Kolmogorov complexity and games

Nikolay Vereshchagin[*]
Lomonosov Moscow State University

January 16, 2008

In memoriam Andrey Muchnik (24.02.1958 – 18.03.2007)

1 Muchnik's theorem

In this survey we consider some results on Kolmogorov complexity whose proofs are based on interesting games. The close relation between Recursion theory, whose part is Kolmogorov complexity, and Game theory was revealed by Andrey Muchnik. In [10], he associated with every statement ϕ of Recursion theory a game G_ϕ that has the following properties. First, G_ϕ is a game with complete information between two players, Muchnik called them *Nature* and *Mathematician*; players make in turn infinitely many moves, every move is 0 or 1. Second, if Mathematician has a computable winning strategy in the game G_ϕ then ϕ is true. Third, if Nature has a computable winning strategy then ϕ is false.

For all natural statements ϕ the winning condition in the game G_ϕ is defined by a Borel set (Mathematician wins if the sequence of moves made by the players belongs to a certain Borel subset of $\{0,1\}^{\mathbb{N}}$); so we assume that this is the case. By Martin's theorem [9] this assumption implies that G_ϕ is a determined game, that is, either Nature or Mathematician has a winning strategy. It might happen however that neither player has a computable winning strategy. In this case we cannot say anything about validity of ϕ.

Muchnik's general theorem applies to all statements on Kolmogorov complexity (with no time or space restrictions on description modes). Thus we can use Muchnik's theorem to find out whether a statement ϕ on Kolmogorov complexity we are interested in is true or false. To this end, given ϕ we first find the game G_ϕ. Usually the game G_ϕ does not look very natural and

[*]Supported by grant 06-01-00122 from RFBR. Paper written while visiting CWI, Amsterdam.

easy-to-analyze. We thus find a more natural game that is equivalent to G_ϕ (sometimes it is just a reformulation of G_ϕ in different terms). Then we find out which of the players has a winning strategy in the game G_ϕ. It turns out that in all particular cases either we can find a computable winning strategy for one of the players in G_ϕ (and thus we know whether ϕ is true or false), or the game is so hard to analyze that we cannot find who wins G_ϕ at all.

Sometimes the resulting game is interesting in its own right. The goal of this paper is to present five examples of this kind. Each example consists of: (1) a statement ϕ on Kolmogorov complexity, (2) a game G used in the proof of ϕ, which is usually just a reformulation of Muchnik's game G_ϕ, (3) the analysis of the game G, and (4) a proof that the existence of a computable winning strategy in G implies that ϕ is true. The last item is usually straightforward and we sometimes omit it.

In item (4) we could use also Muchnik's general theorem. We do not do that, as usually the direct proof it easier than the proof that the considered game G is equivalent to Muchnik's game G_ϕ. Muchnik's general theorem is only a source inspiration for this paper. Therefore we do not present its exact formulation.

In the example of Section 3 the game G is very easy to analyze; the main part of the proof of ϕ is thus finding an easy form of Muchnik's game. In Section 4, both finding G and its analysis are not easy. In the examples of Sections 5 and 7 the resulting games are so hard that we omit their full analysis (for some of the games from Section 7 we even do not know who wins the game).

The next section is an introductory one. We define there plain Kolmogorov complexity and present its properties needed for the following sections. In Section 6 we define a priori probability and prefix Kolmogorov complexity, which are needed to understand the meaning of the result of Section 7. We prove in Section 6 Levin's theorem relating a priori probability and prefix Kolmogorov complexity. The proof is also based on an analysis of an interesting game.

The reader familiar with Kolmogorov complexity can skip Sections 2 and 6. For such readers we present here our notation introduced in those sections. We use $|x|$ to denote the length of a string x; we denote by $C(x), K(x), m(x)$ plain Kolmogorov complexity, prefix Kolmogorov complexity and a priori probability of x, respectively.

2 Kolmogorov complexity

Compressing information

Roughly speaking, Kolmogorov complexity means "compressed size". Programs like `zip`, `gzip`, etc., compress a given file into a presumably shorter one. The original file can then be restored by a "decompressing" program (sometimes both compression and decompression are performed by the same program).

A file that has a regular structure can be compressed significantly. Its compressed size is small compared to its length. On the other hand, a file without regularities hardly can be compressed, and its Kolmogorov complexity is close to its original size.

This explanation is very informal and contains several inaccuracies. Instead of files (sequences of bytes) we will consider binary strings, that is, finite sequences of zeros and ones. Here are the more essential points: (1) We consider only decompressing programs; we do not worry at all about compression. More specifically, a *decompressor* is any algorithm that receives a binary string as an input and returns a binary string as an output. If a decompressor D on input x terminates and returns string y, we write $D(x) = y$ and say that x is a *description* of y with respect to D. Decompressors are also called *description modes.* (2) A description mode is not required to be total. For some x, the computation $D(x)$ may never terminate. Also we do not put any constraints on the computation time of D: on some inputs the program D may halt only after an extremely long time.

Using the recursion theory terminology, we say that a description mode is a partial computable (=partial recursive) function from $\{0,1\}^*$ to $\{0,1\}^*$, where $\{0,1\}^*$ stands for the set of all binary strings. We associate with any algorithm D whose inputs and outputs are binary strings a function d computed by D; namely, $d(x)$ is defined for a string x if and only if D halts on x and $d(x)$ is the output of D on x. A partial function from $\{0,1\}^*$ to $\{0,1\}^*$ is called *computable* if it is associated with (=computed by) some algorithm D. Usually we use the same letter to denote the algorithm and the function it computes. So we write $D(x)$ instead of $d(x)$ unless it causes a confusion.

Assume that a description mode (a decompressor) D is fixed. For a string x consider all its descriptions, that is, all y such that $D(y)$ is defined and equals x. The length of the shortest string y among them is called the *Kolmogorov complexity* of x with respect to D:

$$C_D(x) = \min\{\, |y| : D(y) = x\}.$$

Here $|y|$ denotes the length of the string y; we use this notation throughout the paper. The subscript D indicates that the definition depends on the choice of the description mode D. The minimum of the empty set is defined as $+\infty$, thus $C_D(x)$ is infinite for all the strings x outside the range of the function D (they have no descriptions).

At first glance this definition seems to be meaningless, as for different D we obtain quite different notions, including ridiculous ones. For instance, if D is nowhere defined, then C_D is infinite everywhere. A more reasonable example: consider a decompressor D that just copies its input to output, that is, $D(x) = x$ for all x. In this case every string is its own description and $C_D(x) = |x|$.

For any given string x we can find a description mode D that is tailored to x and with respect to which x has small complexity. Indeed, let $D(\Lambda) = x$. This implies $C_D(x) = 0$. It may seem that the dependence of complexity on the choice of the decompressor makes impossible any general theory of complexity. However, it is not the case.

Optimal description modes

A description mode is better when descriptions are shorter. According to this, we say that a description mode (decompressor) D_1 is *not worse* than a description mode D_2 if

$$C_{D_1}(x) \leqslant C_{D_2}(x) + c$$

for some constant c and for all strings x.

Let us comment on the role of the constant c in this definition. We consider a change in the complexity bounded by a constant as "negligible". One could say that such a tolerance makes the complexity notion practically useless, as the constant c can be very large. However, nobody managed to get any reasonable theory that overcomes this difficulty and defines complexity with better precision.

The following theorem, which states that there exists an optimal decompressor, is a basis of the theory of Kolmogorov complexity.

Theorem 1 (Solomonoff [11] and Kolmogorov [5]). *There is a description mode D that is not worse than any other one: for every description mode D' there is a constant c such that*

$$C_D(x) \leqslant C_{D'}(x) + c$$

for every string x.

A description mode D having this property is called *optimal*.

Proof. Recall that a description mode by definition is a computable function. Every computable function has a program. We assume that programs are binary strings. Moreover, we assume that reading the program bits from left to right we can determine uniquely where it ends, that is, programs are "self-delimiting". Note that every programming language can be modified in such a way that programs are self-delimiting. For instance, we can double every bit of a given program (changing 0 to 00 and 1 to 11) and append the pattern 01 to its end.

Define now a new description mode D as follows:

$$D(py) = p(y)$$

where p is a program (in the chosen self-delimiting programming language) and y is any binary string. That is, the algorithm D scans the input string from the left to the right and extracts a program p from the input. (If the input does not start with a valid program, D does whatever it wants, say, goes into an infinite loop.) Then D applies the extracted program p to the rest of the input (y) and returns the obtained result. (So D is just an "universal algorithm", or "interpreter"; the only difference is that program and input are not separated and therefore we need to use self-delimiting programming language.)

Let us show that indeed D is not worse than any other description mode P. Let p be a program computing a function P and written in the chosen programming language. If y is a shortest description of the string x with respect to P then py is a description of x with respect to D (though not necessarily a shortest one). Therefore, compared to P, the shortest description is at most $|p|$ bits longer, and

$$C_D(x) \leqslant C_P(x) + |p|.$$

The constant $|p|$ depends only on the description mode P (and not on x). $\square$

The same idea is used in practice. A *self-extracting archive* is an executable file starting with a small program (a decompressor); the rest is considered as input to that program. The program is loaded into the memory and then it decompresses the rest of the file.

Note that in our construction optimal decompressor works very long on some inputs (some programs have large running time), and is undefined on some inputs.

Kolmogorov complexity

Fix an optimal description mode D and call $C_D(x)$ the (plain) *Kolmogorov complexity* of the string x. In the notation $C_D(x)$ we drop the subscript D and write just $C(x)$.

