Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 49 (2021)

Kolmogorov complexity and nondeterminism versus determinism for polynomial time computations

Juraj Hromkovic

Dept. of Computer Science, ETH Zurich
Universitätsstrasse 6, CAB, 8092 Zurich, Switzerland
juraj.hromkovic@inf.ethz.ch

February 2021

Abstract

We call any consistent and sufficiently powerful formal theory that enables to algorithmically verify whether a text is a proof **algorithmically verifiable mathematics** (av-mathematics). We study the question whether nondeterminism is more powerful than determinism for polynomial time computations in the framework of av-mathematics. Our main results are as follows.
"$\mathsf{P} \subsetneq \mathsf{NP}$ or for any deterministic, polynomial time compression algorithm A there exists a nondeterministic, polynomial time compression machine M that reduces infinitely many binary strings logarithmically stronger than A."
"$\mathsf{P} \subsetneq \mathsf{NP}$ or f-time resource bounded Kolmogorov complexity of any binary string x can be computed in deterministic polynomial time for each polynomial, time constructible function f."
For computing models with "efficient" interpreters we prove the following theorem:
"For each polynomial, time constructible function f, $\mathsf{TIME(f)} \subsetneq \mathsf{NTIME(f)}$ or one can essentially stronger compress words nondeterministically in time $O(f(n))$ than deterministically in time $f(n)$."

1 Introduction

In [HR20] one used the notion of algorithmically verifiable mathematics in order to question the power of mathematics as a research instrument for proving lower bounds on the computational complexity of concrete problems, and so to question whether P vs NP is solvable inside of mathematics. The starting point in [HR20] was to use the Kolmogorov complexity argument. In contrast to that, here we want to use Kolmogorov complexity in order to suggest trying to prove that nondeterminism is more powerful than determinism for polynomial time computations and space-bounded computations in a general setting. An algorithmically verifiable mathematics (av-mathematics) is any

ISSN 1433-8092

formal theory that is consistent, sufficiently powerful to "speak" about algorithms and computational complexity, and for which there exists an algorithm, that for any word over its alphabet verifies whether that word is a correct proof of a claim. We consider problems like P vs NP [Coo71, Lev73, Kar72] to be meta-problems and suggest to discuss them on a corresponding meta-level. This is the main reason to consider a restriction of av-mathematics to **efficiently verifiable mathematics** (ev-mathematics) by adding the constraint that there exists a polynomial time algorithm that can verify whether a given text t is a proof of a mathematical statement or not. Here we consider polynomial time in the length $|t|$ of a text t and consider the size of the description of the formal system used as a constant independent on t.

We know that $\mathsf{NP} = \mathsf{VP}$ [Kar72], i.e. NP is a class of decision problems that have "short" certificates that can be verified in polynomial time in the length of the input instances. From the point of view of $\mathsf{NP} = \mathsf{VP}$ the question whether nondeterminism is more powerful than determinism for polynomial time computations is equivalent to the question whether it is easier to verify given proof candidates than to "find" the proofs. Hence, everything is about deriving and verifying proofs and so it is natural to discuss P vs. NP in this framework. Since a proof can be viewed as a sequence of applications of syntactic rules (axioms) , it is natural to consider the verification process to be efficient with respect to the proof length, and hope that this assumption can be helpful in better understanding the problems about the relative power of nondeterminism and determinism.

Another starting point here is to use Kolmogorov complexity [Cha69, Kol63, Kol65, Kol68, Sol64a, Sol64b] as a powerful research instrument that has been approved to be instrumental in offering proofs of several fundamental results. Thus, the idea here is to investigate simultaneously P vs. NP and nondeterministic polynomial time compression machines versus deterministic polynomial time compression algorithms.

The main results are as follows:

(1) "$\mathsf{P} \subsetneq \mathsf{NP}$ or for every deterministic, polynomial time compression algorithm A there exists a nondeterministic, polynomial time compression machine M that compresses infinitely many binary words logarithmically stronger than A."

(2) "$\mathsf{P} \subsetneq \mathsf{NP}$ or f-time bounded Kolmogorov complexity of binary strings can be computed in deterministic polynomial time for any polynomial, time constructible function f."

Taking models of computation in which an interpreter can simulate one step of a program by one own step or a constant number of steps, we can prove the following result for sufficiently fast growing, time constructible functions f:

(3) "($\mathsf{P} \subsetneq \mathsf{NP}$ and $\mathsf{TIME(f)} \subsetneq \mathsf{NTIME(f)}$) or nondeterminism in time $O(f(n))$ is stronger than determinism in time $f(n)$ for the problem of word compression."

This paper is organized as follows. Section 2 repeats some fundamental definitions and concepts and introduces new ones as compression algorithms and nondeterministic compression machines, and corresponding complexity classes. In Section 3 we use the concepts introduced to prove our main results. In Section 4 we discuss the meaning and consequences of our results.

2 Preliminaries

Let us consider an efficiently verifiable mathematics (ev-mathematics) for which there exists a polynomial time algorithm A_{ver} that for any input as a word over the alphabet of the ev-mathematics verifies whether the input is a claim followed by a valid proof of this claim. Let p_{ver} be a polynomial function that bounds the time complexity of A_{ver}. Let, for any Algorithm A, $\boldsymbol{A(x)}$ denote the output of A on an input x, and let **Time**$_{\boldsymbol{A}}$ $\boldsymbol{(x)}$ denote the time complexity of algorithm A.

