Kolmogorov time hierarchy and novelty games

Ulysse Léchine, Thomas Seiller

Kolmogorov time hierarchy and novelty games

Ulysse Léchine
LIPN, IRIF
lechine@lipn.univ-paris13.fr

Thomas Seiller
CNRS
thomas.seiller@cnrs.fr

Abstract

In this paper we improve on the state-of-the-art time hierarchy for time-bounded Kolmogorov complexity. More precisely, we prove that there are infinitely many $n \in \mathbb{N}$ such that there are strings of size n which are the output of programs of size $f = o(\log n)$ running in $o(2^f T)$ steps but are not the output of any program of size f running in T steps. The previous gap was exponential in 2^f. This result is established by studying a new problem in combinatorics: a list E of pk numbers in $[1; N]$ is shared evenly between p players, who each get to output an answer depending on their share of E. The players must establish a communication-free strategy ensuring that one of them outputs a number not belonging to E.

1 Introduction

The Kolmogorov complexity of a word $w \in \{0; 1\}^*$ is the size of the smallest program p which prints w and stops. This gives rise to a natural encoding of words: a word can be encoded/described by a program which prints it. To "decode" a program one just runs it until it stops, obtaining in this way the original word. But decoding those, i.e. running the programs, may take some time. We thus may consider time-bounded Kolmogorov complexity, where one restricts to programs of size $f(n)$ running in time $T(n)$.

It is known that allowing programs to have a larger size leads to encoding strictly more words, meaning that for all f there exists words having a description of size $f + 10$ but no description of size f. Can this result be extended when considering bounded time? For arbitrary T, are there words which can be printed by programs running in $2T$ steps but not by programs running in T steps? Stated otherwise: if we denote by $[f, T](n)$ the set of strings of length n outputted by a program of length $f(n)$ running in time $T(n)$ [1], how do $[f, T_1](n)$ and $[f, T_2](n)$ compare? More specifically: given $\alpha > 0$, is $[f, \alpha T](n)$ strictly bigger than $[f, T](n)$? A first remark is that we have to restrict our considerations to words of a given size, smaller than the

[1] Actually running in $T(n)$ steps when simulated through a universal Turing Machine; this is detailed and discussed later.

allotted time bound. Indeed, the question becomes trivial otherwise: with 2000 time steps one can write the string 1^{1900}, but this is not possible in 1800 time steps. We therefore fix a size for our outputs and only consider time bounds larger than this size.

This open question, quoted in [6], is a very natural question to ask when it comes to time bounded Kolmogorov complexity. It is similar to the time hierarchy theorem, a standard result in complexity theory which states that one can solve strictly more decision problems if one allows Turing machines to run for longer. It would seem at first glance that one could use the time hierarchy theorem to solve the problem of the "Kolmogorov time hierarchy", but that line of reasoning has not proven successful.

In his thesis [8], Luc Longpré proves that[2]

$$\exists_\infty n, [f, T](n) \subsetneq [f, c2^f T](n),$$

where $\subsetneq$ means "is included but not equal to", and $\exists_\infty n$ means that this is true for an infinite number of values of n. This result establishes that with exponentially more time in the size of the programs one can prove that it is possible to output new strings. Whether one can lower that $2^{f(n)}$ exponential factor has been an open question since Longpré's result. In this paper, we provide a (partial) positive answer.

Longpré proves his result using a standard diagonalization technique, considering a program running all programs of size f (there are 2^f of them) for T steps each, hence the $2^f T$ time factor, and outputting a string which is not the output of one of those programs. In order to improve on the result, we exploit the following idea: we want to consider several programs each running a fraction of those 2^f programs. These programs will therefore run for less long, allowing us to get a better bound in the theorem. Of course the problem now is that each of the program will only see a fraction of the possible outputs of programs of size f. How can one ensure that they can still output a new string?

We first notice that only one of those program has to output a new string. The question can then be expressed as a new combinatorics problem presented in section 4, called novelty games, describing how they can coordinate to achieve that. We then establish a solution to this combinatorics problem which we then use to get a better Kolmogorov time hierarchy theorem than Longpré for size functions $f = o(\log(n))$. The factor $o(\log(n))$ is directly related to bounds on the solution we obtain for novelty games. Better bounds for novelty games would thus translate (modulo some technicalities) into a better bounded time hierarchy theorem. We call this method

[2]Longpré are also establishes variants of this theorem using space, which we do not mention in this paper.

"parallel diagonalization with advice": the epithet "parallel" is self explanatory, while "with advice" refers to the fact that we can give some additional information to the programs. We believe this technique may have implications elsewhere in complexity theory. The analysis of novelty games may also be of independent interest to researchers in combinatorics.

Kolmogorov time bounded complexity has been studied these recent years and used for breakthrough results in meta complexity [5] [7] [4], but not only [3] [9].

1.1 Outline

Here is a succinct outline of the paper. In section 2, we establish notations and provide some background on time-bounded kolmogorov complexity. Section 3 then provides a high level view of the paper and of our technique. The remaining sections then contain the technical material. We define the new combinatorial problem – which we call *novelty games* – in Section 4. We also prove general results and give winning strategies for these games on specific cases. In Section, 5 we formally define the program in $[f, c2^f T](n)$ but not in $[f, T](n)$ assuming the existence of a general winning strategy for novelty games. We then present such a strategy in Section 6. The results are then combined in Section 7 to establish the main theorem of the paper. Lastly, Section 8 sketches some directions for future work.

2 Definitions

2.1 Notations and useful facts

We will use $\mathbb{N}$ to denote the set of integers. Intervals $\{N, N+1, \dots M\}$ will be denoted by $[N; M]$, and we will abusively write $[N]$ for $[1; N]$. We will write $\binom{[N]}{k}$ to denote the set of multisets of size at most k included in $[1, N]$.

For a given set $\{0; 1\}$, we write $\{0; 1\}^*$ the set of strings, i.e. finite sequences of elements of oi. Given x and y in $\{0; 1\}^*$, $|x|$ will denote the length of the string x, and we will write xy the concatenation of the strings x and y.

We will abusively speak of Turing machine as programs and vice versa. In the following we interchangeably view integers, programs and bitstrings as the same thing, meaning a bitstring represents an integer and a program and vice versa.

Time bounds will be written as T, while α and f are *reasonable* functions of $\mathbb{N}^{\mathbb{N}}$ (see definition below). Functions g and h are reasonable slow growing functions (see definition below). We will often abusively write T, f, and α instead of $T(n)$ $f(n)$ and $\alpha(n)$.

As already mentioned, we write $\forall_\infty n, A(n)$ to express that $A(n)$ is true for all but finitely many n. We will also write $\exists_\infty n, A(n)$ to express that there are infinitely many n such that $A(n)$.

Let $E_1(x) = 1^{|x|}0x$. Let $E_2(x) = E_1(|x|)x$. The proof of the following result can be found in [6].

Theorem 1. E_2 is a prefix code and $|E_2(x)| = x + 2log(|x|) + 1$. E_2 being a prefix code means that $\forall x, y \in (\{0;1\}^*)^2$ one can uniquely retrieve x and y from the string $E_2(x)y$.

We will consider that our encoding of programs is such that if a string is outputted by a program of size f it is also outputted by a program of size $f + c$ for c a universal constant.

We define $C^T(x)$ for x a string in the same way as in Li and Vitanyi [6] i.e. it is the length of the smallest description p such that when p is interpreted by U a universal Turing machine fixed in advance, U prints x in less that T steps.

Let g, f be two integer functions we write $g = \tilde{O}(f)$ to mean $g = O(f * polylog(f))$

When we call a function f reasonable we mean it in an informal way: "any function not designed to explicitly break our theorem". A formal definition most of the time looks like this: given a time bound T a function f is reasonable if there exists $c \in \mathbb{R}$ computing $f(n)$ in less than $cT(n)$ steps (equivalently $f \in \mathsf{DTIME}(T)$). Example of reasonable functions are: usual functions, composition of reasonable functions, $log^*(n)$ (where log^* is the iterated logarithm).

When we call a function slow-growing we informally mean that it tends towards infinity and it does so really slowly (if g is called slow growing one may think $g = \sqrt{\log^*}$).

2.2 Kolmogorov related notions

We define Time-space bounded Kolmogorov complexity as it is usually defined, for instance in the book of Li and Vitanyi [6, section 7.1.2], or in Longpré [8].

Definition 2.1 (Time-space bounded Kolmogorov complexity)**.** Let ϕ be a computable function from strings to strings. Let $T \in \mathbb{N}, x \in \{0;1\}^*$. We define $C_\phi^T(x) = min\{|p|; \phi(p)$ outputs x and stops in T steps or less $\}$

Theorem 2. There exists a recursive function ϕ_0 such that for every other recursive function ϕ, there is a constant c such that $C_{\phi_0}^{ctlog(t)}(x) \leq C_\phi^t(x)$ for all x. The constant c depends on ϕ only.