If we switch to another optimal description mode, the change in complexity is bounded by an additive constant: for every optimal description modes D_1 and D_2 there is a constant $c(D_1, D_2)$ such that

$$|C_{D_1}(x) - C_{D_2}(x)| \leqslant c(D_1, D_2)$$

for all x.

Could we then consider the Kolmogorov complexity of a particular string x without having in mind a specific optimal description mode used in the definition of $C(x)$? No, since by adjusting the optimal description mode we can make the complexity of x arbitrarily small or arbitrarily large. Similarly, the relation "string x is simpler than y", that is, $C(x) < C(y)$, has no meaning for two fixed strings x and y: by adjusting the optimal description mode we can make any of these two strings simpler than the other one.

One may wonder whether Kolmogorov complexity has any sense at all. Let us recall the construction of the optimal description mode used in the proof of the Solomonoff–Kolmogorov theorem. This construction uses some programming language, and two different choices of this language lead to two complexities that differ at most by a constant. This constant is in fact the length of the program that is written in one of these two languages and interprets the other one. If both languages are "natural", we can expect this constant to be not that huge, just several thousands or even several hundreds. Therefore if we speak about strings whose complexity is, say, about 10^5 (i.e., a text of a novel), or 10^6 (DNA sequence) then the choice of the programming language is not that important.

Nevertheless one should always remember that all statements about Kolmogorov complexity are inherently asymptotic: they involve infinite sequences of strings. This situation is typical also for computational complexity: usually upper and lower bounds for complexity of some computational problem are asymptotic bounds.

The number of strings of small complexity

Let n be an integer. Then there are less than 2^n strings x such that $C(x) < n$. Indeed, let D be the optimal description mode used in the definition of Kolmogorov complexity. Then only strings $D(y)$ for all y such that $|y| < n$

have complexity less than n. The number of such strings does not exceed the number of strings y such that $|y| < n$, i.e., the sum

$$1 + 2 + 4 + 8 + \ldots + 2^{n-1} = 2^n - 1$$

(there are 2^k strings for each length $k < n$).

Complexity and information

One can consider the Kolmogorov complexity of x as the *amount of information* in x (we do not try to distinguish between useful and meaningless information, a string has much information unless it has a short description).

If the complexity of a string x is equal to k, we say that x has k bits of information. One can expect that the amount of information in a string does not exceed its length, that is, $C(x) \leqslant |x|$. This is true (up to an additive constant, as we have already said): $C(x) \leqslant |x| + O(1)$. This inequality implies, in particular, that Kolmogorov complexity is always finite, that is, every string has a description.

Here is another property of "amount of information" that one can expect: the amount of information does not increase when algorithmic transformation is performed. (More precisely, the increase is bounded by an additive constant depending on the transformation algorithm.)

Theorem 2. *For every algorithm A there exists a constant c such that*

$$C(A(x)) \leqslant C(x) + c$$

for all x such that $A(x)$ is defined.

Proof. Let D be an optimal decompressor that is used in the definition of the Kolmogorov complexity. Consider another decompressor D':

$$D'(p) = A(D(p)).$$

(We apply first D and then A.) If p is a description of a string x with respect to D and $A(x)$ is defined, then p is a description of $A(x)$ with respect to D'. Let p be a shortest description of x with respect to D. Then we have

$$C_{D'}(A(x)) \leqslant |p| = C_D(x) = C(x).$$

By optimality we obtain

$$C(A(x)) \leqslant C_{D'}(A(x)) + c \leqslant C(x) + c$$

for some c and all x. □

This theorem implies that the amount of information "does not depend on the specific encoding". For instance, if we reverse all bits of some string (replace 0 by 1 and vice versa), or add a zero bit after each bit of that string, the resulting string has the same Kolmogorov complexity as the original one (up to an additive constant). Indeed, the transformation itself and its inverse can be performed by an algorithm.

So we can define Kolmogorov complexity for any finite objects, like natural numbers, pairs of binary strings, finite sets of binary strings, etc. To define Kolmogorov complexity for elements in a class L of finite objects, fix a computable bijection $\pi : L \to \{0,1\}^*$ and let $C(x) = C(\pi(x))$ for all $x \in L$. Theorem 2 guarantees that for any other computable bijection $\tau : L \to \{0,1\}^*$ we have $C(\pi(x)) = C(\tau(x)) + O(1)$ for all $x \in L$. Indeed, both functions $\pi(\tau^{-1}(p))$ and $\tau(\pi^{-1}(p))$ are computable. Thus Kolmogorov complexity is well defined (up to an additive constant) for objects in L.

A theorem, similar to Theorem 2, holds for algorithmic transformation of several objects into one object. Let us state it for algorithmic transformation of 2 strings into one string.

Theorem 3. *For every algorithm A that has two input strings and one output string there exists a constant c such that*

$$C(A(x,y)) \leqslant C(x) + C(y) + 2\log C(x) + c$$

for all x, y such that $A(x,y)$ is defined.

Proof. We can find $A(x,y)$ given a description of A (in constant number of bits) and minimal descriptions of x, y (in $C(x)$ and $C(y)$ bits, respectively). This informal observation yields a stronger bound $C(A(x,y)) \leqslant C(x) + C(y) + O(1)$, without the term $2\log C(x)$, which is actually wrong.

What is wrong with this argument? Let D be the optimal description mode that is used in the definition of Kolmogorov complexity. We try to define the following description mode D'. If $D(p) = x$ and $D(q) = y$ we consider pq as a description of xy, that is, we let $D'(pq) = A(x,y)$. The problem is that D' is not well defined, as D' has no means to separate p from q.

To fix this bug let us prepend the string pq by the length $|p|$ of string p (in binary notation). This allows us to separate p and q. However, we need to find where $|p|$ ends, so let us double all the bits in the binary representation of $|p|$ and then put 01 as separator. More specifically, let $bin(k)$ denote the binary representation of integer k and let $\overline{x}$ be the result of doubling each bit in x. (For example, $bin(5) = 101$, and $\overline{bin(5)} = 110011$.) Let

$$D'(\,\overline{bin(|p|)}\,01pq) = A(D(p), D(q)).$$

Thus D' is well defined: the algorithm D' scans $\overline{bin(|p|)}$ while all the digits are doubled. Once it sees 01, it determines $|p|$ and then scans $|p|$ digits to find p. The rest of the input is q and the algorithm is able to compute $A(D(p), D(q))$.

Now we see that $C_{D'}(xy)$ is at most $2|bin(|p|)| + 2 + |p| + |q|$. The length of the binary representation of $|p|$ is at most $\log_2 |p| + 1$. Therefore, xy has a description of length at most $2\log_2 |p| + 4 + |p| + |q|$ with respect to D'. By optimality,

$$C(A(x,y)) \leqslant C_{D'}(A(x,y)) + O(1) \leqslant C_D(x) + C_D(y) + 2\log C_D(x) + O(1). \quad \square$$

For algorithms receiving k input strings, we obtain the following inequality

$$C(A(x,\ldots,x_k)) \leqslant C(x_1) + \cdots + C(x_k) + 2\log C(x_1) + \cdots + 2\log C(x_{k-1}) + c.$$

Conditional complexity

When transmitting a file, one could try to save communication charges by compressing it. The transmission could be made even more effective if an old version of the same file already exists at the other side. In this case we need only to describe the changes made. This could be considered as a kind of motivation for the definition of conditional complexity of a given string x relative to (known) string y.

A *conditional decompressor* is any computable function D of two arguments (both arguments and the value of D are binary strings). If $D(y,z) = x$ we say that y is a (conditional) *description of x when z is known* (or *relative to z*) The complexity $C_D(x|z)$ is then defined as the length of the shortest conditional description:

$$C_D(x|z) = \min\{|y| : D(y,z) = x\}.$$

We say that (conditional) decompressor D_1 is *not worse* than D_2 if

$$C_{D_1}(x|z) \leqslant C_{D_2}(x|z) + c$$

for some constant c and for all x and z. A conditional decompressor is *optimal* if it is not worse than any other conditional decompressor.

Theorem 4. *There exist optimal conditional decompressors.*

This "conditional" version of Kolmogorov–Solomonoff theorem can be proved in the same way as the unconditional one (Theorem 1, p. 4). Again, we fix some optimal conditional decompressor D and omit index D in the notation.

3 Information distance between two strings and on-line edge coloring of a graph

Information distance and conditional complexity

The “information distance” between strings x and y is the minimal length of a binary string p such that that x can be effectively found from y and p, and conversely, y can be effectively found from x and p.

To define this notion rigorously, we act as we did when we defined Kolmogorov complexity. For any pair A, B of algorithms (which are thought as procedures to find x form p, y and y from x, p, respectively) define

$$C_{AB}(x \leftrightarrow y) = \min\{|p| : A(p, y) = x,\ B(p, x) = y\}.$$

Similar to Solomonoff–Kolmogorov theorem, we can show that there are optimal algorithms A, B, for which C_{AB} is minimal up to an additive constant. Fixing optimal algorithms A, B we obtain the definition of *information distance $C_{AB}(x \leftrightarrow y)$ between x and y.* It is defined, like Kolmogorov complexity, up to an additive constant.

