity measure for data compression, in the following precise sense. If a string x can be encoded/decoded by *some* uniform compression scheme in time $\leq T$ and using a description of length $\leq \log T$, then its Kt complexity is at most $O(\log T)$. As a consequence, compressing a string x to $O(\mathsf{Kt}(x))$ bits is not far from optimal *in a strong sense*.

Now suppose we have a string x of length n, and we estimate its Kt complexity to be at most, say, $O(\sqrt{n})$. Then finding a valid Kt representation of this complexity via *exhaustive search* would take time $2^{O(\sqrt{n})}$. Assuming the widely believed hardness of estimating the Kt complexity of a string (which is provably true for rKt by [Oli19]), one is tempted to conjecture that nothing better can be done. Still, Theorem 2 tells us that there is an algorithm that can produce a valid Kt representation of x of complexity $O(\sqrt{n})$ in time just $\mathsf{poly}(n)$.[6] This result addresses to some extent in the time-bounded setting a question from [Lev84, Page 5] on the efficiency of generating programs of length close to the optimal description length.

It is unclear to us whether there are *deterministic* search-to-decision reductions for Items (i) and (ii) of Theorem 2. Although this requires more investigation, it is possible that these reductions provide natural examples of randomized algorithms that cannot be replaced by deterministic ones with only a polynomial overhead.[7]

1.1.2 Further applications and open problems

Efficient construction of time-bounded programs and complexity lower bounds for estimating rKt. By executing the efficient search-to-decision reduction from Theorem 2 with all values of $k = O(n)$ that are powers of 2, the following corollary is immediate.

Corollary 3 (Effective short lists with short programs in short time)**.** *Given an arbitrary string x of length n, it is possible to compute with high probability and in polynomial time a collection of at most $d = \log(n) + O(1)$ strings $w_1, \ldots, w_d$ such that at least one of these strings is a valid rKt encoding of x of complexity $O(\mathsf{rKt}(x))$.*

The same result can be obtained for Kt complexity. Corollary 31 should be contrasted with the results from [BZ14, BMVZ18] in the context of (time-unbounded) Kolmogorov complexity. They achieve optimal compression rates, by mapping a string of Kolmogorov complexity k to a collection of strings that contains a valid representation of length $k + O(1)$. While we are not able to achieve this level of compression, our setting is more stringent, since we need to construct a short representation that can be decompressed under a time constraint. It would be interesting to

[6] This does not contradict the intractability result from [Oli19]. Indeed, it implies that *checking* if a given representation generates a particular string is the problem that requires super-polynomial time.

[7] Note that this is not inconsistent with $\mathsf{P} = \mathsf{BPP}$ because these are not decision problems.

explore connections between our techniques and those employed in Kolmogorov complexity to see if our parameters can be further improved.

In light of Corollary 3 and the complexity lower bound for estimating the rKt complexity of a string ([Oli19]; see Theorem 18), it follows that the intractability of estimating rKt does not lie in the exhaustive search required to *find* a succinct representation. Instead, the hardness is a consequence of the intractability of *checking* if a given rKt representation is valid for a string x.

It is unlikely that circuit minimization of a given truth-table (MCSP) admits a search-to-decision reduction of the form given by Theorem 2. This would allow one to construct in time polynomial in the size of the input truth-table a collection of circuits that contains a circuit of near-optimal size for the truth-table. As opposed to rKt, checking if a circuit correctly encodes a string can be done in polynomial time, which implies that such a search-to-decision reduction provides a natural property in the sense of [RR97]. Consequently, under cryptographic assumptions, minimizing circuit size and minimizing Kt/rKt behave differently with respect to instance-based search-to-decision reductions.

Hardness of approximately sampling distributions. It is not hard to show that if Promise-BPP $\subseteq$ Promise-P then BPTIME[$\cdot$] admits a time hierarchy theorem. This easily follows from the deterministic time hierarchy theorem for decision problems. As a consequence of our results, we observe that a similar derandomization hypothesis for *decision problems* implies a strong time hierarchy theorem for *sampling distributions*.

To our knowledge, a uniform time hierarchy theorem for sampling distributions was first established in [Wat14]. While the results of his work are *unconditional*, [Wat14] left open the problems of proving an *almost-everywhere* lower bound and of achieving a *larger statistical gap* [Wat14, Section 5]. Our next result provides a conditional solution to these questions.

Theorem 4 (A strong time hierarchy theorem for sampling distributions)**.** *Under the assumption that* Promise-BPE $\subseteq$ Promise-E*, there is a constant $\zeta > 0$ for which the following holds. Let $n^{1/\zeta} \leq T(n) \leq 2^n$ be any constructive time bound. There is an ensemble $\{\mathcal{D}_n\}_{n\geq 1}$ of distributions $\mathcal{D}_n$ such that:*

- *(i) Each distribution $\mathcal{D}_n$ is supported over a single string $z_n \in \{0,1\}^n$.*
- *(ii) There is a deterministic algorithm $A(1^n)$ that samples $\mathcal{D}_n$ and runs in time $O(T(n))$.*
- *(iii) For each randomized algorithm $B(1^n)$ that runs in time $O(T(n)^\zeta)$ and for every large enough n, the statistical distance of $\mathcal{D}_n$ and $B(1^n)$ is at least $1 - 1/T(n)^\zeta$.*

We are not aware of a previous time hierarchy theorem for sampling distributions able to produce hard distributions of support size one, even under computational assumptions. (Interestingly, the unconditional results of [Wat14] hold for support size $k \geq 2$.) Note that Theorem 4 exploits *uniformity* in a crucial way: each distribution $\mathcal{D}_n$ can be sampled by a (non-uniform) linear-size circuit C_n that ignores its random input and outputs the unique string z_n in the support of $\mathcal{D}_n$.

Computational patterns in prime numbers. We now step back and revisit one of the most basic questions associated with the power of probabilistic representations. The problem stated below is concerned with the computational (in)compressibility of n-bit prime numbers.

Problem 5 (Primes with short descriptions)**.** *Is there an infinite sequence $\{p_n\}_{n\geq 1}$ of prime numbers $p_n \in [2^{n-1}, 2^n - 1]$ such that $\mathsf{rKt}(p_n) = o(n)$?*

This would show that there are primes of *every* large length that admit effective short encodings, i.e., such primes possess computational patterns that translate into effective representations (e.g. Mersenne primes). A partial result appears in [OS17], where this is proved for *infinitely* many n. Consequently, some primes can have short descriptions. *Is this a rare phenomenon, or does it happen for primes of all lengths?* This is the question captured by Problem 5.

We note that obtaining a solution to Problem 5 is necessary before showing the existence of a deterministic algorithm that generates n-bit primes in time $2^{o(n)}$. We refer to [TCH12] for more background on this problem.

Theorem 1 offers a path to solving this problem. In other words, it shows that it is enough to sample n-bit numbers in time $2^{o(n)}$ in a way that assigns enough weight to some (possibly unknown) prime. Given that many advances to our understanding of prime numbers employ *probabilistic ideas* (see e.g. [Ten15, May19] and references therein), this perspective could be fruitful.

Our last result shows that the existence of a sampler of this form is in fact *equivalent* to a positive solution to Problem 5.

Theorem 6 (Equivalence between faster samplability and improved time-bounded descriptions)**.** *The following statements are equivalent:*

(*i*) Sampling Algorithm. *For every $\varepsilon > 0$, there is a randomized algorithm $A(1^n)$ sampling strings in $\{0,1\}^*$ that runs in time $T(n) = O(2^{\varepsilon n})$ and for which the following holds. For every large n, there is an n-bit prime q_n such that $\Pr[A(1^n)$ outputs $q_n] \geq 2^{-\varepsilon n}$.*

(*ii*) Short Descriptions. *Let $\delta > 0$ be an arbitrary constant. For every large n, there is an n-bit prime p_n such that $\mathsf{rKt}(p_n) \leq \delta n$.*

We stress that the equivalence in Theorem 6 is not particular to prime numbers and to exponential time bounds. It can be seen as the analogue for rKt of Levin's fundamental insight that a sequence of objects (such as n-bit primes or solutions to search problems) can be deterministically generated in time $T(n)$ if and only if they have Kt complexity of order $\log(T(n))$.

Finally, complementing the applications and open problems mentioned above, it would be interesting to understand when a mathematical method employed to show the existence of certain combinatorial objects also implies the existence of objects of bounded rKt complexity. This is particularly interesting in settings where the desired objects are "rare" and the probability of producing them is small (e.g. Lovász local lemma and techniques from discrepancy theory). Extracting rKt upper bounds from existential proofs offers an alternate way of designing non-trivial algorithms for generating the corresponding objects, since it is sufficient to exhaustively search for objects of bounded description length.[8]

1.2 Techniques

In this section, we describe the main conceptual ideas behind Theorems 1 and 2. Since our goal is to establish a coding theorem in a time-bounded setting and our applications require an algorithm that produces a valid encoding in polynomial time, it is not clear if arguments employed in the context of (unbounded) Kolmogorov complexity can be adapted to our setting. For instance,

[8] In some applications, one might need to consider *conditional* rKt complexity. The techniques employed in this work also extend in this direction (see Section 3.3 for more details).

it is not hard to construct an encoding given all strings in the support of the distribution and their corresponding probabilities, but we cannot assume in the time-bounded setting that this is available. Similarly, under additional assumptions on the distribution, such as the ability to compute its cumulative probability function, producing an encoding is easier. However, we are aiming for a result that applies to the larger class of samplable distributions.

Sketch of the proof of Theorem 1. We are given a randomized algorithm $A(1^m)$ that runs in time T and samples from a distribution $\mathcal{D}_m$. Fix a string $x \in \mathsf{Support}(\mathcal{D}_m)$, and assume that its probability weight $\mathcal{D}_m(x) \geq \delta$. Our goal is to (efficiently) produce a succinct probabilistic representation of x in the sense of rKt complexity. The first thing to notice is that there are at most $1/\delta$ strings $y \in \mathsf{Support}(\mathcal{D}_m)$ such that $\mathcal{D}_m(y) \geq \delta$. Let $S_\delta \stackrel{\text{def}}{=} \{y \in \{0,1\}^* \mid \mathcal{D}_m(y) \geq \delta\}$ be the set of such strings, which includes our target string x. We assume for simplicity of the exposition that $S_\delta \subseteq \{0,1\}^n$, where $n = |x|$ is the length of x.

Let's pretend for now that we know the set S_δ. Note that this is not really a realistic assumption, since we are aiming to output a valid rKt description of x in a number of steps that might not even allow us to sample a single string from $\mathcal{D}_m$. We will revisit this assumption later on, and argue that explicit knowledge of S_δ is not needed to produce a probabilistic representation of x.

Our current goal is to be able to identify x among the elements of S_δ, ideally with an advice string of length close to $O(\log |S_\delta|) = O(\log(1/\delta))$. A natural way to try to achieve this goal is by producing a "fingerprint" or "hash value" from x that uniquely specifies this string. In other words, we would like to have a function $h \colon S_\delta \to \{0,1\}^*$ such that $h(x) \neq h(y)$ for every $y \in S_\delta \backslash \{x\}$. Then we can identify x in S_δ using h and the value $z = h(x)$. Assuming that we know S_δ, the total description length of x would be upper bounded by roughly $|h| + |z|$, where $|h|$ is the description length of h and $|z|$ is the length of z.

This is a basic algorithmic problem, and one way to achieve this goal with a reasonable upper bound on the description length is as follows. Given an explicit polynomial-time computable error-correcting code $E \colon \{0,1\}^n \to \{0,1\}^{O(n)}$, let $T_\delta \stackrel{\text{def}}{=} E(S_\delta)$, i.e., each string $y' \in T_\delta$ is obtained by applying E to a string $y \in S_\delta$. Consequently, for every distinct pair $y', y'' \in T_\delta$, their relative hamming distance $d(y', y'') = \Omega(1)$. For this reason, it follows by a simple probabilistic analysis that if we *randomly project* about $O(\log(1/\delta))$ coordinates of the strings in T_δ, we are likely to produce a "fingerprint" that uniquely specifies each string. Thus to specify x in S_δ it is enough to compute $E(x)$ and to store $O(\log(1/\delta))$ random pairs (i, b_i), where $i \sim [O(n)]$ and $b_i \stackrel{\text{def}}{=} E(x)_i$, the i-th bit of the string $E(x)$. Following our notation from above, one can think of h as being given by the sequence of coordinates, and z by the bits obtained from the projection of $E(x)$.[9]

There are two potential issues with this approach:

(a) In order to store x's fingerprint information (h and z), we still need $O(|h|+|z|) = O(\log(1/\delta) \cdot \log(n) + \log(1/\delta))$ bits, instead of just $O(\log(1/\delta))$.