Proof. You may find the proof in [6] □

The log factor in this theorem comes from the time overhead of the best known simulation by universal Turing machines.

Definition 2.2 (Universal efficient Turing machine)**.** For any Turing machine M, $\#M \in \{0;1\}^*$ denotes a reasonable encoding of M. Let $\mathbb{U}$ be a Turing machine. We say that $\mathbb{U}$ is a universal efficient Turing machine if:

- the function ϕ_0 defined by $\mathbb{U}$ respects the condition of theorem 2, and
- for all Turing machines M and strings x, $\mathbb{U}(< \#M, x >) = M(x)$ where $< \#M, x >= E_2(\#M)x$.

In the following we fix $\mathbb{U}$ to be a universal efficient Turing machine. We will abusively treat $\mathbb{U}$ as a program when necessary.

Definition 2.3 (Universal time-space bounded Kolmogorov complexity)**.** Let $T \in \mathbb{N}^2, w \in \{0;1\}^*$. We define $C^T(w)$ as

$$\min\{|p| \mid \mathbb{U}(p) =_t w\},$$

where $\mathbb{U}(p) =_t w$ means that $\mathbb{U}$ computes w on input p in at most t steps.

Remark 1. Note that here p is treated as the encoding of a pair consisting of a machine and its input. The size of a program is actually the size of the encoding of the program and its input.

Remark 2. For a string w, C^T is less than f if there is a program p of size less than f whose running time *when simulated by* $\mathbb{U}$ is less than T. This should be opposed to: a program p of size less than f whose "real" running time is less than T. As a consequence, the log factor caused by the simulation is already taken into account when considering the time-Kolmogorov complexity of a string. In section 1 and 3 we abusively did not make that difference, but this will become important when we formalize the proof in section 7. In a first reading this difference may be ignored.

For a second reading, let us add some details. When we simulate α distinct programs for T steps in section 5 and 7, we need to simulate them for T steps and not $T \log T$. Because of the tightness of our result, the small speeding factor we achieve would be overshadowed by this $\log T$ factor if it were not already built in the definition of time-bounded Kolmogorov complexity. Let us note that this is not a specifity of our approach, as Longpré result also uses this fact.

Definition 2.4 (Class of time-space bounded Kolmogorov complexity with fixed size)**.** Let T, f be functions from $\mathbb{N}$ to $\mathbb{N}$, and $n \in \mathbb{N}$. We define

$$[f, T](n) = \{w; |w| = n \wedge C^{T(n)}(w) \leq f(n)\}.$$

Longpré denotes this class by $[f(n), T(n)]$ instead.

In this paper we sometimes consider slow growing functions, the slower they grow the better, but we need those functions to exceed $K^{T(n)}(n)$ for infinitely many n. We prove in the next theorem that for $T(n) = 2n$ and for infinitely many n, $K^{2n}(n)$ is less than $\log^* \log^*(n)$. Thus most reasonable slow growing functions exceed $K^{T(n)}(n)$ infinitely many times.

Theorem 3. $\forall c \in \mathbb{N}, \exists_\infty n \in \mathbb{N}$, $K^{2n}(n) + c \leq \log^*(\log^*(n))$.

Proof. Taking $c = 0$ the proof being similar for any c: consider the sequence $n_0 = 2$, $n_k = 2^{n_{k-1}}$. $n_k = (\log^*)^{-1}(k)$. n_k is of size n_{k-1} and can be printed in time $2n_{k-1}$ by a program of size $k = \log^*(n)$. So we have a small and fast program to print $n_k = (\log^*)^{-1}(k)$ by composing the program twice we can print $n'_k = (\log^*)^{-1}(\log^*)^{-1}(k)$ in time $2n'_k$ with a program of size k. □

3 Goal and general strategy

High level description of the paper. Our goal is to study the question

$$\exists_\infty n, [f(n), T(n)] \subsetneq [f(n), \alpha T(n)]?$$

Longpré has shown

$$\exists_\infty n, [f(n), T(n)] \subsetneq [f(n), c2^{f(n)}T(n)].$$

The proof of Longpré follows a standard diagonalization argument. One considers a program $p(n)$ which runs all programs of size $f(n)$ for $T(n)$ steps (this takes time $2^f T$), and then outputs a string of length n which was not the ouput of any of the 2^f enumerated programs. Intuitively, the program p is of size $K^{T(n)}(n) + O(1)$ since we just gave its description (the term $O(1)$) and instead of giving n as an input we can give it a representation of n decodable in $T(n)$ steps (the term $K^{T(n)}(n)$). As a consequence the stronger result we obtain is in fact

$$\exists_\infty [f(n), T(n)] \subsetneq [K^{T(n)}(n) + O(1), c2^{f(n)}T(n)],$$

since $K^{T(n)}(n) + O(1)$ is significantly lower than $f(n)$ for any reasonable functions f and T, and infinitely many n. When f is bigger than $\log(n) + c$, one can change the quantifier $\exists_\infty$ to $\forall_\infty$, since $K^{T(n)}(n) < \log n + c$ for some $c \in \mathbb{N}$.

One try variant. Our idea to prove

$$[f(n), T(n)] \subsetneq [f(n), \alpha T(n)]$$

is to do the following: instead of running every program of size f for T steps (which takes time $2^f T$), we only run α of them for T steps. First we

divide the set of 2^f programs of size f into chunks of size α, these chunks are called C_i for $i \in [0, 2^f/\alpha]$. We then consider $2^f/\alpha$ programs $(p_i)_{i\in[2^f/\alpha]}$. Program p_i will run all the programs of chunk C_i for T steps: some of those may stop before T steps and we remember their outputs (there are at most α such outputs). Call those outputs $c_i^1, c_i^2 \ldots$. Then each program p_i will look at these outputs and produce a new string which hopefully is not the output by *any* program of size f (not only the ones of chunk C_i). Now of course this is where the difficulty of the approach lies: each program p_i only knows a fraction of the possible outputs of programs of size f, and there is no reason to think that it can output something new. However, we only need *one* of those programs to output something new. To that effect, the programs will play a collective game ensuring that at least one of them outputs a "new" string. The second task of our programs, after computing the values c_i^j, is thus to apply a strategy ensuring that one of them succeeds in outputting a new string. Note that this might be possible because the programs *collectively* know all the outputs of programs of size f running for T steps.

In this paper, instead of considering multiple programs p_i we will have one global program [3] p taking as input n and i, n tells it which size we are currently looking at and i tells it to run the i-th chunk of size α programs of size $f(n)$. These programs are of approximate size $\log n + \log(\frac{2^f}{\alpha}) \approx \log n + f - \alpha$. This has to be smaller than f in order to be of any use. To reduce the size we also apply a trick and give a description of n decodable in $T(n)$ steps noted $\#^T n$: the programs are then of approximate size $\mathsf{K}^{T(n)}(n) + f - \alpha$. For any reasonable increasing function α growing to infinity, this is smaller than f for an infinite number of n, that is because $\liminf K^{T(n)}(n)$ grows incredibly slowly as shown in theorem 3. We even have a bit of leeway which allows us to cram in some additional information which we'll use in the multiple tries variants.

As for the running time, simulating α programs for T steps takes αT steps in total. We then have to apply our strategy on the α inputs. The strategy needs to be described, which adds some extra size to program p. When the strategies are uniform this only adds 0(1) bits of size. Running this strategy also adds some extra running time which depends on the precise strategy we implement. In general, this extra time could be greater than αT and forbid us to conclude. However, in practice, it turns out that this is not the limiting factor.

Regardless of size and time constraint, it is not clear at all at this stage that a strategy allowing this proof technique even exists. We will first rephrase

[3] Actually this adds some uniformity to the programs which is not formally needed and might hinder the approach. Without uniformity, one may give additional advice to the programs, for instance information about the inputs of the other programs (hence the terminology of parallel diagonalization *with advice*).

the expected outcome of the strategy as a purely combinatorics problem. We start from an initial list $E \subset [1; N]$, where N is to be thought of as all the possible strings of size n, therefore $N = 2^n$. Here E should be understood as the set of outputs of size n of all the programs of size f running for less than T steps, and therefore[4] $|E| = 2^f$. Then we divide this list E into chunks of size k and give those to p players. Here k is to be thought of as α ($k = \alpha$), and p as the number of different programs (one per chunk), hence $p = \frac{|E|}{\alpha} = \frac{2^f}{\alpha}$. Each of the p players may produce an answer depending only on their k inputs. Their answer must be an element of $[N]$. The players collectively win if at least one of the answers given by the players is not in E. The question is, for the game $\mathcal{G}^N(p, k)$ with p player having k numbers each answering in $[N]$, is there a strategy the players can apply which will work no matter the initial list E. Note here that each player has its own strategy, and no communications are allowed between players during the process.