The trivial lower bound for $C(x \leftrightarrow y)$ is $\max\{C(x|y), C(y|x)\}$ (up to a constant error term). Indeed, if $A(p, y) = x$ then given p and y we can find x thus $C(y|x) \leqslant |p| + O(1)$. Similarly, $C(x|y) \leqslant |p| + O(1)$ if $B(p, x) = y$. The constants in $O(1)$ notation depend on A, B.

It turns out that $C(x \leftrightarrow y)$ is close to this lower bound for all x, y, which was proved in [1].

Theorem 5. *For all x, y we have*

$$C(x \leftrightarrow y) = \max\{C(x|y), C(y|x)\} + O(\log \max\{C(x|y), C(y|x)\}).$$

The edge coloring game

The proof of Theorem 5 is based on the following edge coloring game. Let G be an undirected graph. An edge coloring of G is a mapping assigning an integer number to each edge of G. An edge coloring is *proper* if every two adjacent edges (edges sharing a common vertex) have different colors. We want to find a proper edge coloring of a given graph G using minimum number of colors. Let $d(G)$ denote the degree of G (the maximal degree of a node in G). Clearly, we need at least $d(G)$ colors, as all edges incident to any node must have different colors. By Vizing’s theorem (see [3, 16]) for every graph G there is a proper edge coloring using $d(G) + 1$ colors.

We are interested, however, in how many colors we need if the graph is not given to the coloring algorithm in advance. That is, assume that the edges of an unknown graph appear one by one and we need to color every edge immediately after it appears.

More specifically, let d, l be integer parameters. Consider the following game. Players, call them Alice and Bob, make infinitely many moves in turn. On her moves Alice generates edges of a graph (one edge per move) so that after each of her moves the degree of the generated graph is at most d (otherwise she looses immediately). On his moves Bob colors the edge generated by Alice on the preceding move using one of available l colors. Bob wins if the resulting coloring of the graph is proper (that is, every two adjacent edges have different colors).

An analysis of the game

It is not hard to see that, in contrast to Vizing's theorem, $d+1$ colors are not enough for Bob to win (for all $d > 2$). The next lemma shows that $2d-1$ colors suffice.

Lemma 1. *If $l = 2d-1$ then Bob has a computable winning strategy in the game.*

Proof. Apply the greedy strategy: color each edge e in the first color that is different from all colors used so far for edges adjacent to e. The edge e has 2 ends, and each of them has at most $d-1$ incident edges different from e. Thus there are at most $2(d-1)$ edges adjacent to e and hence we do not need more than $2(d-1)+1$ colors. □

Proof of Theorem 5

How do we use the lemma? Let $k = \max\{C(x|y), C(y|x)\}$. Consider the graph G_k, where nodes are binary strings, and nodes x' and y' are connected by an edge if $K(x'|y') \leqslant k$ and $K(y'|x') \leqslant k$. Clearly the degree of this graph is less than $d = 2^{k+1}$. Let Alice generate edges of this graph (enumerating the set of pairs x', y' with $K(x'|y') \leqslant k$): once she has found a new pair such that $K(x'|y') \leqslant k$ and $K(y'|x') \leqslant k$, she makes the next move (x', y'). Apply Bob's computable strategy against Alice's strategy. We obtain an algorithm to color properly the graph G_k using $2d-1 < 2^{k+2}$ colors. Call this algorithm T, it receives k as input and it prints on its output tape all the pairs $\langle$an edge of G_k, its color$\rangle$.

We will now design particular algorithms A, B such that

$$C_{AB}(x \leftrightarrow y) \leqslant \max\{C(x|y), C(y|x)\},$$

up to a logarithmic error term (actually A will coincide with B).

Let $\hat{k}$ is a self-delimiting description of k, say, $\hat{k} = \overline{bin(k)}01$ where $\overline{p}$ stands for the string p with all bits doubled, and $bin(k)$ is the binary notation of k. Consider the following algorithm A: on inputs $\hat{k}i$ and y' it runs $T(k)$ and outputs x' once $T(k)$ prints the pair $\langle(x', y'), i\rangle$. As both $C(x|y)$ and $C(y|x)$ do not exceed k, the edge (x, y) belongs to G_k and thus $A(\hat{k}i, x) = y$ and $A(\hat{k}i, y) = x$ for some i. Hence

$$C_{AA}(x \leftrightarrow y) \leqslant |\hat{k}i| = k + O(\log k).$$

By optimality, we have

$$C(x \leftrightarrow y) \leqslant C_{AA}(x \leftrightarrow y) + O(1) \leqslant k + O(\log k)$$

as well.

4 Producer vs. Consumer game

The number of shortest descriptions

How many minimal length descriptions can have a string x (with respect to an optimal description mode D)? The number of such descriptions is bounded by a constant.

Indeed, assume that there are 2^m length-k descriptions of x. The number of y's that have at least 2^m descriptions of length k is at most 2^{k-m}. Given m and k we can enumerate all such y's. Hence every such y can be found from m and its $(k-m)$ bit index in that enumeration (k can be retrieved from $k-m$ and m). Thus if x has 2^m descriptions of length k then its complexity is bounded by

$$C(x) \leqslant k - m + O(\log m).$$

For $k = C(x)$ this implies that m is bounded by a constant.

Is an analogous statement true for approximate descriptions? We call p an approximate description of x if the number of positions where x and $D(p)$ differ (that is, Hamming distance between x and $D(p)$ is at most d) is small. Here D is an optimal description mode in the definition of $C(x)$.

More formally, let d be an integer. Let x be a binary string of length n. Let $C_d(x)$ denote the minimal Kolmogorov complexity of a string of length n such that the Hamming distance between x and y is at most d. Is it true that the number of y's of complexity $C_d(x)$ at Hamming distance d (or less) from x is small? More specifically, is it bounded by a polynomial in $|x|$? More generally, is it true that $C_d(x) \leqslant k - m + O(\log n + \log m)$ whenever there are at least 2^m strings y of complexity at most k at Hamming distance d from x? If the second question answers in positive then so does the first one (substitute $C_d(x)$ for k). We will show that this is indeed the case.

To see why this is not straightforward, let us try first the arguments used in the case of precise (normal) descriptions. The number of x's that have at least 2^m different y's of complexity k at Hamming distance d or less from x is bounded by

$$2^{k-m}B(n,d),$$

where $B(n,d)$ stands for the cardinality of a Hamming ball of radius d in $\{0,1\}^n$. Such x's can be enumerated given n, m, k, d and thus the complexity of each such x is at most

$$k - m + \log B(n,d) + O(\log n + \log m).$$

We need to prove that for a smaller quantity $C_d(x)$ we have a better bound

$$C_d(x) \leqslant k - m + O(\log n + \log m).$$

To this end we have to cover all enumerated x's by $2^{k-m+O(\log n+\log m)}$ balls of radius d. Moreover, we need to produce such covering on-line. We need to cover every enumerated x immediately, we cannot wait until all such x's have been enumerated, as we do not know when it happens.

As we said above, the answer to the question is positive: if x belongs to at least 2^m different Hamming balls of radius d and complexity k then x belongs to a Hamming ball of radius d and complexity $k - m + O(\log n)$ (up to an additive error term $O(\log n)$, the complexity of a Hamming ball coincides with complexity of its center, thus this is an equivalent formulation of the question). This is true not only for Hamming balls but for any family of subsets of $\{0,1\}^n$ of low Kolmogorov complexity.

Let $\mathcal{S}$ be an arbitrary family of subsets of $\{0,1\}^n$ and k, m natural numbers. Let $C(\mathcal{S})$ denote the complexity of $\mathcal{S}$ regarded as a finite object. For example, if $\mathcal{S}$ consist of all Hamming balls of radius d then $C(\mathcal{S}) = O(\log n + \log d) = O(\log n)$.

Theorem 6 ([14]). *If a string x belongs to at least 2^m sets in $\mathcal{S}$ of complexity at most k then x belongs to a set in $\mathcal{S}$ of complexity at most*

$$k - m + O(\log n + \log k + \log m + C(\mathcal{S})).$$

For instance, in the case of Hamming balls Theorem says: if x belongs to 2^m balls of radius d and complexity at most k then x belongs to a ball of radius d and complexity at most $k - m + O(\log n)$. (Note that the total number of balls of fixed radius is 2^n hence $m \leqslant n$ and the term $O(\log m)$ is absorbed by $O(\log n)$, as well as terms $O(\log k)$ and $O(C(\mathcal{S}))$.)

The reader may wonder what happens if Kolmogorov complexity of all sets in $\mathcal{S}$ is close to k. In this case the statement of the theorem just yields an upper bound $C(\mathcal{S}) + O(\log n + \log k)$ for m. Note that in this case $C(\mathcal{S})$ is at least $k - O(1)$ and thus this bound is not very interesting.

To prove the theorem let us try the following argument. Let $P_1, \dots, P_N$ where $N < 2^{k+1}$ be all sets in $\mathcal{S}$ of complexity at most k. Let T consist of all $x \in \{0,1\}^n$ of multiplicity $\geqslant 2^m$ in the sets $P_1, \dots, P_N$. Let us show first that it is possible to cover T by at most $N2^{-m}\mathit{poly}(n,k,m)$ sets of $P_1, \dots, P_N$ (here, $\mathit{poly}(k,m,n)$ denotes a polynomial).