(b) We have assumed explicit knowledge of S_δ to recover x from h and z.

The first issue is of a quantitative nature, and it can be handled via standard techniques. By projecting coordinates of $E(x)$ according to a random walk on an explicit constant-degree expander graph on $O(n)$ vertices, the total description length can be reduced to $O(\log(n) + \log(1/\delta))$.

[9] Notice that an *explicit* error-correcting code together with a bounded number of random coordinates of the string $E(x)$ were used to minimize the description length of x's fingerprint. We can also generate a fingerprint by collecting the value of random XORs χ_S applied to x, but storing the relevant sets S would have been expensive.

Regarding the more challenging issue (b), first notice that the argument we have described so far uses randomness only to produce h. In particular, it gives a *randomized* algorithm that with high probability produces a *deterministic* representation of x from h, $z = h(x)$, and S_δ. Therefore, we have not yet exploited the power of *probabilistic representations*.

A simple but crucial idea is that we can use the *code* of the sampler A and m to efficiently *compute a valid* rKt *representation* of x, without ever running $A(1^m)$. More precisely, the rKt *instructions* to output x include running $A(1^m)$ about $O((1/\delta) \cdot \log(1/\delta))$ times to collect (with high probability) a superset $W_\delta \supseteq S_\delta$. Let's assume for simplicity that $W_\delta = S_\delta$ (it is possible to take care of extra elements by a more delicate argument). Given the set W_δ, $n = |x|$, and an independently generated h obtained before we compute the rKt representation, h and the hash value $z = h(x)$ are likely to isolate x among the elements in W_δ. A careful implementation of these ideas allows us in randomized polynomial-time to generate an rKt representation of x whose description length is $O(\log(m) + \log(1/\delta) + \log(n))$ and whose logarithm of the running time is $O(\log(T(m)) + \log(1/\delta))$. Since generating an n-bit string x takes time $T \geq n$, the overall rKt complexity (description length + log of the running time) is $O(\log(1/\delta) + \log(T(m)) + \log(m))$.

Sketch of the proof of Theorem 2. First, we discuss a reduction for rKt. Given a string x and an upper bound $k \geq \mathsf{rKt}(x)$, we need to output a valid rKt representation of x of complexity $O(k)$. This is not a lot of information, but we have a general tool at our disposal: the Coding Theorem for rKt (Theorem 1). In particular, if we could sample x in time $2^{O(k)}$ and with probability $2^{-\Omega(k)}$ using an *explicit* algorithm A, we would be done by the moreover part of this result. The only potential challenge is to uniformly construct an explicit sampler of this form.

Fortunately, it is not hard to show that there is a *universal* sampler U that works for all strings of rKt complexity at most k. In more detail, the sampler randomly selects the code of a randomized machine M of length at most k, simulates M with its internal randomness for at most 2^k steps, and outputs whatever is left on the output tape of M after this simulation. Using that $\mathsf{rKt}(x) \leq k$, which implies that some randomized machine of length at most k outputs x within 2^k steps with probability at least $2/3$, it is easy to see that x has probability weight at least $2^{-\Omega(k)}$ under U.

Given that the code of U is explicit and we know x and k, the desired representation of x can be generated with high probability in polynomial time via Theorem 1.

We now consider a search-to-decision reduction for Kt. Recall that, under a derandomization assumption, $\mathsf{Kt}(x) = \Theta(\mathsf{rKt}(x))$. Moreover, as we observe in Lemma 32, there is an efficient deterministic algorithm that converts a probabilistic representation in the sense of rKt into a deterministic one in the sense of Kt. For this reason, it is not hard to show, under a plausible computational assumption, that there is an instance-based search-to-decision reduction for Kt, given the same result for rKt.

The most interesting aspect of Item (ii) of Theorem 2 is that it is possible to *unconditionally* establish the result. This is obtained by a more careful investigation of the elements employed in the proofs of Theorem 1 and Theorem 2 Item (i), which reveals that the derandomization assumption is not really needed. We refer the reader to the main body of the paper for the details.

Organization. The proof of Theorem 1 appears in Section 3, while the remaining results are established in Section 4. In Appendix A, we provide alternate proofs of Theorem 1, compare the result to the coding theorem in Kolmogorov complexity, and discuss its parameters in detail. In Appendix B, we optimise the space complexity of the encoding and decoding procedures behind Theorem 1.

2 Preliminaries

2.1 Basic notions

We use $|x|$ to denote the length of a binary string $x \in \{0,1\}^*$. We abuse notation and use $|M|$ denote the length of the binary encoding of a machine M with respect to a fixed universal machine. Although this will not be essential, we assume a prefix-free encoding of machines, and remark that our statements are robust with respect to encoding choices.

Probabilistic machines have an extra tape with random bits. We use $\boldsymbol{M_{\leq t}}$ to denote a random variable that represents the content of the output tape of M when it computes for t steps over the empty string ε (or its final content if the machine halts before that).

Definition 7 (rKt_δ Complexity)**.** *For $\delta \in [0,1]$ and a string $x \in \{0,1\}^*$, we let*

$$\mathsf{rKt}_\delta(x) = \min_{M,t} \left\{ |M| + \lceil \log t \rceil \;\middle|\; \Pr[\boldsymbol{M_{\leq t}} = x] \geq \delta \right\}.$$

The randomized time-bounded Kolmogorov complexity of x is given by $\mathsf{rKt}(x) \stackrel{\text{def}}{=} \mathsf{rKt}_{2/3}(x)$.

Levin's complexity $\mathsf{Kt}(x)$ can be defined similarly, either by taking $\delta = 1$ in the definition above or by restricting the minimization over M and t to deterministic machines. For more information about rKt, see [Oli19].

We will assume from our encoding that for every string x of length n, $\mathsf{rKt}(x) \leq \gamma \cdot n$, where $\gamma \in \mathbb{N}$ is a universal constant. This is achieved by a trivial machine that simply stores x and prints it when given an empty string. Since printing a string x takes time at least $|x|$, we have $\mathsf{rKt}(x) \geq \mathsf{Kt}(x) \geq \log |x|$. We might implicitly use this fact to omit certain $\log n$ additive factors in our upper bounds.

These definitions and conventions are sufficient for the formalization of our results, given that the proof of Theorem 1 incurs a constant factor overhead in the resulting rKt upper bound. For the interested reader, we present a careful discussion of the parameters of Theorem 1 in Appendix A. For more background in time-bounded Kolmogorov complexity, see [LV08].

For a discrete probability distribution $\mathcal{D}$, we use $\mathsf{Support}(\mathcal{D})$ to denote its support. For $x \in \mathsf{Support}(\mathcal{D})$, we let $\mathcal{D}(x)$ denote its probability weight under $\mathcal{D}$. We extend this definition to a set T in the natural way, i.e., $\mathcal{D}(T) = \sum_{x \in T} \mathcal{D}(x)$. If $\mathcal{D}$ and $\mathcal{D}'$ are distributions with support contained in a set S, their statistical distance $|\mathcal{D} - \mathcal{D}'|$ is defined as $\max_{T \subseteq [S]} |\mathcal{D}(T) - \mathcal{D}'(T)|$.

Definition 8 (Entropy)**.** *The entropy $H(\mathcal{D})$ of a discrete probability distribution $\mathcal{D}$ is defined as*

$$H(\mathcal{D}) \;=\; \sum_{x \in \mathsf{Support}(\mathcal{D})} \mathcal{D}(x) \cdot \log \frac{1}{\mathcal{D}(x)},$$

where log *is the binary logarithm function.*

We will also require a standard application of expander graphs in order to minimize the amount of randomness in one of our constructions.

Definition 9 (Expander Graph)**.** *An m-vertex undirected graph G is an (m, d, λ)-expander if G is d-regular and $\lambda(G) \leq \lambda$, where $\lambda(G)$ denotes the second largest eigenvalue (in absolute value) of the normalized adjacency matrix of G (i.e., the adjacency matrix of G divided by d).*

2.2 Technical tools

The version of the concentration bound appearing below can be found for instance in [Val19].

Theorem 10 (Chernoff Bound)**.** *Let $X = \sum_{i=1}^{n} X_i$, where each X_i is an independent 0/1-valued random variable. The following inequalities hold.*

(i) For every $\gamma > 0$, if $\mu \geq \mathbf{E}[X]$ then $\mathbf{Pr}[X \geq (1+\gamma)\mu] \leq \left(\frac{e^{\gamma}}{(1+\gamma)^{(1+\gamma)}}\right)^{\mu}$.

(ii) For every $0 < \gamma \leq 1$, if $\mu \leq \mathbf{E}[X]$ then $\mathbf{Pr}[X \leq (1-\gamma)\mu] \leq \left(\frac{e^{-\gamma}}{(1-\gamma)^{(1-\gamma)}}\right)^{\mu}$.

Theorem 11 (Explicit ECCs; see e.g. [Spi96])**.** *There is a constant $C \in \mathbb{N}$ and a polynomial-time computable function $E_n \colon \{0,1\}^n \to \{0,1\}^{Cn}$ such that for each $n \geq 1$ and for any distinct strings $a, b \in \{0,1\}^n$, the relative hamming distance between $E_n(a)$ and $E_n(b)$ is at least $1/10$.*

We will rely on the following explicit construction of expander graphs.

Theorem 12 ((Strongly) Explicit Expander Graphs [GG81])**.** *There are constants $d \in \mathbb{N}$ and $0 < \lambda < 1$ for which the following holds. There is an (m, d, λ)-expander family $\{G_m\}$ of m-vertex graphs and a deterministic algorithm A that on inputs $m \in \mathbb{N}$, $v \in [m]$, and $i \in [d]$ outputs the i-th neighbor of v in G_m in time polynomial in $\log(m)$.*

We will also need the following well-known property of a random walk on an expander graph.

Theorem 13 (Expander Chernoff Bound [Gil98])**.** *Let $G_m = (V, E)$ be an (m, d, λ)-expander, $f \colon V \to \{0,1\}$ be an arbitrary function, and $\mu \stackrel{\text{def}}{=} \mathbf{E}_{v \sim V}[f(v)]$. Let $v_1 \sim V$ be a uniformly chosen vertex and $v_1, \ldots, v_t$ be a random walk on G of length t. Then, for any $\alpha > 0$,*

$$\Pr_{v_1,\ldots,v_t}\left[\frac{1}{t}\sum_{i=1}^{t} f(v_i) < \mu - \alpha\right] \;\leq\; e^{-(1-\lambda)\alpha^2 t/4}.$$

For convenience of the reader, we recall the following basic result from information theory. (We state the lower bound component only.)

Theorem 14 (Shannon's Source Coding Theorem; see e.g [CT06])**.** *Let $\mathcal{D}$ be a discrete probability distribution, and $f \colon \mathsf{Support}(\mathcal{D}) \to \{0,1\}^*$ be an injective function. Then its expected encoding length satisfies*

$$\mathop{\mathbf{E}}_{x \sim \mathcal{D}}[|f(x)|] \;\geq\; H(\mathcal{D}).$$

Some results rely on the following hardness hypothesis and its consequences.

Conjecture 15 (Exponential circuit lower bounds for a language in E)**.** *There exists a language $L \in \mathsf{E}$ and a constant $\varepsilon > 0$ such that L_n requires circuits of size $2^{\varepsilon n}$ for all but finitely many n.*

Theorem 16 ([NW94, IW97], see also [Uma03])**.** *If Conjecture 15 holds, then there exists a deterministic algorithm G such that for any integer $s > 0$, $G(1^s)$ runs in $s^{O(1)}$ time and outputs a collection R of $s^{O(1)}$ strings, each of which is s-bit, such that for any circuit C of size at most s, R $(1/s)$-fools C, meaning that*

$$\left|\mathop{\mathbf{Pr}}_{x \sim \{0,1\}^s}[C(x) = 1] - \mathop{\mathbf{Pr}}_{x \sim R}[C(x) = 1]\right| \;\leq\; 1/s.$$

Theorem 17 (Conditional equivalence of Kt and rKt [Oli19])**.** *If* $\mathsf{Promise\text{-}BPE} \subseteq \mathsf{Promise\text{-}E}$ *then* $\mathsf{Kt}(x) = \Theta(\mathsf{rKt}(x))$ *for every string* $x \in \{0,1\}^*$*. In particular, the same conclusion holds under Conjecture 15.*

It is known that estimating the rKt complexity of an input string is intractable.