Definition 1. *Let Σ be an alphabet. Let (Σ, L) be a recursive decision problem for an $\mathrm{L} \subseteq \Sigma^*$. We say that the* ***decision problem*** $\boldsymbol{(\Sigma, \mathrm{L})}$ *or shortly* ***the language*** $\mathbf{L}$ ***has short proofs****, if there exists a polynomial p such that for any $w \in \mathrm{L}$, there exists a proof of "$w \in \mathrm{L}$" that has the length at most $p(|w|)$.*

Note that if an algorithm A provably decides a decision problem, then the description of the algorithm as well as the proof of the correctness of the algorithm have together a length d which is a constant with respect to the length of inputs. In this way the proof of the correctness of the algorithm A plus the computation of A on x can be viewed as a proof of "$x \in \mathrm{L}$" (or "$x \notin \mathrm{L}$"). Therefore the length of the proofs is bounded by $d + \mathrm{Time}_A\,(x)$ for every $x \in \Sigma^*$. Hence, if A is a polynomial time algorithm, then L and L^C have short proofs.

Moreover if A is a polynomial time nondeterministic machine accepting L, then we can guarantee that L has short proofs. The following claim is a version of the famous theorem of Karp that NP is a class of decision problems with "short" proof certificates.

Claim 1 [Kar72]**.** *A language* L *has short proofs iff* $\mathrm{L} \in \mathsf{NP}$

Proof. We already proved the direction "$\Leftarrow$". If A has short proofs, then, for any input $x \in \mathrm{L}$, a string R as a proof of "$x \in L$" can be nondeterministically guessed in polynomial time, and then verified in polynomial time in ev-mathematics whether the guessed string R is a valid proof of $x \in L$. $\square$

Let $\boldsymbol{K(x)}$ for any $x \in \{0,1\}^*$ denote the Kolmogorov complexity of x [Kol63, Kol68, Cha69]. It does not matter which model of computation (Turing machine, programming language, etc.) we consider. For generating words our models have a special output tape which they are not allowed to read from or change its content. We denote by p_{comp} the time complexity of the interpreter of this model, that, for any input string over $\{0,1\}$, verifies whether the string is a binary code of a machine (program) in the model. Note that w.l.o.g. one assumes that p_{comp} is a polynomial function. For any program P in the model we assume that the interpreter can execute one step of P in time $p_{sim}(|code(P)|)$ for a polynomial function p_{sim}.

We consider also time-bounded Kolmogorov complexity [Ko86, Ko91, Sip83, Har83, Lon86, BF97]. For any time-constructible function f we denote by ***time***$\boldsymbol{(f)}$**-**$\boldsymbol{K(x)}$ the length of the shortest program generating the string $\{0,1\}^*$ in time at most $f(|x|)$. It is well known that $K(x)$ is not computable (there does not exist any algorithm that, for a given $x \in \{0,1\}^*$, computes $K(x)$), but that *time*(f)-$K(x)$ is computable for any f [Cha69]). Let us shortly remind the proof idea.

Claim 2. *For each time-constructible function f, there exists an algorithm AK(f) that computes time(f)-$K(y)$ for any given $y \in \{0,1\}^*$.*

Proof. An algorithm *AK*(f) can work as follows:

- **Input:** $y \in \{0,1\}^*$
- **AK(f)**
  (a) Compute the value of $f(|y|)$
  (b) Generate one after the other words $z \in \{0,1\}^*$ in the canonical order. Verify with A_{comp} whether z is a binary code of a program. If z is a code of a program $P(z)$, let $P(z)$ run for $f(|y|)$ steps in order to check whether $P(z)$ generates y in $f(|y|)$ steps.
- **Output**: $|z|$ of the canonically first z such that $P(z)$ generates y in $f(|y|)$ steps.

Note that $\text{Time}_{\text{AK}(f)}(n)$ is exponential in n.

□

We consider also nondeterministic Kolmogorov complexity of binary strings. We say that a **nondeterministic machine P generates an $x \in \{0,1\}^*$** if there exists at least one computation of P that halts and outputs x, all computations are finite, and there does not exist any other computation of P generating a word $z \neq x$. $\text{Time}_P(x)$ denotes the length of the shortest computation of P generating x.

In what follows we consider a special version of resource bounded nondeterministic Kolmogorov complexity [All03]. The **nondeterministic Kolmogorov complexity of x**, denoted ***nK*(x)** for any $x \in \{0,1\}^*$, is the binary length of the shortest nondeterministic machine generating x. We say that a **nondeterministic machine P generates $x \in \{0,1\}^*$** in time m, if all computations of P have length at most m, and all accepting computations output x. For any time-constructible function f, ***time*(f)-*nK*(x)** is the length of the shortest nondeterministic machine computing x in time at most $f(|x|)$. *time*(f)-*nK*(x) is called the **time(f)-bounded nondeterministic Kolmogorov complexity of x**.

Claim 3. *For any time-constructible function f, time(f)-nK(y) is computable for any $y \in \{0,1\}^*$.*

Proof. The same as the proof for Claim 2, except one has to simulate all computations of $P(z)$ for every z in order to fix that $P(z)$ generates only y and that all computations are of length at most $f(|y|)$. This works because we consider only nondeterministic machines whose all computations are of the length at most $f(|y|)$. □

We introduce the following decision problems:

$$\textbf{UpperK} = \{(x,n) \mid x \in \{0,1\}^*, n \in \mathbb{N}, K(x) \leq n\}$$
$$\textbf{LowerK} = \{(y,m) \mid y \in \{0,1\}^*, m \in \mathbb{N}, K(y) > m\}$$

Observe that LowerK $=$ (UpperK)^C.