Consider the bipartite graph whose left nodes are elements of T and right nodes are sets $P_1, \dots, P_N$. Draw an edge between x and P_i if $x \in P_i$. The degree of each left node is at least 2^m hence the number of edges in the graph is at least $2^m|T|$. Thus there is a right node of degree at least $2^m|T|/N$. In other words, there is a set P_u that covers at least $2^m|T|/N$ element in T. Include in the covering any such set P_u. Then apply the same argument to the set $T \setminus P_u$ in place of T and include in the covering a set P_v that covers at least $2^m/N$ fraction of elements in $T \setminus P_u$, and so on. Each time the fraction of non-covered $x \in T$ decreases by a factor of $1 - 2^m/N$. After including $2^{-m}Nn \ln 2$ sets, the number of non-covered strings in T is at most

$$|T|(1 - 2^m/N)^{2^{-m}Nn\ln 2} < |T|e^{-n\ln 2} = |T|2^{-n} \leqslant 1.$$

That is, all $x \in T$ are covered. (Alternatively, we could show that $2^{-m}Nn \ln 2$ randomly chosen sets cover T with positive probability. We prefer the described greedy covering algorithm, as it is more constructive.)

Why this argument does not suffice? The problem is that we are not given the set $\{P_1, \dots, P_N\}$. This set might have a huge Kolmogorov complexity. We can only assume that a procedure to enumerate $P_1, \dots, P_N$ is available, such procedure can be described in $O(\log k + C(\mathcal{S}))$ bits. More specifically, there is a non-halting algorithm with inputs $k, C(\mathcal{S})$ that prints out $P_1, \dots, P_N$ is some order. More formally, the algorithm prints the codes of sets $P_1, \dots, P_N$ with respect to a fixed encoding of finite subsets of $\{0,1\}^*$ by binary strings. And we do not know at which moment the last set is printed.

The game

Thus we naturally come to the following game of two players, we call them Producer (P) and Consumer (C). Let m, n, k, L be natural parameters. The game consists of alternating moves by the players, each making 2^k moves, starting with P's move. A move of P consists in producing a subset of $\{0,1\}^n$. A move of C consists in marking some sets previously produced by P (the number of marked sets on any move can be 0). The total number of marked sets must not exceed L. C wins if, following every one of its moves, every $x \in \{0,1\}^n$ that is covered at least 2^m times by P's sets[1] belongs to a marked set. It is important that this condition is checked after every Consumer's move. Consumer cannot wait until all 2^k sets appear.

An analysis of the game

Consumer can easily win if $L = 2^k$: she marks every set produced by P. On the other extreme, Producer wins if $L < 2^{k-m}$ and n is large enough (specifically, we need $2^n \geqslant (m+1)2^{k-m}$): he generates 2^m groups of sets, each group has 2^{k-m} sets and all sets within a group have a common element and sets from different groups are disjoint[2]. What happens for L between 2^{k-m} and 2^k? It turns out that for most such L Consumer wins.

Lemma 2. *Consumer has a winning strategy in the game provided*

$$L = 2^{k-m} poly(k, m, n).$$

There are two proofs of Lemma 2: a constructive one and a non-constructive one (the latter belongs to An. Muchnik). Given the above analysis of the "off-line" version of the game, is is easier to understand the constructive proof.

The constructive proof of the lemma. Consumer simultaneously uses k strategies denoted by $j = 1, 2, \dots, k$. Strategy j works as follows. Divide the sequence of Producer's sets into 2^{k-j} segments of 2^j sets each. After receiving

[1]This means that x belongs to at least 2^m *different* Producer's sets.

[2]To run this strategy, we need enough strings of length n. That's why n should be large. The union of sets of one group must have $m+1$ elements—one shared element and m other elements to obtain 2^m different sets.

each segment $Q_1, \dots, Q_{2^j}$, consider the set T consisting of all $x \in \{0,1\}^n$ of multiplicity $\geqslant 2^m/k$ in the sets $Q_1, \dots, Q_{2^j}$. Using the greedy algorithm described above mark $2^{j-m}kn \ln 2$ sets among $Q_1, \dots, Q_{2^j}$ so as to cover the set T by marked sets.

Since there are 2^{k-j} segments (for fixed j), the total number of marked sets C needs to use is $2^{j-m}2^{k-j}kn \ln 2 = 2^{k-m}kn \ln 2$ (for fixed j). Summing over all j, this comes to $2^{k-m}k^2 n \ln 2$ marked sets.

We claim that after every move $t = 1, \dots, 2^k$ of C, each $x \in \{0,1\}^n$ of multiplicity 2^m belongs to a marked set. Assume to the contrary, that there is an x that has multiplicity 2^m following step t of C, and x belongs to no set marked on step t or earlier. Let $t = 2^{j_1} + 2^{j_2} + \dots$ where $j_1 > j_2 > \dots$ be the binary expansion of t. The element x has multiplicity less than $2^m/k$ in the first segment of 2^{j_1} sets of P, multiplicity of less than $2^m/k$ in the next segment of 2^{j_2} sets, and so on. Thus its total multiplicity among t first sets is less than $k2^m/k = 2^m$. The contradiction proves the claim. $\square$

The non-constructive proof of the lemma. The non-constructiveness of the proof is two-fold. (1) Instead of proving that C has a winning strategy, we will prove that P has no winning strategy (by König's theorem one of the players must have a winning strategy); (2) To prove that P has no winning strategy we will design a randomized strategy for C that beats every fixed P's strategy with positive probability.

The randomized C's strategy is very simple: mark each set of P with probability $p = 2^{-m}(n+1)\ln 2$. Fix a strategy S of P. It suffices to prove that (1) with probability more than $\frac{1}{2}$, following each move of Consumer, every element of multiplicity 2^m or more is covered by marked sets; and (2) with probability more than $\frac{1}{2}$, Consumer marks at most $2^{k-m+1}(n+1)\ln 2$ sets.

To prove (2) note that the expected number of marked sets is $p2^k$. Thus by Markov's inequality, the probability that it exceeds $p2^{k+1}$ is less than $\frac{1}{2}$.

To prove (1) fix $x \in \{0,1\}^n$ and estimate the probability that there is move of C following which x is covered 2^m times by sets of P but belongs to no marked set. We need to show that this happens with probability less than 2^{-n-1}. To this end denote by R_i the event "following a move of C, string x is covered at least i times by sets of P but none of them is marked". Let us prove by induction that $\mathrm{Prob}[R_i] \leqslant (1-p)^i$. For $i = 0$ the statement is trivial. To prove the induction step we need to show that $\mathrm{Prob}[R_{i+1}|R_i] \leqslant 1 - p$. Let $z = z_1, z_2, \dots, z_s$ be a sequence of decisions by C: $z_j = 1$ if C marks the jth set of P and $z_j = 0$ otherwise. Call z *bad* if following C's sth move it happens for the first time that x belongs to i sets of P but none of them is marked. Then R_i is the disjoint union of the events "C has made the decisions z" (denoted by Q_z) over all bad z. Thus it is enough to prove that $\text{Prob}[R_{i+1}|Q_z] \leqslant 1-p$. Given that C has made the decisions z, the event R_{i+1} means that after those decisions the strategy S will ever produce $i+1$st set including x but C will not mark it. C's decision not mark that set does not depend on all previous decisions and is made with probability $1-p$. Hence

$$\text{Prob}[R_{i+1}|Q_z] = \text{Prob}[\text{P produces } i+1\text{st set including } x|Q_z]\cdot(1-p) \leqslant 1-p.$$

The induction step is proved. By the choice of p we have $\text{Prob}[R_{2^m}] \leqslant (1-p)^{2^m} < e^{-p2^m} = 2^{-n-1}$. By union bound the probability that some x of length n belongs to 2^m Producer's set but does not belong to a marked set is at most 1/2. □

Proof of Theorem 6

The strategy of Lemma 2 can be found by the brute force search given n, k and m and $\mathcal{S}$. Enumerate the sets in $\mathcal{S}$ of complexity at most k, and consider appearing sets as the moves of the Producer. Use the strategy of Lemma 2 against it. We will mark at most $2^{k-m}\mathit{poly}(m,k,n)$ of the generated sets that cover all the strings of multiplicity 2^m, i.e., that are covered 2^m times by the generated sets. We do not know when the last generated set appears, but the winning rule ensures that following our next move, all the strings of multiplicity 2^m will be covered. The complexity of each marked set is bounded by the logarithm $k-m+O(\log k+\log n+\log m)$ of the number of marked sets plus the amount of information needed to run the whole process. The latter is $O(\log k+\log m+\log n)$.

5 Naming branches in a growing tree

Kolmogorov complexity of computable infinite 0-1-sequences

Let ω be an infinite sequence of zeros and ones. The sequence ω is called computable if there is an algorithm that on input n finds nth bit of ω. (Equivalently, there is an algorithm that on input n computes the length-n prefix of ω.)

Let ω_n denote the length-n prefix of ω. There is the following criterion of computability of ω in terms of Kolmogorov complexity:

$$\omega \text{ is computable } \Leftrightarrow C(\omega_n|n) = O(1). \tag{1}$$

This result is attributed in [8] to A.R. Meyer (see also [17, 7]). Our goal is to investigate quantitative versions of Meyer's criterion.

The following two natural complexity measure are related to the left hand side of (1). Fix a programming language and let $U(p, n)$ denote the output of the program p on input n (here p is a binary string and n a natural number). Define

$$C_U(\omega) = \min\{|p| : U(p, n) = \omega_n \text{ for all } n\},$$

and

$$\tilde{C}_U(\omega) = \min\{|p| : U(p, n) = \omega_n \text{ for all but finitely many } n\},$$

Clearly, Solomonoff-Kolmogorov theorem holds both for complexities $C_U(\omega), \tilde{C}_U$ as well. Fixing an optimal programming language we obtain complexity measures $C(\omega), \tilde{C}(\omega)$ of an infinite binary sequence ω. Both $C(\omega), \tilde{C}(\omega)$ are finite if and only if ω is computable. These two complexities correspond to the left hand side of the equivalence (1).