Theorem 18 (Hardness of estimating rKt [Oli19])**.** *Let* $\varepsilon \in (0,1)$ *and* $C \in \mathbb{N}$*. There is no randomized algorithm* A *running in time* $T(n) = O(n^{(\log n)^C})$ *such that for every large enough* n*:*

- *For every* $x \in \{0,1\}^n$ *such that* $\mathsf{rKt}(x) \leq n^{\varepsilon}$*,* $\Pr_A[A(x) = 1] \geq 2/3$*.*
- *For every* $x \in \{0,1\}^n$ *such that* $\mathsf{rKt}(x) \geq n - 10$*,* $\Pr_A[A(x) = 0] \geq 2/3$*.*

3 A Coding Theorem via Probabilistic Representations

3.1 An efficient String Isolation Lemma

The next lemma allows us to efficiently isolate a string x from a collection W of strings using a short advice string v whose length depends on the logarithm of the size of W. We follow a construction described in the proof of [CJW20, Lemma 6.1].

Lemma 19 (String Isolation Lemma)**.** *There is a deterministic algorithm* M *for which the following holds. For any set* $W \subseteq \{0,1\}^n$ *of size* ℓ*, there exists a string* $u \in \{0,1\}^{O(\log(n\cdot\ell))}$ *such that* $M(1^n, u)$ *runs in* $\mathsf{poly}(n)$ *time and outputs a Boolean circuit that computes a function* $H\colon \{0,1\}^n \to \{0,1\}^{O(\log \ell)}$ *with the following property:*

$$H(w) \neq H(w') \text{ for every distinct pair } w, w' \in W.$$

Moreover, the same guarantee is achieved by a random string u *of the same length with probability at least* 0.99*.*

Proof. Let $E_n\colon \{0,1\}^n \to \{0,1\}^{Cn}$ be the error-correcting code from Theorem 11. Moreover, let $t = c_1 \log \ell$, where c_1 is a large enough universal constant. Finally, let G_{Cn} be the expander graph from Theorem 12, where $m = Cn$.

Given a walk $\gamma = (v_1, \ldots, v_t)$ in G_{Cn}, we define the function $H_\gamma\colon \{0,1\}^n \to \{0,1\}^t$ as follows. On a string z, let $z' = E_n(z)$, and set $H_\gamma(z)_i = z'_{v_i}$, where $v_i \in [Cn]$ is given by γ and z'_j denotes the j-th bit of z'.

For distinct strings $z^1, z^2 \in \{0,1\}^n$, $E_n(z^1)$ and $E_n(z^2)$ have relative distance at least $1/10$. As a consequence, if $\boldsymbol{\gamma}$ is a length-t *random walk* on G_{Cn}, it follows from the Expander Chernoff Bound (Theorem 13) that

$$\Pr_{\boldsymbol{\gamma}}\left[H_{\boldsymbol{\gamma}}(z^1) = H_{\boldsymbol{\gamma}}(z^2)\right] \;\leq\; \Pr_{\boldsymbol{\gamma}}\left[\frac{1}{t}\cdot\sum_{i=1}^{t} 1_{[H_{\boldsymbol{\gamma}}(z^1)_i \neq H_{\boldsymbol{\gamma}}(z^2)_i]} < \frac{1}{20}\right] \;\leq\; e^{-\Omega(t)} \;<\; \frac{1}{100\ell^2},$$

where we have used that c_1 is a large enough constant in the definition of t. By a union bound over all distinct pairs of strings in W, there is some length-t random walk such that the corresponding function H satisfies the condition of the lemma.

Note that any length-t walk $\gamma = (v_1, \ldots, v_t)$ can be described by a string of length $\log(Cn) + O(t) = O(\log(n \cdot \ell))$, given that G_{Cn} is an m-vertex d-regular graph with $d = O(1)$ and $m = Cn$.

Finally, given 1^n and a description u of a length-t walk γ, it is possible to produce a circuit that computes as the function H_γ in time polynomial in n. This is because E_n, G_{Cn} and the walk encoded by u can be computed in time $\mathsf{poly}(n,t) = \mathsf{poly}(n, \log \ell) = \mathsf{poly}(n)$, where the last step uses that $\ell \leq 2^n$. This completes the proof of the lemma. □

The power of Lemma 19 comes from the fact that we don't need to know the set W, i.e., an upper bound on its size is sufficient. We remark that it is possible to achieve better parameters in Lemma 19 if we do not consider the efficiency of M. However, we need an efficient M for our applications in time-bounded Kolmogorov complexity. We also note that in our proofs we will only need that for a particular string $x \in W$ fixed in advance the value $H(x)$ is not contained in the image of H on $W \backslash \{x\}$.

We provide alternate proofs of Lemma 19 in Appendix A, where we take a closer look at the asymptotic parameters of Theorem 1.

3.2 An efficient Coding Theorem for rKt

We prove Theorem 1 in this section, restated below for convenience of the reader.

Theorem 20 (Efficient Coding Theorem for rKt)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^m)$ runs in time $T(m)$ and outputs a string $x \in \{0,1\}^*$ of length $n \leq T(m)$ with probability at least $\delta > 0$. Then*

$$\mathsf{rKt}(x) = O(\log(1/\delta) + \log(T(m)) + \log(m)),$$

where the constant behind the $O(\cdot)$ depends on $|A|$ and is independent of the remaining parameters. Moreover, given x, m, the code of A, and δ, it is possible to compute with probability ≥ 0.99 some rKt *encoding of x of at most this complexity in randomized time* $\mathsf{poly}(|A|, \log(m), |x|, \log(1/\delta))$.

Proof. Let x be a string generated with probability at least δ by a sampler $A(1^m)$. Since we only need a *lower bound* on the probability and our rKt upper bound is stated asymptotically, for representation purposes we assume without loss of generality that δ is of the form $2^{-\ell}$ for some $\ell \in \mathbb{N}$. We now show how to obtain an rKt description of x.

With the binary descriptions of m and $1/\delta$, we first run the sampler $A(1^m)$ for $t = 64 \cdot (1/\delta) \cdot \log(1/\delta)$ times to obtain a multi-set of strings S_0 of size t. Let V be the set

$$V \stackrel{\text{def}}{=} \{y \mid A(1^m) \text{ outputs } y \text{ with probability at least } \delta\}.$$

Note that $x \in V$ and that $|V| \leq 1/\delta$. Also, without loss of generality, we assume $\delta < 2/3$; otherwise the sampler $A(1^m)$ yields a desired rKt description of x.

Claim 21. *With probability at least $5/6$ (over S_0), every $y \in V$ appears in S_0 at least $\alpha = 32 \cdot \log(1/\delta)$ times.*

Proof of Claim 21. Fix a $y \in V$. Note that the expected number of times that y appears in S_0 is at least

$$\mu \stackrel{\text{def}}{=} t \cdot \delta \;=\; 64 \cdot \log(1/\delta).$$

By the Chernoff bound (Item (ii) of Theorem 10), we have

$$\mathop{\mathbf{Pr}}_{S_0}[y \text{ appears in } S_0 \text{ less than } \alpha \text{ times}] \;\leq\; \left(\frac{e^{-1/2}}{(1/2)^{(1/2)}}\right)^{64\cdot\log(1/\delta)} \;<\; \delta^6.$$

By a union bound over the strings in V, where $|V| \leq 1/\delta$, we have that the probability that there exists a $y \in V$ such that y appears in S_0 less than α times is less than $\delta^5 < 1/6$. $\square$ (Claim 21)

Claim 22. *With probability at most* $1/6$ *(over* S_0*), there exists a string* z *in* S_0 *such that* z *appears at least* $\alpha = 32 \cdot \log(1/\delta)$ *times in* S_0 *and that* $A(1^m)$ *outputs* z *with probability less than* $\delta/32$*.*

Proof of Claim 22. Let's view S_0 as an ordered multi-set $(z_1, z_2, \ldots, z_t)$. For $z_i \in S_0$, we say "z_i is bad" if that $A(1^m)$ outputs z_i with probability less than $\delta/32$ and that it appears in S_0 at least α times. Let $\mathcal{E}$ be the event that there exists some $z_i \in S_0$ such that z_i is bad. Then we have

$$\mathbf{Pr}[\mathcal{E}] \;\leq\; \sum_{i=1}^{t} \mathbf{Pr}[z_i \text{ is bad}]. \tag{1}$$

Fix an $i \in [t]$, we have

$$\begin{aligned}
\mathop{\mathbf{Pr}}_{z_1,z_2,\ldots,z_t\sim A}[z_i \text{ is bad}] \;&=\; \sum_{\substack{z\,:\,A \text{ outputs } z\\ \text{w.p. less than } \delta/32}} \mathbf{Pr}[z_i = z \text{ AND } z_i \text{ appears in } S_0 \text{ at least } \alpha \text{ times}] \\
&\leq\; \sum_{z \text{ as above}} \mathbf{Pr}[z_i \text{ appears in } S_0 \text{ at least } \alpha \text{ times} \mid z_i = z] \cdot \mathbf{Pr}[z_i = z] \\
&\leq\; \max_{z \text{ as above}} \mathbf{Pr}[z_i \text{ appears in } S_0 \text{ at least } \alpha \text{ times} \mid z_i = z] \\
&\leq\; \max_{z \text{ as above}} \mathbf{Pr}[z \text{ appears } S_0\backslash\{z_i\} \text{ at least } \alpha - 1 \text{ times}]. \qquad (2)
\end{aligned}$$

Note that if for a string z, A outputs z with probability less than $\delta/32$, then the expected number of times that z appears in the multi-set $S_0\backslash\{z_i\}$ is less than

$$\mu \stackrel{\text{def}}{=} (t-1)\cdot\delta/32 \;=\; 2\cdot\log(1/\delta) - \delta/32,$$

and hence $\alpha - 1 > 11 \cdot \mu$. Then by the Chernoff bound (Item (i) of Theorem 10), we have

$$\textit{Equation (2)} \;\leq\; \left(\frac{e^9}{10^{10}}\right)^{\log(1/\delta)} \;<\; \delta^{18}.$$

Therefore, we have

$$\textit{Equation (1)} \;\leq\; t\cdot\delta^{18} \;<\; 64\cdot(1/\delta)^2\cdot\delta^{18} \;<\; 1/6,$$

as desired. $\square$ (Claim 22)

Next, from S_0, we build a set S by removing every string in S_0 that appears less than $\alpha = 32 \cdot \log(1/\delta)$ times in S_0, and keeping only one copy for each of the remaining strings. Let W be the set

$$W \stackrel{\text{def}}{=} \{w \mid A(1^m) \text{ outputs } w \text{ with probability at least } \delta/32\}.$$

Note that $|W| \leq 32/\delta$. From Claim 21 and Claim 22, we get that with probability at least $2/3$ over the choices of S_0, we obtain a "good" set S in the sense that $V \subseteq S$ (hence $x \in S$) and $S \subseteq W$.

Consider the algorithm in the String Isolation Lemma (Lemma 19), and let $n = |x| \leq T(m)$. Let u be a string of length $O(\log(n \cdot |W|)) = O(\log(T(m)) + \log(1/\delta))$ such that the algorithm in Lemma 19 running on u produces a good hash function $H\colon \{0,1\}^n \to \{0,1\}^\tau$ for all length-n strings in W, where $\tau = O(\log(|W|)) = O(\log(1/\delta))$. That is, for every distinct $w, w' \in W \cap \{0,1\}^n$, we have $H(w) \neq H(w')$. Then for every "good" set S, H maps the strings in S of length n into different buckets. Therefore, given u, n, and the hash value for x, $H(x)$, we can recover x from a good set S.

The whole algorithm runs in randomized time $\mathrm{poly}(m, T(m), 1/\delta)$ and requires an advice of length $O(\log(1/\delta) + \log(m) + \log(n))$, which gives the desired upper bound for $\mathsf{rKt}(x)$.

For the moreover part of the theorem, first observe that to output a description of x it is not necessary to run the sampling algorithm $A(1^m)$ nor to know an upper bound on its running time. We need instead the following information: the code of A, the input parameter m, the probability lower bound δ, the length $n = |x|$, the advice string u given by Lemma 19, and the hash value $H(x)$. Crucially, the proof of Lemma 19 shows that a *random* string u (encoding a random walk) satisfies the conditions of the lemma with probability ≥ 0.99. Furthermore, the same proof shows that, given u and x, we can compute $H(x)$ in time $\mathsf{poly}(n) = \mathsf{poly}(|x|)$. Therefore, by sampling a random string u of an appropriate length that depends on n and δ, it is possible to compute a correct description of x with probability at least 0.99. The necessary information can be computed from x, the code of A, n, and δ in time $\mathsf{poly}(|A|, \log(m), |x|, \log(1/\delta))$. □

3.3 Further remarks

In this section, we make a few additional comments about the coding theorem for rKt and its applications.