For any time-constructible function f we introduce the languages

$$\begin{aligned}\mathbf{Upper(f)\text{-}K} &= \{(x,n) \mid x \in \{0,1\}^*, n \in \mathbb{N}, \textit{time}(f)\text{-}K(x) \le n\}\\ \mathbf{Lower(f)\text{-}K} &= \{(y,m) \mid y \in \{0,1\}^*, m \in \mathbb{N}, \textit{time}(f)\text{-}K(y) > m\}\end{aligned}$$

Note again, that Lower(f)-K $=$ (Upper(f)-K)^C for any function f.

Claim 4. Upper(f)-K $\in$ NP *and* Lower(f)-K $\in$ coNP *for any polynomial, time-constructible function f.*

Proof. A nondeterministic machine B accepting Upper(f)-K works for any given pair (x,n) as follows. First, B computes deterministically the value $f(|x|)$ in time $O(f(|x|))$. Then, B generates nondeterministically a string $z \in \{0,1\}^{\le n}$ and deterministically verifies in time $p_{comp}(n)$ whether z is the binary code of a program $P(z)$. If yes, B simulates the work of $P(z)$ for at most $f(|x|)$ steps in order to find whether $P(z)$ generates x or not. If $P(z)$ generates x in time $f(|x|)$, then B accepts its input (x,n), else B rejects.

Time$_{\mathsf{B}}$ $((x,n))$ is bounded by $O(p_{comp}(n) + f(|x|) \cdot p_{sim}(|z|))$ and all p_{comp}, p_{sim} and f are polynomial functions. B is a polynomial algorithm, because $n \in |x| + O(1)$. $\square$

Lemma 1. Lower(f)-K $\in$ NP *iff* Lower(f)-K $\in$ NP $\cap$ coNP *iff there exists a proof of polynomial length of "f-$K(x) > n$" for every $(x,n) \in \{0,1\}^* \times \mathbb{N}$.*

Proof. The first part is obvious because (Lower(f)-K)^C $=$ Upper(f)-K $\in$ NP. If there exist a short proof of "f-$K(x) > n$", then a nondeterministic machine D can guess it and verify it in polynomial time in an ev-mathematics.

If Lower(f) $\in$ NP, there exists a nondeterministic polynomial time machine D accepting Lower(f). Hence, for each $(x,n) \in$ Lower(f), there exists a proof of a length at most $O(\text{Time}_D\ ((x,n)))$. $\square$

Claim 5. *Let f be any polynomial, time-constructible function. If there does not exist a polynomial function p such that the lengths of the shortest proofs of the claims "f-$K(x) > n$" are bounded by $p(|x|)$, then*

$$\text{Lower(f)-K} \in \mathsf{EXPTIME} - \mathsf{NP}.$$

Proof. Lower(f)-K $\in$ EXPTIME is obvious, because one can generate all (at most $2^n - 1$ many) binary codes of programs shorter than n and let them run $f(|x|)$ many steps. $\square$

Let, for any two strings $x, y \in \{0,1\}^*$, $\boldsymbol{x \ll y}$ denote that x is "smaller" than y with respect to the canonical order.

We will introduce the following decision problem for every time-constructible function f.

$$\textbf{First(f)-K} = \{(x,n) \mid n \in \mathbb{N},\ x \in \{0,1\}^*,\ \textit{time}(f)\text{-}K(x) \geq n \text{ and } \textit{time}(f)\text{-}K(y) < n \text{ for all } y \ll x\}.$$

Obviously

$$\textbf{(First(f)-K)}^{\mathbf{C}} = \{(x,n) \mid n \in \mathbb{N},\ x \in \{0,1\}^*,\ \textit{time}(f)\text{-}K(x) < n \text{ or } \exists y : y \ll x \text{ with } \textit{time}(f)\text{-}K(y) \geq n\}.$$

Claim 6. *For any polynomial, time-constructible function f, if* Lower(f)-K $\in$ NP*, then* $(\text{First(f)-K})^C \in$ NP *and* First(f)-K $\in$ coNP.

Proof. Let there exist a nondeterministic machine C for Lower(f)-K working in polynomial time. A nondeterministic machine B for $(\text{First(f)-K})^C$ works as follows. For any input $(x,n) \in \{0,1\}^* \times \mathbb{N}$, B guesses

– either a $z \in \{0,1\}^{<n}$ and verifies whether z is a binary code of a program $P(z)$ that generates x in time $f(|x|)$

– or a $y \in \{0,1\}^{\leq n}$, $y \ll x$ and uses the nondeterministic subroutine C to verify whether $(y, n-1) \in$ Lower(f)-K

□

In the following sections we aim to use the Kolmogorov complexity concept as a research instrument for the study of the relationship between nondeterminism and determinism for polynomial time computations. The following result shows the power of the Kolmogorov complexity argument. We can prove in a new setting that more complexity resources increase the power of computations. Here we show that more time improves the compressibility of strings.

Theorem 1. *Let f be an arbitrary fast growing function that is time-constructible. Let $\{x_n\}_{n=1}^{\infty}$, $x_n \in \{0,1\}^*$ for each $n \in \mathbb{N}$, be a sequence of strings such that $(x_n, n) \in$* First(f)-K*. Then $\exists c \in \mathbb{N}$ such that*

$$K(x_n) \leq \log_2 n + c = \log_2 (\textit{time}(f)\text{-}K(x_n)) + c.$$

Proof. We present an algorithm A, that, for a given $n \in \mathbb{N}$, generates x_n. A generates words $y \in \{0,1\}^*$ in canonical order and estimates for each y its *time*(f)-$K(y)$ (Claim 2). A halts for the first y with *time*(f)-$K(y) = n$ and outputs y.