The right hand side of (1) corresponds to the following two complexity measures:

$$M(\omega) = \max_n C(\omega_n|n) \text{ and } \tilde{M}(\omega) = \limsup_n C(\omega_n|n).$$

Both these quantities are finite if and only if $C(\omega_n|n)$ is bounded by a constant.

Obviously, $M(\omega) \leqslant C(\omega) + O(1)$ and $\tilde{M}(\omega) \leqslant M(\omega)$. In terms of the introduced complexity measures, Meyer's criterion reads: for every $f \in \{C, \tilde{C}\}$ and every $g \in \{M, \tilde{M}\}$,

$$f(\omega) < \infty \Leftrightarrow g(\omega) < \infty.$$

The question is: how we can upper bound $f(\omega)$ in terms of $g(\omega)$ and vice versa?

It is a straightforward from the definition that $M(\omega) < C(\omega) + O(1)$ and $\tilde{M}(\omega) < \tilde{C}(\omega) + O(1)$, which is a quantitative version of the easy part of Meyer's criterion. What about the converse inequalities? It is not very surprising and not hard to prove that $C(\omega)$ is not bounded by any computable function of $M(\omega)$.[3] However, for $\tilde{C}(\omega), \tilde{M}(\omega)$ the situation is quite different. We have $\tilde{M}(\omega) \leqslant 2\tilde{C}(\omega) + O(1)$ and this inequality is tight.

[3]One can show also that $M(\omega)$ (and hence $C(\omega)$) is not bounded by any computable function of $\tilde{C}(\omega)$.

Theorem 7 ([2]). *For all ω we have $\tilde{C}(\omega) < 2\tilde{M}(\omega) + O(1)$. On the other hand, for every m there exists a sequence ω such that $\tilde{M}(\omega) \leqslant m + O(1)$ and $\tilde{C}(\omega) \geqslant 2m$.*

The game

The game corresponding to this theorem is as follows. Let k, l be integer parameters. On her moves Alice enumerates a set of strings T that must have at most k strings of each length. More specifically, in his turn Alice may include a finite number of binary strings in the set T, which is empty at the start of the game. Every string included in T cannot be removed from T on later steps. On his moves, Bob defines l functions $h_1 \dots, h_l$ from $\mathbb{N}$ to $\{0,1\}^*$; in his turn Bob can define each of $h_1 \dots, h_l$ on any finite set of arguments. Once $h_i(n)$ is defined, it cannot be changed on later steps. The game lasts infinitely long. After having done infinitely many moves Alice has defined a set T and Bob has defined partial functions $h_1, \dots, h_l : \mathbb{N} \to \{0,1\}^*$.

Bob wins if the following holds. Call an infinite binary sequence ω a *branch* of T if almost all prefixes of ω are in T. Bob wins if for every branch ω of T there is i such that $h_i(n)$ is defined and equal to ω_n for almost all n. (We call such i a name of ω.)

Call this game the k, l-game. For each k, l the winning rule in k, l-game is defined by a Borel set. By Martin's theorem [9] every Borel game is deterministic. In particular, for each k, l either Alice, or Bob has a winning strategy in k, l-game.

An analysis of the game

As T has at most k strings of each length, T has at most k branches. (Indeed, if T had $k+1$ branches, for large enough n the length-n prefixes of those branches would be pair-wise different strings of length n in T.) Alice can easily construct a set T having exactly k branches. As different branches must have different names, Alice wins if $l < k$. On the other hand, it is even not evident that for every k there is l such Bob wins in k, l-game.

It turns out that the border line separating (k, l)-games won by Alice and Bob is close to the parabola $l = k^2$, which implies Theorem 7. More specifically, Theorem 7 follows more or less directly from the following lemmas.

Lemma 3. *For every k Bob has a computable winning strategy in the k, k^2-game (the winning algorithm has k as an input).*

Lemma 4. *Alice has a computable winning strategy in the $k, k^2/4$-game (the winning algorithm has k as an input). Moreover, Alice's winning strategy always constructs a tree (if x is a prefix of $y \in T$ then $x \in T$). This implies that we can require the inequality $M(\omega) \leqslant m + O(1)$ instead of $\tilde{M}(\omega) \leqslant m + O(1)$ in Theorem 7.*

We will present here Muchnik's proof of Lemma 3 and will omit the proof of Lemma 4, as it is too involved.

Proof of Lemma 3. Bob's functions will be indexed by pairs a, b, where a and b are natural numbers in the range $1, \ldots, k$. Let us explain how Bob defines $h_{ab}(n)$. Observing the growing set T, Bob looks for all pairs of strings u and v such that:

(a) the ordinal number of u in the lexicographic ordering of all (already appeared) strings of length $|u|$ in T is a;

(b) the ordinal number of v in the reverse lexicographic ordering[4] of all (already appeared) strings of length $|v|$ in T is b;

(c) u is a prefix of v.

After such a pair of strings is found, Bob sets $h_{ab}(n) = u_n$ (length-n prefix of u) for all $n \leqslant |u|$ such that $h_{ab}(n)$ has not been defined yet. Then Bob looks for another pair of strings u and v with the same properties, etc.

Let T be the set constructed by Alice after infinite number moves. We need to prove that this strategy guarantees that at the end of the game any infinite branch in T has a name. Let ω be an infinite branch, so $\omega_n \in T$ for all sufficiently large n. For these n let a_n denote the lexicographic number of ω_n in the set T_n of all strings of length n that are in T, and let b_n denote the reverse lexicographic number of ω_n in T_n. Let $a = \limsup a_n$ and $b = \limsup b_n$. We claim that for all sufficiently large n, $h_{ab}(n) = \omega_n$.

It is easy to see that $h_{ab}(i)$ is defined for all i. In other words, for all i there is a pair u, v satisfying the above conditions and such that $|u| \geqslant i$. According to the definition of a and b there are infinitely many n such that $a_n = a$ and infinitely many m such that $b_m = b$. Choose a pair of such n and m; assume that $i \leqslant n \leqslant m$. The strings $u = \omega_n$ and $v = \omega_m$ will be discovered after all strings of length n and m appear in T. These u, v qualify all the conditions listed above. (Note that this does not prove that

[4]We say that x is less that y w.r.t. the reverse lexicographic ordering if $y < x$ w.r.t. the normal lexicographic ordering.

$h_{ab}(i) = \omega_i$, as we might find a pair u, v that is different from the specified pair.)

It remains to prove that for all sufficiently large n we have $h_{ab}(n) = \omega_n$ provided $h_{ab}(n)$ is defined. Fix N such that $a_n \leqslant a$ and $b_n \leqslant b$ for all $n \geqslant N$. We will show that if both $|u|$ and $|v|$ are at least N and u, v satisfy the conditions listed above then u is a prefix of ω. Indeed, the inequality $a_{|u|} \leqslant a$ implies that $\omega_{|u|} \leqslant u$ w.r.t. the lexicographical ordering.[5] Similarly, the inequality $b_{|v|} \leqslant b$ implies that $v \leqslant \omega_{|v|}$ w.r.t. the same (normal lexicographical) ordering. The latter implies that $v_{|u|} \leqslant \omega_{|u|}$ and hence $v_{|u|} \leqslant \omega_{|u|} \leqslant u$. Therefore, the only chance for u to be a prefix of v is when both u and $v_{|u|}$ are prefixes of ω. $\square$

We omit the proof that Lemmas 3 and 4 imply Theorem 7. This proof is rather straightforward. For the first statement of the theorem we fix Alice's strategy in $2^{\tilde{M}(\omega)+1}, 2^{2(\tilde{M}(\omega)+1)}$-game (she includes in T all x with $C(x|n) \leqslant \tilde{M}(\omega)$, here n stands for the length of x) and use Bob's computable winning strategy of Lemma 3 against it. For the second statement of the theorem we fix Bob's strategy in $2^{m+1}, 2^{2m}$-game (he indexes his functions by strings of length less than $2m$ and lets $h_p(n) = U(p, n)$) and use Alice's computable winning strategy of Lemma 4 against it.

6 Prefix complexity and a priori measure

Randomized algorithms and lower semi-computable semimeasures

Let M be an algorithm (=machine) with one infinite input tape and one infinite output tape. At the start the input tape contains an infinite binary sequence ω called the input to M. The output tape is empty at the start. We say that the algorithm M on an input ω prints a natural number n if M halts after having printed n (say in binary notation) followed by a certain marker on the output tape. In this case we write $M(\omega) = n$.

Consider the uniform probability distribution on inputs to such algorithm. The algorithm then becomes a randomized (=probabilistic) algorithm without input whose outputs are natural numbers.

Speaking formally, the probability that such algorithm M prints a result n is defined as follows. Consider the uniform Bernoulli distribution on the space Ω of all infinite 0-1-sequences. The measure of the set Ω_u of all infinite

[5] Indeed, item (a) implies that there are at least a strings of length $|u|$ in the resulting set T (at the end of the game) that are less than or equal to u. Thus the string $\omega_{|u|}$ cannot be greater than u, as otherwise $a_{|u|} > a$.

continuations of a finite string u is equal to $2^{|u|}$. Consider the set $A = \{\omega \mid M(\omega) = n\}$. This set is the union of intervals Ω_p over all strings p such that M prints n after having scanned p on its input tape. The probability that M outputs n is equal to the measure of this set.