Remark 3. The combinatorial approach disregards any restriction on the number of bits necessary to describe the strategy and the running time of the strategy. As a consequence, once we will have a winning strategy, we will need to take care of these aspects in order to use it for proving our result about time-bounded Kolmogorov complexity classes.

Remark 4. At this point it is not clear that a strategy even exists. Also, achieving a winning strategy for *any* list E is more than what is needed for our specific problem. Indeed, for the intended application for time-bounded Kolmogorov complexity, we only need a winning strategy when the initial list is composed of the outputs of the programs of size $f(n)$ running for $T(n)$ steps.

Remark 5. The bound $[N]$ is really important. Our programs *must* answer a bitstring of size n (recall the definition of $[f, T](n)$) i.e. a integer in $[N]$. We are given $f(n)$ and $\alpha(n)$ which determine the number $p(n)$ of players and the number $k(n)$ of inputs. Now it may be that the game $\mathcal{G}^N(p(n), k(n))$ has a solution when N is very large but not when N is small [5]. But for our purpose we need N to be less than 2^n. This leads us to an analysis in section 4 of the bound $\mathfrak{B}(p, k)$ which is the smallest N for which game $\mathcal{G}^N(p, k)$ has a solution. This bound $\mathfrak{B}(p, k)$ proves to be the crux and the limiting factor of our results.

Multiple tries variant. Up until this point we have considered that the i-th chunk of α programs was attributed to program $p(n, i)$ of size $\approx K^T(n) + f - \alpha$. As mentioned earlier, we have some leeway on the size of our programs, so we can relax the setting to provide some extra information to

[4] To be more precise, this should be an inequality since there may be repeats or programs that do not stop. But we just consider the worst case here.

[5] We will establish later in the paper that the game is indeed easier when N gets large.

our programs. We present here *one* way of using this extra information. We will now consider replacing $p(n,i)$ by a program $p(\#^T n, i, j)$. This program is the j-th program attributed to the i-th chunk of α programs. It has an additional input j in $[1; 2^\alpha/g]$, where g is any reasonable slow growing function. The size of $p(\#^T n, i, j)$ is therefore approximately $K^T(n) + f - \alpha + \alpha - \log(g) \approx K^T(n) + f - \log(g)$, which is smaller than f for infinitely many n. As for the one-try variant, the program $p(\#^T n, i, j)$ runs the i-th chunk of programs of size $f(n)$ for $T(n)$ steps but can then give an answer which depends on j.

Expressed using our combinatorics problem, each player can now propose m answers instead of giving out a unique answer. The players collectively win the game if one of the answers of one of the players is not in the initial list E. This game is denoted by $\mathcal{G}^N(p,k,m)$.

For the moment, we have not been able to use the ability to use multiple tries to improve on our solution for the one try variant, and we only came up with basic theorems. We nevertheless mention it since it is a natural extension of the underlying combinatorics problem. It may also be a way forward to lower the bound $\mathfrak{B}(p,k,m)$ presented in section 4.

4 Novelty games

In this section, we introduce the combinatorial game behind our technique, under the name of *Novelty games*.

We remind the reader that $\binom{[N]}{k}$ denotes the set of all multisets included in $[1,N]$ of size at most k.

4.1 Definition

Definition 4.1. We now define the game $\mathcal{G}^N(p,k,m)$.

- There are p players.
- Player i receives a multiset $A_i \in \binom{[N]}{k}$; elements of A_i are called inputs.
- Each player has m tries.
- On each try a player answers an element of $[1;N]$

An *occurrence* of the game $\mathcal{G}^N(p,k,m)$ is a specific family $(A_i)_{i=1}^p$.

A *strategy* $\mathfrak{S}^N(p,k,m)$ for $\mathcal{G}^N(p,k,m)$ is a family of functions $(s_i)_{i=1,\dots,p}$ where

$$s_i : \binom{[N]}{k} \times [m] \to [N].$$

The players collectively win the occurrence $(A_i)_{i=1}^p$ of $\mathcal{G}^N(p,k,m)$ if there exists a pair $(i,j) \in [p] \times [m]$ such that $s_i(A_i,j) \notin \cup_{i=1}^p A_i$.

A *winning strategy* for $\mathcal{G}^N(p,k,m)$ is a strategy such that all occurrences of the game are won.

Notations 1. When $m = 1$, we may write $\mathcal{G}^N(p,k)$ instead of $\mathcal{G}^N(p,k,1)$. We may sometimes drop the superscript and subscript altogether when they are not relevant or clear from the context.

When all players have the same strategy, i.e. when $\mathfrak{S} = (s_i)_{i=1}^p$ is such that there exists a function s with $s_i = s$ for all $i = 1, \dots, p$, we say that $\mathfrak{S}^M(p,k,m)$ is *oblivious*.

Definition 4.2. We define $\mathfrak{B}(p,k,m)$ as the smallest integer N such that there exists a winning strategy for the game $\mathcal{G}^M(p,k,m)$.

We note that $\mathfrak{B}(p,k,m)$ is well defined, as proven in section 6.2. Establishing an estimate of $\mathfrak{B}(p,k,m)$ in general remains an open question.

4.2 General remarks about Novelty games

Sets. The first remark is that one may only consider sets of size exactly k instead of multisets of size at most k. This is because the hardest case for the strategy is when each player has k different numbers. Indeed, suppose there exists a strategy $\mathfrak{S}$ winning in all occurrences of the game for which players have a set of size k (or, equivalently, a multiset of size k without repetitions). Then one can easily construct a winning strategy for all multisets: when the considered multiset have less than k distinct numbers, it suffices to change repeated occurrences by arbitrary new ones and if you still don't have k values you can add arbitrary numbers to reach k. One can can apply the strategy $\mathfrak{S}$ on the resulting set. A routine check suffices to show that this strategy is winning.

In the following, we will therefore always assume that inputs are pairwise distinct, and that we may add arbitrary new inputs so that sets have size exactly k.

We also note that inputs are not ordered, since we work with sets and not tuples. But we may suppose, if need be, that they are by chosing an arbitrary order relation.

The next theorem states that the existence of strategy for inputs in $[1;N]$ implies the existence of a strategy for inputs in $[1;M]$ for all $M \geqslant N$. This motivates the definition of $\mathfrak{B}$.

Theorem 4. Let $M, N \in \mathbb{N}$ such that $M \geqslant N$. If $\mathcal{G}^N(p,k,m)$ has a winning strategy then so does $\mathcal{G}^M(p,k,m)$

Proof. We treat the case $M = N+1$ and then we can conclude by induction. Let $\mathfrak{S}$ be a winning strategy for $\mathcal{G}^N(p,k,m)$. We define a strategy $\mathfrak{S}'$ for the game $\mathcal{G}^M(p,k,m)$ as follows. If M is not an input of strategy $\mathfrak{S}'$, then

each player simply runs $\mathfrak{S}$. Otherwise, the player replaces M by N and runs the strategy $\mathfrak{S}$ on the resulting set. One easily checks that $\mathfrak{S}'$ is a winning strategy for $\mathcal{G}^M(p,k,m)$ since the output of $\mathfrak{S}'$ is in $[N] \subset [M]$. □

Theorem 5. For all p, k, m in $\mathbb{N}$, and all a, b, c, d in $[0,1]^3$,

$$\mathfrak{B}(p-a, k-b, m+c) \leq \mathfrak{B}(p,k,m).$$

Proof. This is proven by noticing that a strategy for the game $\mathcal{G}^M(p,k,m)$ is also a strategy for $\mathcal{G}^M(p-a,k-b,m+c)$: it suffices to add b random inputs, apply the strategy on those (since there are less players, the strategy determines a map for each player – and the maps s_i for $i = p-a+1,\ldots,p$ are unused), and guess c arbitrary additional outputs. □

We believe establishing recurrent relations between values of $\mathfrak{B}$ is hard when varying the number of players, numbers, tries or the size of the set of inputs. For instance, it is not even clear how the inequality of theorem 5 can be made strict.

We now start by proving a simple lower bound on $\mathfrak{B}$.

Theorem 6. For all $p,k,m \in \mathbb{N}$, $\mathfrak{B}(p,k,m) \geq pk+1$.

Proof. It is easy to realise that there are no winning strategy if $N = pk$: if for all $i \in [1;p]$, player i has as inputs $[(i-1)k+1; ik]$, then the set of all inputs is equal to $[1;pk]$ and no player may output a new number. □

Another easy result to establish is that if one allows for enough tries, the value of $\mathfrak{B}$ can be determined (and is low).

Theorem 7. For all $p,k \in \mathbb{N}$, $\mathfrak{B}(p,k,k) = pk+1$.