Short time-bounded programs from many time-bounded programs. As a consequence of Theorem 20, by defining an appropriate sampler (see Section 4.1), it is possible to show that if a string admits *many* time-bounded programs, then it admits a *short* time-bounded program. To our knowledge, this is the first unconditional proof of a phenomenon of this form in the setting of time-bounded Kolmogorov complexity. We remark that a quantitatively tighter connection can be obtained if one could show that the dependence on δ in Theorem 20 can be improved from $O(\log(1/\delta))$ to just $\log(1/\delta)$. This is discussed in detail in Appendix A.

On the optimality of the rKt *upper bound.* A dependency of order $\log(1/\delta)$ in the rKt upper bound appearing in Theorem 20 is clearly needed, since the uniform distribution on $\{0,1\}^n$ is trivially samplable, and most n-bit strings have nearly maximum rKt complexity.

A bit more precisely, for a polynomial-time computable sampler $A(1^m)$, Theorem 20 gives the upper bound

$$\mathop{\mathbf{E}}_{x \sim A(1^m)}[\mathsf{rKt}(x)] \;\leq\; \mathop{\mathbf{E}}_{x \sim A(1^m)}[O(\log(m)) + O(\log(1/p_x))] \;=\; O\big(\log(m) + H(A(1^m))\big),$$

where p_x denotes the probability of x under $A(1^m)$, and $H(\cdot)$ is the entropy function. Since a valid rKt representation specifies a unique string, it follows from Shannon's Source Coding Theo-

rem (Theorem 14) that the *expected encoding length* achieved by Theorem 20 is also nearly optimal.

On the robustness of the definition of rKt *complexity.* We have defined $\mathsf{rKt}(x)$ with a probability threshold of $2/3$ for machines generating x. It is not hard to see that any constant threshold probability $> 1/2$ leads essentially to the same definition, using standard amplification. On the other hand, it is not hard to prove from Theorem 20 that, for every string $x \in \{0,1\}^*$,

$$\mathsf{rKt}(x) \;=\; O\Big(\min_{\delta\in(0,1)} \mathsf{rKt}_\delta(x) + \log(1/\delta)\Big).$$

The proof uses that a machine witnessing the bound $\mathsf{rKt}_\delta(x)$ for x can be seen as a *sampler*, and that for every x we have $\mathsf{rKt}(x) \geq \log|x|$. (A weaker result was observed in [Oli19, Footnote 6].)

A coding theorem for rKt *with advice.* It is possible to consider a sampler $A(1^m, a_m)$ in the statement of Theorem 20, where A receives in addition to 1^m an advice string $a_m \in \{0,1\}^*$. In this case, our techniques easily extend to show an upper bound on the *conditional* rKt complexity $\mathsf{rKt}(x \,|\, a_m)$ of x given the string a_m, where $\mathsf{rKt}(x \,|\, a_m)$ is defined in the natural way.

In Appendix B, we optimise the space of the encoding and decoding algorithms implicit in the proof of Theorem 20.

4 Applications

4.1 Equivalence between samplability and succinct probabilistic descriptions

It will be useful in the proof of some results to introduce the following sampler.

Definition of $U(1^m)$:

1. Given 1^m, U samples an integer $\ell \sim [m]$. It then samples a uniformly random string z of length ℓ, and an independent uniformly random string r of length 2^m.

2. U interprets z as the code of a randomized machine M_z, simulates M_z on the empty input string with randomness r for 2^m steps, and outputs the string y that is left on the output tape of M_z after this simulation.

This sampler satisfies the following properties.

Claim 23. *On every input 1^m and for every choice of its randomness, $U(1^m)$ runs in time at most $2^{O(m)}$.*

Proof. This is immediate from the definition. □

Claim 24. *Suppose $x \in \{0,1\}^*$ is a string such that $\mathsf{rKt}(x) \leq k$. Then the probability of x under $U(1^k)$ is at least $(1/k)\cdot 2^{-k}\cdot(2/3)$.*

Proof. Since $\mathsf{rKt}(x) \leq k$, there is a randomized machine M running in time at most t such that

$$\Pr[\boldsymbol{M}_{\leq \boldsymbol{t}} = x] \;\geq\; 2/3 \quad \text{and} \quad |M| + \log t \;\leq\; k.$$

Let $\ell = |M| \leq k$, and note that $t \leq 2^k$. It is clear that x is output by $U(1^k)$ with probability at least $(1/\ell) \cdot 2^{-\ell} \cdot (2/3) \geq (1/k) \cdot 2^{-k} \cdot (2/3)$. □

There are some additional properties of U that make it a "universal" time-bounded sampler. We explain two of these properties next.

First, we note that if a function f is hard under some (possibly unknown) polynomial-time samplable distribution $\mathcal{D}_n$, in the sense that any algorithm from a class $\mathcal{C}$ fails to compute f with probability at least $1/\mathsf{poly}(n)$ under $\mathcal{D}_n$, then f is also average-case hard for $\mathcal{C}$ under the *explicit* polynomial-time samplable distribution $\mathcal{V}_n \stackrel{\text{def}}{=} U(1^{\alpha \cdot \log n})$, where α is a constant that depends on $\mathcal{D} = \{\mathcal{D}_n\}_{n \in \mathbb{N}}$.

Lemma 25 ("Hardness Universality" of $U(1^{O(\log n)})$ among efficiently samplable distributions)**.** *Let $f = \{f_n\}_{n \geq 1}$ be a sequence of functions $f_n \colon \{0,1\}^n \to \{0,1\}$, and assume that there is a polynomial $p(n)$ and a polynomial-time samplable ensemble $\mathcal{D} = \{\mathcal{D}_n\}_{n \geq 1}$, where each $\mathcal{D}_n$ is supported over $\{0,1\}^n$, such that for every function $g_n \in \mathcal{C}$, $\Pr_{x \sim \mathcal{D}_n}[f_n(x) \neq g_n(x)] \geq 1/p(n)$. Then there is a constant $\alpha \in \mathbb{N}$ and a polynomial $q(n)$ such that for each large enough n and for every function $g_n \in \mathcal{C}$, $\Pr_{x \sim \mathcal{V}_n}[f_n(x) \neq g_n(x)] \geq 1/q(n)$, where $\mathcal{V}_n \stackrel{\text{def}}{=} U(1^{\alpha \cdot \log n})$.*

Proof. Suppose that $\mathcal{D} = \{\mathcal{D}_n\}_{n \geq 1}$ is samplable by a randomized algorithm $S(1^n)$ running in time $\leq c_1 n^{c_1}$. Then there is a sequence $\{M^n\}_{n \geq 1}$ of randomized machines M^n running in time at most $\leq c_2 n^{c_2}$ when given the empty string such that $M^n(\varepsilon)$ and $S(1^n)$ sample the same distribution and $|M^n| \leq \beta \cdot \log n$, for constants β and c_2 that depend on S but are independent of n. We set $\alpha \stackrel{\text{def}}{=} \max\{c_2 + 1, \beta\}$ and $q(n) \stackrel{\text{def}}{=} p(n) \cdot 2^{|M^n|} \cdot (\alpha \cdot \log n)$. Note that $q(n)$ is upper bounded by some polynomial in n.

Assume that n is sufficiently large, and fix a function $g_n \in \mathcal{C}$. Then, recalling the definitions of $\mathcal{V}_n \stackrel{\text{def}}{=} U(1^{\alpha \cdot \log n})$ and of the sampler $U(\cdot)$, let $\mathcal{E}$ be the event that $(\ell = |M^n|) \wedge (M_z = M^n)$. We have

$$\begin{aligned}
\Pr_{y \sim \mathcal{V}_n}[f_n(y) \neq g_n(y)] &\geq \Pr_{(\ell,z,y) \sim U(1^{\alpha \cdot \log n})}[f_n(y) \neq g_n(y) \mid \mathcal{E}] \cdot \Pr_{(\ell,z,y) \sim U(1^{\alpha \cdot \log n})}[\mathcal{E}] \\
&\geq \Pr_{x \sim \mathcal{D}_n}[f_n(x) \neq g_n(x)] \cdot \frac{1}{\alpha \cdot \log n} \cdot 2^{-|M^n|} \\
&\geq \frac{1}{p(n) \cdot (\alpha \cdot \log n) \cdot 2^{|M^n|}} = \frac{1}{q(n)}.
\end{aligned}$$

This completes the proof of the lemma. □

A similar argument establishes the following lemma.

Lemma 26 ("Weight Universality" of $U(1^{O(\log n)})$ among efficiently samplable distributions)**.** *If a string $x \in \{0,1\}^n$ is assigned weight $\delta \geq 1/\mathsf{poly}(n)$ under some efficiently samplable distribution $A(1^n)$, then the weight of x under $U(1^{O(\log n)})$ is at least $1/\mathsf{poly}(n)$.*

Analogues of Lemmas 25 and 26 with optimal parameters are known in the context of Kolmogorov complexity (see e.g. [LV08]). We refer to [AF09] for related results in the time-bounded setting, with some of them requiring computational assumptions.

We state next an immediate consequence of the coding theorem for rKt and the existence of universal time-bounded samplers. For concreteness, we focus on the generation of prime numbers in the context of Problem 5.

Theorem 27 (Equivalence between fast samplability and short time-bounded descriptions)**.** *The following statements are equivalent:*

(*i*) Sampling Algorithm. *For every $\varepsilon > 0$, there is a randomized algorithm $A(1^n)$ sampling strings in $\{0,1\}^*$ that runs in time $T(n) = O(2^{\varepsilon n})$ and for which the following holds. For every large n, there is an n-bit prime q_n such that $\Pr[A(1^n)$ outputs $q_n] \geq 2^{-\varepsilon n}$.*

(*ii*) Short Descriptions. *Let $\delta > 0$ be an arbitrary constant. For every large n, there is an n-bit prime p_n such that $\mathsf{rKt}(p_n) \leq \delta n$.*

Proof. That samplability as in the first item leads to short descriptions follows immediately from Theorem 20 by taking $\varepsilon > 0$ small enough as a function of the given $\delta > 0$. For the other direction, for a given $\varepsilon > 0$, we consider the sampler $A(1^n) \stackrel{\text{def}}{=} U(1^m)$ with $m = \varepsilon' n$, where $\varepsilon' = \varepsilon/C$ for a large constant C. By Claim 23, $A(1^n)$ runs in time $O(2^{\varepsilon n})$. Under the assumption that (*ii*) holds, Claim 24 guarantees that for large n there is an n-bit prime q_n that is output by $A(1^n)$ with probability at least $2^{-\varepsilon n}$. □

This should be contrasted with the equivalence between the existence of primes of bounded Kt complexity and fast deterministic generation of primes. As mentioned in Section 1.1.2, the equivalence in Theorem 27 also holds in a broader sense.

4.2 Instance-based search-to-decision reduction for rKt and its consequences

In this section, we show that there is a *uniform* polynomial-time "approximate" search-to-decision reduction for rKt. More precisely, we show that given a *linear* approximation of the rKt complexity of the input string, it is possible to output a valid rKt representation that is optimal up to a *constant factor*.

We will need the following definition.

Definition 28. *Let $\gamma\colon \mathbb{N} \to \mathbb{R}$. We say that a function $\mathcal{O}\colon \{0,1\}^* \to \mathbb{N}$ γ-approximates rKt complexity if for every string $x \in \{0,1\}^*$,*

$$\frac{\mathsf{rKt}(x)}{\gamma(|x|)} \;\leq\; \mathcal{O}(x) \;\leq\; \gamma(|x|) \cdot \mathsf{rKt}(x).$$

If this holds for a constant $\gamma \in \mathbb{N}$, we say that $\mathcal{O}$ linearly approximates rKt.