Consider an example: the algorithm reads the input sequence ω until it encounters a 1 and then outputs the number of scanned 0s and halts. The probability p_n of the event "the output is n" is equal to 2^{-n-1}.

We assign to every probabilistic machine (having no input and producing natural numbers) a sequence $p_0, p_1, \ldots$ of real numbers: p_n is the probability that the machine prints the number n. We say that the probabilistic machine *generates* the sequence $p_0, p_1, \ldots$.

Which sequences $p_0, p_1, \ldots$ can be obtained in this way? There is an obvious necessary condition: $\sum p_i \leqslant 1$ (since the machine cannot produce two different outputs). However, this inequality is not sufficient, as there are countably many randomized algorithms and uncountably many sequences satisfying this condition.

A sequence $p_0, p_1, p_2, \ldots$ is called *lower semi-computable* if there is a function $p(i, n)$, where i, n are integers and $p(i, n)$ is either a rational number or $-\infty$, with the following properties: the function $p(i, n)$ is non-decreasing in the second argument:

$$p(i, 0) \leqslant p(i, 1) \leqslant p(i, 2) \leqslant \ldots,$$

and

$$p_i = \lim_{n \to \infty} p(i, n)$$

for all i.

An equivalent definition: a sequence $p_0, p_1, p_2, \ldots$ is *lower semi-computable* if the set of all pairs $\langle r, i \rangle$, where i is an integer and r is a rational number less than p_i, is computably enumerable (a set is called computably enumerable if there is an algorithm without input that prints on its output tape all elements from the set in some order, and no other elements). The following lemma identifies the class of all generatable sequences (we omit its proof).

Lemma 5. *A sequence $p_0, p_1, p_2, \ldots$ is generated by a probabilistic algorithm if and only if it is lower semi-computable and $\sum p_n \leqslant 1$.*

Any sequence p_i satisfying the conditions of the previous lemma is called a *lower semi-computable semimeasure* (or *enumerable from below semimeasure*) on $\mathbb{N}$. We thus have two alternative definitions of a lower semi-computable semimeasure: (1) a probability distribution generated by a randomized algorithm; (2) a lower semi-computable sequence of non-negative reals whose sum does not exceed 1. [6]

We have considered so far (lower semi-computable) semimeasures on the natural numbers. The definition of a lower semi-computable semimeasure can be naturally generalized to the case of binary strings or any other constructive objects in place of natural numbers. For example, to define a notion of a lower semi-computable semimeasure on the set of binary strings we have to consider probabilistic machines whose output is a binary string.

The a priori probability

Comparing two semimeasures on $\mathbb{N}$ we will ignore multiplicative constants. A lower semi-computable semimeasure m is called *maximal* if for any other lower semi-computable semimeasure m' the inequality $m'(i) \leqslant cm(i)$ holds for some c and for all i.

Theorem 8. *There exists a maximal lower semi-computable semimeasure on* $\mathbb{N}$.

Proof. We have to construct a probabilistic machine M with the following property. The machine M should print every number i with a probability that is at most constant times less than the probability that any other machine M' prints i (the constant may depend on M' but not on i).

Let the machine M pick at random a probabilistic machine M' and then simulate M'. The probability to pick each machine M' should be positive. If a machine M' is chosen with probability p then M will print a number i with probability at least $p\cdot$(the probability that M' prints i). Thus one can let $c = 1/p$.

It remains to explain how to implement the random choice of a probabilistic machine. Enumerate all the probabilistic machines in a natural way; let $M_0, M_1, M_2, \dots$ be the resulting sequence. We toss a coin until the first 1 appears. Then we simulate the machine M_i where i is the number of zeros preceding the first 1. $\square$

Fix any maximal lower semi-computable semimeasure $p_0, p_1, p_2, \dots$ on natural numbers. We will use the notation $m(i)$ for p_i and the notation m

[6]The word "semimeasure" may look strange, but unfortunately there is no other appropriate term in the literature. Dropping semi-computability requirement, one can call any function $i \mapsto p_i$ with $\sum_i p_i \leqslant 1$ a *semimeasure* on $\mathbb{N}$. Every semimeasure on $\mathbb{N}$ defines a probability distribution on the set $\mathbb{N} \cup \{\perp\}$ where $\perp$ is a special symbol meaning "undefined". The probability of the number i is p_i and the probability of $\perp$ is $1 - \sum_i p_i$.

for the semimeasure itself. The value $m(i)$ is called the *a priori probability* of i.[7] Here is an explanation of this term. Assume that we are given a device (a black box) that after being turned on produces a natural number. For each i we want to get an upper bound for the probability that the black box outputs i. If the device is a probabilistic machine then *a priori* (without any other knowledge about the box) we can estimate the probability of i as $m(i)$. This estimate can be much more than the unknown true probability, but only $O(1)$ times less than it.

The a priori probability of a number i is closely related to its complexity. Roughly speaking, the less the complexity is, the larger the a priori probability is. More specifically, a slightly modified version of complexity (the so-called prefix complexity) of i is equal to the minus logarithm of $m(i)$.

Prefix Kolmogorov complexity

The difference between prefix complexity and plain complexity can be explained as follows. Defining prefix complexity, we consider only "self-delimiting descriptions". This means that the decoding machine does not know where the description ends and has to find this information itself. Let f be a function whose arguments and values are binary strings. We say that f is *prefix-stable*, if the following holds for all strings x, y:

> $f(x)$ is defined and x is a prefix of y $\Rightarrow$ $f(y)$ is defined and is equal to $f(x)$.

Theorem 9. *There exists an optimal prefix-stable decompressor (for the family of all prefix-stable decompressors).*

We omit the proof of this theorem. Let us fix some optimal prefix-stable decompressor D and let $K(x)$ denote $C_D(x)$. We call $K(x)$ the *prefix complexity* of x. As well as the plain complexity, the prefix complexity is defined up to an $O(1)$ additive term.

It is easy to see that $K(x)$ and $C(x)$ differ by a logarithmic additive term:

$$C(x) \leqslant K(x) + O(1) \leqslant C(x) + 2\log C(x) + O(1).$$

The first inequality here is straightforward as the class of all decompressors, used in the definition of $C(x)$, includes the class of all prefix-free decompressors, used in the definition of $K(x)$. To prove the second inequality it suffices to construct a prefix free decompressor D' such that $C_{D'}(x) \leqslant C(x) + 2\log C(x) + O(1)$.

[7]Another name for m is the *universal semimeasure* on $\mathbb{N}$.

Fix an optimal decompressor D used in the definition of the plain complexity. Let decompressor D' be defined on all strings of the form $\hat{n}pq$ where n is the length of p and $D(p)$ is defined, and q is arbitrary string.[8] For such inputs we let $D'(\hat{n}pq) = D(p)$. The function D' is well defined: if a string has two different representations $\hat{n}pq = \widehat{n'}p'q'$ where $n = |p|$ and $n' = |p'|$ then $n = n'$ and thus $pq = p'q'$. The latter implies that $p = p'$.

By construction D' is a prefix-stable function. As the length of $\hat{n}p$ is $|p| + 2\log n + O(1)$ we have $C_{D'}(x) \leqslant C_D(x) + 2\log C_D(x) + O(1)$.

Prefix complexity and a priori probability

Now we can state the relation between the a priori probability of a string x and its complexity:

Theorem 10. *$K(x) = -\log m(x) + O(1)$.*

We present a sketch of proof of this theorem. The statement of the theorem is a conjunction of two inequalities.

$$-\log m(x) \leqslant K(x) + O(1), \qquad K(x) \leqslant -\log m(x) + O(1).$$

The first of them follows directly from the fact that the function $2^{-K(x)}$ is a lower semi-computable semimeasure, which is easy to verify. The proof of the converse inequality, uses a certain game.

The semimeasure $m(x)$ is lower semi-computable, so we can generate lower bounds for $m(x)$ that converge to $m(x)$ but no estimates for the approximation error are given. The larger $m(x)$ is, the smaller $K(x)$ should be, that is, the shorter description p we have to provide for x. The descriptions reserved for different strings must be incompatible. (The descriptions p_1 and p_2 are incompatible if the intervals Ω_{p_1} and Ω_{p_2} are disjoint. Recall that the interval Ω_p consists of all infinite binary sequences beginning with p.) The inequality $|p| \leqslant -\log_2 m(x)$ means that the measure of the interval Ω_p is at least $m(x)$: $2^{-|p|} \geqslant m(x)$. Thus we have to assign to every string x an interval of measure at least $m(x)$ so that the intervals assigned to different strings do not overlap. Actually, it suffices to reserve an interval of the length $\varepsilon m(x)$ rather than $m(x)$, for some fixed positive ε. This relaxation causes the complexity increase at most by a constant.

[8] Notation $\hat{n}$ refers to the "self delimiting" description of a natural number n in $2\log n + O(1)$ bits.

The space allocation game

So we arrive to the following game between two players, called Client and Server. Fix a positive rational number ε. Client defines an infinite sequence $a(1), a(2), \ldots$ of non-negative reals. At the start of the game all $a(i)$ are zeroes. In her turn Client may increase any $a(i)$ by a rational value so that $\sum_i a(i) \leqslant 1$ (if after some Client's move this sum becomes greater than 1, she looses immediately).

