Search-To-Decision Reductions for Kolmogorov Complexity

Noam Mazor
Tel Aviv University, Israel

Rafael Pass
Tel Aviv University, Israel
Cornell Tech, New York, NY, USA

Abstract

A long-standing open problem dating back to the 1960s is whether there exists a *search-to-decision* reduction for the time-bounded Kolmogorov complexity problem – that is, the problem of determining whether the length of the shortest time-t program generating a given string x is at most s.

In this work, we consider the more "robust" version of the time-bounded Kolmogorov complexity problem, referred to as the GapMINKT problem, where given a size bound s and a running time bound t, the goal is to determine whether there exists a $\text{poly}(t, |x|)$-time program of length $s + O(\log |x|)$ that generates x. We present the first non-trivial search-to-decision reduction R for the GapMINKT problem; R has a running-time bound of $2^{\epsilon n}$ for any $\epsilon > 0$ and additionally only queries its oracle on "thresholds" s of size $s + O(\log |x|)$. As such, we get that any algorithm with running-time (resp. circuit size) $2^{\alpha s}\text{poly}(|x|, t, s)$ for solving GapMINKT (given an instance (x, t, s), yields an algorithm for finding a witness with running-time (resp. circuit size) $2^{(\alpha+\epsilon)s}\text{poly}(|x|, t, s)$.

Our second result is a polynomial-time search-to-decision reduction for the time-bounded Kolmogorov complexity problem in the average-case regime. Such a reduction was recently shown by Liu and Pass (FOCS'20), heavily relying on cryptographic techniques. Our reduction is more direct and additionally has the advantage of being *length-preserving*, and as such also applies in the exponential time/size regime.

A central component in both of these results is the use of Kolmogorov and Levin's *Symmetry of Information Theorem*.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Kolmogorov complexity, search to decision

Digital Object Identifier 10.4230/LIPIcs.CCC.2024.34

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/003/` [18]

Funding *Noam Mazor*: Research partly supported by NSF CNS-2149305 and DARPA under Agreement No. HR00110C0086.
Rafael Pass: Supported in part by AFOSR Award FA9550-23-1-0387, AFOSR Award FA9550-23-1-0312, and an Algorand Foundation grant. This material is based upon work supported by DARPA under Agreement No. HR00110C0086. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government, DARPA, AFOSR or the Algorand Foundation.

1 Introduction

In his historical account, Trakhtenbrot [22] describes efforts in the 1960s in the Russian Cybernetics program to understand problems requiring *brute-force search* to solve [22, 23, 24]. The so-called *Perebor* (Russian for brute-force search) conjectures refer to the conjectures that certain types of, what today are referred to as "meta-complexity", problems require brute-force search to be solved. These include (a) the *Minimimum Circuit Size problem*

(MCSP) [12, 22] – finding the smallest Boolean circuit that computes a given function x, and (b) the *Time-Bounded Kolmogorov Complexity Problem* [14, 21, 2, 13, 6, 20] – computing the length, denoted $\mathrm{K}^t(x)$ of the shortest program (evaluated on some particular Universal Turing machine U) that generates a given string x, within time t, where $t = poly(|x|)$ is a polynomial.

Our focus in this paper is on the Time-Bounded Kolmogorov Complexity problem. As explained by Trakhtenbrot, two versions of this problem were considered since the 1960s.

- **The "existential" (i.e., decisional) version:** Given a string x and a threshold s, determining whether $\mathrm{K}^t(x)$ is less than "roughly" s.
- **The "constructive" (i.e., search) version:** Given a string x and a threshold s, if $\mathrm{K}^t(x)$ is less than "roughly" s, finding a program π of length "roughly" s that certifies this.

Both of these problems are conjectured to require *brute-force search*: that is, to require algorithms with running time close to 2^n where $n = |x|$ is the size of the given instance x. This is referred to as the *Perebor conjecture* (with respect to the time-bounded Kolmogorov complexity problem) and can be viewed as an early precursor, and stronger form, of the $\mathbf{NP} \neq \mathbf{P}$ conjecture (as the problem trivially resides in $\mathbf{NP}$). In fact, not only no non-trivial uniform algorithms are known for the search versions[1], but there are also no non-trivial (uniform) algorithms (i.e., beating brute-force search) even if we have access to an oracle solving the decisional version: That is, the only *search-to-decision reduction* is simply to ignore the decision oracle and solve the search version using brute-force search.[2]

The central result of this paper is developing the first non-trivial search-to-decision reduction for a gap-version of time-bounded Kolmogorov complexity; more precisely, we develop such a reduction with running time $2^{\epsilon n}$ for every $\epsilon > 0$.

We additionally address search-to-decision reductions in the *average-case* regime (w.r.t. the uniform distribution over instances). There, recently, Liu and Pass demonstrated a polynomial time reduction [16], but the reduction is not *length preserving* and as such it cannot be applied in the exponential regime. As our second result, we present a new direct proof of the result of [16], but achieve also a length-preserving polynomial-time reduction (which thus also applies in the exponential regime).

1.1 Our Results

To explain our results, let us first recall the MINKT and GapMINKT problems.

The GapMINKT Problem

Following Ko [13], we let MINKT denote the set of strings $(x, 1^t, 1^s)$ such that $\mathrm{K}^t(x) \leq s$. Since the notion of Kolmogorov complexity is highly dependent on the choice of the universal Turing machine, a natural – and more "robust" – variant of this problem allows for (a) some polynomial overhead in terms of the running time, and (b) some logarithmic slackness in terms of the threshold [15, 13]. Following Hirahara [9], we refer to $\mathrm{Gap_pMINKT}$ as the promise problem where:

[1] As we shall discuss shortly, two recent works [17, 11] demonstrate *circuits* (i.e., non-uniform algorithms) of size $2^{4n/5n+o(n)}$ that solve it.

[2] As just mentioned, in the non-uniform regime, non-trivial algorithms are known, but the same algorithm solves both the search and the decisional problem, so also in the non-uniform setting, the best known approach is to simply ignore the decisional oracle and solving the search problem from scratch.

- YES-instances consist of strings $(x, 1^t, 1^s)$ where $\mathrm{K}^t(x) \leq s$;
- NO-instances consist of strings $(x, 1^t, 1^s)$ where $\mathrm{K}^{p(t,|x|)}(x) > s + \log p(t, |x|)$;

and we say that an algorithm A solves GapMINKT if there exists some polynomial p such that A decides $\mathrm{Gap_p MINKT}$. Furthermore, we say that A solves the search version of the problem, search-GapMINKT if there exists some polynomial p such that given any $\mathrm{Gap_p MINKT}$ YES-instance $(x, 1^t, 1^s)$, A outputs a program Π certifying that $(x, 1^t, 1^s)$ is not a NO-instance (i.e, the program can run in time $p(t, |x|)$ and have length at most $s + \log p(t, |x|)$.

Non-trivial Search-to-Decision for GapMINKT

Our first (and main) result, is a non-trivial search-to-decision reduction for GapMINKT.

▶ **Theorem 1** (Informal). *For any $\varepsilon > 0$, and every polynomial τ, there exists a randomized oracle-aided algorithm F such that for every A that decides Gap_τMINKT, F^A solves* search-GapMINKT.

Moreover, on input $(x, 1^t, 1^s)$ F runs in time $2^{\epsilon s}\mathrm{poly}(|x|, t, s)$, and only queries A on inputs $(y, 1^{t'}, 1^{s'})$ with $|y| = \mathrm{poly}(|x|, t, s)$, $t' = \mathrm{poly}(|x|, t, s)$, and $s' \leq s + \log \mathrm{poly}(|x|, t, s)$.

F^A solves search-GapMINKT with gap $\tau^{O(1/\epsilon)}$. We remark that the reduction is not fully length preserving – the reduction invokes its oracle on statements x' that are longer than the original statement x, and at first sight, it may thus seem that the reduction is not useful in the regime of exponential hardness.

The key point, however, is that it only invokes the oracle on thresholds s' that are of roughly the same size as the original threshold s. Therefore, since the hardness of the GapMINKT problem most naturally should be thought to be a function of the threshold s (as there is a trivial $\mathrm{poly}(|x|, t, s)2^s$ algorithm, namely brute-force search), this reduction still yields a non-trivial search-to-decision reduction in the exponential regime:

▶ **Corollary 2.** *For any $\epsilon > 0, \alpha > 0$, $\tau \in \mathrm{poly}$, assume that there is an algorithm that solves Gap_τMINKT in time (resp. size) $2^{\alpha \cdot s}\mathrm{poly}(|x|, t, s)$. Then there exists an algorithm that solves* search-GapMINKT *in time (resp. size) $2^{(\alpha+\epsilon)s} \cdot \mathrm{poly}(|x|, t, s)$ on inputs $(x, 1^t, 1^s)$.*

An Average-case Search-to-Decision Reduction

We turn to considering the average-case regime. Here we provide a polynomial-time reduction that additionally is *length-preserving* and as such directly also applies in the exponential regime. (This is in contrast to the earlier average-case search-to-decision reduction of [16] that did not apply in the exponential regime.)

▶ **Theorem 3.** *For every $p, t \in \mathrm{poly}$ there exists $\widehat{p} \in \mathrm{poly}$ and $t' \in \mathrm{poly}$, and an efficient oracle-aided algorithm F such that the following holds. Let A be an algorithm that computes $\mathrm{K}^{t'}$ with probability $1 - 1/\widehat{p}$ on the uniform distribution. Then F^A solves* search-K^t *with probability $1 - 1/p$ on the uniform distribution.*

Moreover, on input $x \in \{0,1\}^n$, F makes only queries of the form $x||y$ with $y \in O(\log n)$.