The existence of algorithm A guarantees the existence of the infinite sequence $\{A_n\}_{n=1}^{\infty}$ of programs such that Output$(A_n) = x_n$. All A_n are the same except the parameter n that can be saved by $\lceil \log_2 (n+1) \rceil$ bits. Hence, the upper bound on $K(x_n)$ follows. □

Note, that, for slowly growing f, the time complexity of A can be exponential in $f(n)$. But, for fast growing f such as 2^{2^n}, the time complexity of A can be in $O(f(n) \cdot \log_2 f(n))$ or even smaller. Hence, for fast growing functions we can have results such as *time*$(f \cdot \log_2 f)$-$K(x) \leq \log_2 (\textit{time}(f)\text{-}K(x)) + c$ for infinitely many $x \in \{0,1\}^*$.

Definition 2. *Any algorithm A computing an injective function $h\colon \{0,1\}^* \to \{0,1\}^*$ is called a* ***compression algorithm****, if, for each $x \in \{0,1\}^*$, $A(x) = comp_A(x)$ is the binary code of a program $P(comp_A(x))$ that generates the string x.*

We define the following compression complexity class for any time-constructible function f:

$$\begin{aligned}\textsf{\textbf{compTIME(f)}} = \{f_A \colon \{0,1\}^* \to \{0,1\}^* \mid & \text{ there exists a compression algorithm } A \\ & \text{computing } f_A \text{ within } \mathrm{Time}_A(|x|) \le f(|x|) \text{ and } P(\mathrm{comp}_A(x)) \\ & \text{generates } x \text{ in time } f_A(|x|) \text{ for every } x \in \{0,1\}^*\}\end{aligned}$$

$$\textsf{\textbf{compP}} = \bigcup_{c\in\mathbb{N}} \textsf{compTIME}(\textsf{n}^{\textsf{c}}).$$

Definition 3. *We say that a nondeterministic machine M computes an injective function $h\colon \{0,1\}^* \to \{0,1\}^*$ if all computations on any input x are either accepting or rejecting, and all accepting computations finish with the same output $M(x) = comp_M(x)$ for each input $x \in \{0,1\}^*$. M is called a* ***nondeterministic compression machine*** *if, for each $x \in \{0,1\}^*$, $comp_M(x)$ is the binary code of a nondeterministic machine $M(comp_M(x))$ that generates x.*

We define the following compression complexity classes for any time-constructible function f:

$$\begin{aligned}\textsf{\textbf{compNTIME(f)}} = \{f_M \colon \{0,1\}^* \to \{0,1\}^* \mid & \exists \text{ nondeterministic compression machine} \\ & M \text{ computing } f_M \text{ within } \mathrm{Time}_M(|x|) \le f(|x|) \text{ and} \\ & M(\mathrm{comp}_M(x)) \text{ generates } x \text{ in time } f(|x|) \text{ for every } x \in \{0,1\}^*\}\end{aligned}$$

$$\textsf{\textbf{compNP}} = \bigcup_{c\in\mathbb{N}} \textsf{compNTIME}(\textsf{n}^{\textsf{c}}).$$

We say that an infinite sequence $\{x_n\}_{n=1}^{\infty}$ of strings is **hard** for a compression class $\mathcal{A}$, if, for each compression function $g \in \mathcal{A}$, $|g(x_n)| \in \Omega(|x_n|)$.

We say that $\{x_n\}_{n=1}^{\infty}$ is **easy** for $\mathcal{A}$, if there exists $h \in \mathcal{A}$ and a constant $c \in \mathbb{N}$, such that $|h(x_n)| \le \log_2 |x_n| + c$ for all $n \in \mathbb{N}$.

For two compression complexity classes $\mathcal{A}$ and $\mathcal{B}$ we say that $\boldsymbol{\mathcal{A}}$ **is stronger than** $\boldsymbol{\mathcal{B}}$ if

(i) $\mathcal{B} \subsetneq \mathcal{A}$

(ii) there exists an infinite sequence of words that is easy for $\mathcal{A}$ but hard for $\mathcal{B}$.

Let A and B be compression algorithms or nondeterministic compression machines. We say that $\boldsymbol{A}$ **is essentially stronger than** $\boldsymbol{B}$, if

(i) $|\text{comp}_A(x)| \leq |\text{comp}_B(x)|$ for all $x \in \{0,1\}^*$

(ii) $\exists \{x_n\}_{n=1}^{\infty}$, $x_i \in \{0,1\}^*$, $x_i \ll x_{i+1}$ for $i \in \mathbb{N}$, and $\exists c \in \mathbb{N}$, such that $|\text{comp}_A(x_n)| \leq \log_2 |\text{comp}_B(x_n)| + c$ for all $n \in \mathbb{N}$.

Analogously, for any two injective functions g and h from $\{0,1\}^*$ to $\{0,1\}^*$ we say that ***g* compresses strings essentially stronger than h**, if

(i) $|g(x)| \leq |h(x)|$ for all $x \in \{0,1\}^*$, and

(ii) there exist a sequence $\{x_n\}_{n=1}^{\infty}$, $x_i \in \{0,1\}^*$, $x_i \ll x_{i+1}$ for all $i \in \mathbb{N}$, and a constant $c \in \mathbb{N}$ such that $|g(x_n)| \leq \log_2 |h(x_n)| + c$ for all $n \in \mathbb{N}$.

3 Main results

In this section we aim to compare nondeterministic polynomial time with deterministic polynomial time simultaneously with respect to decision problems and compression. Our first result is for ev-mathematics.