Proof. Let $N = pk+1$. Each player calls its inputs $i_1,\ldots,i_k$. On its j-th try each player does this: if $i_j + 1 \leq N$ it answers i_j+1 otherwise it outputs 1. It's routine to check that this is a winning strategy. □

The study of multiple tries has proven to be hard. Apart from quite obvious results, we have not found ways to improve on the results we obtain in the one-try variant. We therefore leave the following question for future work.

Question 1. Can one find a better bound for $\mathfrak{B}(p,k,k-1)$ than $\mathfrak{B}(p,k,1)$?

We summarise the results of this paper in fig. 1. The proof are scattered in the paper.

Game	$(p,k,1)$	(p,k,i)	(p,k,k)
Upper bound	$(k^p)^{k^p}$	$(k^p)^{k^p}$	$pk+1$
Reference	section 6.2	section 6.2	section 4.2

(a) Bounds for general values p,k

Game	$(2,2,1)$	$(1,k,1)$	$(p,1,1)$
Upper bound	9	$k+1$	$p+1$
Reference	section 4.4	section 4.2	section 4.2

(b) Bounds for specific values

Figure 1: Results of the paper

4.3 Novelty games for Kolmogorov complexity

If we want to prove that $[f,\alpha T](n) \not\subseteq [f,T](n)$, We are interested in studying game $\mathcal{G}^N(p,k,m)$ where:

- $p = 2^f/\alpha$;
- $k = \alpha$;
- $m = k/g$ where g is any reasonable slow growing function;
- $N = 2^n$.

Our bound $\mathfrak{B}(p,k,m)$ should be less than N because a program must answer a string of size n (i.e. an integer of $[N]$). This gets us to analyze when:

$$\mathfrak{B}(p,k,m) < N \Leftrightarrow \exists g : \mathbb{N} \to \mathbb{N}, \forall n, \mathfrak{B}\left(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n), \frac{\alpha(n)}{g(n)}\right) \leq 2^n.$$

Reminding the reader that f, T and α are all functions of n, we are thus particularly interested in analysis of the bound $\mathfrak{B}(p(n),k(n),m(n))$ in which $p(n)$, $k(n)$, and $m(n)$ are functions going to infinity and $m(n) = k(n)/g(n)$ where $g(n)$ is a very slowly growing function. $p(n)$ must go to infinity as there is one player per chunk of programs to be simulated and the number of chunks grows with n. We have no formal argument establishing that $k(n)$ should go to infinity: each player could have a constant number of inputs as n grows, but this would require to refine the size analysis of our programs, at the very least. Likewise - with no size analysis refinements - $g(n)$ has to grow towards infinity, however the slower g grows the more tries for each player.

Remark 6. This subsection was meant to give precise links between our Kolmogorov problem and values for novelty games. But we believe analysing novelty game $\mathfrak{B}^N(p,k,m)$ for any p,k,m is the right way forward.

4.4 Resolution of $\mathcal{G}(2,2)$ case

This partial solution was chronologically the first to be proposed, by Corentin Henriet. We chose to present it since the strategy is different from the ones presented in the next sections and it provides interesting bounds.

This strategy is oblivious. We will thus define the single map s used by both players. The strategy is winning as long as $N \geq 9$, showing that $\mathfrak{B}(2,2,1) \leq 9$ (much better than the general bound obtained in section 5, which gives 216). In this specific case, we can also prove that $\mathfrak{B}(2,2,1) > 5$ (to be compared to 4 – the general lower bound obtained in theorem 6).

Definition 4.3 (Strategy s)**.** We intuitively understand our inputs as being on a "clock" from 1 to N, i.e. the successor of N is 1. Let (x,y) be our inputs ordered such that the distance between x and y is less than the distance from y to x when read in the clockwise direction. If $y = x+1$ or $y = x+2$, then $s(x,y) = x-2$, otherwise we define $s(x,y) = x+1$.

Theorem 8. $\mathfrak{B}(2,2,1) \leq 9$.

Proof. The proof consists in verifying that s is a winning strategy. This is done by checking every possible scenario. □

Theorem 9. $\mathfrak{B}(2,2,1) > 5$.

Proof. This is easily checked by pen and paper search. □

While this strategy provides good bounds, we have not found a way to generalise it to more players or larger sets. The next section presents a general strategy for the 2-players case.

4.5 Graph strategies

This line of reasoning, and the corresponding winning strategy for $\mathcal{G}(2,k,1)$ was first obtained by Dmitriy Kunisky.

We first recall the definition of colored graphs and establish some useful notations.

Definition 4.4 (Colored graphs)**.** Let $p \in \mathbb{N}$. A p-colored graph is a triple (V,E,C) where V is a set of vertices, $E \subseteq \{(x,y) \in V^2 \mid x \neq y\}$ is a set of edges, and $C : E \to [p]$ a coloring function mapping every edge to a color (an element of $[1;p]$).

The graph is said to be complete when $E = \{(x,y) \in V^2 \mid x \neq y\}$. We will moreover write $G_i = (V, E_i)$ the graph induced by selecting the edges of a given color i, i.e. $w \in E_i \Leftrightarrow w \in E \wedge C(w) = i$. Given $u, v \in V$, we write $u \sim v$ when $\{u;v\} \in E$; in this case we say that u and v are neighbors. We moreover write $u \overset{i}{\sim} v$ when $u \sim v$ and $C(\{u;v\}) = i$.

We will now define a property of colored graphs which will be used later to define a winning strategy.

Definition 4.5 ((p,k)-set property)**.** Let $p, k \in \mathbb{N}$. A p-colored graph $G = (V, E, C)$ is said to have the (p,k)-set property if for every subset of vertex $A \subseteq V$ of size k and every color $i \in [1;p]$, the vertices in A have a mutual neighbor v in G_i, i.e. such that for all $a \in A$, $a \overset{i}{\sim} v$.

We will now explain how the (p,k)-set property ensures that one can define a strategy for p players and sets of size k from a p-colored graph. The intuition is the following: each input will correspond to a vertex, each player will correspond to a color, and a player will use the (p,k)-set property instantiated on its color to produce an output.

We now recall the notion of polychromatic cycle that will be useful to discuss winning strategies.

Definition 4.6 (Polychromatic cycles)**.** Let $p \in \mathbb{N}$ and $G = (V, E, C)$ be a p-colored graph. A cycle c of G is said to be polychromatic if edges in c have pairwise different colors.

Note that a polychromatic cycle does not necessarily use all colors. The non-existence of polychromatic cycles will ensure that the strategy will be winning (i.e that no all player's answer are an input of another one). For the interested reader, graphs with no polychromatic cycles have been studied here [2, 1].

Definition 4.7 (Good graphs)**.** Let $p, k \in \mathbb{N}$, and let G be a p-colored graph. We say that G is (p,k)-good if it has the (p,k)-set property and has no polychromatic cycles.

We now formally describe the strategy induced by a (p,k)-good graph $G = (V, E, C)$ with $V = [1; N]$. The map $s_i^G : \binom{[N]}{k} \mapsto [N]$ defining the strategy of the i-th player is defined as $s_i^G(A) = y$ where y is any vertex (for instance, the smallest) such that $\forall x \in A, x \overset{i}{\sim} y$. Such a y always exists thanks to the (p,k)-set property. We can then establish that the absence of polychromatic cycles implies that this strategy is winning.

Theorem 10. Let $p, k \in \mathbb{N}$ and let G be a p-colored graph with N vertices. If G is (p,k)-good, the strategy $\mathfrak{S} = (s_i^G(A))_{i \in [1;p]}$ is winning for the game $\mathcal{G}^N(p,k,1)$.

Proof. Suppose the strategy is not winning. Then there exists an occurence $A_1, \ldots, A_p$ of the game for which the strategy fails (here A_i denotes the set of inputs of player i). Consider a graph H where the vertices are players P_1 to P_p, with a directed edge from P_i to P_j if the answer of player i, i.e. $s_i^G(A_i)$, belongs to the set of inputs A_j of player j. By assumption, there exists a

cycle in this graph (otherwise the strategy is winning on this occurence). But one can check that a cycle in H implies the existence of a polychromatic cycle in G, leading to a contradiction. □

The last step is now to show that $(2, k)$-good graphs exist. We will then discuss, in subsection 4.5.2, the unlikely existence of (p, k)-good graphs for $p > 2$ and $k \geq 2$.

4.5.1 Graph strategies for $(2, k)$

Theorem 11. For any $k \in \mathbb{N}$, there exists a $(2, k)$-good graph.

Proof. We will prove that there exists an undirected uncolored graph $G = (V, E)$ such that for every $A \in \binom{[N]}{k}$:

- $\exists a, \forall x \in A, x \sim a$;
- $\exists b, \forall x \in A, x \nsim b$.