As corollaries, we thus get:

▶ **Corollary 4** (reproving [16]). *For every $p, t \in \mathrm{poly}$ there exists $\widehat{p}, t' \in \mathrm{poly}$ such that the following holds: if there exists a polynomial time (reps. $2^{o(|x|)}$ time) algorithm that computes $\mathrm{K}^{t'}$ with probability $1 - 1/\widehat{p}$ over the uniform distribution, then exists polynomial time (resp. $2^{o(|x|)}$ time) algorithm that solves* search-K^t *with probability $1 - 1/p$ on the uniform distribution. Moreover, the same holds also in the non-uniform setting (i.e., w.r.t. polynomial-size and respectively $2^{o(|x|)}$ size algorithms).*

▶ **Corollary 5** (new). *For every $p, t \in \text{poly}$ there exists $\widehat{p}, t' \in \text{poly}$ such that the following holds: if there exists a constant $\alpha > 0$ and a $2^{\alpha \cdot |x|}\text{poly}(|x|)$ time (reps. size) algorithm that computes $\text{K}^{t'}$ with probability $1 - 1/\widehat{p}$ over the uniform distribution, then there exists a $2^{\alpha \cdot |x|}\text{poly}(|x|)$ time (resp. size) algorithm that solves* search-K^t *w.p $1 - 1/p$ over the uniform distribution.*

We note that while our reduction improves on [16] in the length-preserving aspect (and additionally is significantly simpler), it also has some disadvantages: in particular, in [16] an oracle for $\text{K}^{t'}$ for *any* polynomial t' can be used to solve search-K^t for any other polynomial t, whereas in our case, the reduction only works as long as t is sufficiently larger than t'. Additionally, the same thing holds also with respect to the error probability polynomials $\widehat{p}, p$. The reasons for these "amplifications" is that [16] passes through cryptographic techniques (hardness amplification [25], and constructions of pseudorandom generators [7]) that blow up the input size.

1.2 Related Works

While, as far as we know, no non-trivial search-to-decision reductions were previously known for GapMINKT in the worst-case regime, there are several works that consider variants of this question:

Slightly Subexponential Search-to-Decision for MFSP: An elegant work by Ilango considers a formula size variant of the classic Perebor conjecture problem, MFSP, where the goal is to find the shortest formula computing some given function. He demonstrates a search-to-decision reduction with running-time $2^{0.67n}$ for MFSP. As far as we know, this is the first result to demonstrate any non-trivial search-to-decision reduction for a Perebor-style problem. (We note that in contrast, we here consider the standard time-bounded Kolmogorov complexity problem, and we also get a smaller running time of $2^{\epsilon n}$ for any $\epsilon > 0$.) Ilango also gets an improved running time of $2^{n/\log\log n}$ if only requiring an algorithm that succeeds on *most* (i.e., a $1 - 1/o(1)$) fraction of instances. In contrast, in this setting, we get a polynomial running time.

Average-case Search-to-Decision for MINKT: Liu and Pass [16] show a *polynomial-time* algorithm that solves the search-GapMINKT) *on average* over the uniform (over x, and for every t, s) given access to an oracle that solves GapMINKT on average. Our second result is a strict strengthening of this result since our reduction is length-preserving (i.e., it only queries its oracle on input lengths that are $O(\log n)$ longer), and as such it also applies in the exponential regime (whereas the result of [16] only apply in the polynomial to subexponential regimes).

Conditional and Non-black-box Search-to-Decision for GapMINKT: An intriguing work by Hirahara [10] presents a *non-black-box* search to decision reduction for GapMINKT in the polynomial regime, under standard derandomization assumptions. More precisely, assuming that E does not have subexponential-size circuits, he shows that if GapMINKT has a (deterministic, wlog due to the assumption) polynomial-time decider, then search-GapMINKT has a polynomial-time algorithm. His result does not extend to the non-uniform setting, or to algorithms running in time even just $n^{\log n}$ due to the fact that the code of the GapMINKT attacker gets incorporated into the witness for the search-GapMINKT problem. In contrast, ours is unconditional; on the other hand, ours is only meaningful in the exponential regime (as the running time of the reduction is subexponential).

A different paper by Hirahara [8] gets an unconditional non-black-box search-to-decision reduction for the polynomial regime for $\text{Gap}_\tau\text{MINKT}$ w.r.t $\tau = 2^{\sqrt{n}}$.[3] This result also does not apply in the non-uniform setting, but does extend to the subexponential (but not exponential) regime.

Search-to-Decision Reductions w.r.t. Black-box Solvers: In a very recent work, the current authors consider *black-box* solvers for the MINKT problem that solve the problem no matter what the underlying Universal Turing Machine U is, given black-box access to it. A polynomial-size black-box search-to-decision reduction is demonstrated with respect to such attackers. In contrast, we here consider all, and not just black-box, solvers.

Non-uniform Algorithms Beating Perebor: As mentioned above, two independent recent works [17, 11] develop algorithms solving the MINKT problem using a circuit of size $2^{4n/5}\text{poly}(n)$, disproving the "non-uniform" version of the Perebor conjecture. These algorithms directly also work for the search version of the problem and as such, even in the non-uniform regime, it was not known how to make use of an GapMINKT oracle to solve the search version better than simply solving it from scratch.

1.3 Proof Overview

We provide a brief overview of the proofs of Theorem 1 and 3, starting with Theorem 1, which proceeds in two steps.

Worst-case Search-to-Decision Reduction for "Shallow" Instances

As an intermediary step, which may be of independent interest, we start by providing a search-to-decision reduction whose running time is a function of the so-called *computational depth* of the instance x we are reducing from (i.e., that we want to find a witness for). Recall that the computational depth [1] of an instance x is defined as $cd^t(x) = \text{K}^t(x) - \text{K}(x)$. Note that by a standard counting argument, we have that "computationally deep" strings (i.e., string x such that $cd^t(x) > O(\log(|x|))$ are rare.

We start by presenting a search-to-decision reduction with running time

$$2^{cd^t(x)}\text{poly}(|x|, t, s)$$

(which thus for most strings runs in polynomial time). The key idea behind the reduction is the following. Given a string x, and a minimal-length t-time program Π generating x, the $\text{K}^{t'}$-complexity of the string $x||\Pi$, for $t' = \text{poly}(|x|, t)$, is not significantly higher than the K^t-complexity of the string x – since the string $x||\Pi$ also can be generated by a self-printing variant of Π. Furthermore, the above argument also holds even if we concatenate not only the whole of Π but even just a prefix of it.

Thus, if we have access to a GapMINKT oracle, we ought to be able to find Π "bit-by-bit" by simply concatenating a bit to x and checking if the $\text{K}^{t'}$-complexity remains below $s + O(\log |x|)$. In more detail, we keep track of a set $\mathcal{S}$ of candidates y (whose prefix is x) and at each iteration concatenate each bit $b \in \{0, 1\}$ to y and check whether the $\text{K}^{t'}$-complexity of $y||b$ remains small, and if so adding $y||b$ to the set $\mathcal{S}$. By the argument above and a standard induction, we have that at iteration i, $y = x||\Pi_{\leq i}$ (where $\Pi_{\leq i}$ denotes the i first bit of Π) must be in the set $\mathcal{S}$, so we can finally find Π by simply going over all the elements $y = x||\Pi'$ of $\mathcal{S}$ and checking whether Π' generates x.

[3] The results is actually a bit stronger – the running time only increases by a polynomial, but the gap increases by $\sqrt{n}$, as opposed to the desired $O(\log n)$.

The problem, of course, is that the set $\mathcal{S}$ could contain lots of other elements. This is where computational depth enters the picture. To see why, let us first start by showing that if we had been dealing with Kolmogorov complexity, as opposed to t-bounded Kolmogorov complexity, then the size of $\mathcal{S}$ can never be more than of polynomial size (in $|x|, t$). In fact, this follows almost directly from Kolmogorov and Levin's celebrated *symmetry of information (SoI)* theorem [26] which states that for any strings a, b, we have that[4]

$$\mathrm{K}(a||b) \geq \mathrm{K}(a) + \mathrm{K}(b|a) - O(\log(|a| + |b|).$$

Indeed, recall that $\mathcal{S}$ consists of all strings $y = x||\Pi$ whose K-complexity is roughly that of x; by the SoI theorem, setting $a = x$ and $b = \Pi$, we get that $\mathrm{K}(\Pi|x) \leq O(\log |x|)$ and thus there can be at most $\mathrm{poly}(|x|)$ such strings.[5]

Finally, note that if considering a string x whose computational depth is d, then $\mathrm{K}^t(x) - \mathrm{K}(x) \leq d$, and as such for each element $x||\Pi$ that remains in $\mathcal{S}$, we have that

$$\begin{aligned}
\mathrm{K}(x||\Pi) - \mathrm{K}(x) &\leq \mathrm{K}^t(x||\Pi) - \mathrm{K}(x) \\
&\leq \mathrm{K}^t(x) + O(\log |x|) - \mathrm{K}(x) \\
&\leq \mathrm{K}^t(x) + O(\log |x|) - (\mathrm{K}^t(x) - cd^t(x)) \\
&\leq d + O(\log |x|)
\end{aligned}$$

Thus, by the SoI theorem, we then get that $\mathrm{K}(\Pi|x) \leq d + O(\log |x|)$, and therefore we have that $|\mathcal{S}| \leq 2^d \mathrm{poly}(|x|)$. As such, the running of our algorithm becomes $2^{cd^t(x)}\mathrm{poly}(|x|, t, s)$, as desired.

Dealing with Deep Instances

Note that given any instances x whose computational depth is bounded by $\epsilon|x|$, then the running time of the above algorithm becomes $2^{\epsilon|x|}\mathrm{poly}(|x|, t, s)$, as desired. If not, and in case the algorithm's running time becomes larger than this, we must have that the set $\mathcal{S}$ produced is bigger than $2^{\epsilon|x|}$. Our key idea now is to simply stop the algorithm once the size of $\mathcal{S}$ reaches $2^{\epsilon|x|}$, and at this point selecting a *random element* in $x' \in \mathcal{S}$, and restarting the algorithm on x' instead (since a program generating x' can easily be modified to a program generating x). The reason for doing this is that since the set $\mathcal{S}$ is "big", by choosing a random element, we are guaranteed that the *actual* (i.e., not time-bounded) Kolmogorov complexity of the chosen string x' is roughly ϵn larger than that of x, yet since all strings in $\mathcal{S}$ have roughly the same time-bounded Kolmogorov complexity ($s + O(\log |x|)$, we must have that the computational depth of x' is at least ϵn smaller than that of x. In essence, by picking this random element, we are able to get a new instance x' such that (a) the witness for x' is also a valid witness for x, yet (b) the computational depth of x' is $\epsilon|x|$ smaller than that of x.