Theorem 2. *In any ev-mathematics, the following is true: "*$\mathsf{P} \subsetneq \mathsf{NP}$ *or for any deterministic, polynomial time compression algorithm A there exists a nondeterministic, polynomial time compression machine M such that M is essentially better than A."*

Proof. We distinguish a few cases with respect to the existence of short proofs for some of the Kolmogorov decision problems introduced in the previous section.

(i) Let there exist a polynomial, time-constructible function f such that Lower(f)-K does not have short proofs, i.e. Lower(f)-K $\notin \mathsf{NP}$. Since (Lower(f)-K)^C = Upper(f)-K $\in \mathsf{NP}$ (Claim 4), NP is not closed under complement and hence $\mathsf{P} \subsetneq \mathsf{NP}$.

(ii) Let, for any polynomial, time-constructible function f, Lower(f)-K have short proofs, i.e. Lower(f)-K $\in \mathsf{NP}$. Following Claim 6, (First(f)-K)^C $\in \mathsf{NP}$ for all polynomial, time-contractible functions f. Now, we distinguish two cases:

(ii.1) Let First(f)-K does not have short proofs, i.e. First(f)-K $\notin \mathsf{NP}$, for a polynomial, time constructible function f. Since following (ii) (First(f)-K)^C $\in \mathsf{NP}$, NP is not closed under complement and hence $\mathsf{P} \subsetneq \mathsf{NP}$.

(ii.2) There exist short proofs for Lower(f)-K as well for First(f)-K for any polynomial, time constructible function f.
Note, that for any compression algorithm A with $\text{Time}_A(z) \leq f(|z|)$ for all $z \in \{0,1\}^*$, $|\text{comp}_A(z)| \geq$ *time*(f)-$K(z)$. Let p_{lower} be a polynomial that bounds the length of the short proofs for Lower(f)-K, and let p_{first} be a polynomial that bounds the length of the short proofs for First(f)-K.
Let us consider the infinite sequence $\{x_n\}_{n=1}^{\infty}$ of strings (for a fixed f) such that $(x_n, n) \in$ First(f)-K.

Now we show that infinitely many words x_n can be essentially stronger compressed in nondeterministic polynomial time. Let $\{x_n\}_{n=1}^{\infty}$ be a sequence of words such that $(x_n, n) \in \mathrm{First(f)\text{-}K}$. We construct an infinite sequence $\{B_n\}_{n=1}^{\infty}$ of nondeterministic machines such that B_n generates x_n.

```
**Input** $n$
**Step 1** $B_n$ computes $f(n)$ in time $O(f(n))$
**Step 2** $B_n$ guesses two words $x \in \{0,1\}^{\leq n}$ and $u \in \{0,1\}^{\leq p_{\text{lower}}(|x|)}$.
         $B_n$ verifies whether $u$ is a proof of "*time*$(f)$-$K(x) \geq n$" in time
         $p_{\text{ver}}(p_{\text{lower}}(|x|))$.
         **If** "yes" $B_n$ continues with Step 3 **else** $B_n$ halts and rejects.
**Step 3** $B_n$ guesses a $v \in \{0,1\}^n$, verifies in time $p_{\text{comp}}(n)$ whether $v$ is
         a code$(M)$ of a program $M$, and then verifies in time $p_{sim}(n) \cdot$
         $f(|x|)$ whether $M$ generates $x$ in time $f(|x|)$.
         **If** both verifications succeeded **then** $B_n$ continues with Step 4 **else**
         $B_n$ halts and rejects.
**Step 4** $B_n$ guesses a $w \in \{0,1\}^{p_{\text{first}}(|x|)}$, and verifies in time $p_{\text{ver}}(p_{\text{first}}(|x|))$
         whether $w$ is a proof of $(x, n) \in \mathrm{First(f)\text{-}K}$.
         **If** $w$ is a proof of $(x, n) \in \mathrm{First(f)\text{-}K}$ **then output**$(x)$ **else** $B_n$
         halts and rejects.
**Output** $x = x_n$
```

All the algorithms B_n generating x_n are the same, they differ only in the input n. Because of that the decriptional complexity of B_n is bounded by $\log_2(n) + c$, where c is the size of the binary code of the common part of all B_n's. Hence, there exists a constant $c \in \mathbb{N}$, such that for all $n \in \mathbb{N}$

$$\textit{time}(g)\textit{-nK}(x_n) \leq \log_2 n + c$$

for a polynomial function

$$g(n) \in O(p_{sim}(n) \cdot f(n) + p_{\text{ver}}(p_{\text{lower}}(n)) + p_{\text{comp}}(n) + p_{\text{ver}}(p_{\text{first}}(n)))\,.$$

Note that p_{lower} and p_{first} are parameterized by f, and one is not allowed to say that $g(n) \in O(f(n))$ for a sufficiently fast growing function f.

Now we construct a nondeterministic compression machine C that is essentially better than any compression algorithm for a compression function in compTIME(f):

```
**Input** $w \in \{0,1\}^*$
**Step 1** $C$ constructs the value $f(|w|)$ in $O(f(|w|))$ steps.
**Step 2** $C$ guesses whether "$w = x_n$ for some $n \in \mathbb{N}$" or "$w \neq x_i$ for all $i \in \mathbb{N}$".
    **Step 2.1** **If** the guess of $C$ is "$w = x_n$ for some $n \in \mathbb{N}$", **then**
        • $C$ guesses an $n \in \mathbb{N}$ and an $v \in \{0,1\}^{\leq p_{\text{lower}}(|w|)}$ and
          verifies in time $p_{\text{ver}}(p_{\text{lower}}(|w|))$ whether $v$ is a proof of
          "$f$-$K(w) \geq n$".
          **If** "yes" **then** $C$ continues with the next step **else** $C$ halts
          and rejects.
```

- C guesses a string $u \in \{0,1\}^n$ and verifies in time $p_{\text{comp}}(u)$ whether $u = \text{Code}(P)$ for a program P that deterministically generates w in time $f(|w|)$.
  If "yes" **then** C continues with the next step **else** C halts and rejects.
- C guesses a string $z \in \{0,1\}^{\leq p_{\text{first}}(|w|)}$ and verifies whether z is a proof of "$(w,n) \in \text{First(f)-K}$".
  If "yes" **then** C outputs the binary code of B_n **else** C halts and rejects.