This is sufficient to prove the existence of a $(2, k)$-good graph by converting every edge to a 1-colored edge and adding a 2-colored edge for every pair of vertices $x \neq y$ such that $x \nsim y$. Note that this graph has no polychromatic cycles of size 2 since two vertices cannot be simultaneously neighbors and non-neighbors.

We now prove the existence of such graphs using the probabilistic method. Let G be an Erdős-Rényi graph with edge probability $\frac{1}{2}$ on the vertex set $[1; N]$. Then, the probability $\mathbb{P}[G \text{ is } k\text{-good}]$ goes to 1 as N goes to ∞:

$$\begin{aligned}
&\mathbb{P}[G \text{ is not } k\text{-good}] \\
&\quad \leq \mathbb{P}[\text{some } i_1, \ldots, i_k \in [N] \text{ have no mutual neighbor}] \\
&\qquad + \mathbb{P}[\text{some } i_1, \ldots, i_k \in [N] \text{ have no mutual non-neighbor}] \\
&\quad \leq 2 \cdot \mathbb{P}[\text{some } i_1, \ldots, i_k \in [N] \text{ have no mutual neighbor}] \\
&\quad \leq 2\binom{N}{k}\mathbb{P}[1, \ldots, k \text{ have no mutual neighbor}] \\
&\quad \leq 2\binom{N}{k}\left(1 - \frac{1}{2k}\right)^{N-k} \\
&\quad \leq 2\exp\left(k \log N - \frac{N-k}{2k}\right)
\end{aligned}$$

Note we used symmetry to deduce the second inequality. □

For large values of k, we have that $2\exp\left(k \log N - \frac{N-k}{2k}\right)$ is less than 1 when $N = O(k^3 2^k)$. The general bound of section 6 gives a worse bound $O(2^{k^2 \log k})$.

Size and running time of the graph strategy. If we want to use a strategy based on graphs in the programs of section 5 to show $[f, \alpha T] \subsetneq [f, T]$, we need a succinct description of such graphs. Indeed, the program needs to be of size less than f. Note however that we have only demonstrated the existence of a $(2, k)$-good graph thanks to the probabilistic method. The best we can naively do is then to hardcode those graphs in the strategy. This would make the strategy really large in terms of bits (close to $N^2 = 2^{2n}$), but our programs can never be bigger than $f < n$ bits. Another approach is to let the programs find those graphs by itself: every program $p(\#^T, i)$ enumerates and tests graphs in ascending lexicographic order until it finds a good graph, and then use it for its strategy. While this strategy takes $O(1)$ bits to describe, it takes *at least* 2^n steps to run implying that one could only establish bounds for $T > 2^n$.

We also hint (but not formally prove) at a method of constructing $(2, k)$-good graphs without using probability theory. We briefly describe such a potential graph using the notion of pseudorandom *Paley graph* from number theory. A Paley graph is a graph G on a prime number $p =\equiv 1 \mod 4$ of vertices. Vertices of G are identified with integers modulo p, and there exists an edge $i \sim j$ if and only if $i - j$ is congruent to a square modulo p, i.e. if and only if $\exists k, i - j = k^2 \mod p$. The condition $p \equiv 1 \mod 4$ ensures that the relation $\sim$ is symmetric.

Finding a mutual neighbor of $x_1, \ldots, x_k$ then amounts to solving the system of equations $\{y - x_i = z_i^2\}$ in variables $y, z_1, \ldots, z_k$. One may check, using character sum estimates, that many integers y satisfying these equations exist, for any choice of $z_1, \ldots, z_k$, as long as p is sufficiently large.

4.5.2 Graph Strategies for (p, k)

In the previous section, we established the existence of winning strategies for the games $\mathcal{G}^N(2, k, 1)$, defined from colored graphs. Can we generalise the approach to obtain winning strategies for all games $\mathcal{G}^N(p, k, 1)$? If we were to follow the same method, we would need the existence of (p, k)-good graphs. Unfortunately, we have not been able to prove or disprove the existence of such graphs for $p > 2$. However, we can establish that their are no complete (p, k)-good graphs (note that the $(2, k)$-good graphs we came up with are complete).

Definition 4.8. A p-colored graph G is said to be connected for color i if there is a i-colored path between any two vertices in G

Theorem 12. (Proven in [2]) Let G be a complete p-colored graph. If G has no polychromatic cycles, then G is connected for at most 2 colors.

Theorem 13. There are no complete (p, k)-good graphs for $p > 2$ and $k \geq 2$.

Proof. Let $G = (V, E, C)$ be a complete (p, k)-good graph. Then for any color i and any two vertices $u, v \in V$, there exists $w \in V$ such that $u \overset{i}{\sim} w$ and $v \overset{i}{\sim} w$. This implies that u and v are connected for every color. Whenever there are $p \geq 3$ colors, this contradicts Theorem 12 □

Can graph strategies be salvaged? Graph strategies are an interesting approach, and they provide – in the case $p = 2$ –, better bounds that the general strategy defined below. However it is unclear if (p, k)-good graphs exist. An approach would be to get rid of them altogether and try to understand what kind of graphs we could be used for the multiple tries setting. Surely, such graphs would have less stringent requirements (but which requirements this is remains unclear at the moment). Another approach would be to consider uncolored but *directed* graphs instead of colored undirected graphs, requiring the following properties:

- there exists no cycles of length less than p;
- for any set $i_1, \dots i_k$ there exists a vertex b such that $\forall j \in [1; k], i_j \to b$ (i.e. there exists a directed edge of source i_j and target b).

If we could prove the existence of such graphs, it could be used to define a winning strategy for the game $\mathcal{G}(p, k, 1)$. Indeed, every player would answer a common "directed" neighbor of its $k-$inputs using the second property. The fact that the strategy is winning would then directly follow from the first property. However, the existence of such graphs also remains an open question at the moment.

5 Formal description and analysis of the programs

Recall that we want in the end to show that $[f, \alpha T] \nsubseteq [f, T]$. The functions $f(n)$, $\alpha(n)$ and $T(n)$ are all supposed to be in $\mathsf{DTIME}(T(n))$, i.e. there is a program which computes them in $O(T(n))$ steps and which can be hardcoded into any program for a cost of $0(1)$ bits. We also suppose that α is less than $K^{T(n)}(n)$ for infinitely many n; as established by theorem 3, this is not a stringent condition.

In the following, we will often drop the arguments of the functions and write, for instance, f instead of $f(n)$. We also suppose w.l.o.g. that $T(n) > n$ and $f < n$. Indeed, if $T(n) < n$, then it is not possible to write any string of length n in the allocated time, and if $f \geq n$ then all inputs have a description of size bounded by f and there are no gap in the hierarchy. We also assume for simplicity that 2^f is divisible by α.

We will in the end only be interested in values of n for which there exists a program $\#^T n$ outputting n in T steps or less, and such that $\alpha(n) > K^T(n)$. A careful reading of the proof of theorem 3 ensures the existence of infinitely many such n.

For the moment, we will assume that for all $p, k, m, N \in \mathbb{N}$, we have a strategy (not necessarily winning) $\mathfrak{S}^N(p,k,m) = (\sigma_i^N(p,k,m))_{i\in[p]}$ for the game $\mathcal{G}^N(p,k,m)$, where σ_i^N is the map defining the strategy for player i. The following definition of programs does not depend on the specific strategy considered, but it should be clear that their analysis will be dependent on this choice.

5.1 One try variant

We formally define the program which is intended to output a string in $[f, \alpha T](n)$ and not in $[f, T](n)$ when the strategy $\mathfrak{S}^N(p,k,1)$ is winning. We then proceed with the size and running time analyses.

Definition of the program p The program p takes as input two strings of $\{0;1\}^*$:

- $\#^T n$ is taken to be the smallest program outputting n in $T(n)$ steps or less;
- i which is the number of the chunk of programs to be run: it ranges between 1 and $2^f/\alpha$.

The program $p(\#n^T, i)$ then performs the following computations:

1. It runs the program $\#^T n$ until it stops. We call n its output.
2. It computes $f(n)$, $T(n)$, and $\alpha(n)$.
3. It creates an empty list l which will be used to store the outputs.
4. For j ranging from 0 to $\alpha - 1$, it simulates the program number $(i-1)\frac{2^f}{\alpha} + j$ running for $T(n)$ *simulation* steps[6]: if this program outputs a value before stopping, it adds the output to the list l.
5. It runs the strategy $\sigma_i^N(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n), 1)$, where $N = 2^n$ and m is the size of list l, on inputs $l[0], l[1], \ldots, l[m-1]$, and outputs the result.