By iteratively continuing this process, we eventually (after $1/\epsilon$ steps) end up with an element with small computational depth and thus manage to find a witness in the desired running time.

---

[4] Recall that the conditional Kolmogorov complexity of b given a, denoted by $\mathrm{K}(b \mid a)$ is the minimal length of a program that outputs b given input a.

[5] This result may be of independent interest. It shows a polynomial-time "list-to-decision" reduction for Kolmogorov-complexity – that is, a polynomial-time algorithm that given access to a decision oracle outputs a polynomial-length list of candidate witnesses, one of which is correct. The reason why this does not yield a *search*-to-decision reduction is that we cannot, in polynomial-time, determine if a witness is correct by running it.

Let us highlight why this approach only gives an algorithm with subexponential running time: The issue is that each time we pick a random element $x' \in \mathcal{S}$, the time-bounded Kolmogorov complexity of the element may increase by $O(\log |x|)$, so we can only afford a constant number of iterations, which is why we need to make sure that we can eliminate a constant fraction of the computational depth in each step. (An additional reason is that the running-time t' blows up as a polynomial of t in each iteration, so again, we can only afford a constant number of iterations.)

Search-to-Decision in the Average-case Regime

We turn to discussing our search-to-decision reduction in the average-case regime. The goal is to show how to use an oracle that (decides, or equivalently, computes) K^t with high probability on the uniform distribution to find a $\mathrm{K}^{t'}$ witness with high probability over the uniform distribution for a polynomially related t'.

Towards this, we will show a reduction that again works in the worst-case regime, but only on computationally shallow instances – that is, instances x with computational depth $O(\log |x|)$. This reduction will improve on the one above in the sense that it is *length preserving*; additionally, due to the length-preserving aspect of the reduction, it will also follow that if we only require the reduction to work with high probability (over the uniform distribution) over instances, then it suffices for the oracle to also work with high probability.

The idea is to, given an instance x, consider strings $y = x||i||\Pi_i$, where Π_i is the ith "chunk" (of length $O(\log|x|)$) of the smallest time-bounded program Π generating x; such strings still have roughly the same time-bounded Kolmogorov complexity as x, and by the same argument based on symmetry-of-information, we can argue that there cannot be more than polynomially many strings y that have x as a prefix and also have roughly the same time-bounded Kolmogorov complexity as x. This enables us to recover a small set of candidates for (most) of the "coordinates" of Π. But, even if there are just 2 candidates for each such coordinate, there will still be too many options to try out, as the number of coordinates is polynomial in $|\Pi|/|\Pi_i|$ which can be as large as $|x|/\log|x|$.

To solve this problem, we will rely on the notion of a *list-recoverable error-correcting code* [5, 3] – in essence, a type of an error-correcting code (ECC) from which we recover a polynomial-length list of candidate messages (one of which is guaranteed to be the true one) given a polynomial-length candidate list for each symbol of the encoding. Roughly speaking, we find all strings $y = x||i||z_i$ that have small time-bounded Kolmogorov complexity, and then apply the list-recoverable procedure of the ECC. By the existence of *efficient* list-recoverable codes [5] (where both the encoding and decoding can be done efficiently), we are still guaranteed that when z_i is the ith symbol of the encoding of Π, then y indeed has small time-bounded Kolmogorov complexity; next, the above symmetry-of-information based argument will ensure that we can only have a "small" number of candidates for most coordinates i, and as such, the list-recovering procedure will indeed find some short program Π.

There is just one catch with this argument: using the above SoI based argument we will get a too weak bound on the number of possible candidates for each symbol. We note, however, that we can use the same argument to bound the *total* number of strings of the form (x, i, z) with small time-bounded Kolmogorov complexity, and as such, use an averaging argument to argue that for, say 90% of the coordinates, we get a sufficiently small list of symbols. The issue remaining is that we can no longer rely on the list-recoverability property to recover the message (as we no longer have a short list for *every* symbol of the codeword). Luckily, there exist list-recoverable codes satisfying exactly this property (i.e.,

that we can recover a polynomial-length list of messages, even if we only have a bound on the set of symbols, for a constant fraction of the coordinates) – indeed, as shown in [5, 4], the Reed-Solomon code also satisfies such list-recoverability "with errors".

To finally see why this reduction also works in the average-case regime, first recall that computationally deep strings are rare, so the reduction will work with high probability over x, as long as the oracle works on all instances. Next, note that the reduction, given an instance x, only queries its oracles on instances of roughly the same length as x, and that have x as a prefix, which suffices to argue that we only need an oracle that works with high probability.

2 Preliminaries

2.1 Notations

All logarithms are taken in base 2. We use calligraphic letters to denote sets and distributions, uppercase for random variables, and lowercase for values and functions. Let poly stand for the set of all polynomials. Given a vector $v \in \Sigma^n$, let v_i denote its i^{th} entry, let $v_{<i} = (v_1, \ldots, v_{i-1})$ and $v_{\leq i} = (v_1, \ldots, v_i)$. For $x, y \in \{0,1\}^*$, we let xy and $x||y$ denote the concatenation of the strings x an y. An oracle-aided algorithm A is an algorithm with an oracle access. Given a (randomized) function $\mathcal{O}$, we use $A^{\mathcal{O}}$ to denote the algorithm A when using $\mathcal{O}$ as the oracle.

2.2 Distributions and Random Variables

When unambiguous, we will naturally view a random variable as its marginal distribution. The support of a finite distribution $\mathcal{P}$ is defined by $\text{Supp}(\mathcal{P}) := \{x \colon \Pr_{\mathcal{P}}[x] > 0\}$. For a (discrete) distribution $\mathcal{P}$, let $x \leftarrow \mathcal{P}$ denote that x was sampled according to $\mathcal{P}$. Similarly, for a set $\mathcal{S}$, let $x \leftarrow \mathcal{S}$ denote that x is drawn uniformly from $\mathcal{S}$.

2.3 Kolmogorov Complexity

Roughly speaking, the *t-time-bounded Kolmogorov complexity*, $\mathrm{K}^t(x)$, of a string $x \in \{0,1\}^*$ is the length of the shortest program $\Pi = (M, y)$ such that, when simulated by a universal Turing machine, Π outputs x in t steps. Here, a program Π is simply a pair of a Turing Machine M and an input y, where the output of Π is defined as the output of $M(y)$. When there is no running time bound (i.e., the program can run in an arbitrary number of steps), we obtain the notion of Kolmogorov complexity.

In the following, let $\mathsf{U}(\Pi, 1^t)$ denote the output of Π when emulated on U for t steps. We now define the notion of Kolmogorov complexity with respect to the universal TM U.

▶ **Definition 6.** *Let $t \in \mathbb{N}$ be a number. For all $x \in \{0,1\}^*$, define*

$$\mathrm{K}^t_{\mathsf{U}}(x) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : \mathsf{U}(\Pi, 1^t) = x\}$$

where $|\Pi|$ is referred to as the description length *of Π. Similarly, for every $z \in \{0,1\}^*$ define*

$$\mathrm{K}^t_{\mathsf{U}}(x \mid z) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : \mathsf{U}(\Pi(z), 1^t) = x\}.$$

When there is no time bound, we define

$$\mathrm{K}_{\mathsf{U}}(x) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : \exists t \in \mathbb{N} \text{ s.t. } \mathsf{U}(\Pi, 1^t) = x\}$$

and

$$\mathrm{K}_{\mathsf{U}}(x \mid z) = \min_{\Pi \in \{0,1\}^*} \{|\Pi| : \exists t \in \mathbb{N} \text{ s.t. } \mathsf{U}(\Pi(z), 1^t) = x\}.$$

It is well known that for every x, $\mathrm{K}^t(x) \leq |x| + c$, for some constant c depending only on the choice of the universal TM U.

▶ **Fact 7.** *For every universal TM* U*, there exists a constant* c *such that for every* $x \in \{0,1\}^*$*, and for every* t *such that* $t(n) > 0$*,* $K^t_{\mathsf{U}}(x) \leq |x| + c$*.*

We will also use the following fact, which states that we can efficiently encode a pair (x, y) with a small overhead.

▶ **Fact 8.** *There exists* $q \in \mathrm{poly}$ *such that the following holds or every* $x, y \in \{0,1\}^*$*,*

$$\mathrm{K}^{q(|xy|)}(x, y) \leq |x| + |y| + \log|x| + 2\log\log|x| + O(1).$$

We will use the following bound on the Kolmogorov complexity of strings sampled from the uniform distribution.

▶ **Lemma 9.** *For any universal TM* U*, any string* $x \in \{0,1\}^*$ *and any set* $\mathcal{S}$*, it holds that*

$$\Pr_{y \leftarrow \mathcal{S}}[\mathrm{K}_{\mathsf{U}}(y \mid x) < \log|\mathcal{S}| - i] \leq 2^{-i}.$$

In this paper, unless otherwise stated, we fix some universal Turing machine U that can emulate any program Π with polynomial overhead, and let $\mathrm{K}^t = \mathrm{K}^t_{\mathsf{U}}$ and $\mathrm{K} = \mathrm{K}_{\mathsf{U}}$.

The computational depth of a string is the difference between its Kolmogorov complexity and its time-bounded Kolmogorov complexity.

▶ **Definition 10** (Computational depth [1])**.** *For* $x \in \{0,1\}^*$ *and* $t \in \mathbb{N}$*, the* computational depth *of* x *is defined to be* $cd^t(x) = \mathrm{K}^t(x) - \mathrm{K}(x)$*.*

Since, by a simple counting argument, most strings $x \in \{0,1\}^n$ have $\mathrm{K}^t(x)$ close to n, it holds that most strings have small computational depth.

▶ **Fact 11.** *For every* $n \in \mathbb{N}$ *and every* $t \in \mathbb{N}$*,* $\Pr_{x \leftarrow \{0,1\}^n}[cd^t(x) > i] \leq 2^{-i}$*.*

We will also use the Symmetry of Information lemma.