Step 2.2 **If** C guessed "$w \neq x_i$ for all $i \in \mathbb{N}$", **then**

- C computes nondeterministically *time*(f)-$K(w)$ by
  (i) guessing a string $v \in \{0,1\}^*$ and verifying whether v codes a program $\tilde{P}$ that generates w in time $f(|w|)$
  (ii) guessing a string $u \in \{0,1\}^{\leq p_{\text{lower}}(|w|)}$ and verifying whether u is a proof of "*time*(f)-$K(w) \geq |v|$".
- C guesses a string $y \ll w$ and a $z \in \{0,1\}^{\leq p_{\text{lower}}(|w|)}$ and verifies whether z is a proof of *time*(f)-$K(y) \geq |v| =$ *time*(f)-$K(w)$. **If** all verifications succeed **then output**$(v = \text{Code}(\tilde{P}))$ **else** C halts and rejects.

We observe that $|\text{comp}_c(w)| \leq$ *time*(f)-$K(w)$ for all $w \in \{0,1\}^*$ and so $|\text{comp}_c(w)| \leq |\text{comp}_A(w)|$ for each compression algorithm A computing a function from compTIME(f), since for any A $|\text{comp}_A(w)| \geq f$-$K(w)$ for all $w \in \{0,1\}^*$.

There exists a $c \in \mathbb{N}$ such that, for all $n \in \mathbb{N}$,

$$\begin{aligned}|\text{comp}_c(x_n)| = |\text{code}(B_n)| &\leq \log_2 n + c \leq \log_2(\textit{time}(f)\text{-}K(x_n)) + c \\ &\leq \log_2(|\text{comp}_A(x_n)|) + c\end{aligned}$$

for any compression algorithm A working in time $f(n)$.

C works in time

$$O(p_{sim}(n) \cdot f(n) + p_{\text{comp}}(n) + p_{\text{ver}}(p_{\text{first}}(n)) + p_{\text{ver}}(p_{\text{lower}}(n)))$$

and so C is a polynomial time, nondeterministic compression machine. □

Corollary 1. $\mathsf{P} \subsetneq \mathsf{NP}$ *or, for any polynomial, time-constructible function f, time(f)-$K(x)$ can be computed in deterministic polynomial time in $|x|$.*

Proof. For cases (i) and (ii.1) we proved in Theorem 1 that $\mathsf{P} \subsetneq \mathsf{NP}$.

If $\mathsf{P} = \mathsf{NP}$ (i.e. $\mathsf{P} \subsetneq \mathsf{NP}$ does not hold) in case (ii.2), then all the languages Upper(f)-K, Lower(f)-K, and First(f)-K are in P for any polynomial, time-constructible function f. But then one can use the binary search to estimate *time*(f)-$K(x)$ for any $x \in \{0,1\}^*$ because there exists a constant c such that $K(x) \leq |x| + c$ for all $x \in \{0,1\}^*$. □

Up to now we did not specify our computing model. We only asked for an efficient interpreter and an efficient (polynomial time in size of the simulated program) simulation of any program by the interpreter. In the following we consider computation models with an interpreter that can simulate any program with a constant number of steps for each simulated step (i.e. the computational complexity of simulations is independent of the program being simulated). In the following, we call such interpreters $\mathbf{O(1) - interpreters}$. Note that having $O(1)$ interpreters is not obvious for models of computation. For instance, an MTM (multiple-tape Turing machine) does not have an $O(1)$-interpreter.

Theorem 3. *In each av-mathematics and for any computation model having an $O(1)$-interpreter, the following is true:*

"($\mathsf{P} \subsetneq \mathsf{NP}$ and $\mathsf{TIME(f)} \subsetneq \mathsf{NTIME(f)}$) or $\mathsf{compNTIME(O(f))}$ is essentially stronger than $\mathsf{compTIME(f)}$ for any polynomial, time constructible function $f(n) \geq p_{comp}(n)$."

Proof. $\mathrm{Upper(f)\text{-}K} \in \mathsf{NP}$ (Claim 4). If $\mathrm{Lower(f)\text{-}K} = (\mathrm{Upper(f)\text{-}K})^C \notin \mathsf{NP}$, then NP is not closed under complement and $\mathsf{P} \subsetneq \mathsf{NP}$. Assume $\mathrm{Lower(f)\text{-}K} \in \mathsf{NP}$. Following Claim 6 we have $(\mathrm{First(f)\text{-}K})^C \in \mathsf{NP}$. If $\mathrm{First(f)\text{-}K} \notin \mathsf{NP}$, then NP is not closed under complement and $\mathsf{P} \subsetneq \mathsf{NP}$.