In his turn Server defines a mapping $h : \{0,1\}^* \to \{1, 2, \ldots\}$. At the start of the game h is undefined on all arguments. On each move Server may define h on a new argument (previously defined values cannot be changed). If $h(p) = i$ then we say that Server has allocated Ω_p to ith job and the value $a(i)$ will be called space request for ith job.

If intervals Ω_p and Ω_q are allocated to different jobs then Ω_p and Ω_q must be disjoint (otherwise Server looses immediately after that has happened).

Players make in turn infinitely many moves. Server wins the game if (at the end of the game) for every i he has allocated to ith job an interval Ω_p of measure at least $\varepsilon a(i)$. Let us stress that Client is interested not in the total space allocated to every job, but in the largest interval, which makes "space management" job difficult.

Clearly, Client wins this "space allocation" game provided $\varepsilon > 1$. Moreover, Client wins if $\varepsilon > 1/2$. Let us show this, say, for $\varepsilon = 9/16$. Client requests $1/9 + \delta$ of space for each of the first 8 jobs, where δ a small number. Server has to allocate an interval of measure $\varepsilon(1/9 + \delta) > 1/16$ to each of them. So each of 8 jobs is allocated an interval of measure 1/8, which means the entire space Ω is exhausted. If δ is small enough, the sum of all requests is still less than 1 and Client requests a small amount of space for the ninth job, which Server is unable to allocate.

In this example Client has not used her right to increase requests many times. Using this option Client can win the game for $\varepsilon = 1/2$ and even for some $\varepsilon < 1/2$. However for $\varepsilon = 1/4$ the game is won by Server.

Lemma 6. *Server has a computable winning strategy in the described game for $\varepsilon = 1/4$.*

Proof. [9] The main idea is as follows: Server takes into account only those increases of requests when $a(i)$ becomes greater than a number of the form 2^{-j}. If on step t certain $a(i)$ becomes greater than 2^{-j} we allocate to client i a fresh interval (an interval that is disjoint with all previously allocated intervals) of measure $b_t = 2^{-j-1}$. (If on step t no $a(i)$ becomes greater than

[9]The proof is similar to the proof of the so called Kraft–Chaitin lemma, see [4].

a number of the form 2^{-j}, then we let $b_t = 0$ and do not allocate any space on that step.)

Before to describe the allocation strategy note that the total space we will allocate to job i does not exceed the sum

$$2^{-j-1} + 2^{-j-2} + 2^{-j-3} + \dots = 2^{-j} < a(i),$$

where j is the integer numbers with $2^{-j} < a(i) \leqslant 2^{-j+1}$. This shows that $\sum b_t \leqslant 1$ provided $\sum_i a(i) \leqslant 1$ (Server will not run out of space). In particular, at any moment of the game, the total measure $b_{t+1} + b_{t+2} + \dots$ we have to allocate on future steps does not exceed the total measure of unallocated space.

The allocation strategy is as follows: we maintain the representation of the free space (part of Ω that is not allocated) as the union of intervals of different measures. Initially this list contains one interval Ω. Assume that we need to allocate an interval of measure $w = 2^{-j-1}$ to a job. First note that one of the free intervals has measure at least w. Indeed, the total measure of free intervals is at least w provided $\sum_i a(i) \leqslant 1$. If all the free intervals had measures smaller than w, their total measure would be less than w since they have different measures and the sum of powers of 2 less than w is less than w.

If there is a free interval in the list that has measure exactly w, our task is simple. We just allocate this interval and delete it from the free list (maintaining the invariant relation).

Assume that this is not the case. Then we have some intervals in the list that are bigger than requested. Using the best fit strategy, we take the smallest among these intervals. Let $w' > w$ be its length. Then we split free interval of measure w' into intervals of measure $w, w, 2w, 4w, 8w, \dots, w'/2$ (note that $w + w + 2w + 4w + 8w + \dots + w'/2 = w'$. The first interval (of measure w) is allocated, all the other intervals are added to the free list. We have to check out the invariant relation: all new intervals in the list have different measures starting with w and up to $w'/2$; old free intervals cannot have this measure since w' was the best fit in the list.

Let us prove that Server wins. Let $a(i)$ denote the space request for ith job at the end of the game. Assume that $a(i)$ is positive and let j be the integer numbers with $2^{-j} < a(i) \leqslant 2^{-j+1}$. At some time in the game $a(i)$ becomes larger than 2^{-j} and Server then allocates an interval of measure $2^{-j-1} \geqslant a(i)/4$ to ith job. □

Let us finish the proof of Theorem 10. Let Client lower semi-compute the a priori probability $m(i)$ so that after any her move $a(i)$ is equal to the best lower bound of $m(i)$ discovered up to that time. Let Server use the computable strategy of Lemma. Then he defines a computable function $h(p)$ such that $C_h(i) \leqslant -\log m(i) + O(1)$. That function can be extended to a computable prefix stable decompressor D by letting $D(pq) = h(p)$ for all p, q. Thus we have $K(i) \leqslant C_D(i) + O(1) \leqslant -\log m(i) + O(1)$.

7 Solovay's theorem and "cats vs. ants" game

Prefix complexity and a priori probability of enumerable sets

Kolmogorov complexity (as well as prefix complexity and a priori probability) can be defined for algorithm problems of a very general type. Roughly speaking, Kolmogorov complexity of an algorithmic problem is the minimal length of a program that solves that problem, with respect to an optimal programming language. From this viewpoint the plain complexity $C(x)$ is the complexity of the algorithmic problem "print x".

In this section, we will focus on algorithmic problems of the type "enumerate the set A" (where A is a computably enumerable set of natural numbers). The following definitions are attributed by Solovay in [12] to G. Chaitin (wee keep Solovay's notations).

First we define prefix complexity $I(A)$ of the problem "enumerate A". Let M be an algorithm with one infinite input tape and one infinite output tape. At the start the input tape contains an infinite binary string ω called the input to M. The output tape is empty at the start. We say that a program p *enumerates* a set $A \subset \mathbb{N} = \{1, 2, \dots\}$ with respect to M, if in the run on every input ω extending p the algorithm M prints all the elements of A in some order and no other elements. We do not require M to halt in the case when A is finite.[10] Let $I_M(A)$ denote the minimal length of a program enumerating A.

There is an algorithm M_0 (called a *universal* algorithm) such that for every other algorithm M there is a constant c such that

$$I_{M_0}(A) \leqslant I_M(A) + c$$

for all $A \subset \mathbb{N}$. This is proved just as Solomonoff–Kolmogorov theorem. Fix any such M_0 and call $I(A) \overset{def}{=} I_{M_0}(A)$ the *complexity of enumeration of*

[10]In the case of finite sets any such program is called an *implicit description of* A, as opposed to *explicit description of* A when M is required to halt after having printed the last element of A.

A. This complexity, like the plain Kolmogorov complexity, depends on the choice of the universal algorithm and is defined up to an additive constant.

Second, define the a priori probability distribution on enumerable sets. Let M be an algorithm with one infinite input tape and one infinite output tape as described above. For every infinite 0-1-sequence ω let $M(\omega)$ denote the set enumerated by M when ω is written on its input tape. For every $A \subset \mathbb{N}$ consider the probability

$$p_M(A) = \text{Prob}[M(\omega) = A].$$

We say that M *generates* p_M. A theorem of de Leeuw, Moore, Shannon and Shapiro [6] states that if $p_M(A) > 0$ for some algorithm M then A is enumerable.

The class of generatable distributions has a maximal one up to a multiplicative constant. In other words, there is a algorithm M_1 (called *optimal*) such that for every algorithm M there is a constant c such that

$$c \cdot p_{M_1}(A) \geqslant p_M(A)$$

for all $A \subset \mathbb{N}$. This is proved just as the theorem on existence of maximal lower semi-computable semimeasure on natural numbers.

Fix any optimal M_1 and call $m(A) \overset{def}{=} p_{M_1}(A)$ the *a priori* probability of enumerating A. The a priori distribution thus depends on the choice of the optimal algorithm and is defined up to a multiplicative constant. Let $H(A)$ denote the negative binary logarithm of the a priori probability of A: $H(A) = \lceil -\log m(A) \rceil$.

Comparing M_0, the algorithm defining $I(A)$, with M_1, the algorithm defining $m(A)$, we see that

$$H(A) = \lceil -\log p_{M_1}(A) \rceil \leqslant I_{M_1}(A) \leqslant I_{M_0}(A) + O(1) = I(A) + O(1)$$

for all A. Solovay [12] has proved that conversely $I(A) \leqslant 3H(A) + O(\log H(A))$ for all A, which can be viewed as a sharpening of de Leeuw et al.'s result. It is unknown whether we can replace the constant 3 in this inequality by a smaller constant. In the case of *finite* sets we can do it.

Theorem 11 ([15]). *For every finite set $A \subset \mathbb{N}$ we have $I(A) \leqslant 2H(A) + O(\log H(A))$.*

The proof of Theorem 11 is based on a computable winning strategy in the following game.[11]

[11]The proof of Solovay's theorem is also based on a game. That game is more complicated and we omit it in this survey.

Ants versus cats game

The game is specified by a positive rational number ε and a natural number k. Consider finite subsets of $\mathbb{N}$ as nodes on the infinite directed graph, where there is an arc from a set A to a set B if A is a proper subset of B.