Size analysis. We now analyze the size of $p(\#^T n, i)$:

- $\#^T n$ is of size $K^T(n)$ (by definition);
- i is an integer in $[1; 2^f/\alpha]$, hence it is of size $f - \log(\alpha)$.
- $(\#^T n, i)$ can be uniquely described by a string of size $K^T(n) + \log(K^T(n)) + f - \log(\alpha)$ from theorem 1;

[6] On the universal Turing machine used to define our Kolmogorov complexity

- the program p is of size $O(1)$.

We conclude that $p(\#^T n, i)$ is of size $K^T(n) + f - \log(\alpha) + O(1)$. This is strictly less than $f(n)$ for infinitely many values of n (for any reasonable function α see theorem3).

Running time analysis. The steps 1 to 4 take time αT to compute. The running time of step 5 depends on the specific strategy $\mathfrak{S}^N(2^f/\alpha, \alpha)$ we choose to implement. Hopefully this running time can be kept in $O(\alpha T)$, otherwise we cannot (directly) conclude.

Correctness analysis. Suppose that the strategy $\mathfrak{S}^{2^n}(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n))$ used in the program is a winning strategy for the game $\mathcal{G}^{2^n}(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n))$. Let $\#n$ be a program outputting n. Then, by construction, there exists $i \in [1, 2^f/\alpha]$ such that $p(\#n, i)$ outputs a number of size n which is not the output of any program of size f in less than T steps.

Conclusion Size and running time analysis are meant to prove that the output of $p(\#^T n, i)$ is in $[f, \alpha T](n)$. Correctness analysis shows that for at least one i, $p(\#^T n, i)$ is not in $[f, T](n)$. Thus showing $[f, T](n) \subsetneq [f, \alpha T](n)$.

5.2 Multiple tries variant

Let g be a very slow-growing reasonable function computable in $T(n)$ steps, such as $g = \sqrt{\log^*}$). The multiple tries setting is quite similar to the one try setting, except that we will have an additional input j indicating which try is computed. We formally define the program which is intended to output a string in $[f, \alpha T](n)$ and not in $[f, T](n)$ when the strategy $\mathfrak{S}^N(p, k, m)$ is winning.

Definition of the program p. The program p now takes as input three strings of $\{0; 1\}^*$:

- $\#^t n$ as in the one-try case;
- i as in the one-try case;
- an integer j ranging in $[1; \alpha/g]$.

The program $p(\#n, i)$ then performs the following computations.

1. It follows the steps 1, 2, 3, and 4 of the program defined for the one try variant.
2. It runs the strategy $\sigma_i^N(\frac{2^{f(n)}}{\alpha(n)}, \alpha(n), j)$, where $N = 2^n$ and m is the size of list l, on inputs $l[0], l[1], \ldots, l[m-1]$, and output the j-th answer.

Size analysis. We analyze the size of $p(\#^T n, i)$:

- $\#^T n$ can be chosen to be of size $K^T(n)$ (by definition);
- i is of size $f - \log(\alpha)$;
- j is of size $\log(\alpha) - \log(g)$;
- $(\#^T n, i)$ can be uniquely described by a string of size $O(K^T(n)) + f - \log \alpha$;
- the program p is of size $O(1)$.

As a consequence, $p(\#^T n, i)$ is of size $O(K^T(n)) + f - \log(\alpha) + O(1)$.

Running time and correctness. The analysis is similar to the one try case.

6 General Strategy for (p, k)

We will now exhibit a general strategy. This is an oblivious[7] strategy: the same function is used by all players. Intuitively, we will be using the remainder of arguments modulo some large enough number m to ensure disjointness. The strategy however remains involved as it requires to keep track of the remainders modulo m of antecedents.

We will start by detailing the $(3, 2)$ case before generalising to any pair (p, k).

6.1 Strategy for (3,2)

The function $M^2 \to M$ behind the strategy can be explained informally as follows. Given two arguments x, y, we will suppose their digits when written in base m correspond to the remainders modulo m of their antecedents through F. Here m will be chosen to be equal to 15, as explained below. Since we need to ensure that no cycles of size $\leqslant 3$ appear, we will need to keep track of the values of such remainder up to depth-2 antecedents (i.e. antecedents of antecedents). For instance, x will be written as $x = x_7x_6x_5x_4x_3x_2x_1$ in base m, and we understand:

- x_2, x_3 as the remainders of the antecedents of x through F,
- x_4, x_5 as the remainders of the antecedents of x_2 through F,
- x_6, x_7 as the remainders of the antecedents of x_3 through F.

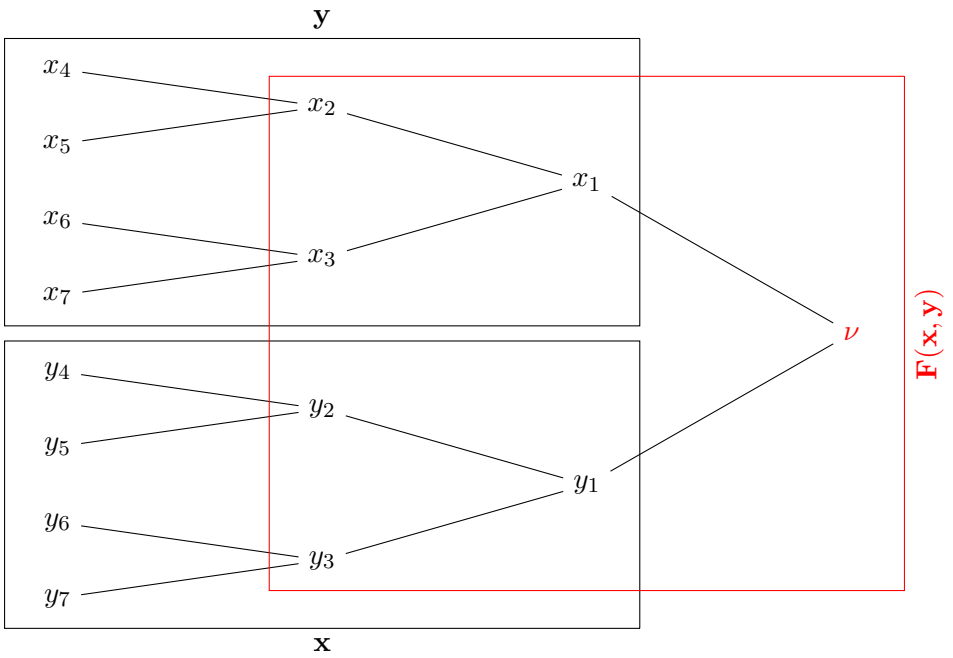
[7] We chose not to use the term uniform as to not create confusion with the uniformity of computability theory

Similarly, we suppose the writing of $y = y_7y_6y_5y_4y_3y_2y_1$ to keep track of the remainders modulo m of the antecedents of y.

The function $F_{(3,2)}$ then just implements this notion of keeping track of the antecedents, while producing a fresh value modulo m. More precisely, given $x = x_7x_6x_5x_4x_3x_2x_1$ and $y = y_7y_6y_5y_4y_3y_2y_1$, the function F will pick a value $\nu \notin \{x_1, \ldots, x_7, y_1, \ldots, y_7\}$. Since there are 14 values in the latter set, choosing $m = 15$ ensures that this is always possible. The function is then defined as

$$F_{(3,2)}(x_7x_6x_5x_4x_3x_2x_1, y_7y_6y_5y_4y_3y_2y_1) = x_3x_2y_3y_2x_1y_1\nu.$$

The following illustration should clarify the definition of $F_{(3,2)}$.



Definition 6.1. The function $F_{(3,2)}$ is defined on $[0, 15^7]$ as follows:

$$F_{(3,2)}(x, y) = x_3.15^6 + x_2.15^5 + y_3.15^4 + y_2.15^3 + x_1.15^2 + y_1.15 + \nu,$$

where $x = \sum_{i=0}^{6} x_{i+1}.15^i$, $y = \sum_{i=0}^{6} y_{i+1}.15^i$, and $\nu \in \{0, \ldots, 14\}$ does not belong to $\{x_i \mid i = 1, \ldots, 7\} \cup \{y_i \mid i = 1, \ldots, 7\}$.

Note that by definition $F_{(3,2)}(x, y)$ is necessarily different from x and y.

Theorem 14. The function $F_{(3,2)}$ provides a uniform winning strategy for 3 players with 2 numbers.

Proof. We will denote by a, b the numbers given to player 1, by c, d the numbers given to player 2, and by e, f the numbers given to player 3. The goal is to show that if each player applies the function $F_{(3,2)}$ then they always collectively win the game.

As in the definition above, we write $a = a_6a_5a_4a_3a_2a_1a_0$ the base 15 representation of a, $b = b_6b_5b_4b_3b_2b_1b_0$ the base 15 representation of b, etc.