Consider the case that $\mathrm{Lower(f)\text{-}K}$ and $\mathrm{First(f)\text{-}K}$ are in NP. If $f(n) \geq p_{comp}(n)$ and the model of computation has an $O(1)$-interpreter, then the proof of Claim 4 shows that $\mathrm{Upper(f)\text{-}K} \in \mathsf{NTIME(f)}$. If $\mathrm{Lower(f)\text{-}K} = (\mathrm{Upper(f)\text{-}K})^C \notin \mathsf{NTIME(f)}$, then $\mathsf{NTIME(f)}$ is not closed under complement and so $\mathsf{TIME(f)} \subsetneq \mathsf{NTIME(f)}$. If $\mathrm{Lower(f)\text{-}K} \in \mathsf{NTIME(f)}$ and $f(n) \geq p_{comp}(n)$ for all $n \in \mathbb{N}$, then the proof of Claim 6 shows that $(\mathrm{First(f)\text{-}K})^C \in \mathsf{NTIME(f)}$. In this case, if $\mathrm{First(f)\text{-}K} \notin \mathsf{NTIME(f)}$, then $\mathsf{TIME(f)} \subsetneq \mathsf{NTIME(f)}$.

The remaining case is $\mathrm{Upper(f)\text{-}K}, \mathrm{Lower(f)\text{-}K}, \mathrm{First(f)\text{-}K} \in \mathsf{NTIME(f)}$ for all $f(n) \geq p_{comp}(n)$. This immediately implies that $time(f)\text{-}K(x)$ can be computed nondeterministically in time $O(f)$. Note that up to now we are even allowed to assume $\mathsf{TIME(f)} = \mathsf{NTIME(f)}$, and so to assume that all these problems are in $\mathsf{TIME(f)}$.

Consider (similarly to the proof of Theorem 2) the infinite sequence $\{x_n\}_{n=1}^{\infty}$ of words such that $(x_n, n) \in \mathrm{First(f)\text{-}K}$. A nondeterministic machine $B(n)$ generates x_n for any $n \in \mathbb{N} - \{0\}$ as follows:

```
**Input** $n$
**Step 1** $B_n$ deterministically computes $f(n)$ in time $O(f(n))$.
**Step 2** $B_n$ guesses a word $x \in \{0,1\}^*$ and verifies whether $(x, n) \in \mathrm{First(f)\text{-}K}$.
**Output** $B_n$ outputs $x = x_n$ if $(x, n) \in \mathrm{First(f)\text{-}K}$, else $B_n$ rejects
```

$B(n)$ works in time $O(f(n))$ and generates x_n. The consequence is that there exists a constant k such that $\mathsf{Time}_B(n) \leq k \cdot f(n)$ for all $n \geq 1$, and hence there exist constants k and e such that:

$$time(k \cdot f(n))\text{-}nK(x_n) \leq \log_2 n + e\,.$$

Therefore $\{x_n\}_{n=1}^{\infty}$ is hard for $\mathsf{compTIME(f)}$ and easy for $\mathsf{compNTIME(k \cdot f(n))}$. Now, we design a nondeterministic machine C such that

$$|\mathsf{comp}_C(w)| \leq \textit{time}(f)\text{-}K(w) \leq |\mathsf{comp}_A(w)|$$

for any compression algorithm A working in time $f(n)$ and simultaneously $|\mathsf{comp}_C(x_n)| = |\mathsf{code}(B(n))| \leq \log_2 n + c$.

C works as follows:

Input $w \in \{0,1\}^*$

Step 1 C deterministically constructs the value $f(|w|)$ in time $O(f)$.

Step 2 C deterministically computes $n = \textit{time}(f)\text{-}K(w)$ in time $O(f)$.

Step 3 C guesses "$w = x_n$" or "$w \neq x_n$".

- If the guess is "$w = x_n$", C verifies $(w, n) \in \mathrm{First(f)\text{-}K}$ in time $O(f(n))$ and outputs $B(n)$ if it is true.
- If the guess is "$w \neq x_n$", C verifies $(w, n) \notin \mathrm{First(f)\text{-}K}$ in time $O(f(n))$ and guesses a $z \in \{0,1\}^n$ such that z codes a program generating w in time $f(|w|)$. C verifies in time $p_{comp}(|w|)$ that z codes a program $P(z)$ and in time $O(f(|w|))$ that $P(z)$ generates w in time $f(|w|)$.

Summarizing, C is at least as good as any compression algorithm working in time $f(n)$ and $\{x_n\}_{n=1}^{\infty}$ is easy for C, but hard for $\mathsf{compTIME(f)}$. $\square$

Let s_{comp} be a function that bounds the space complexity of the compiler in the considered model. Similar arguments using space-bounded Kolmogorov complexity offer the following result:

Theorem 4. *In av-mathematics for a computation model with $O(1)$-interpreter and each space constructible function f with $f(n) \geq \max\{n, s_{comp}(n)\}$ it holds:*

"$\mathsf{SPACE(f)} \subsetneq \mathsf{NSPACE(f)}$ or $\mathsf{compNSPACE(O(f(n)))}$ is essentially stronger than $\mathsf{compSPACE(f)}$."

4 Discussion

It would be nice to get a stronger result than in Theorem 2 and 3. For instance:

(*) "$\mathsf{P} \subsetneq \mathsf{NP}$" or "compNP is stronger than compP".

Why are our proofs and thoughts not sufficient to calculate such a strong result? Because we cannot exclude the possibility that, for each deterministic polynomial time compression algorithm A, there exists another deterministic polynomial time algorithm B such that B is essentially stronger than A. In other words, for each nondeterministic polynomial time compression machine C from the proof of Theorem 2, there can exist a deterministic polynomial time algorithm B that is at least as good as C. If (*) does not hold, it means that each deterministic polynomial time algorithm can be essentially improved inside of the class compP.