Theorem 29 (An instance-based search-to-decision reduction for rKt)**.** *Let $\mathcal{O}$ be a function that linearly approximates rKt complexity. There is a randomized polynomial-time algorithm with access to $\mathcal{O}$ that, when given an input string x, outputs with probability ≥ 0.99 a valid rKt representation of x of complexity $O(\mathsf{rKt}(x))$. Furthermore, this algorithm makes a single query q to $\mathcal{O}$, where $q = x$.*

Proof. Given an input $x \in \{0,1\}^*$, let $\widetilde{k} \stackrel{\text{def}}{=} \mathcal{O}(x)$ be the estimate provided by the oracle, and recall that $\mathsf{rKt}(x)/C \leq \widetilde{k} \leq C \cdot \mathsf{rKt}(x)$ for a universal constant C. We proceed as in the proof of Theorem 30, but this time we will invoke the "moreover" part of Theorem 20 with sampler $A = U(1^{C \cdot \widetilde{k}})$. By Claim 24, since $\mathsf{rKt}(x) \leq C \cdot \widetilde{k}$, we get that $U(1^{C \cdot \widetilde{k}})$ outputs x with probability at least $\delta \stackrel{\text{def}}{=} (1/(C \cdot \widetilde{k})) \cdot 2^{-C \cdot \widetilde{k}} \cdot (2/3)$. In addition, the running time of this sampler is bounded by $2^{O(C \cdot \widetilde{k})}$. It then follows from Theorem 20 that it is possible to efficiently compute with probability at least 0.99 a valid rKt representation of x of complexity $O(C \cdot \widetilde{k}) = O(C^2 \cdot \mathsf{rKt}(x)) = O(\mathsf{rKt}(x))$. □

As explained in Section 1, an interesting aspect of this reduction is that it works on an *input by input* basis. In other words, if we are able to linearly approximate the rKt complexity of the input string, then we can compute a near-optimal representation for the same string.

We note that a reduction that works on an input by input basis is easy to come up with in the setting of (time-unbounded) Kolmogorov complexity: from an upper bound on $K(x)$, it is possible to compute a string that represents x of complexity at most $K(x)$ simply by running in parallel all machines of at most this description length. Theorem 29 can be interpreted as an analogue result in the more challenging setting of time-bounded Kolmogorov complexity.

The next result should be contrasted with the unconditional lower bound of [Oli19] for estimating the rKt complexity of an input string x.

Theorem 30 (Efficient generation of rKt descriptions)**.** *There is a randomized polynomial-time algorithm E such that on any input string $x \in \{0,1\}^n$, given as advice a string $\alpha = \alpha(x)$ of length $\leq \log\log n + O(1)$, $E(x,\alpha)$ outputs with probability ≥ 0.99 an rKt description of x of complexity $O(\mathsf{rKt}(x))$.*

Proof. Algorithm E computes on $x \in \{0,1\}^n$ as follows. It expects $\alpha \in \{0,1\}^{\log\log n + O(1)}$ to encode a tight approximation $\tilde{k}$ to the value $k \stackrel{\text{def}}{=} \mathsf{rKt}(x) \in [\gamma \cdot n]$, where $\gamma \in \mathbb{N}$ is a universal constant, and $\gamma \cdot n$ is an upper bound on $\max\{\mathsf{rKt}(x) \mid x \in \{0,1\}^n\}$. Since we are only aiming for a constant factor approximation of $\mathsf{rKt}(x)$, we use α to encode the smallest integer β such that $2^\beta \geq \mathsf{rKt}(x)$. Since $\beta \leq \log n + O_\gamma(1)$, α can be encoded with just $\log\log n + O(1)$ bits, and a value $\tilde{k}$ such that $k \leq \tilde{k} \leq 2k$ can be obtained from α. $E(x,\alpha)$ considers the universal sampler $U(1^{\tilde{k}})$ as the algorithm $A(1^{\tilde{k}})$ in Theorem 20. Furthermore, it sets $\delta \stackrel{\text{def}}{=} (1/\tilde{k}) \cdot 2^{-\tilde{k}} \cdot (2/3)$. By Claims 23 and 24, $A(1^{\tilde{k}}) \equiv U(1^{\tilde{k}})$ outputs x with probability at least δ and in time at most $T = 2^{O(\tilde{k})}$. As a consequence, from the "moreover" part of Theorem 20 it follows that $E(x,\alpha)$ can compute with probability at least 0.99 a valid rKt representation of x of complexity $O(\log(\tilde{k}) + \log(T) + \log(1/\delta)) = O(\tilde{k}) = O(\mathsf{rKt}(x))$ in time $\mathsf{poly}(|U|, \log(\tilde{k}), n, \log(1/\delta)) = \mathsf{poly}(n)$. □

The next corollary is immediate from Theorem 29.

Corollary 31 (Time-bounded short lists with short programs in short time)**.** *Given an arbitrary string x of length n, it is possible to compute with high probability and in polynomial time a collection of at most $d = \log(n) + O(1)$ strings $w_1, \ldots, w_d$ such that at least one of these strings is a valid rKt encoding of x of complexity $O(\mathsf{rKt}(x))$.*

Proof. We can enumerate all values of $k \in [O(n)]$ that are a power of 2, and run the instance-based search-to-decision reduction from Theorem 29 on x using k as the query answer. One of such k will be a linear approximation of $\mathsf{rKt}(x)$, and the output of the search-to-decision reduction for this k will be a rKt description of x with complexity $O(\mathsf{rKt}(x))$. □

4.3 Implications for Levin's Kt complexity

In this section, we investigate applications of rKt to Levin's Kt complexity [Lev84]. We start with the following lemma, which provides a conditional but general way of translating results for rKt to Kt.

Lemma 32 (Conditional rKt-to-Kt Conversion Lemma)**.** *Assume that Conjecture 15 holds, and let G be the algorithm from Theorem 16. For every string $x \in \{0,1\}^*$, given any valid* rKt *representation w of x and an upper bound T on the running time of the corresponding randomized machine M_w, we can compute in deterministic time* $\mathrm{poly}(|w|, \log(T))$ *a valid* Kt *representation of x of complexity $O(|w| + \log(T))$.*

Proof. Consider the following computation: Let $R \subseteq \{0,1\}^s$ be the output of the algorithm $G(1^s)$, where $s = \mathrm{poly}(T)$. Run M_w over the empty string using as its source of randomness each string in R and output the most common string output by M_w.

We claim that this deterministic computation produces x. Let C be a (multi-output) deterministic circuit on at most T input bits that interprets its input as a random string z and outputs M_w on the empty string using randomness z. Note that C has size at most $\mathrm{poly}(T)$, and that it outputs x with probability at least $2/3$. Next consider the (single-output) Boolean circuit C' obtained by adding a circuit on top of C that checks if the output of C is x, i.e., C' outputs 1 if and only if this is the case. Note that C' also has size $\mathrm{poly}(T) \leq s$, since $|x| \leq T$. Since the set R $(1/10)$-fools C', on more than a $1/2$ fraction of the strings in R, C' outputs 1. The only way for this to happen is if C (and also M_z) outputs x on every such string in R. Therefore, the computation described above must produce x.

It is not hard to see that, given the string w, the code of G, and the upper bound T, we can produce the code of a deterministic machine that performs the computation described above. We need to hardcode $\log(T)$ in this description, which extends the description length from $|w|$ to $O(|w| + \log(T))$, where the constant in $O(\cdot)$ depends on $|G|$ as well. Since the computation runs in time $\mathrm{poly}(T)$, the overall Kt complexity is bounded by $O(|w| + \log(T))$. □

Lemma 32 provides a natural example of a deterministic algorithm manipulating probabilistic data representations.

Theorem 33 (Conditional Effective Coding Theorem for Kt)**.** *Assume that Conjecture 15 holds, and let G be the algorithm from Theorem 16. If there is a randomized algorithm A for sampling strings such that $A(1^m)$ runs in time $T(m)$ and outputs a string $x \in \{0,1\}^*$ of length $n \leq T(m)$ with probability at least $\delta > 0$, then*

$$\mathsf{Kt}(x) = O(\log(1/\delta) + \log(T(m)) + \log(m)),$$

where the constant behind the $O(\cdot)$ depends on $|A|$ and $|G|$, and is independent of the remaining parameters. Moreover, given x, m, $T(m)$, δ, and the codes of A and G, it is possible to compute with probability ≥ 0.99 some Kt *encoding of x of at most this complexity in randomized* $\mathrm{poly}(n)$ *time.*

Proof. We can assume without loss of generality that $\max\{|A|, \log(m), \log(T(m)), \log(1/\delta)\} \leq n$, since otherwise outputting a trivial Kt representation for x of complexity $O(n)$ is sufficient. The result is then immediate from the Coding Theorem for rKt (Theorem 20) and Lemma 32. □

Given the result above, it is not hard to show a *conditional* instance-based search-to-decision reduction for Kt. However, using ideas from the proof the coding theorem for rKt, we can also show an *unconditional* instance-based search-to-decision reduction for Kt.

Theorem 34 (An instance-based search-to-decision reduction for Kt)**.** *Let $\mathcal{O}$ be a function that linearly approximates Kt complexity. There is a randomized polynomial-time algorithm with access to $\mathcal{O}$ that, when given an input string x, outputs with probability ≥ 0.99 a valid Kt representation of x of complexity $O(\mathsf{Kt}(x))$. Furthermore, this algorithm makes a single query q to $\mathcal{O}$, where $q = x$.*

Proof. We are given an input $x \in \{0,1\}^*$ where $|x| = n$, and $\mathsf{Kt}(x)/C \leq \widetilde{k} \leq C \cdot \mathsf{Kt}(x)$ for a universal constant C, where $\widetilde{k}$ is obtained via a single query to $\mathcal{O}$ on x. Our encoding algorithm is as follows:

1. Pick $u \in \{0,1\}^d$ uniformly at random, where $d = O\Big(\log\Big(n \cdot 2 \cdot 2^{C \cdot \widetilde{k}}\Big)\Big) = O(\widetilde{k} + \log(n))$.
2. Invoke the deterministic algorithm $M(1^n, u)$ from Lemma 19 to obtain a hash function $H\colon \{0,1\}^n \to \{0,1\}^{O(\widetilde{k})}$, and compute $z = H(x)$.
3. Output $(n, \widetilde{k}, u, z)$.

It is easy to verify that the output of the above procedure has length

$$O(\log(\widetilde{k}) + \log(n)) = O(\mathsf{Kt}(x) + \log(n)) = O(\mathsf{Kt}(x)).$$

Next, we claim that with probability at least 0.99 over the choice of u, the tuple generated in Step 3 can be easily converted into a valid Kt representation of x. (Note that the randomness is only over the encoding algorithm that generates the Kt representation, which provides a deterministic description.) To decode x from this information, we first run $M(1^n, u)$ from Lemma 19 to recover the hash function H. We then enumerate each deterministic machine of description length at most $C \cdot \widetilde{k}$ and simulate it for at most $2^{C \cdot \widetilde{k}}$ steps. For each output y of these machines that is in $\{0,1\}^n$, we compute $H(y)$ and output the first y such that $H(y) = z$. Note that there are at most $\ell = 2 \cdot 2^{C \cdot \widetilde{k}}$ distinct machines of length at most $C \cdot \widetilde{k}$, and each machine produces at most one output string. By the fact that $\mathsf{Kt}(x) \leq C \cdot \widetilde{k}$, at least one of them will output x. Also, by Lemma 19, for at least a fraction of .99 of the u's, the hash function H obtained from u isolates every n-bit string in the set of outputs of these ℓ machines, and x is the only string such that $H(x) = z$. Therefore, for any such "good" u, the string x can be decoded correctly from n, $\widetilde{k}$, u, and z. It is easy to see that the running time of the decoding algorithm is $\mathrm{poly}(n) \cdot 2^{O(\widetilde{k})}$, so the Kt complexity of the resulting representation for x is $O(\mathsf{Kt}(x))$. □

Note that Theorem 29 (search-to-decision reduction for rKt) and Theorem 34 (search-to-decision reduction for Kt) complete the proof of Theorem 2 from Section 1.

It is unclear to us how to use the hardness hypothesis in Conjecture 15 to derandomize the reduction. Indeed, even under stronger assumptions, we don't see how the reductions for rKt and Kt can be derandomized in polynomial time. We suspect that the use of randomness might be inherent in any efficient algorithm that produces succinct descriptions with respect to Kt complexity.

4.4 Time hierarchies for sampling distributions

In this section, we observe that a derandomization hypothesis for *decision problems* implies a strong time hierarchy theorem for *sampling distributions*.

Theorem 35 (A strong time hierarchy theorem for sampling distributions)**.** *Under the assumption that* Promise-BPE $\subseteq$ Promise-E*, there is a constant $\zeta > 0$ for which the following holds. Let $n^{1/\zeta} \leq T(n) \leq 2^n$ be any constructive time bound. There is an ensemble $\{\mathcal{D}_n\}_{n\geq 1}$ of distributions $\mathcal{D}_n$ supported over $\{0,1\}^n$ such that:*

(i) Each distribution $\mathcal{D}_n$ is supported over a single string z_n.