▶ **Theorem 12** (Symmetry of Information [26])**.** *There exists a constant* $c \in \mathbb{N}$ *such that for every* $x, y \in \{0,1\}^*$*,*

$$\mathrm{K}(x) + \mathrm{K}(y \mid x) + c\log(|x| + |y|) \geq \mathrm{K}(x||y) \geq \mathrm{K}(x) + \mathrm{K}(y \mid x) - c\log(|x| + |y|)$$

We next define MINKT and GapMINKT.

▶ **Definition 13** (MINKT)**.** MINKT *is the following promise problem:*

- $\mathcal{Y} = \{(x, 1^t, 1^s) \colon \mathrm{K}^t(x) \leq s\}$
- $\mathcal{N} = \{(x, 1^t, 1^s) \colon \mathrm{K}^t(x) > s\}$

We say that an algorithm A *solves* search-MINKT *if* A *finds a program* Π *such that* $|\Pi| \leq s$ *and* $\mathsf{U}(\Pi, 1^t) = x$*, for every* $(x, 1^t, 1^s) \in \mathcal{Y}$*.*

We remark that MINKT is actually a language (every possible input is either in $\mathcal{Y}$ or in $\mathcal{N}$), and we define it as a promise problem for easy comparison with GapMINKT.

▶ **Definition 14** (GapMINKT)**.** *Let* $\tau \in \mathrm{poly}$ *be a polynomial such that* $\tau(t, |x|) \geq t$ *for every* t, x*. Then* Gap_τMINKT *is the following promise problem:*

- $\mathcal{Y} = \{(x, 1^t, 1^s) \colon \mathrm{K}^t(x) \leq s\}$
- $\mathcal{N}_\tau = \left\{(x, 1^t, 1^s) \colon \mathrm{K}^{\tau(t,|x|)}(x) > s + \log\tau(t, |x|)\right\}$

We say that an algorithm A *decides* GapMINKT *if there exists* $\tau \in \mathrm{poly}$ *such that* A *decides* Gap_τMINKT*.*

We say that a (randomized) algorithm A *solves* search-GapMINKT *if there exists* $\tau \in \mathrm{poly}$ *such that* A *(with probability* $1/2$*) finds a program* Π *such that* $|\Pi| \leq s + \log\tau(t, |x|)$ *and* $U(\Pi, 1^{\tau(t,|x|)}) = x$*, for every* $(x, 1^t, 1^s) \in \mathcal{Y}$*.*

3 Decision-to-Search for Shallow Instances

In this part we prove our search-to-decision reduction for inputs with small computational depth.

▶ **Theorem 15.** *There exists an oracle-aided algorithm F such that the following holds. Let A be an oracle that decides* GapMINKT*. Then F^A solves* search-MINKT*.*

Moreover, on input $(x, 1^t, 1^s)$, F runs in time $2^{cd^t(x)}\text{poly}(|x|, t, s)$, and only queries A on inputs $(y, 1^{t'}, 1^{s'})$ with $|y| \leq |x| + s$, $t' \in \text{poly}(|x|, t)$, and $s' \leq s + O(\log(|x| + t))$.

Directly from Theorem 15 we get the following corollary.

▶ **Corollary 16.** *The following holds:*

- *Assume that there is a poly-time (resp. poly size) algorithm that solves* GapMINKT*. Then there exists an algorithm that solves* search-GapMINKT *in time (resp. size) $2^{cd^t(x)} \cdot \text{poly}(|x|, t, s)$ on inputs $(x, 1^t, 1^s)$.*
- *Assume that for some $\alpha > 0$ there is an algorithm that solves* GapMINKT *in time (resp. size) $2^{\alpha \cdot s}\text{poly}(|x|, t, s)$. Then there exists an algorithm that solves* search-GapMINKT *in time (resp. size) $2^{cd^t(x)+\alpha \cdot s} \cdot \text{poly}(|x|, t, s)$ on inputs $(x, 1^t, 1^s)$.*

The proof of Theorem 15 is almost immediate from the following lemma, in which the running time of the algorithm that solves MINKT is larger for high values of the threshold s.

▶ **Lemma 17.** *There exists an oracle-aided algorithm F' such that the following holds. Let A be an oracle that decides* GapMINKT*. Then F'^A solves* search-MINKT*.*

Moreover, on input $(x, 1^t, 1^s)$, F' runs in time $2^{s-\text{K}(x)}\text{poly}(|x|, t, s)$, and queries A on inputs $(y, 1^{t'}, 1^{s'})$ with $|y| \leq |x| + s$, $t' \in \text{poly}(t)$, and $s' \leq s + O(\log(|x| + t))$.

Proof of Theorem 15. Let F be the algorithm that given $(x, 1^t, 1^s)$ runs F' on input $(x, 1^t, 1^{s'})$ for every $s' = 1, \ldots, s$, until the first execution that outputs a program Π with $\mathsf{U}(\Pi, 1^t) = x$. The theorem follows since the algorithm halts when $s' = \text{K}^t(x)$. ◀

We next prove Lemma 17. In the proof of Lemma 17 we will use the following claim.

▷ Claim 18. There exists a polynomial $q \in \text{poly}$ and a constant c_0, such that the following holds for every $x \in \{0,1\}^*$ with $|x| \geq 2$, and every $t \in \mathbb{N}$. Let Π a program of length $\text{K}^t(x)$ such that $\mathsf{U}(\Pi, 1^t) = x$. Then for every $i \leq \text{K}^t(x)$, $\text{K}^{q(t,|x|)}(x||\Pi_{\leq i}) \leq \text{K}^t(x) + c_0 \log|x|$.

Proof of Claim 18. Let U' be an efficient program such that $\mathsf{U}'(\Pi, i) = \mathsf{U}(\Pi)||\Pi_{\leq i}$ (where we encode Π, i using Fact 8), and let q be a polynomial such that $q(t, |x|)$ is an upper bound on the running time of U' where t is a bound on the running time of $\mathsf{U}(\Pi)$ (recall that $|\Pi| = \text{K}^t(x) \leq |x| + O(1)$).

Then $\text{K}^{q(t,|x|)}(\mathsf{U}(\Pi)||\Pi_{\leq i}) \leq |(\Pi, i)| + O(1) \leq |\Pi| + 3\log|\Pi| + O(1) \leq |\Pi| + 3\log(|x| + c) + O(1)$, where the last inequality holds by Fact 7, for some constant $c \in \mathbb{N}$. ◁

To prove Lemma 17, let $q \in \text{poly}$ and c_0 be the polynomial and constant promised by Claim 18, and consider the following algorithm that finds a minimal K^t-witness.

▶ **Algorithm 19** (F').
Oracle: GapMINKT *decider* A.
Input: $(x, 1^t, 1^s)$ *for* $x \in \{0,1\}^*$, $t, s \in \mathbb{N}$.
1. *Set* $\mathcal{S}_0 = \{x\}$ *and* $k = s + c_0 \log|x|$.
2. *For every* $i = 1, 2, \ldots, s$:
   a. *Compute* $\mathcal{S}_i = \left\{yb\colon y \in \mathcal{S}_{i-1}, b \in \{0,1\} \text{ and } A(yb, 1^{q(t,|x|)}, 1^k) = \text{Yes}\right\}$
   b. *If exists* $y \in \mathcal{S}_i$ *such that* $y = x||\Pi$ *and* $\mathsf{U}(\Pi, 1^t) = x$, *output* Π *and terminate.*
3. *Output* $\perp$.

We will show that the correctness of the above algorithm follows directly by Claim 18. To bound the running time of Algorithm 19, we will use the following claim.

▷ Claim 20. On input $(x, 1^t, 1^s)$, Algorithm 19 runs in time $2^{s-\mathrm{K}(x)} \cdot \mathrm{poly}(|x|, t, s)$.

Before proving Claim 20, let us use Claim 18 and Claim 20 to prove Lemma 17.

Proof of Lemma 17. We start with the correctness of Algorithm 19. Let $(x, 1^t, 1^s)$ be the input for the algorithm, and assume that $\mathrm{K}^t(x) \leq s$. Let Π be a program of minimal length such that $\mathsf{U}(\Pi, 1^t) = x$. By Claim 18, the correctness of the oracle A, and by a simple induction, $x||\Pi_{\leq i}$ is in the set $\mathcal{S}_i$ for every $i \leq |\Pi| \leq s$. Therefore, $x||\Pi$ is in $\mathcal{S}_{\mathrm{K}^t(x)}$, and thus Algorithm 19 outputs a correct answer.

By Claim 20, Algorithm 19 runs in time $2^{s-\mathrm{K}(x)} \cdot \mathrm{poly}(|x|, t, s)$. Finally, it is not hard to see that Algorithm 19 makes only queries of the form $(yb, 1^{q(t,|x|)}, 1^k)$ with $k = s + c_0 \log|x|$, and $|yb| \leq |x| + s$. ◀

3.1 Proving Claim 20

We will use the following lemma, that bounds the number of k-bit strings y such that xy has low Kolmogorov complexity

▶ **Lemma 21.** *There exists a constant* $c \in \mathbb{N}$ *such that the following holds for every* $x \in \{0,1\}^*$ *and for every* $k, \ell \in \mathbb{N}$.

$$\left|\left\{y \in \{0,1\}^{\leq k}\colon \mathrm{K}(xy) \leq \ell\right\}\right| \leq 2^{\ell+1-\mathrm{K}(x)} \cdot (|x|+k)^c$$

Proof of Lemma 21. Let c be the constant from Theorem 12. By a simple counting argument, there are at most $2^{\ell + c\log(|x|+k)+1-\mathrm{K}(x)}$ strings $y \in \{0,1\}^{\leq k}$ such that $\mathrm{K}(y \mid x) \leq \ell + c\log(|x|+k) - \mathrm{K}(x)$. It thus enough to show that for every $y \in \{0,1\}^{\leq k}$ with $\mathrm{K}(y \mid x) > \ell + c\log(|x|+k) - \mathrm{K}(x)$, it holds that $\mathrm{K}(xy) > \ell$, which is true By Theorem 12. ◀

We are now ready to prove Claim 20.