Definition 4. *An algorithm (a machine) A is called* ***almost optimal*** *in a compression class $\mathcal{A}$ if A satisfies the complexity restrictions of $\mathcal{A}$, and for any compression function $f \in \mathcal{A}$ there exist constants c and $d > 1$ such that, for all $x \in \{0,1\}^*$,*

$$|A(x)| \le c + |f(x)|^d,$$

i.e., no algorithm for $\mathcal{A}$ is essentially better than A.

Now, we can reformulate Theorem 2 as follows:

Theorem 5. *"$\mathsf{P} \subsetneq \mathsf{NP}$" or "$\mathsf{compNP}$ is stronger than compP" or "there does not exist any almost optimal compression algorithm in compP".*

In this way Theorem 5 claims, that if "$\mathsf{P} = \mathsf{NP}$ and compNP is not stronger than compP", then increasing polynomially the time complexity of polynomial time algorithms one can compress infinitely many strings essentially (polylogarithmically) stronger. Taking Theorem 3 into account, one can formulate this result as follows:

Corollary 2. *"$\mathsf{P} \subsetneq \mathsf{NP}$" or "$\mathsf{compNP}$ is stronger than compP" or "for any $c \in \mathbb{N}$ there exists a $k \in \mathbb{N}$ such that $\mathsf{compTIME}(\mathsf{k} \cdot \mathsf{n})$ is stronger than $\mathsf{compTIME}(\mathsf{c} \cdot \mathsf{n})$".*

The exponential improvement of the quality of compressions by an increase of complexity by a multiplicative constant factor would be a surprising property. Similarly surprising as the result of Corollary 1. To compute the Kolmogorov complexity $K(x)$ for a given string is an especially hard problem among the algorithmically unsolvable problems. There are at most finitely many $x \in \{0,1\}^*$ for which the claim "$K(x) = m$" for some $m \in \mathbb{N}$ is provable in mathematics [Cha69]. From this point of view one would not expect that *time*(f)-$K(x)$ is computable in deterministic polynomial time for every polynomial function f. In fact, we even do not expect that *time*(f)-$K(x)$ is computable in nondeterministic polynomial time for polynomial functions f.

Another interesting point to observe is the descriptional complexity of the machine B_n from the proof of Theorem 2. If one parameterizes the descriptional complexity of B_n with respect to the time complexity function f, then the descriptional complexity of B_n is

$$\text{“}\log_2 n + c + \textit{descriptional complexity of } f\text{”}.$$

If one applies Theorem 2 infinitely many times (always again for a potential deterministic polynomial time algorithm reaching the quality of the nondeterministic machine C from the proof of Theorem 2), then there is no upper bound on the descriptional complexity of this infinite sequence of functions f. If such upper bound would exist (or a polynomial function growing faster than all of them), then we would get a contradiction, and so the proof of "$\mathsf{P} \subsetneq \mathsf{NP}$ or compNP is stronger than compP". This is because the sets of words $\{x_n\}_{n=1}^{\infty}$ in two different applications of Theorem 2 in a sequence of iterative applications have always at most an intersection of finite size. If $c +$ *descriptional complexity of f* would be bounded by a constant, then, for a sufficiently large n, all pairs of sets of words from different applications of Theorem 2 would be disjoint. Consequently, in each iteration one would reduce the descriptional complexity of another word in $\{0,1\}^n$. At the very end, all words in $\{0,1\}^n$ would be compressed, which is impossible.

Finally we observe that using the strategy for proving Theorem 3 one can remove the assumption about ev-mathematics from Theorem 2. It is enough to assume in the case (ii.2) that $\mathsf{P} = \mathsf{NP}$ and then one does not need to verify any guessed proof. The only remaining assumption is that p_{comp} is polynomial, which is true for any reasonable machine model.

References

[All03] Eric Allender. NL-printable sets and nondeterministic Kolmogorov complexity. *Electronic Notes in Theoretical Computer Science*, 84:1–15, 2003.

[BF97] Harry Buhrman and Lance Fortnow. Resource-bounded Kolmogorov complexity revisited. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 105–116, 1997.

[Cha69] Gregory J Chaitin. On the simplicity and speed of programs for computing infinite sets of natural numbers. *Journal of the ACM*, 16:407–422, 1969.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[Har83] Juris Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *24th IEEE FOCS*, pages 439–445, 1983.

[HR20] Juraj Hromkovič and Peter Rossmanith. What one has to know when attacking P vs. NP. *Journal of Computer and System Sciences*, 107:142–155, 2020.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. 1972.

[Ko86] Ker-I Ko. On the notion of infinite pseudorandom sequences. *Theoretical Computer Science*, 48(5):9–33, 1986.

[Ko91] Ker-I Ko. On the complexity of learning minimum time-bounded Turing machines. *SIAM Journal on Computing*, 20(5):962–986, 1991.

[Kol63] Andrei N Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–375, 1963.

[Kol65] Andrei N Kolmogorov. Three approaches to the quantitative definition of information. *Problems of information transmission*, 1(1):1–7, 1965.

[Kol68] Andrei Kolmogorov. Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, 14(5):662–664, 1968.

[Lev73] Leonid A Levin. Universal sorting problem. *Problemy Predaci Informacii*, 9:265–266, 1973.

[Lon86] Luc Longpré. Resource bounded Kolmogorov complexity, a link between computational complexity and information theory. *Cornell TR86*, 776, 1986.

[Sip83] Michael Sipser. A complexity theoretic approach to randomness. In *Proc. 15th ACM STOC*, pages 330–335, 1983.

[Sol64a] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

[Sol64b] Ray J Solomonoff. A formal theory of inductive inference. part ii. *Information and control*, 7(2):224–254, 1964.