(ii) There is a deterministic algorithm $A(1^n)$ that samples $\mathcal{D}_n$ and runs in time $O(T(n))$.

(iii) For each randomized algorithm $B(1^n)$ that runs in time $O(T(n)^\zeta)$ and for every large enough n, the statistical distance of $\mathcal{D}_n$ and $B(1^n)$ is at least $1 - 1/T(n)^\zeta$.

Proof. The assumption implies via Theorem 17 that there is a universal constant C such that

$$\mathsf{Kt}(x) \;\leq\; C \cdot \mathsf{rKt}(x) \tag{3}$$

for all $x \in \{0,1\}^*$. Consider the constructive time bound $T(n)$, and let $\lambda \stackrel{\text{def}}{=} \varepsilon \cdot \log(T(n))$, where $\varepsilon > 0$ is a small constant independent of the remaining parameters in the statement of the theorem. By exhaustive search, we can enumerate in time $O(T(n))$ all strings in $\{0,1\}^n$ of Kt complexity at most λ. Note that this does not exhaust the set of all n-bit strings, since $\lambda < n$ under our assumption that $T(n) \leq 2^n$. Let z_n be the lexicographic first n-bit string that is not in this list, and let $\mathcal{D}_n$ be the distribution supported on z_n. Note that Items (i) and (ii) hold due to our construction.

In order to prove Item (iii), let $B(1^n)$ be a randomized algorithm running in time $O(T(n)^\zeta)$, and assume that n is large enough. Finally, toward a contradiction, suppose $|\mathcal{D}_n - B(1^n)| \leq 1 - 1/T(n)^\zeta$. From the definition of statistical distance,

$$|\mathcal{D}_n(z_n) - (B(1^n))(z_n)| \;=\; 1 - (B(1^n))(z_n) \;\leq\; 1 - 1/T(n)^\zeta,$$

i.e., $B(1^n)$ outputs z_n with probability $\geq 1/T(n)^\zeta$. By Theorem 20,

$$\mathsf{rKt}(z_n) \;\leq\; C' \cdot (\log n + \zeta \cdot \log(T(n))),$$

where C' is a universal constant. Finally, due to Equation (3) and the definitions of z_n and λ,

$$\varepsilon \cdot \log(T(n)) \;\leq\; \mathsf{Kt}(z_n) \;\leq\; C \cdot C' \cdot (\log n + \zeta \cdot \log(T(n))).$$

Since $T(n) \geq n^{1/\zeta}$, the last equation is contradictory if ζ is small enough compared to the universal constants C, C', and ε. □

Acknowledgements

We are grateful to Lijie Chen for a suggestion that allowed us to improve the list size in Corollary 3. We also thank Rahul Santhanam and Rahul Ilango for discussions on search-to-decision reductions, and James Maynard for an email exchange on number-theoretic techniques and the problem of generating primes. Igor would like to thank Michal Koucký for asking a question about the probability threshold in the definition of rKt during the Dagstuhl Workshop "Computational Complexity of Discrete Problems" (2019), which we address in Section 3.3. This work received support from the Royal Society University Research Fellowship URF\R1\191059.

References

[Aar14] Scott Aaronson. The equivalence of sampling and searching. *Theory Comput. Syst.*, 55(2):281–298, 2014.

[ABK+06] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006.

[AF09] Luis Filipe Coelho Antunes and Lance Fortnow. Worst-case running times for average-case algorithms. In *Conference on Computational Complexity* (CCC), pages 298–303, 2009.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.

[All92] Eric Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In *Kolmogorov complexity and computational complexity*, pages 4–22. Springer, 1992.

[All01] Eric Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *International Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), pages 1–15. Springer, 2001.

[All17] Eric Allender. The complexity of complexity. In *Computability and Complexity*, pages 79–94. Springer, 2017.

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

[BLvM05] Harry Buhrman, Troy Lee, and Dieter van Melkebeek. Language compression and pseudorandom generators. *Comput. Complex.*, 14(3):228–255, 2005.

[BMVZ18] Bruno Bauwens, Anton Makhlin, Nikolai K. Vereshchagin, and Marius Zimand. Short lists with short programs in short time. *Comput. Complex.*, 27(1):31–61, 2018.

[BZ14] Bruno Bauwens and Marius Zimand. Linear list-approximation for short programs (or the power of a few random bits). In *Conference on Computational Complexity* (CCC), pages 241–247, 2014.

[Che10] Mahdi Cheraghchi. *Applications of Derandomization Theory in Coding.* PhD thesis, Swiss Federal Institute of Technology, 2010.

[CIKK16] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Conference on Computational Complexity* (CCC), pages 10:1–10:24, 2016.

[CJW19] Lijie Chen, Ce Jin, and R. Ryan Williams. Hardness magnification for all sparse NP languages. In *Symposium on Foundations of Computer Science* (FOCS), pages 1240–1255, 2019.

[CJW20] Lijie Chen, Ce Jin, and R. Ryan Williams. Sharp threshold results for computational complexity. In *Symposium on Theory of Computing* (STOC), 2020.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2006.

[For04] Lance Fortnow. Kolmogorov complexity and computational complexity. *Complexity of Computations and Proofs. Quaderni di Matematica*, 13, 2004.

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, 1981.

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Symposium on Theory of Computing* (STOC), pages 812–821, 2002.

[Gil98] David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4):1203–1220, 1998.

[GS91] Andrew V. Goldberg and Michael Sipser. Compression and ranking. *SIAM J. Comput.*, 20(3):524–536, 1991.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[Hir18] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *Symposium on Foundations of Computer Science* (FOCS), pages 247–258, 2018.

[Hir20] Shuichi Hirahara. Non-disjoint promise problems from meta-computational view of pseudorandom generator constructions. In *Conference on Computational Complexity* (CCC), 2020.

[Ila20] Rahul Ilango. Connecting Perebor conjectures: Towards a search to decision reduction for minimizing formulas. In *Computational Complexity Conference* (CCC), 2020.

[ILO20] Rahul Ilango, Bruno Loff, and Igor C. Oliveira. NP-hardness of circuit minimization for multi-output functions. In *Computational Complexity Conference* (CCC), 2020.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Symposium on Theory of Computing* (STOC), pages 220–229. ACM, 1997.

[Lee06] Troy Lee. *Kolmogorov complexity and formula lower bounds*. PhD thesis, University of Amsterdam, 2006.

[Lev74] Leonid A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

[Lev84] Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984.

[LM93] Luc Longpré and Sarah Mocas. Symmetry of information and one-way functions. *Inf. Process. Lett.*, 46(2):95–100, 1993.

[LP20] Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. *IACR Cryptol. ePrint Arch.*, 2020:423, 2020.

[LV92] Ming Li and Paul M. B. Vitányi. Average case complexity under the universal distribution equals worst-case complexity. *Inf. Process. Lett.*, 42(3):145–149, 1992.

[LV08] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in Computer Science. Springer, 2008.

[LW95] Luc Longpré and Osamu Watanabe. On symmetry of information and polynomial time invertibility. *Inf. Comput.*, 121(1):14–22, 1995.

[May19] James Maynard. The twin prime conjecture. *Japanese Journal of Mathematics*, 14:175–206, 2019.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[Oli19] Igor C. Oliveira. Randomness and intractability in Kolmogorov complexity. In *International Colloquium on Automata, Languages, and Programming* (ICALP), pages 32:1–32:14, 2019.

[OPS19] Igor C. Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In *Computational Complexity Conference* (CCC), pages 27:1–27:29, 2019.

[OS17] Igor C. Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. In *Symposium on Theory of Computing* (STOC), pages 665–677, 2017.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

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

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inf. Theory*, 42(6):1723–1731, 1996.

[SUV17] Alexander Shen, Vladimir A. Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*. American Mathematical Society, 2017.

[TCH12] Terence Tao, Ernest Croot, III, and Harald Helfgott. Deterministic methods to find primes. *Math. Comp.*, 81(278):1233–1246, 2012.

[Ten15] Gérald Tenenbaum. *Introduction to analytic and probabilistic number theory*, volume 163. American Mathematical Society, 2015.

[TVZ05] Luca Trevisan, Salil P. Vadhan, and David Zuckerman. Compression of samplable sources. *Comput. Complex.*, 14(3):186–227, 2005.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[Val19] Gregory Valiant. Lecture Notes for CS265/CME309: Randomized Algorithms and Probabilistic Analysis (Lecture 6), 2019.

[Wat14] Thomas Watson. Time hierarchies for sampling distributions. *SIAM J. Comput.*, 43(5):1709–1727, 2014.

A On the Parameters of the Coding Theorem for rKt

The contents of this section are organized as follows.

Appendix A.1. We compare the parameters of the coding theorem for rKt with the analogous result in Kolmogorov complexity.

Appendix A.2. We describe two alternate proofs of the String Isolation Lemma (Lemma 19) which provide a small constant in the asymptotic bounds.

Appendix A.3. We explain Levin's conditional result showing that simultaneously achieving *efficient* encoding *and* decoding is impossible, even with probabilistic descriptions. This has implications to the discussion on optimality appearing in the preceding sections.

As a consequence of the results from this section, the constant in front of $\log(1/\delta)$ in the rKt upper bound provided by Theorem 1 is at most 5.

A.1 An optimal coding theorem for rKt?

As mentioned in Section 1, a fundamental result in the study of Kolmogorov complexity is the following coding theorem.

Theorem 36 (Coding Theorem in Kolmogorov Complexity [Lev74])**.** *Let $\mathcal{D}$ be an enumerable distribution. Then for every x,*

$$\mathsf{K}(x) \leq \log(1/\delta) + C_A,$$

where $\delta = \mathcal{D}(x)$ and $C_{\mathcal{D}}$ is a constant that depends only on the distribution $\mathcal{D}$.

Among many interesting implications of Theorem 36 and its variants in complexity theory (see e.g. [Lee06]), we would like to bring to the attention of the reader the following results:

- Li and Vitányi [LV92] used it to show that there is a probability distribution $\mathcal{D}_{\mathsf{universal}}$ over $\{0,1\}^*$ such that if a language L is hard in the *worst case* then it is hard in the *average case* with respect to $\mathcal{D}_{\mathsf{universal}}$

- Under a strong derandomization assumption, Fortnow and Antunes [AF09] implicitly proved a coding theorem for P-samplable distributions in the setting of time-bounded Kolmogorov complexity, and used it to characterize the worst-case running time of languages that are in average polynomial-time over all P-samplable distributions.

- Aaronson [Aar14] employed Theorem 36 to show a beautiful equivalence between sampling and search problems for a given computational model $\mathcal{C}$.

Given these consequences of Theorem 36, it is natural to ask whether we can use the unconditional coding theorem for rKt to show analogous results in the setting of time-bounded Kolmogorov complexity. It turns out that all the aforementioned applications also crucially rely on the fact that the coding theorem has an optimal dependence on $\log(1/\delta)$, in the sense that the constant before the $\log(1/\delta)$ term is 1. In contrast with Theorem 36, our coding theorem for rKt (Theorem 1) has a fixed but arbitrary constant factor in the $\log(1/\delta)$ term. This motivates the investigation of optimized coding theorems for rKt with a better dependence on the probability parameter δ.

In order to be very precise about the rKt complexity of a string x, it is helpful to consider a definition where one fixes an efficient universal Turing machine U that specifies the encoding length of a machine M, and to distinguish the input corresponding to M from auxiliary information a that is used to produce x. One them redefines rKt in the natural way by minimizing over $|M|+|a|+\log t$, where $U(M,a)$ outputs x with probability at least $2/3$ when it simulates $M(a)$ for t steps.

Problem 37. *Establish a coding theorem for* rKt *with optimal dependence on* $\log(1/\delta)$. *In particular, suppose there is a randomized algorithm A for sampling strings such that $A(1^n)$ runs in time $T(n)$ and outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Is it the case that*

$$\mathsf{rKt}(x) \;\leq\; 2.01 \cdot \log(1/\delta) + O(\log T(n)) \;?$$

(*In order to obtain even more precise bounds, one might need to decouple description length and running time in the definition of* rKt.)

From the proof of our coding theorem for rKt (Theorem 1), one thing that contributes to the constant factor in the $\log(1/\delta)$ term is the use of the String Isolation Lemma, where we need to include in the rKt description the encoding of a hash function and a hash value. Using Lemma 19, this requires $C \cdot (\log(1/\delta) + \log(n))$ bits, for some arbitrary constant C. Therefore, a natural attempt toward an improved coding theorem would be to prove a string isolation lemma with better parameters.