Proof of Claim 20. The running time of *Algorithm* 19 is bounded by $|\bigcup \mathcal{S}_i| \cdot \mathrm{poly}(|x|, t, s)$, and thus it is enough to bound the size of $\bigcup \mathcal{S}_i$. Toward this goal, let $\tau \in \mathrm{poly}$ be the polynomial for which A decides $\mathrm{Gap}_\tau\mathrm{MINKT}$. We get that

$$\begin{aligned}
\left|\bigcup \mathcal{S}_i\right| &\leq \left|\left\{y \in \{0,1\}^{\leq s}\colon A(xy, 1^{q(t,|x|)}, 1^{s+c_0\log|x|}) = \text{Yes}\right\}\right| \\
&\leq \left|\left\{y \in \{0,1\}^{\leq s}\colon \mathrm{K}^{\tau(q(t,|x|),|x|)}(xy) \leq s + c_0\log|x| + \log\tau(q(t,|x|),|x|)\right\}\right| \\
&\leq \left|\left\{y \in \{0,1\}^{\leq s}\colon \mathrm{K}(xy) \leq s + c_0\log|x| + \log\tau(q(t,|x|),|x|)\right\}\right| \\
&\leq 2^{s+c_0\log|x|+\log\tau(q(t,|x|),|x|)+1-\mathrm{K}(x)} \cdot \mathrm{poly}(|x|+s) \\
&= 2^{s-\mathrm{K}(x)} \cdot \mathrm{poly}(|x|, t, s),
\end{aligned}$$

where the second inequality holds by the correctness of A, the third since $\mathrm{K}(xy) \leq \mathrm{K}^t(xy)$ and the last inequality holds by Lemma 21. ◁

3.2 A List-to-Decision Reduction for K-complexity

We note that Algorithm 19 also gives "list-to-decision" reduction for Kolmogorov-complexity: given access to an oracle that decides the threshold problem of Kolmogorov-complexity, a simple variant of Algorithm 19 outputs a list of polynomial length, containing the witness.

▶ **Theorem 22.** *There exists an efficient oracle-aided algorithm F such that the following holds. Let A be an oracle that given $x \in \{0,1\}^*$ and $s \in \mathbb{N}$, decides if $\mathrm{K}(x) \leq s$. Then F^A outputs a list $\mathcal{L}$, such that $|\mathcal{L}| \in \mathrm{poly}(|x|)$, and $\mathcal{L}$ contains a K-witness for x: that is, there exists $\Pi \in \mathcal{L}$ for which $|\Pi| = \mathrm{K}(x)$ and $\mathsf{U}(\Pi) = x$.*

As in Theorem 15, the same holds when the oracle A only solves the gap version of the threshold problem (given x and s, A decides if $\mathrm{K}(x) \leq s$ or $\mathrm{K}(x) \geq s + O(\log|x|)$.)

The algorithm is as follows.

▶ **Algorithm 23** (F).
Oracle: A.
Input: $x \in \{0,1\}^*$.
1. *Use A to compute $s = \mathrm{K}(x)$.*
2. *Set $\mathcal{S}_0 = \{x\}$ and $k = s + c_0 \log|x|$.*
3. *For every $i = 1, 2, \ldots, s$:*
   a. *Compute $\mathcal{S}_i = \{yb : y \in \mathcal{S}_{i-1}, b \in \{0,1\}$ and $A(yb, 1^k) = \mathrm{Yes}\}$*
4. *Output $\mathcal{S}_s$.*

Proof of Theorem 22. Fix an input $x \in \{0,1\}^*$, and let Π be a K-witness for x. By Claim 18 and simple induction, there exists a constant c_0 such that $\Pi \in \mathcal{S}_s$. By Lemma 21, it holds that $|\mathcal{S}_s| \in \mathrm{poly}(|x|, \mathrm{K}(x)) = \mathrm{poly}|x|$.

Finally, by Lemma 21 we get that $|\mathcal{S}_i| \in \mathrm{poly}(|x|, i)$, which implies that Algorithm 23 runs in polynomial time. ◀

4 Decision-to-Search Everywhere

In this part we prove our main search-to-decision reduction for GapMINKT.

▶ **Theorem 24.** *Let $\varepsilon > 0$ be a constant. Then there exists a randomized oracle-aided algorithm F such that the following holds for every $\tau \in \mathrm{poly}$. Let A be an oracle that decides $\mathrm{Gap}_\tau\mathrm{MINKT}$. Then $F^A_\tau = F^A(\tau, \cdot, \cdot, \cdot)$ solves* search- GapMINKT.

Moreover, on input $(x, 1^t, 1^s)$ F^A_τ runs in time $2^{\epsilon s}\mathrm{poly}(|x|, t, s)$, and only queries A on inputs $(y, 1^{t'}, 1^{s'})$ with $|y| = \mathrm{poly}(|x|, t, s)$, $t' = \mathrm{poly}(|x|, t, s)$, and $s' \leq s + \log \mathrm{poly}(|x|, t, s)$.

Directly from Theorem 24 we get the following corollary.

▶ **Corollary 25.** *The following holds for every $\tau \in \mathrm{poly}$ and $\epsilon > 0$: Assume that for some $\alpha > 0$ there is an algorithm that solves $\mathrm{Gap}_\tau\mathrm{MINKT}$ in time (resp. size) $2^{\alpha \cdot s}\mathrm{poly}(|x|, t, s)$. Then there exists an algorithm that solves* search- GapMINKT *in time (resp. size) $2^{(\alpha+\epsilon)s} \cdot \mathrm{poly}(|x|, t, s)$ on inputs $(x, 1^t, 1^s)$.*

We next prove Theorem 24. In the following, let $q \in \text{poly}$, $c_0 \in \mathbb{N}$ be the polynomial and constant from Claim 18, and let c be the constant from Theorem 12. We start with the following algorithm, that with high probability outputs a program Π such that x is a prefix of the output of Π. We later change the algorithm such that the output will be a program that outputs x.

▶ **Algorithm 26** (Find).
Parameters: $\epsilon > 0, \tau \in \text{poly}$
Oracle: $\text{Gap}_\tau\text{MINKT}$ *decider* A.
Input: $(x, 1^t, 1^s)$ *for* $x \in \{0,1\}^*$, $t, s \in \mathbb{N}$.

1. *Set* $x^1 = x$, $t^1 = t$ *and* $s^1 = s$.
2. *For every* $j = 1, \ldots, \lceil 1/\varepsilon \rceil + 1$*:*
   a. *Set* $\mathcal{S}_0^j = \{x^j\}$, *and* $k^j = s^j + c_0 \log |x^j|$.
   b. *Set* $r^j = \epsilon s + (k^j - s^j) + c \log(|x^j| + s^j) + \log \tau(q(t^j, |x^j|), |x^j| + s^j) + \log 4/\epsilon$.
   c. *For every* $i = 1, 2, \ldots, s^j$*:*
      i. *Compute* $\mathcal{S}_i^j = \left\{ yb \colon y \in \mathcal{S}_{i-1}^j, b \in \{0,1\} \text{ and } A(yb, 1^{q(t^j, |x^j|)}, 1^{k^j}) = \text{Yes} \right\}$
      ii. *If exists* $y \in \mathcal{S}_i^j$ *such that* $y = x^j || \Pi$ *and* $\mathsf{U}(\Pi, 1^{t^j}) = x_j$, *output* Π *and terminate.*
      iii. *If* $\left|\mathcal{S}_i^j\right| \geq 2^{r^j}$, *set* $\mathcal{S}^j = \mathcal{S}_i^j$ *and move to Item 2d.*
   d. *Randomly choose* $x^{j+1} \leftarrow \mathcal{S}^j$.
   e. *Set* $t^{j+1} = \tau(q(t^j, |x^j|), |x^{j+1}|)$ *and* $s^{j+1} = k^j + \log \tau(q(t^j, |x^j|), |x^{j+1}|)$.
3. *Output* $\perp$.

We start with a simple observation on the parameters in Claim 28.

▷ Claim 27. For every $j \leq \lceil 1/\epsilon \rceil + 1$ it holds that $t^j \in \text{poly}(|x|, t, s)$, $|x^j| \in \text{poly}(|x|, t, s)$, $s^j = s + \log(\text{poly}(|x|, t, s))$, $k^j = s + \log(\text{poly}(|x|, t, s))$, and $r^j = \epsilon \cdot s + \log(\text{poly}(|x|, t, s))$.

Proof of Claim 27. The claim holds since ϵ is a constant, τ and q are polynomials, and by the definition of t^j, x^j, s^j, k^j and r^j. ◁

We next bound the running time of Algorithm 26.

▷ Claim 28. On input $(x, 1^t, 1^s)$, Algorithm 26 runs in time $2^{\epsilon s} \cdot \text{poly}(|x|, t, s)$. Moreover, Algorithm 26 only queries A on inputs $(y, 1^{t'}, 1^{s'})$ with $|y| = \text{poly}(|x|, t, s)$, $t' = \text{poly}(|x|, t, s)$, and $s' \leq s + \log \text{poly}(|x|, t, s)$.

Proof of Claim 28. Fix an input $(x, 1^t, 1^s)$. Similarly to the proof of Claim 20, the running time of the j-th iteration in Step 2 of Algorithm 26 is at most

$$\left| \bigcup_i \mathcal{S}_i^j \right| \cdot \text{poly}(|x^j|, t^j, s^j) \leq s^j \cdot 2^{r^j} \cdot \text{poly}(|x^j|, t^j, s^j).$$

It thus enough to show that $t^j \in \text{poly}(|x|, t, s)$, $|x^j| \in \text{poly}(|x|, t, s)$, $s^j = s + \log(\text{poly}(|x|, t, s))$, and $r^j = \epsilon \cdot s + \log(\text{poly}(|x|, t, s))$ for every $j \leq \lceil 1/\epsilon \rceil + 1$, which holds by Claim 27. ◁

To see that Algorithm 26 indeed outputs a program that outputs x, we have the following claim.