If $F_{(3,2)}(a, b)$ is different from c, d, e, f, the players have won the game. Suppose now that we are in the case where $F_{(3,2)}(a, b) \in \{c, d, e, f\}$. Without loss of generality, we can suppose that $F_{(3,2)}(a, b) = c$. In particular: $c_0 \neq a_i$ for all $i \in [0, 6]$, $c_0 \neq b_i$ for all $i \in [0, 6]$ $c_2 = a_0$, $c_5 = a_1$, and $c_6 = a_2$. We now consider the result produced by player 2, that is $F_{(3,2)}(c, d)$. Once again, if this value is different from a, b, e, f then the game is won. If this is not the case, there are two cases.

The first case is $F_{(3,2)}(c, d) \in \{a, b\}$. We now show that this is impossible. By definition, $F_{(3,2)}(c, d)$ is different modulo 15 from all c_i ($i \in [0, 6]$), but c_1 is the remainder modulo 15 of b and and c_2 is the remainder modulo 15 of a. Hence the value of $F_{(3,2)}(c, d)$ modulo 15 is different from the values of a, b modulo 15.

The second case is $F_{(3,2)}(c, d) \in \{e, f\}$. Without loss of generality, we suppose $F_{(3,2)}(c, d) = e$. We will now have to show that $F_{(3,2)}(c, d) \notin \{a, b, c, d\}$. Let us write $z = z_6z_5z_4z_3z_2z_1z_0$ the representation of $F_{(3,2)}(c, d)$ in base 15. By definition of $F_{(3,2)}$, $z_0 \neq e_i$ for all $i \in [0, 6]$. But since $F_{(3,2)}(c, d) = e$ and $F_{(3,2)}(a, b) = c$, one has the following properties: $e_1 = d_0$, $e_2 = c_0$, $e_5 = c_1 = b_0$, $e_6 = c_2 = a_0$. Thus z_0 is different from a_0, b_0, c_0, and d_0. This proves that $F_{(3,2)}(e, f) \neq a, b, c, d$, and therefore the game is won. □

6.2 General strategy for (p,k)

We now expose the general strategy. The construction follows the $(3, 2)$ case just described. In that case, the main insight of the strategy was to book-keep the remainders modulo m of the potential antecedents up to depth 2. For general (p, k), we will therefore keep track of the potential antecedents up to depth $p-1$. This means that for each of the k arguments, we will need values modulo m of the potential $1+k+k^2+k^3+\cdots+k^{p-1}$ antecedents. Since there are k arguments, this means that we will be manipulating $k(1+k+k^2+k^3+\cdots+k^{p-1})$ values modulo m, and produce a fresh value. This implies that m should be greater or equal to $1+k(1+k+k^2+k^3+\cdots+k^{p-1}) = \frac{k^{p+1}-1}{k-1}$.

Now, since we need to book-keep $1 + k + k^2 + k^3 + \cdots + k^{p-1} = \frac{k^p-1}{k-1}$ antecedents, we will require that:

$$M \geqslant \left(\frac{k^{p+1}-1}{k-1}\right)^{\frac{k^p-1}{k-1}}.$$

We will here suppose that M is minimal, i.e. the above inequality is in fact an equality. We introduce the notation $k^{(d)} = \frac{k^d-1}{k-1}$; by convention, we define $k^{(0)} = 0$. Suppose given arguments $x_i \in [0, M]$. We write those in base $m = \frac{k^{p+1}-1}{k-1}$ as follows:

$$x_i = \sum_{d=0}^{p-1} \sum_{c=0}^{k^d-1} x_i(d,c) m^{k^{(d)}+c}.$$

The definition of $F_{(p,k)}$ then simply generalises the definition of $F(3,2)$ given above.

Definition 6.2. For any p, k we define the function $F_{(p,k)}$ as follows:

$$F_{(p,k)}(x_0, \ldots, x_{k-1}) = \mu + \sum_{d=1}^{p-1} \sum_{i=0}^{k-1} \sum_{c=0}^{k^{d-1}-1} x_i(d-1,c) m^{i.k^{(d-1)}+c},$$

where μ is a value different from all $x_i(d,c)$ (say the smallest one for a deterministic function).

Following the proof of Theorem 14, one can establish that the strategy is winning, leading to the following result.

Theorem 15. $\mathfrak{B}(p,k,1) \leq \left(\frac{k^{p+1}-1}{k-1}\right)^{\frac{k^p-1}{k-1}}$.

Corollary 1. For large values of p and k, $\mathfrak{B}(p,k,1) \leq (k^p)^{k^p}$.

7 Results

We now combine the program of section 5 and the general strategy described in section 6.2 for the game $\mathcal{G}^N(p,k,1)$ with $N = (k^p)^{(k^p)}$. We obtain in this way the following result which improves on the result by Longpré that $\exists_\infty n [f, 2^f T](n) \not\subseteq [f,T](n)$.

Theorem 16. Let T and f be functions of $\mathbb{N} \mapsto \mathbb{N}$ in $\mathsf{DTIME}(T(n))$ such that $T(n) \geq n$ and $f(n) = o(\log(n))$. Let $\alpha = \frac{f 2^f}{\log n}$. Then:

$$\exists c \in \mathbb{R}, \exists_\infty n, [f, c\alpha T](n) \not\subseteq [f,T](n).$$

Proof. The proof consists of two parts. First, we analyze the bound $\mathfrak{B}(p,k)$ which will give us the restrictions on f and α. We will then check that there are programs implementing the general strategy and having the proper size and running time.

Let us recall that that we need $\mathcal{B}(p,k) = (k^p)^{(k^p)}$ to be bounded above by 2^n. Here we have:

- $p = 2^f/\alpha$
- $k = \alpha$
- $pk = 2^f$
- We need $(k^p)^{k^p} \leq 2^n$

We can then compute:

$$(k^p)^{k^p} \leq 2^n \quad \Leftrightarrow \quad (1)$$
$$k^p p \log(k) \leq n \quad \Leftrightarrow \quad (2)$$
$$\alpha^{2^f/\alpha} 2^f/\alpha \log(\alpha) \leq n \quad \Leftrightarrow \quad (3)$$
$$2^f/\alpha \log \alpha + f - \log \alpha + \log\log \alpha \leq \log n \quad \Leftrightarrow \quad (4)$$

This implies that $f(n)$ must be in $o(\log n)$. Under this assumption, taking any $\alpha \geq \frac{f2^f}{\log n}$ makes the inequality hold . We thus set $\alpha = \frac{f2^f}{\log n}$ [8]. Note that $\alpha = o(2^f)$ which is necessary to improve on the result of Longpré.

Now, consider the programs $p(\#^T n, i)$ from section 5 implementing the one-try strategy from section 6.2, with $f = o(\log(n))$ and $\alpha = \frac{f2^f}{\log n}$. We recall that i ranges in $[1; 2^f/\alpha]$ and that $\#^T n$ is the smallest program outputting n in $T(n)$ steps. The infinite sequence of $(n_j)_{j\in\mathbb{N}}$ we consider are all the values of n such that $K^{T(n)}(n) < \log\log n)$ in ascending order. This is indeed an infinite sequence by Theorem 3. We now claim that:

1. $\forall j \in N$, $p(\#^T n_j, i)$ is of size $\leq f(n_j)$;
2. with a proper implementation of the strategy of section 4.5.2,$p(\#^T n_j, i)$ runs in $c\alpha T(n_j)$ for all $j \in \mathbb{N}$;
3. for all $j \in \mathbb{N}$, there exists $i \in [2^{f(n_j)}/\alpha(n_j)]$, such that no program of size $f(n_j)$ running for $T(n_j)$ steps outputs a string equal to the output of $p(\#^T n_j, i)$; moreover the output of $p(\#^T n_j, i)$ is of size n_j.

Once those 3 points are established, the theorem will be proven.

[8] Actually in all generality we should just take $\alpha = o(2^f)$, because when f grows too slowly for instance $f = \log\log n$, α as defined here takes non sensical value. The "issue" is that that we have to take a specific function α in order to perform size and running time analysis, so we picked one which worked for "most"$f = o(log(n))$. For slower growing f we could just adjust the α.

Size of the program. We first check that our program is of size less than $f(n_i)$ for all n_i.

It has the same structure as the program described in section 5 and the strategy of section 6.2 may be described in c bits. Therefore the size of p is

$$K^T(n) + f - \log(\alpha) + c = K^T(n) + log(f) - \log\log n + c,$$

which is less than $f(n)$ for infinitely many values of n.

Running time of the program. We check that our program runs in time less than $c\alpha T(n)$ for all n_i. In particular this involves checking that one can implement the strategy of section 6.2 in time $\mathrm{O}(\alpha T)$.