In the next subsection, we provide different proofs of the lemma which imply that the constant in front of $\log(1/\delta)$ in Theorem 1 is small, and in particular ≤ 10. We then discuss some difficulties on getting an optimal coding theorem for rKt using our techniques.

A.2 Alternate proofs of the string isolation lemma with better parameters

We describe two alternate proofs of the String Isolation Lemma (Lemma 19). While we can use Lemma 38 or Lemma 45 below to improve the description length in terms of the constant factor in the $\log(1/\delta)$ term, in order to reason about rKt complexity, we also need to consider the running time of the decoding procedure for such a description. In our proof of Theorem 1, we need to run the sampler at least $\Omega(1/\delta)$ times to observe x, so the running time is at least $\Omega(1/\delta)$. This adds another $\log(1/\delta)$ to our upper bound on the rKt complexity of x.

We are not aware of an off-the-shelf explicit construction that would allow us to obtain the parameters mentioned in Problem 37. However, as far as we know, a coding theorem for rKt with these parameters might be true and within reach of existing techniques.

A.2.1 String isolation lemma via modular arithmetic

Our first proof uses hashing that is based on modular arithmetic and prime numbers. We will identify $\{0,1\}^n$ with $\{0,1,\ldots,2^n-1\}$ in the natural way.

Lemma 38 (String Isolation Lemma via Modulo Primes)**.** *There is a randomized algorithm M for which the following holds. For any set $W \subseteq \{0,1\}^n$ of size (at most) K, and $x \in W$, with probability at least 0.99, $M(1^n, K)$ runs in time $\mathrm{poly}(n)$ and outputs a prime p with the following properties:*

- *$x \not\equiv y \bmod p$ for every $y \in W$ such that $y \neq x$.*
- *$p \leq O(K \cdot n^2)$.*

Proof. We follow a similar argument that appears in [BFL01, Lemma 3.1]. Fix any $x \in W$, and note that by the Chinese Remainder Theorem, for every $y \in W$ such that $y \neq x$, there can be at most n primes for which $x \equiv y \bmod p$. Therefore, there are at most $(K-1)\cdot n$ many primes such that $x \equiv y \bmod p$ for *some* $y \in W$ such that $y \neq x$. Let's call such primes *bad* for x.

Consider the set of all positive integers with magnitude at most $m \stackrel{\text{def}}{=} 1000 \cdot K \cdot n^2$. By the prime number theorem, there are at least $m/\log(m) > 200 \cdot (K-1) \cdot n$ primes in this set, and hence less than a 0.005 fraction of these primes are bad.

Consider the following procedure M for outputting a good prime:

Repeats $t \stackrel{\text{def}}{=} 6 \cdot \log(m)$ times:

1. Randomly pick a positive integer p with magnitude at most m.
2. Check if p is prime. If so, output p and halt.

Then, the probability that M fails to output a "good" prime is

$$\begin{aligned}
&\leq \mathbf{Pr}[M \text{ fails to output a prime}] + \mathbf{Pr}[\text{the prime output by } M \text{ is bad}] \\
&< (1 - 1/\log m)^t + 0.005 \\
&< 0.01.
\end{aligned}$$

It is easy to verify that the running time of M is $\mathrm{poly}(n)$. □

Note that for every $x \in W$, a "good" prime for x as described in Lemma 38 yields a hash function that isolates x from all other elements in W. Such a hash function can be encoded using at most $\log(K) + O(\log n)$ bits, and a hash value can be specified using at most $\log(K) + O(\log n)$ bits, where K is a upper bound on the size of the set W. Then using such a hash function in our decoding procedure described in the proof of Theorem 1, in order to recover an n-bit string x which is output by a uniform sampler $A(1^m)$ with probability at least δ, we need to know the code of A, the values of n, m, δ, a "good"prime p for x, and the value $x \bmod p$. Recall that in this decoding procedure we need to consider a set W of size at most $O(1/\delta)$. Also note that, since we can assume

$1/\delta$ is a power of 2, we can recover the value of δ given the value of $\log(1/\delta)$, so we can specify δ using only $\lceil \log\log(1/\delta) \rceil = O(\log n)$ bits. In total, we get a description of length

$$2 \cdot \log(1/\delta) + O(\log(m) + \log(n)).$$

A.2.2 String isolation via lossless condensers

Our second proof will use *lossless condensers* in the very low error regime. We first introduce some basic definitions and facts regarding condensers.

A.2.2.1 Technical tools

A distribution has high *min-entropy* if no element in its sample space has large probability weight.

Definition 39 (Min-Entropy)**.** *Let $\mathcal{X}$ be a distribution over some sample space Ω. The* min-entropy *of $\mathcal{X}$, denoted by $H_\infty(\mathcal{X})$, is the largest real number k such that $\mathbf{Pr}[\mathcal{X} = x] \leq 2^{-k}$ for every $x \in \Omega$. We call $\mathcal{X}$ a k-source if $H_\infty(\mathcal{X}) \geq k$.*

Definition 40 (Flat Distribution)**.** *A distribution $\mathcal{X}$ over some finite sample space Ω is called flat if it is uniform on a set $S \subseteq \Omega$. That is, for every $x \in S$, $\mathbf{Pr}[\mathcal{X} = x] = 1/|S|$.*

Note that a flat distribution on a set S has min-entropy $\log(|S|)$.

A *condenser* is a function that converts a source with a certain amount of min-entropy into another source with about the same amount of min-entropy but on a much smaller domain, so that the output source has a high *min-entropy rate*.

Definition 41 (Strong Condenser)**.** *For any integers n, d, τ, k, k' and $0 < \varepsilon < 1$, let $\mathcal{X}$ be a distribution over $\{0,1\}^n$ with min-entropy at least k and Y be the uniform distribution over $\{0,1\}^d$. A function $f\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^\tau$ is called a strong $k \to_\varepsilon k'$ condenser if the distribution $(Y, f(\mathcal{X}, Y))$ is ε-close in statistical distance to some distribution with min-entropy at least k'. The condenser f is called lossless if $k' = k$, and is called explicit if f can be computed in time* $\mathrm{poly}(n)$.

We will need the following useful properties about strong condensers.

Lemma 42 (See e.g. [Che10, Corollary 2.13])**.** *Let $f\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^\tau$ be a strong $k \to_\varepsilon k'$ condenser. Let $\gamma > 0$ and $\mathcal{X}$ be a k-source. Then for all but a γ fraction of the seeds $u \in \{0,1\}^d$, the distribution $f(\mathcal{X}, u)$ is (ε/γ)-close to a k'-source.*

Lemma 43 (See e.g. [Che10, Proof of Proposition 2.14])**.** *Let $\mathcal{X}$ be a flat distribution with min-entropy* $\log(K)$ *over a finite sample space Ω, and let $g\colon \Omega \to \Gamma$ be a mapping to a finite set Γ such that $g(\mathcal{X})$ is ε'-close to having min-entropy* $\log(K)$*. Finally, let*

$$T \stackrel{\text{def}}{=} \{y \in \Gamma \mid |g^{-1}(y) \cap \mathsf{Support}(\mathcal{X})| = 1\}.$$

Then $|T| \geq (1 - 2\varepsilon') \cdot K$.

We will use the following construction of a strong lossless condenser which has a short seed length and a high min-entropy rate.

Theorem 44 ([GUV09], see also [Che10, Theorem 2.22])**.** *For every constant $0 < \alpha < 1$, every integer $0 < k \leq n$, and every $0 < \varepsilon < 1$, there is an explicit strong $k \to_\varepsilon k$ condenser $f\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^\tau$, with seed length $d = (1 + 1/\alpha) \cdot (\log(n) + \log(1/\varepsilon) + \log(k)) + O(1)$ and output length $\tau = d + (1 + \alpha) \cdot k$.*

A.2.2.2 Proof of string isolation via lossless condenser

We obtain the following optimized version of Lemma 19. (Recall that in our applications we assume that an upper bound on K is known.)

Lemma 45 (String Isolation Lemma via Condensers)**.** *For every constant $\alpha \in (0,1)$, there is a deterministic polynomial-time algorithm M for which the following holds. For any set $W \subseteq \{0,1\}^n$ of size (at most) K, for at least a 0.99 fraction of the strings $u \in \{0,1\}^d$, $M(1^n, u)$ outputs a Boolean circuit that computes a function $H_u \colon \{0,1\}^n \to \{0,1\}^\tau$ with the following properties:*

- Isolating: *$H_u(w) \neq H_u(w')$ for every distinct pair $w, w' \in W$.*
- Succinct Encoding: *$d = (1 + 1/\alpha) \cdot (\log(n) + \log(K) + \log\log(K)) + O(1)$.*
- Small Range: *$\tau = d + (1+\alpha) \cdot \log(K)$.*

Proof. The algorithm M from the statement of the lemma outputs

$$H_u(\cdot) \stackrel{\text{def}}{=} f(\cdot, u),$$

where f is the explicit condenser from Theorem 44 with $k = \log(K)$ and $\varepsilon = 1/(300K)$. It is easy to verify that the parameters in Items 2 and 3 above are as claimed.

We now show that with probability at least 0.99 (over u), H_u isolates every string in the set W. Let $\mathcal{X}_W$ be the uniform distribution over the set W. Note that $\mathcal{X}_W$ is a flat distribution with min-entropy $\log(K)$. Since f is a strong $k \to_\varepsilon k$ condenser, where $k = \log(K)$ and $\varepsilon = 1/(300K)$, by Lemma 42, with probability at least 0.99 over the u's, $H_u(\mathcal{X}_W)$ is ε'-close to having min-entropy $\log(K)$, where

$$\varepsilon' = \varepsilon/0.01 = 1/(3K).$$

Then, by Lemma 43, the set

$$T \stackrel{\text{def}}{=} \{y \in \{0,1\}^\tau \mid |H_u^{-1}(y) \cap W| = 1\}$$

has size at least

$$(1 - 2\varepsilon') \cdot K = K - \frac{2}{3K} \cdot K = K - \frac{2}{3}.$$

This means that there are at least K buckets in H_u that contain exactly one string from W. Since $|W| = K$, this implies that every string in W was mapped to a bucket on its own. This concludes the proof of Item 1. □

Note that in Lemma 45, we have

$$\begin{aligned} d + \tau &= 2 \cdot (1 + 1/\alpha) \cdot (\log(n) + \log(K) + \log\log(K)) + (1+\alpha) \cdot \log(K) + O(1) \\ &= (3 + 2/\alpha + \alpha) \cdot \log(K) + O(\log n). \end{aligned}$$

Choosing $\alpha = \sqrt{2}$, we get

$$d + \tau = (3 + 2\sqrt{2}) \cdot \log(K) + O(\log n).$$

Therefore, we can get a description length of

$$(3 + 2\sqrt{2}) \cdot \log(1/\delta) + O(\log(m) + \log(n)).$$

A.3 On the (im)possibility of a coding theorem with efficient encoders and decoders

Recall that our randomized "encoding" algorithm in Theorem 1 is efficient, i.e., we can efficiently output a valid rKt description w of x with high probability. On the other hand, "decoding" x from w is not necessarily an efficient randomized computation. In this section, we observe under a standard assumption that this is unavoidable.

For simplicity of the exposition, our definition of one-way functions refers to security against (non-uniform) polynomial-size adversaries.

Theorem 46. *If one-way functions exist, then for any $\varepsilon > 0$, there exists a P-samplable distribution $\mathcal{D}$ over $\{0,1\}^n$ such that the set*

$$T_n \stackrel{\text{def}}{=} \{x \mid \mathcal{D}(x) \geq 1/2^{n^\varepsilon}\}$$

is not compressible to length $n-3$ by any encoding/decoding scheme that runs in randomized polynomial time when n is sufficiently large.

Proof. The proof is adapted from that of [Lee06, Theorem 5.3.1], attributed to Levin.

If one-way functions exist, then by [HILL99], for any $\varepsilon > 0$, there is a pseudorandom generator $G\colon \{0,1\}^{n^\varepsilon} \to \{0,1\}^n$ such that for any polynomial-size circuit C, it is the case that

$$\left| \Pr_{z \sim \{0,1\}^{n^\varepsilon}}[C(G(z)) = 1] - \Pr_{x \sim \{0,1\}^n}[C(x) = 1] \right| \leq 1/n. \tag{4}$$

Let $\mathcal{D}$ be the distribution over $\{0,1\}^n$ that is induced by the pseudorandom generator G, and let

$$T \stackrel{\text{def}}{=} \mathsf{Support}(\mathcal{D}).$$

Note that for every $x \in T$, $\mathcal{D}(x) \geq 1/2^{n^\varepsilon}$. Moreover, $\mathcal{D}$ is P-samplable.