▷ Claim 29. There exists a constant $c \in \mathbb{N}$ such that the following holds. Assume that on input $(x, 1^t, 1^s)$, Algorithm 26 enters the j-th iteration in Step 2. Then,

1. x is a prefix of x^j,
2. $\mathrm{K}^{t^j}(x^j) \leq s^j$, and,
3. With probability at least $1 - j \cdot \epsilon/4$, $\mathrm{K}(x^j) \geq (j-1) \cdot \epsilon \cdot s + (s^j - s)$.

Proof of Claim 29. The first item holds by the definition of the algorithm. The second item holds since in Algorithm 26, x^j is in the set $\mathcal{S}_i^{j-1}$ only if $A(x^j, 1^{q(t^{j-1}, |x^{j-1}|)}, 1^{k^{j-1}}) = \text{Yes}$, which implies by the correctness of A that

$$\mathrm{K}^{\tau(q(t^{j-1}, |x^{j-1}|), |x^j|)}(x^j) \leq k^{j-1} + \log \tau(q(t^{j-1}, |x^{j-1}|), |x^j|),$$

and since $t^j = \tau(q(t^{j-1}, |x^{j-1}|), |x^j|)$, $s^j = k^{j-1} + \log \tau(q(t^{j-1}, |x^{j-1}|), |x^j|)$.

The proof of the last item is by induction on j. Assume that

$$K(x^j) \geq (j-1) \cdot \epsilon \cdot s + (s^j - s).$$

Observe that by definition of the algorithm, it holds that x^j is a prefix of every element in $\mathcal{S}^j$. That is, we can write $\mathcal{S}^j = x^j || \mathcal{S}'^j$ for a set $\mathcal{S}'^j$ of size at least 2^{r^j}.

Thus, $x^{j+1} = x^j || z$, for $z \in \{0,1\}^{\leq s^j}$ which is randomly chosen from a set of size at least 2^{r^j}. Using Lemma 9, it holds that with probability at least $1 - \epsilon/4$ that $\mathrm{K}(z \mid x^j) \geq r^j - \log 4/\epsilon$. by Symmetry of Information (Theorem 12) we get that

$$\begin{aligned}
\mathrm{K}(x^{j+1}) &\geq \mathrm{K}(x^j) + r^j - \log 4/\epsilon - c \log(|x^j| + s^j) \\
&\geq (j-1) \cdot \epsilon \cdot s + (s^j - s) + (\epsilon s + (k^j - s^j) + \log \tau(q(t^j, |x^j|), |x^j| + s^j)) \\
&\geq j \cdot \epsilon \cdot s + (k^j + \log \tau(q(t^j, |x^j|), |x^j| + s^j)) - s \\
&\geq j \cdot \epsilon \cdot s + (k^j + \log \tau(q(t^j, |x^j|), |x^{j+1}|)) - s \\
&= j \cdot \epsilon \cdot s + s^{j+1} - s
\end{aligned}$$

The proof now follows by the union bound. ◁

We can now prove Theorem 24.

Proof of Theorem 24. We start with the definition of the algorithm F. Let F be the algorithm that first executes Algorithm 26, to get a program Π such that $\mathsf{U}(\Pi)_{\leq |x|} = x$. Then, F outputs a program Π' that simulates Π, and outputs the $|x|$ first bits of its output. It follows that $\mathsf{U}(\Pi') = x$, and $|\Pi'| \leq |\Pi| + O(\log |x|)$. Moreover, the running time of Π' is bounded by a polynomial of the running time of Π.

Next, by Claim 28, F runs in the stated time. We now prove the correctness of F. First, recall that by Fact 7, we can assume without loss of generality that $s \leq |x| + O(1)$. Next, observe that by Claim 29, in every iteration j it holds that x is a prefix of x^j, and that $\mathrm{K}^{t^j}(x^j) \leq s^j$. Thus, by the correctness of Algorithm 19, if for every iteration i, $\mathcal{S}_i^j$ is not larger than 2^{r^j}, then Algorithm 26 outputs a program of length at most

$$s^j = s + \log \mathrm{poly}(|x|, t, s) = s + \log \mathrm{poly}(|x|, t)$$

that outputs x^j in time

$$t^j = \mathrm{poly}(|x|, t, s) = \mathrm{poly}(|x|, t).$$

We are left to deal with the case that in every iteration j of Step 2, $\mathcal{S}_i^j$ is larger than 2^{r^j} for some i. In this case, by the third item of Claim 29, with probability at least $1 - (\lceil 1/\epsilon \rceil + 1)\epsilon/4 \geq 1/2$, in the last iteration we get that

$$\mathrm{K}(x^{\lceil 1/\epsilon \rceil + 1}) \geq (\lceil 1/\epsilon \rceil + 1)\epsilon s + (s^{\lceil 1/\epsilon \rceil + 1} - s) > s^{\lceil 1/\epsilon \rceil + 1},$$

which is in contradiction to the second item of Claim 29, as $\mathrm{K}(x^{\lceil 1/\epsilon \rceil + 1}) \leq \mathrm{K}^t(x^{\lceil 1/\epsilon \rceil + 1})$ for every $t \in \mathbb{N}$. ◀

5 Decision-to-Search for MINKT using List Recoverable Codes

In this part we use list recoverable codes to get length-preserving decision-to-search for instances with small computational depth. List recoverable codes [3] are defined next.

▶ **Definition 30.** *For $\Sigma = \{\Sigma_n\}_{n\in\mathbb{N}}$ and functions $m\colon \mathbb{N}\to\mathbb{N}$, $p\colon \mathbb{N}\to[0,1]$, $\ell\colon \mathbb{N}\to\mathbb{N}$ and $L\colon \mathbb{N}\to\mathbb{N}$, an ensemble $\mathsf{Enc} = \{\mathsf{Enc}_n\colon \{0,1\}^n \to \Sigma_n^{m(n)}\}_{n\in\mathbb{N}}$ is an efficient (p,ℓ,L)-list-recoverable code if Enc can be computed in polynomial time, and there exists an efficient procedure Rec such that the following holds for every $n\in\mathbb{N}$. Given sets $\mathcal{S}_1,\ldots,\mathcal{S}_{m(n)}\subseteq\Sigma_n$ such that $|\mathcal{S}_i|\le\ell(n)$ for every i, $\mathsf{Rec}(\mathcal{S}_1,\ldots,\mathcal{S}_{m(n)})$ outputs a list $\mathcal{R}$ of size at most $L(n)$, containing all $x\in\{0,1\}^n$ with*

$$|\{i\in[m(n)]\colon \mathsf{Enc}(x)_i\notin\mathcal{S}_i\}|\le p(n)\cdot m(n).$$

As shown in [5, 4], Reed-Solomon codes [19] are list recoverable with parameters that are suitable for our needs. In particular:

▶ **Theorem 31** ([5, 4])**.** *For every efficiently computable $w\in O(\log n)$, there exists an efficient $\mathsf{Enc}\colon\{0,1\}^n\to[m(n)]^{m(n)}$, for $m(n)=2^{w(n)}$, such that Enc is an (p,ℓ,L) list recoverable code , for any $p\le 1-\sqrt{\ell(n)\cdot n/m(n)}$ and $L(n)\in O(\ell(n)\cdot m(n))$.*

Moreover, $\mathsf{Enc}\colon\{0,1\}^n\to[m(n)]^{m(n)}$ runs in time $\mathrm{poly}(n,w(n))$.

We now state the main theorem of this part.

▶ **Theorem 32.** *For every $d\in\mathbb{N}$, there exists an oracle-aided algorithm F such that the following holds. Let A be an oracle that decides* MINKT*. Then F^A solves* search-MINKT *on inputs $(x,1^t,1^s)$ with $cd^t(x)\le d\log|x|$.*

Moreover, on input $(x,1^t,1^s)$, F runs in time $\mathrm{poly}(|x|,t,s)$, and only queries A on inputs $(x||y,1^{t'(|x|,t)},1^{s'})$ with $|y|=r(|x|,t)\le\log\mathrm{poly}(|x|,t)$, $t'\in\mathrm{poly}$ and $s'\le s+\log\mathrm{poly}(|x|,t)$.

In the following, let $\mathsf{Enc}=\{\mathsf{Enc}_n\colon\{0,1\}^n\to\Sigma_n^{m(n)}\}$ be an efficient (p,ℓ,L) list recoverable code, and Rec the efficient reconstruction algorithm of Enc, for parameters $\Sigma_n, m(n), p, \ell, L$ we will choose later. Let $q\in\mathrm{poly}$ and $c_0,c_1\in\mathbb{N}$ be a polynomial and constants to be chosen later. We will show that the following algorithm returns a short program that produces x, when the input s is exactly equal to $\mathrm{K}^t(x)$, and then show how to get rid of this assumption. For $i\in[m(n)]$, we let $\langle i\rangle\in\{0,1\}^{\lceil\log m(n)\rceil}$ be the binary representation of i. Let c_K be the constant from Fact 7 such that $\mathrm{K}^t(x)\le|x|+c_K$ for every x.

▶ **Algorithm 33** (Find)**.**
Oracle: MINKT decider A.
Parameters: $\tau\in\mathrm{poly}$, $d\in\mathbb{N}$.
Input: $(x,1^t,1^s)$ for $x\in\{0,1\}^*$, $t,s\in\mathbb{N}$.

1. *Let $n=|x|+c_K$.*
2. *For every $i\in[m(n)]$, compute the set*

$$\mathcal{S}_i=\left\{y\in\Sigma_n\colon A(x||\langle i\rangle||y,1^{q(t,|x|)},1^{s+\log m(n)+c_1\log\log m(n)})=\mathrm{Yes}\right\}.$$

3. *For every $i\in[m(n)]$ such that $|\mathcal{S}_i|>\ell(n)$, set $\widehat{\mathcal{S}}_i=\emptyset$. Otherwise, set $\widehat{\mathcal{S}}_i=\mathcal{S}_i$.*
4. *Compute $\mathsf{Rec}(\widehat{\mathcal{S}}_1,\ldots,\widehat{\mathcal{S}}_{m(n)})$, and let $\mathcal{R}\subseteq\{0,1\}^n$ be the output.*
5. *Find a string $\Pi\in\{0,1\}^s$ such that $\Pi||0^{n-s}\in\mathcal{R}$ and $\mathsf{U}(\Pi,1^t)=x$.*
6. *Output Π.*

Note that in the above algorithm, the set $\mathcal{S}_i$ is the set of all possible values (in Σ_n) for the ith symbol in the encoding of Π. We start by showing that the size of $\mathcal{S}_1, \ldots, \mathcal{S}_{m(n)}$ is not too large.