- Retrieving n_j from $\#^T n_j$ is done in $T(n_j)$ for all n_j by definition.
- computing $f(n_j)$, $T(n_j)$ is done in $cT(n_j)$ steps by supposition. One can check that it is also possible to compute $\alpha(n_j)$ in $cT(n_j)$ steps.
- Simulating the execution of the i-th chunk of size α of programs of size f i.e. getting the α inputs for our strategy takes a time αT as explained in section 5.
- Lastly, we describe an efficient implementation of the strategy described in 6.2. First we must convert every input in base $m = k^p = o(n)$. Notice that the strategy works if we take base $m \leftarrow 2^{\lceil \log m \rceil}$, and converting a binary number to a base 2^k can be done "for free".

  Computing our answer is then done in two steps. First, we compute all digits (in base m) of our answer except for the last one by looking at the digits of the inputs in base m. This takes time $O(\alpha n) = O(\alpha T)$ because $T > n$. Second, we compute the last digit of the output by keeping track of every digits of the α inputs in a table of size $m = o(n)$ (there are m possible values for the last digit). This takes time αn which is $o(\alpha T)$. Thus the whole program is computed in time less than $O(\alpha T)$.

  Correctness. We follow the same outline as in paragraph correctness of section 5. Call $e_{i,j}$ the output of $p(\#^T n_j, i)$. $e_{i,j}$ is an integer of $[N_j] = [2^{n_j}]$ by definition of program p , i.e. a bitstring of size n_j. We proved in section 6.2 that $\mathfrak{S}^{N_j}(\frac{2^f}{\alpha}, \alpha, 1)$ is winning, by construction this entails that there exists an $i \in [2^f/\alpha]$ such that $e_{i,j}$ is not the output of any program of size f running for T steps.

□

8 Additional details and musings for future work

We now sketch a promising way to reason about winning strategies for novelty games. This could be used either to construct new ones, or establish better lower bounds for $\mathfrak{B}(p,k,m)$. The idea is to study the smallest number of antecedents any element y in the image of $f : \binom{[N]}{k} \to [N]$ may have.

Notations 2. Given a sequence $(x_i) \in \mathbb{N}^k$, an element $x \in \mathbb{N}$ and some $j \in [1;k]$, we will denote by $(\hat{x}_i^j(x))$ the sequence such that $\hat{x}_i^j(x) = x$ if $i = j$ and $\hat{x}_i^j(x) = x_i$ otherwise.

Definition 8.1 (Antecedents)**.** Let f be a function, $\binom{[N]}{k} \to [N]$. We define $\mathcal{A}_f(y)$ as the set

$$\mathcal{A}_f(y) = \{x \mid \exists (x_i) \in [N]^k, \exists j \in [1;k], f(\hat{x}_1^j(x), \dots, \hat{x}_k^j(x))) = y\}.$$

The set $\mathcal{A}_f(y)$ contains all values of x such that there is a way to complete x with other numbers in order to produce y by the function f. Said differently: $\mathcal{A}_f(y) = \bigcup_{B \in f^{-1}(y)} B$.

We naturally extend the definition to apply to sets of number. Let $B \subset [N]$, then $\mathcal{A}_f(B) = \bigcup_{y \in B} \mathcal{A}_f(y)$. Thus we can write $\mathcal{A}_f^i(B) = \mathcal{A}_f^{i-1}(\mathcal{A}_f(B))$.

Definition 8.2 (Non self-hitting (NSH) functions)**.** Let f be a function $\binom{[N]}{k} \to [N]$. We say that f is non self-hitting if $\forall B \in \binom{[N]}{k}, f(B) \notin B$.

It is easy to prove that we need only consider strategies which are non self hitting.

Theorem 17. If there is a winning strategy for the game $\mathcal{G}(p,k,m)$, then there is a strategy with no self hitting functions winning for game $\mathcal{G}(p,k,m)$.

As a consequence, all considered functions from now on will be supposed to be non-self hitting.

Theorem 18 (Equivalence theorem for $(2,k)$)**.** Let f be an NSH function $\binom{[N]}{k} \to [N]$. Then there exists g such that (f,g) is a valid strategy for the game $\mathcal{G}(2,k,1)$ if and only if

$$\forall x_1, \dots, x_k \in [N]^k, \bigcup_{i=0}^{n} \mathcal{A}_f(x_i) \neq [N].$$

Proof. For the right to left implication, it is clear that if the property is satisfied then there exists a function g such that

$$\forall x_1, \dots, x_k, g(\{x_1, \dots, x_k\}) \in [N] \setminus \bigcup_{i=0}^{n} \mathcal{A}_f(x_i).$$

To prove the the converse implication, suppose there exists $x_1, \ldots, x_k$ such that $\bigcup_{i=0}^{n} \mathcal{A}_f(x_i) = [N]$, and call $a = g(\{x_1, \ldots, x_k\})$. We know a belongs to $\bigcup_{i=0}^{n} \mathcal{A}_f(x_i)$ and we can w.l.o.g. that it belongs to $\mathcal{A}_f(x_1)$. Then there exists a set B of size k such that $a \in B$ and $f(B) = x_1$. Therefore, the strategy fails on $(B, \{x_1, \ldots, x_k\})$, leading to a contradiction. □

Corollary 2. Let f be an NSH function $\binom{[N]}{k} \to [N]$. If

$$\forall x_1, \ldots, x_k \in [N]^k, \bigcup_{i=0}^{n} |\mathcal{A}_f(x_i)| < \frac{N}{k},$$

then there exists g such that (f, g) is a valid strategy for the game $\mathcal{G}(2, k, 1)$.

Following this line of thought, our idea to establish lower bounds for $\mathfrak{B}$ is to study a subclass of strategies on which it is easier to reason and then claim that any winning strategy can be reduced to a function of this subclass (in the case where we want to prove lower bounds).

We have just seen that only NSH functions need to be considered. If we restrict to oblivious strategies, the analysis should be easier. Lastly we can only consider strategies having many symmetries. The strategy we described for the game $\mathcal{G}(2, 2, 1)$ and for $\mathcal{G}(p, k, 1)$ are functions with lots of symmetries. One way to encapsulate this notion of symmetry was the notion of balanced functions, although it is clear that the notion may be too lenient. Indeed, the winning strategies we have found so far do much more than just being balanced. As such for future work we expect that an analysis of oblivious-NSH-balanced strategies could lead to progress, whether for establishing new strategies or lower bounds on $\mathfrak{B}$

Definition 8.3 (Balanced functions)**.** A function $f : \binom{[N]}{k} \to [N]$ is *balanced* if

$$\forall y \in [N], \left| |\mathcal{A}_f(y)| - \frac{1}{N}\binom{N}{k} \right| \leq 1.$$

A strategy is said to be balanced if the functions for players are all balanced functions.

Musings Here are some additional thoughts and results we expect to hold, but have not yet formally established.

- When f is not time-constructible in T, the result may still stand with adjustements. For instance, we can indicate the value $f(n)$ to p for a cost of $\log f(n)$ bits.
- We believe we can show that $\mathfrak{B}(2, k) \geq 3k + 1$.
- Our results hold under oracles.

- We believe theorem 2 can be formulated in the case of the game $\mathcal{G}(p,k,1)$ as follows. Let f be an NSH function $\binom{[N]}{k} \to [N]$. If

$$\forall x_1,\ldots,x_k \in [N]^k, \bigcup_{i=0}^{n} |\mathcal{A}_f(x_i)| < \frac{N}{k^p},$$

  then there are functions $g_1,\ldots,g_{p-1}$, such that $\mathfrak{S} = (g_1,\ldots,g_{p-1},f)$ is a winning strategy.

- We believe the best upper bound for $\mathfrak{B}(p,k,1)$ is of the order k^p.

- It should be methodically checked if the technique of parallel diagonalization with advice can lead to class separation in complexity theory

References

[1] Atif Abueida, James Lefevre, and Mary Waterhouse. Equitable edge colored steiner triple systems. *The Australasian Journal of Combinatorics [electronic only]*, 50, 06 2011.

[2] N. G. Bate. Complete graphs without polychromatic circuits. *Discret. Math.*, 46(1):1–8, 1983.

[3] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1261–1270. IEEE, 2023.

[4] 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, 2018.

[5] Shuichi Hirahara, Rahul Ilango, Zhenjian Lu, Mikito Nanashima, and Igor C. Oliveira. A duality between one-way functions and average-case symmetry of information. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1039–1050. ACM, 2023.

[6] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition.* Texts in Computer Science. Springer, 2019.

[7] Yanyi Liu and Rafael Pass. On one-way functions and kolmogorov complexity, 2020.

[8] Luc Longpré. *Resource Bounded Kolmogorov Complexity, A Link between Computational Complexity & Information Theory.* PhD thesis, Cornell University, USA, 1986.

[9] Zhenjian Lu and Igor C. Oliveira. Theory and applications of probabilistic kolmogorov complexity. *Bull. EATCS*, 137, 2022.