For the sake of contradiction, suppose there are polynomial-time algorithms Enc and Dec such that for every $x \in T$,

$$\mathsf{Dec}(\mathsf{Enc}(x, r_1), r_2) = x \quad \text{and} \quad |\mathsf{Enc}(x, r_1)| \leq n - 3, \tag{5}$$

with probability at least $3/4$ over $r_1, r_2 \in \{0,1\}^{n^c}$, where $c > 0$ is some constant. Then by an averaging argument, there exist fixings of r_1, r_2 such that Condition 5 holds with probability at least $3/4$ with respect to $x \sim \mathcal{D}$. For such fixed r_1, r_2, define the circuit

$$C(x) = 1 \iff \text{Condition 5 holds for } x.$$

On the one hand, by the fact that Enc and Dec are polynomial-time computable, we have that C can be implemented in polynomial size, and by definition, we have

$$\Pr_{x \sim \mathcal{D}}[C(x) = 1] \geq 3/4.$$

On the other hand, we know by counting that the number of x's from $\{0,1\}^n$ that satisfies Condition 5 is at most 2^{n-2}, so

$$\Pr_{x \sim \{0,1\}^n}[C(x) = 1] \leq 1/4.$$

This contradicts Equation (4). □

This result has the following implication in connection to the problem of optimizing the dependence on $\log(1/\delta)$ in the coding theorem for rKt. Our proof of Theorem 1 provides an efficient encoding algorithm. On the other hand, our decoder, implicit in the rKt representation, has a running time t satisfying $\log t \geq \log(1/\delta)$. (This is how we sidestep Theorem 46.) Given that $\log(1/\delta)$ bits are necessary for description length by Shannon's Coding Theorem (Section 3.3), any approach via efficient encoders is unlikely to lead to a coding theorem for rKt that goes beyond the parameters in Problem 37, since contributions that depend on δ will come both from running time and description length.

B On the Space Complexity of the Encoding and Decoding Algorithms

One way to interpret Theorem 1 is that it probabilistically compresses strings that can be generated with non-trivial probability by a time-bounded procedure. Since data compression is often performed to reduce space usage, it is also important to have space-efficient compression and decompression algorithms. In this section, we revisit the time and space complexities of the compression scheme provided in the proof of Theorem 1, which appears in Section 3 (restated as Theorem 20). For simplicity and concreteness, we assume the RAM computational model for the analysis of the space and time complexities below.

Theorem 47 (Space complexity in the proof of Theorem 20)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^n)$ runs in time $T(n)$ and space $S(n)$, and outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Then given x, δ, and the code of A, it is possible to probabilistically encode x using $O(\log(1/\delta) + \log(n))$ bits in time $O(n) + \log(1/\delta) \cdot \mathrm{polylog}(n)$ and space $O(n)$. Such an encoding can be probabilistically decoded in time $(1/\delta) \cdot T(n) \cdot \mathrm{poly}(n)$ and space $(1/\delta) \cdot \mathrm{poly}(n) + S(n)$.*

Proof Sketch. We first address the complexity of the encoding algorithm. Recall that in the encoding procedure described in the proof of Theorem 20, given x, we first find a hash function H_u by randomly picking a string u which specifies a random walk $(u_1, u_2, \ldots, u_t)$ on an expander with $O(n)$ vertices (given by Theorem 12), where $t = O(\log(1/\delta))$. We then obtain the value of $H_u(x) = E(x)_{u_1}, E(x)_{u_2}, \ldots, E(x)_{u_t}$, where $E\colon \{0,1\}^n \to \{0,1\}^{O(n)}$ is the error-correcting code in Theorem 11. To implement this procedure, we can first compute $E(x)$ in time $O(n)$; it is known that there exists such an error-correcting code whose encoding can be done in linear time [Spi96, GI02]. We then generate the string u by simulating a random walk of length t on the expander and store only the current vertex in the work tape. That is, as we make a move on the graph, we randomly pick a direction and append it to u in the output (except for the first move where we randomly pick a starting vertex). We then compute the next vertex in the walk, replacing the current vertex in the work tape. Note that computing the next vertex can be done in time $\mathrm{polylog}(n)$. Then at the i-th step with the current vertex being u_i, we can write the i-th bit of $H_u(x)$ as $E(x)_{u_i}$ to the output. It is easy to verify that the above can be done in $O(n) + t \cdot \mathrm{polylog}(n)$ time and $O(n)$ space.

Next, we address the complexity of the decoding algorithm, for a non-trivial value $\delta \geq 2^{-n}$. Recall that in the proof of Theorem 20, we run the sampler $t = 64 \cdot (1/\delta) \cdot \log(1/\delta))$ times and store the resulting samples in the multi-set S_0. This requires time $t \cdot T(n)$ and space $t \cdot n + S(n)$. Next, we keep only the elements in S_0 that appear more than $\alpha = 32 \cdot \log(1/\delta)$ times to obtain a

set S. This can be done in time $t \cdot \mathrm{poly}(n)$ by sorting the elements in S_0. Finally, we need to apply hashing to each element in S, which takes time at most $t \cdot \mathrm{poly}(n)$ in total. □

One disadvantage of Theorem 47 is that the encoding and decoding algorithms require large space. In particular, the decoding algorithm requires super-polynomial space if $1/\delta$ is super-polynomial, even when the sampler runs in bounded space. Next, we present encoding and decoding algorithms that have a slightly larger running time but are *space efficient*.

Theorem 48. *Suppose there is a randomized algorithm A for sampling strings such that $A(1^n)$ runs in time $T(n)$ and space $S(n)$, and outputs a string $x \in \{0,1\}^n$ with probability at least $\delta > 0$. Then given x, δ, and the code of A, it is possible to probabilistically encode x using $O(\log(1/\delta) + \log(n))$ bits in time $n \cdot \log(1/\delta) \cdot \mathrm{polylog}(n)$ and space $\mathrm{polylog}(n)$. Such an encoding can be probabilistically decoded in time $(1/\delta)^2 \cdot T(n) \cdot \mathrm{poly}(n)$ and space $\mathrm{poly}(n) + S(n)$.*

Proof. We describe and analyse our encoding and decoding algorithms below. These algorithms are variants of the procedures from the proof of Theorem 20. We start with a modified encoding algorithm.

Encoding. Recall that in the encoding procedure, given an input x, we want to find a good hash function H_u which isolates every element in a set W of size $O(1/\delta)$ (which contains the strings that are output by A with probability at least $\delta/32$), and then compute the value $H_u(x)$. In order to make this space efficient, we will use the following hash family.

Claim 49. *For any $K > 0$, there exists a hash family $\{H_u \colon \{0,1\}^n \to \{0,1\}^t\}$, where $t = O(\log K)$, such that*

1. *Each H_u can be encoded using $O(\log(n) + t)$ bits.*
2. *For every $W \subseteq \{0,1\}^n$ of size K,*
$$\Pr_{H_u}[H_u(x) \neq H_u(x') \text{ for every distinct pair } x, x' \in W] \geq 0.99.$$
3. *There is a randomized algorithm that given K and $x \in \{0,1\}^n$, outputs $(H_u, H_u(x))$ in time $n \cdot t \cdot \mathrm{polylog}(n)$ and space $\mathrm{polylog}(n)$, where H_u is a uniform function from the hash family.*

Proof. We first describe the hash family. Let $S \subseteq \{0,1\}^n$ be a $(1/10)$-biased set. That is, for every $v \in \{0,1\}^n \setminus 0^n$, we have
$$1/2 - 1/10 \leq \Pr_{w \in S}[\langle v, w \rangle = 0] \leq 1/2 + 1/10,$$
where $\langle v, w \rangle = \sum_{i=1}^{n} v_i \cdot w_i \bmod 2$. In particular, the above equation implies that for every distinct x and x',
$$\Pr_{w \in S}[\langle x, w \rangle \neq \langle x', w \rangle] \geq 0.4. \tag{6}$$
It is known that there exists such a set S with $|S| = \mathrm{poly}(n)$, and given $j \in [n]$ and $i \in [|S|]$, the j-bit of the i-th element in S can be computed in $\mathrm{polylog}(n)$ time (see e.g. [AGHP92]). Let's denote $S = \{w_1, w_2, \ldots w_{|S|}\}$.

Let G be an expander with $|S|$ vertices given by Theorem 12, and let $u = (u_1, u_2, \ldots, u_t)$ be a walk on G, where $t = O(\log K)$. Then we define

$$H_u \stackrel{\text{def}}{=} \langle x, w_{u_1} \rangle, \langle x, w_{u_2} \rangle, \ldots, \langle x, w_{u_t} \rangle.$$

We first show Item 3. We will generate the string u, which specifies the hash function H_u, by simulating a random walk of length t on G and store only the current vertex in the work tape. That is, as we make a move on the graph, we randomly pick a direction and append it to u in the output (except for the first move where we randomly pick a starting vertex). We then compute the next vertex in the walk, replacing the vertex in the work tape. Note that computing the next vertex can be done in $\text{polylog}(n)$ time and space. Also, at the i-th step with the current vertex being u_i, we will compute the i-th bit of $H_u(x)$. Note that given u_i, for every $j \in [n]$, we can compute $z_j = x_j \cdot (w_{u_i})_j$ in $\text{polylog}(n)$ time and space (since the set S is strongly explicit). In order to compute the i-th bit of $H_u(x)$, we need to compute the parity of the z_j's for $j \in [n]$, which can be done in a streaming fashion using $\text{polylog}(n)$ space (and $n \cdot \text{polylog}(n)$ time), by computing one z_j at a time. Therefore, we can output $(H_u, H_u(x))$ in time $t \cdot n \cdot \text{polylog}(n)$ and space $\text{polylog}(n)$.

Item 2 can be shown using the Expander Chernoff Bound (Theorem 13) and Equation (6), as in the proof Lemma 19. We omit the details here.

Finally, Item 1 easily follows from the definition of the hash family, using that G is a constant-degree expander and that both G and S are strongly explicit. □

Proceeding as in the original proof, it is easy to see that we can use Claim 49 to get an encoding algorithm with the claimed time and space complexities.

Decoding. Recall that in the decoding procedure described in the proof of Theorem 20, we run the sampler $t = O(1/\delta \cdot \log(1/\delta))$ times and store the resulting samples, which requires space at least t. The idea for our space-efficient decoding algorithm is to process these samples in a streaming fashion without storing them in memory. More specifically, let H_u be an isolating hash function for the set of strings that are output by A with probability at least $\delta/32$, as described in the proof of Theorem 20. Consider Algorithm 1 described below.

Note that we can assume that $\delta \geq 2^{-n}$ without loss of generality. It is easy to verify that Algorithm 1 runs in time $(1/\delta)^2 \cdot T(n) \cdot \text{poly}(n)$ and uses space $O(n) + S(n)$, by noting that we can also compute each bit of H_u in $\text{polylog}(n)$ space.

Next, we show the correctness of this algorithm. Let's look at the part inside the outer for loop. On the one hand, if $y = x$, then by Claim 21, with probability except δ^6, we will end up with $\mathit{count} \geq \alpha$ and x will be recovered successfully. Also note that by our choice of parameter in the outer for loop, with probability except $(1-\delta)^{2/\delta} \leq e^{-2}$, in at least one of the executions we will have $y = x$. On the other hand, if $y \neq x$ but $H(y) = H(x)$, then we know that A outputs y with probability less than $\delta/32$ (since H is isolating for the set of strings that are output by A with probability at least $\delta/32$). Therefore, by Claim 22 the probability that $\mathit{count} \geq \alpha$ is less than $64 \cdot \delta^{16}$. By a union bound (over $i = 1, 2, \ldots, 2/\delta$), the probability that we output an answer is $128 \cdot \delta^{15} \leq 1/3$. As a result, the probability that the above algorithm fails to recover x is at most $\delta^6 + e^{-2} + 128 \cdot \delta^{15} < 1/4$. (Here we assume for simplicity that $\delta < 0.6$.) □

Algorithm 1 A space-efficient decoding algorithm

```
1:  procedure DECODE(A, n, δ, H_u, H_u(x))
2:      t ≝ 64 · (1/δ) · log(1/δ)
3:      α ≝ 32 · log(1/δ)
4:      for i = 1, 2, ..., 2/δ do
5:          y ← A(1^n)
6:          if H_u(y) = H_u(x) then
7:              count ≝ 0
8:              for j = 1, 2, ..., t do
9:                  z ← A(1^n)
10:                 if z = y then
11:                     count ← count + 1
12:         if count ≥ α then
13:             Output y
14:     Output "Fail"
```


ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il