▷ Claim 34. There exists a constant $c_2 \in \mathbb{N}$ such that the following holds for every $c_1, d, w \in \mathbb{N}$. Let $x \in \{0,1\}^*$ such that $cd^t(x) \leq d \log|x|$. Then there are at most

$$M = 2^{w+(c_1+c_2)(\log 2w)+d\log|x|+c_2\log|x|}$$

pairs $(i, \alpha) \in [2^w] \cdot [2^w]$ such that $\mathrm{K}(x||\langle i\rangle||\alpha) \leq \mathrm{K}^t(x) + w + c_1(\log w)$.

Proof. Immediate by Lemma 21. ◁

Next, we will use the following claim to show that Π is in the set $\mathcal{R}$.

▷ Claim 35. For every $w\colon \mathbb{N} \to \mathbb{N}$, and every efficient code $\mathsf{Enc}\colon \{0,1\}^n \to [2^{w(n)}]^{2^{w(n)}}$, there exists a polynomial q such that the following holds for every $x \in \{0,1\}^*$ with $w(|x|) \geq 2$, and every $t \in \mathbb{N}$. Let Π a program of length $\mathrm{K}^t(x)$ such that $\mathsf{U}(\Pi, 1^t) = x$. Then for $n = |x| + c_K$ and every $i \in [2^{w(n)}]$,

$$\mathrm{K}^{q(t,|x|)}(x||\langle i\rangle||\mathsf{Enc}(\Pi||0^{n-|\Pi|})_i) \leq \mathrm{K}^t(x) + w(n) + c_1 \log w(n)$$

for some universal constant c_1.

Proof. Let Π' be the program that given input (Π, i), first simulates $\mathsf{U}(\Pi)$ to get an output x, and then computes $n = |x| + c_K$ and $\mathsf{Enc}(\Pi||0^{n-|\Pi|})_i$. Let $q \in$ poly be the bound of the running time of Π'. Then, using Fact 8, $\mathrm{K}^{q(t,|x|)}(x||\langle i\rangle||\mathsf{Enc}(\Pi)_i) \leq \mathrm{K}^t(x) + w(n) + 4\log w(n)$. ◁

We are now ready to prove Theorem 32.

Proof of Theorem 32. We will show that on input $(x, 1^t, 1^{s'})$ Algorithm 33 returns a program Π such that $\mathsf{U}(\Pi, 1^t) = x$, if $s' = \mathrm{K}^t(x)$. The theorem then follows by considering the algorithm that enumerates over all possible values of $s' < s$.

Let c_1 and c_2 be the constants promised by Claim 35 and Claim 34 respectively. Let $\mathsf{Enc}\colon \{0,1\}^n \to [2^{w(n)}]^{m(n)}$ be an efficient (p, ℓ, L) list recoverable code, for $w(n) = 2(d + c_1 + c_2)\log n + 10$, $p(n) = 1/10$, $\ell(n) = 2^{(c_1+c_2)\log w(n)+(d+c_2)\log n+5}$, $L(n) \in$ poly and $m(n) = 2^{w(n)}$. By Theorem 31 such a code exists as

$$\ell(n) \cdot n/m(n) = n \cdot 2^{(c_1+c_2)\log w(n)-(d+2c_1+c_2)\log n-5} \leq 2^{-5} \leq (1-p)^2$$

for any large enough n. Finally, let q be the polynomial promised by Claim 35 with respect to the code Enc.

We next show that when using Enc as the code in Algorithm 33, Algorithm 33 outputs a minimal program Π that produces the input x. By the list recoverable property of Enc, it is enough to show that $\Pi||0^{n-s}$ is in the list outputted by Rec. It thus enough to show that (1) $\mathsf{Enc}(\Pi)_i \in \mathcal{S}_i$ for every $i \in [m(n)]$ and (2), $\mathcal{S}_i = \widehat{\mathcal{S}}_i$ for at least $(1 - 1/10)m(n)$ indexes $i \in [m(n)]$.

(1) follows immediately by Claim 35 and the definition of $\mathcal{S}_i$. For (2), by Claim 34 the total size $\sum_i |\mathcal{S}_i|$ of the sets $\mathcal{S}_i$ is at most

$$2^{w+(c_1+c_2)\log 2w+d\log n+c_2\log n} = m(n) \cdot \ell(n) \cdot 2^{-5}.$$

By Markov, we get that there are at most $2^{-5} \cdot m(n) < 1/10 \cdot m(n)$ indexes i such that $|\mathcal{S}_i| > \ell(n)$. For all other i's it holds that $\mathcal{S}_i = \widehat{\mathcal{S}}_i$, which concludes the proof. ◀

5.1 Decision-to-Search on Average

▶ **Definition 36.** *For a function $t\colon \mathbb{N} \to \mathbb{N}$, we say that an algorithm A computes K^t if for every $x \in \{0,1\}^*$, $A(x) = \mathrm{K}^{t(|x|)}(x)$. We say that A computes K^t with error ϵ if for every $n \in \mathbb{N}$, $\Pr_{x \leftarrow \{0,1\}^n}\left[A(x) = \mathrm{K}^{t(|x|)}(x)\right] \geq 1 - \epsilon(n)$.*

We say that A solves search-K^t *with error ϵ if A outputs a program Π such that $\mathsf{U}(\Pi, 1^{t(|x|)}) = x$ and $|\Pi| = \mathrm{K}^{t(|x|)}(x)$ with probability $1 - \epsilon(n)$ over $x \leftarrow \{0,1\}^n$.*

We prove the following theorem.

▶ **Theorem 37.** *For every $p, t \in \mathrm{poly}$ there exists $\widehat{p} \in \mathrm{poly}$ and $t' \in \mathrm{poly}$, and an efficient oracle-aided algorithm F such that the following holds. Let A be an that computes $\mathrm{K}^{t'}$ with error $1/\widehat{p}$. Then F^A solves* search-K^t *with error $1/p$.*

Moreover, on input $x \in \{0,1\}^n$, F makes only queries of the form $x||y$ with $y \in O(\log n)$.

To prove Theorem 37, we will use the following theorem, which is followed by the same proof as Theorem 32.

▶ **Theorem 38.** *For every $d \in \mathbb{N}$ and $t \in \mathrm{poly}$, there exists $t' \in \mathrm{poly}$ and an oracle-aided algorithm F such that the following holds. Let A be an oracle that computes $\mathrm{K}^{t'}$. Then F^A solves* search-K^t *on inputs $(x, 1^t, 1^s)$ with $cd^t(x) \leq d\log|x|$.*

Moreover, on input x, F runs in time $\mathrm{poly}(|x|)$, and only queries A on inputs $x||y$ with $|y| = r(|x|, t) \leq \log \mathrm{poly}(|x|, t)$.

Proof. This follows by a similar proof to Theorem 32, together with the observation that MINKT can be decided on inputs $(x, 1^{t(|x|)}, 1^s)$ using oracle that computes K^t. ◀

Proof of Theorem 37. We start with the assumption that the oracle A is deterministic, and later show how to eliminate this assumption. Fix p and t, and let t' and F be the polynomial and algorithm promised by Theorem 32. Let r be the parameters of F as defined in Theorem 32, and let d be such that $|x|^d > 2p(|x|)$ for every x with $|x| > 2$. Finally, let $\widehat{p}(|x|) = 8p(|x|) \cdot 2^{r(|x|,t(|x|))}$, and let A be an oracle the computes $\mathrm{K}^{t'}$ with error $1/\widehat{p}$ (when the probability is taken only over the input x of A).

Let $\widehat{A}$ be an oracle that computes $\mathrm{K}^{t'}$ correctly on every input. By Definition 36, on a random $x \leftarrow \{0,1\}^n$ A and $\widehat{A}$ agree with probability $1 - 1/\widehat{p}(x)$. By Theorem 32, $F^{\widehat{A}}$ solves search-K^t on every input with $cd^{t(|x|)}(x) \leq d\log|x|$, and thus, by Fact 11, $F^{\widehat{A}}$ solves search-K^t with error at most $1/|x|^d \leq 1/(2p(|x|))$. It is thus enough to show that

$$\Pr_{x \leftarrow \{0,1\}^n}\left[F^A(x) = F^{\widehat{A}}(x)\right] \geq 1 - 1/2p(|x|). \tag{1}$$

To see Equation (1), let $\mathcal{B}_n$ be the set of all $x \in \{0,1\}^n$ such that $F^A(x) \neq F^{\widehat{A}}(x)$. By definition of $\mathcal{B}$, on every input $x \in \mathcal{B}_n$, there must be some query $x||y$ that F makes to the oracle, such that $A(x||y) \neq \widehat{A}(x||y)$. Let $\mathcal{R}_n \subseteq \{0,1\}^n$ be the set of all x's, such that there exists $y \in \{0,1\}^{r(n,t(n))}$ with $A(x||y) \neq \widehat{A}(x||y)$. We get that $|\mathcal{R}_n| \geq |\mathcal{B}_n|$, and,

$$1/\widehat{p}(n + r(n, t(n))) \geq \Pr_{z \leftarrow \{0,1\}^{n+r(n,t(n))}}\left[A(z) \neq \widehat{A}(z)\right] \geq |\mathcal{R}_n| \cdot 2^{-n-r(n,t(n))}.$$

Thus, we get that

$$|\mathcal{B}_n| \leq 2^{n+r(n,t(n))+1} \cdot 1/\widehat{p}(n + r(n, t(n))) \leq 2^{n+r(n,t(n))+1} \cdot 1/\widehat{p}(n),$$

which implies that $|\mathcal{B}_n|/2^n \leq 1/4p(n)$, which concludes the claim.

We next move to deal with randomized algorithm A. That is, A that err with probability $1/\widehat{p}$, where the probability is now taken both over the input and the internal randomness of A. To deal with such a randomized oracle A, we observe that by standard amplification, it is enough to replace Equation (1), with $\Pr_{x\leftarrow\{0,1\}^n}\left[\Pr\left[F^A(x)\neq F^{\widehat{A}}(x)\right]>1/2\right]\leq 1/4p(|x|)$. We can thus let $\mathcal{B}_n$ be the set of all $x\in\{0,1\}^n$ such that $\Pr\left[F^A(x)\neq F^{\widehat{A}}(x)\right]>1/2$, and $\mathcal{R}_n\subseteq\{0,1\}^n$ be the set of all x's, such that with probability larger than $1/2$ (over the randomness of A) there exists $y\in\{0,1\}^{r(n,t(n))}$ with $A(x||y)\neq\widehat{A}(x||y)$. We get that $|\mathcal{R}_n|\geq|\mathcal{B}_n|$, and,

$$1/\widehat{p}(n+r(n,t(n)))\geq\Pr_{z\leftarrow\{0,1\}^{n+r(n,t(n))}}\left[A(z)\neq\widehat{A}(z)\right]\geq 1/2\cdot|\mathcal{R}_n|\cdot 2^{-n-r(n,t(n))},$$

and the claim follows as in the deterministic case. ◀

We get the following two corollaries. The first was already proven in [16].

▶ **Corollary 39** (reproving [16]). *For every $p,t\in$ poly there exists $\widehat{p},t'\in$ poly such that the following holds:*

- *Assume that there is a poly-time (resp. poly size) algorithm that computes $\mathrm{K}^{t'}$ with error $1/\widehat{p}$. Then there exists a poly-time (resp. poly-size) algorithm that solves search-K^t with error $1/p$.*
- *Assume that there is an algorithm that computes $\mathrm{K}^{t'}(x)$ with error $1/\widehat{p}$ in time (resp. size) $2^{o(|x|)}$. Then there exists an algorithm that solves search-K^t with error $1/p$ in time (resp. size) $2^{o(|x|)}$.*

The second shows that the same holds also in the exponential regime.

▶ **Corollary 40.** *For every $p,t\in$ poly there exists $\widehat{p},t'\in$ poly such that the following holds: Assume that there exists a constant $\alpha>0$ and an algorithm that computes $\mathrm{K}^{t'}$ with error $1/\widehat{p}$ in time (resp. size) $2^{\alpha\cdot|x|}\mathrm{poly}(|x|)$. Then there exists an algorithm that solves search-K^t with error $1/p$ in time (resp. size) $2^{\alpha\cdot|x|}\mathrm{poly}(|x|)$.*

6 Decision-to-Search for GapMINKT using List Recoverable Codes

In this part we show that Theorem 32 holds even when we start with an oracle that solves GapMINKT instead of MINKT.

▶ **Theorem 41.** *For every $\tau\in$ poly and $d\in\mathbb{N}$, there exists an oracle-aided algorithm F such that the following holds. Let A be an oracle that decides Gap_τMINKT. Then F^A solves search-MINKT on inputs $(x,1^t,1^s)$ with $cd^t(x)\leq d\log|x|$.*

Moreover, on input $(x,1^t,1^s)$, F runs in time $\mathrm{poly}(|x|,t,s)$, and only queries A on inputs $(x||y,1^{t'(|x|,t)},1^{s'})$ with $|y|=\ell(|x|,t)\leq\log\mathrm{poly}(|x|,t)$, $t'\in$ poly and $s'(|x|,t)\leq s+\log\mathrm{poly}(|x|,t)$.

To prove Theorem 41 we will show that Algorithm 33 with different parameters works. When the oracle A only decides GapMINKT, the size of the sets $\mathcal{S}_i$ can be larger. The first claim shows that it still cannot be too large.

▷ Claim 42. There exists a constant $c_2\in\mathbb{N}$ such that the following holds for every $c_1,d,w\in\mathbb{N}$, and every $\tau,q\in$ poly. Let $x\in\{0,1\}^*$ such that $cd^t(x)\leq d\log|x|$. Then there are at most

$$M=2^{w+(c_1+c_2)(\log 2w)+d\log|x|+c_2\log|x|)+\log\tau(q(t,|x|),|x|)}$$

pairs $(i,\alpha)\in[2^w]\cdot[2^w]$ such that $\mathrm{K}(x||\langle i\rangle||\alpha)\leq\mathrm{K}^t(x)+w+c_1(\log w)+\log\tau(q(t,|x|),|x|)$.

Proof. Immediate by Lemma 21. ◁

To get non-trivial bound from Claim 42, we will need to choose out code alphabet size, 2^w, to be larger than $\tau(q(t,|x|),|x|)$. Since the polynomial q is going to be dependent on the code evaluation time, we will need to use codes where the running time of the code does not grow too fast with the alphabet size. By Theorem 31, there is a family of list recoverable codes, such that we can choose the size of the output alphabet Σ, and the running time of the encoder only depends on $\log|\Sigma|$. This property is used in the next claim, which shows that $\Pi||0^{n-s}$ is in the set $\mathcal{R}$.

▷ Claim 43. There exists a constant c_1 and polynomial q', such that the following holds for every $w\colon \mathbb{N}\to\mathbb{N}$, every $x\in\{0,1\}^*$ with $|x|\geq 16$, and every $t\in\mathbb{N}$. Let Π a program of length $\mathrm{K}^t(x)$ such that $\mathsf{U}(\Pi,1^t)=x$, and let $\mathsf{Enc}\colon\{0,1\}^n\to[2^{w(n)}]^{2^{w(n)}}$ be the code from Theorem 31. Then for $n=|x|+c_K$ and for every $i\in[2^{w(n)}]$,

$$\mathrm{K}^{q'(t,|x|,w(n))}(x||\langle i\rangle||\mathsf{Enc}(\Pi||0^{n-|\Pi|})_i)\leq \mathrm{K}^t(x)+w(n)+c_1\log w(n).$$

Proof. The proof is similar to the proof of Claim 35, using the observation that the running time of Π' is polynomial in $t,|x|$ and $w(n)$ by the choice of Enc. ◁

We are now ready to prove Theorem 41.

Proof of Theorem 41. We will show that on input $(x,1^t,1^{s'})$ Algorithm 33 returns a program Π such that $\mathsf{U}(\Pi,1^t)=x$, if $s'=\mathrm{K}^t(x)$. The theorem then follows by considering the algorithm that enumerates over all possible values of $s'<s$.

Let c_1 and q' be the constant and polynomial promised by Claim 43, and let c_2 be the constant from Claim 42. Let c_0 be such that for $w(n)=c_0\lceil\log n+t(n)\rceil$,

$$\begin{aligned} &w(|x|+c_K) \\ &\geq 2((c_1+c_2)\log w(|x|+c_K)+(d+c_2)\log|x| \\ &\quad+\log\tau(q'(t(|x|),n,w(|x|+c_K)),|x|+2w(|x|+c_K))), \end{aligned} \tag{2}$$

and, $2^{w(|x|+c_K)/3}>|x|$. Finally, let $q(t,|x|)=q'(t,|x|,w)$.

The proof now continues similar to the proof of Theorem 32. ◀

References

1. Luis Antunes, Lance Fortnow, Dieter Van Melkebeek, and N Variyam Vinodchandran. Computational depth: concept and applications. *Theoretical Computer Science*, 354(3):391–404, 2006.
2. Gregory J. Chaitin. On the simplicity and speed of programs for computing infinite sets of natural numbers. *J. ACM*, 16(3):407–422, 1969.
3. Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 658–667. IEEE, 2001.
4. Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. *Draft available at* `https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/web-coding-book.pdf`, 2(1), 2012.
5. Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometric codes. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 28–37. IEEE, 1998.
6. J. Hartmanis. Generalized kolmogorov complexity and the structure of feasible computations. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 439–445, 1983. `doi:10.1109/SFCS.1983.21`.

7 Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, pages 1364–1396, 1999.

8 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within np. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 247–258. IEEE, 2018.

9 Shuichi Hirahara. Non-disjoint promise problems from meta-computational view of pseudorandom generator constructions. In *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

10 Shuichi Hirahara. Symmetry of information from meta-complexity. In *37th Computational Complexity Conference (CCC 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

11 Shuichi Hirahara, Rahul Ilango, and Ryan Williams. Beating brute force for compression problems. Technical Report TR23-171, Electronic Colloquium on Computational Complexity, 2023.

12 Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 73–79, 2000.

13 Ker-I Ko. On the notion of infinite pseudorandom sequences. *Theor. Comput. Sci.*, 48(3):9–33, 1986. doi:10.1016/0304-3975(86)90081-2.

14 A. N. Kolmogorov. Three approaches to the quantitative definition of information. *International Journal of Computer Mathematics*, 2(1-4):157–168, 1968.

15 Leonid A. Levin. Universal'nyĭe perebornyĭezadachi (Universal search problems : in Russian). *Problemy Peredachi Informatsii*, pages 265–266, 1973.

16 Yanyi Liu and Rafael Pass. On one-way functions and kolmogorov complexity. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1243–1254. IEEE, 2020.

17 Noam Mazor and Rafael Pass. The non-uniform perebor conjecture for time-bounded kolmogorov complexity is false. *15th Innovations in Theoretical Computer Science*, 2024.

18 Noam Mazor and Rafael Pass. Search-to-decision reductions for kolmogorov1 complexity. *Electronic Colloquium on Computational Complexity*, 2024.

19 Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

20 Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 330–335, 1983.

21 R.J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7(1):1–22, 1964. doi:10.1016/S0019-9958(64)90223-2.

22 Boris A Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

23 Sergey Yablonski. The algorithmic difficulties of synthesizing minimal switching circuits. *Problemy Kibernetiki*, 2(1):75–121, 1959.

24 Sergey V Yablonski. On the impossibility of eliminating perebor in solving some problems of circuit theory. *Doklady Akademii Nauk SSSR*, 124(1):44–47, 1959.

25 A. C. Yao. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

26 Alexander K Zvonkin and Leonid A Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83, 1970.