Imagine that certain very tiny animals, like ants, and also some medium size animals, like cats, move along edges of this graph. At the start of the game all ants and cats are in the initial node $\emptyset$. We regard ants as infinitely divisible. The cats are not divisible and there are k of them.

The movements of cats are controlled by Alice, and the movements of ants by Bob. Players make alternative moves (say, Bob starts). In his turn Bob may make a finite number actions of the following type: move any portion of ants from a node A to a node B that is reachable from A along edges of the graph. In her turn Alice may make a finite number actions of the following type: move a cat to a new node, also only along edges of the graph.

The game lasts infinitely long. Alice wins if after each of her moves there is a cat in every node where the fraction of ants is at least ε. Call this game the k, ε-game. We will assume that $1/\varepsilon$ is integer.

Analysis of the game

Clearly, if $k < 1/\varepsilon$ then Bob wins k, ε-game. Indeed, on the first move, Bob can move a fraction ε of ants in all the nodes $\{1\}, \{2\}, \dots, \{1/\varepsilon\}$. After her first move Alice looses, as she have not enough cats to put them in all these nodes.

It is not hard to see, however, that $1/\varepsilon$ cats are not enough for Alice to win. For example, Bob can win $1/4, 4$-game as follows. Bob first moves $1/4$ of ants in each of the nodes $\{1\}, \{2\}, \{3\}, \{4\}$, forcing cats to move in those nodes. Then $1/4$ of ants move to the node $\{1, 2\}$ and another $1/4$ of ants to the node $\{3, 4\}$. The remaining amount of ants in each of nodes $\{1\}, \{2\}, \{3\}, \{4\}$ is $1/8$. Alice has to move a cat to each of the nodes $\{1, 2\}$ and $\{3, 4\}$. Say, cats leave nodes $\{1\}$ and $\{3\}$. Now both these nodes have $1/8$ of ants out of reach of any cat. These ants move to the node $\{1, 3\}$ and Bob wins.

On the other hand, it turns out that Alice can win with $O(1/\varepsilon^2)$ cats, and her winning strategy is computable. Theorem 11 is basically a reformulation of the existence of such strategy.

Lemma 7. *Alice has a computable winning strategy in $\varepsilon, O(1/\varepsilon^2)$-game. (The winning algorithm receives ε as an input.)*

Proof. [12] Let $K = 1/\varepsilon$. Alice will have $K(K+1)/2$ cats. After each Alice's move each cat will be assigned a *rank*, which is a natural number in the segment $\{1, 2, \dots, K\}$. Also she will assign to each cat a subset of Ω of measure ε, called the *set of ants attended by that cat.* That subset will be a finite union of intervals.

The rank and attended ants will change time to time so that the following be true after each Alice's move (and at the start of the game).

1. For all $r \leqslant K$ there are exactly r cats of rank r.

2. The sets of ants attended by cats of the same rank are pair-wise disjoint. (As there are K cats of rank K, this item implies that every ant is attended by a cat of rank K.)

3. All ants attended by a cat are at its reach (the cat can move to the node where that ant sits).

4. There is a cat in every node A where the amount of ants is at least ε.

At the start all $K(K+1)/2$ cats are in the node $\emptyset$ and ranks and attended ants are assigned so that items 1 and 2 be true. Items 3 and 4 follow.

Ants' movement can destroy only item 4 so we need to explain how we restore item 4 after every Alice's move. W.l.o.g. we may assume that item 4 has become false for only one node A. By item 2 there is a cat that attends an ant in the node A. Choose such cat of the smallest rank, call it *the chosen one*, and move it to A. We have restored item 4 for the node A (however we might have created a similar problem for the previous location of the chosen cat; further we will explain how to handle with that). To restore item 3 change the set of ants attended by the chosen cat to an ε fraction of ants in the node A.

This change can violate item 2 and to restore it we change ranks as follows. Let r denote the rank of the chosen cat. As no cat of rank $r-1$ attends any ant in the node A, the set of ants attended by the chosen cat is disjoint with sets attended by all the cats of rank $r-1$. Thus we can just swap ranks r and $r-1$ (except the rank of the chosen cat).

After these actions all the items are fulfilled possibly but item 4 for the previous location B of the chosen cat. If that case B is a proper subset of A and we repeat the same procedure for B in place of A. After a finite number of times the process will converge. $\square$

[12] The presented strategy is based on Martin's strategy in another game from [9].

It is unknown what happens for $k \approx 1/\varepsilon^{\alpha}$ where α is a constant between 1 and 2.

Proof of Theorem 11

We will prove a general statement that allows to translate existence of computable strategies in ε, k-games to inequalities relating $I(A)$ and $H(A)$.

Lemma 8. *For any constant rational α the following holds. (1) If for some constant c Alice has a computable strategy that for all ε wins $\varepsilon, c/\varepsilon^{\alpha}$-game*[13] *then $I(A) \leqslant \alpha H(A) + O(\log H(A))$. (2) Conversely, if for some positive δ Bob has a computable winning strategy that for arbitrarily small ε wins $\varepsilon, \delta/\varepsilon^{\alpha}$-game*[13] *then $I(A) \geqslant \alpha H(A) - O(\log I(A))$ for infinitely many finite sets A.*

Proof. (1) Assume that Alice has a computable winning strategy in $\varepsilon, c/\varepsilon^{\alpha}$-game. Fix i and let $\varepsilon = 2^{-i}$ and $k = c/\varepsilon^{\alpha}$. Consider the following Bob's strategy. Bob runs the universal machine from the definition of $m(A)$ on all possible random inputs for $t = 1, 2, \dots$ steps. Every infinite sequence $\omega \in \Omega$ is considered by Bob as an infinitesimal ant. Ants move so that after tth Bob's move an ant ω sits in a node A iff the universal machine on input ω in t steps has printed all elements of A and no other elements.

More specifically, let t be a natural number and p a string of length at least t. Let $A^t(p)$ denote the output of the algorithm on inputs with prefix p (in t steps the algorithm cannot scan more than t input symbols, thus $A^t(p)$ is well defined). On tth move for each string p of length t all ants in the set Ω_p move from the node $A^{t-1}(p)$ (where they were immediately after $t-1$st Bob's move) to the node $A^t(p)$. As $A^{t-1}(p) \subset A^t(p)$, Bob does not violate the rules of the game.

Apply Alice's computable winning strategy against this Bob's strategy. Ants and cats move in a computable way. In particular, the location of cat number j at time t can be computed given j, t and i. Let C_j stand for the union of all nodes (i.e. sets) visited by cat number j. There is a machine M' that on every input beginning with $\hat{i}j$ enumerates C_j. For this machine it holds

$$I_{M'}(C_j) \leqslant 2\log i + \log j + O(1) \leqslant 2\log i + \alpha i + O(1)$$

and by universality

$$I(C_j) \leqslant I_{M'}(C_j) + O(1) \leqslant \alpha i + 2\log i + O(1)$$

[13]The winning algorithm receives ε as an input.

for all i.

Assume now that $m(A) \geqslant 2^{-i+1}$. Starting from some Bob's move the fraction of ants in the node A will be at least $2^{-i} = \varepsilon$. [14] Thus starting from some move there is a cat in A. One of them visits A infinitely often, which means that it sits in A starting from some time. In other words, $A = C_j$ for some j and

$$I(A) \leqslant 2i + 2\log i + O(1)$$

for every A with $m(A) \geqslant 2^{-i+1}$. Letting i be the integer part of $-\log m(A)/2$ we obtain the proof of the first part of Theorem.

(2) Assume that for arbitrarily small ε Bob has a computable winning strategy in $\varepsilon, \delta/\varepsilon^{\alpha}$-game. Thus for some constant c for infinitely many i Bob wins $2^{-i}, 2^{\alpha i - c}$-game.[15]

Fix any such i and let Alice use the following strategy. She identifies her cats with binary strings of length less than $\alpha i - c$. She runs the universal algorithm in the definition of $I(A)$ in steps on all inputs of length less than $\alpha i - c$. Once she finds out that the machine on an input p prints in t steps all elements from a set A and no other elements, she moves "cat p" to the node A. This strategy guarantees the following property:

> If A is a finite set such that $I(A) < \alpha i - c$ then starting from some time there is a cat in the node A.

Apply Bob's winning strategy against this Alice's strategy. We can assume that Bob's strategy is "lazy". That is, if in Bob's turn there is a node where the amount of ants is at least 2^{-i} but no cat sits in that node then the strategy just passes (does nothing). Call t *good* if this happens after tth Alice's move.

We are given that Bob wins the game. This means that there is T such that all $t > T$ are good. On all moves $t > T$ ants do not move. Let $A_1, \dots, A_N$ be all nodes where the amount of ants on moves $t > T$ is at least 2^{-i}. For every $t > T$ at least one of the nodes $A_1, \dots, A_N$ has no cat in time t. Therefore, there is a node A_j that has no cat infinitely many times, which implies $I(A_j) \geqslant \alpha i - c$.

[14] Indeed, let β denote the amount of ants that have visited the node A during the entire game. There is t such that the amount of ants that have visited A in first t moves is at least $\beta - \varepsilon$. On steps $t+1, t+2, \dots$ at most ε of ants in total can arrive in A, hence on every step $t' \geqslant t$ the amount of ants in A is at least $m(A) - \varepsilon \geqslant 2^{-i+1} - \varepsilon = \varepsilon$.

[15] Indeed, assume that Bob has a computable winning strategy in $\varepsilon, \delta/\varepsilon^{\alpha}$-game. Let i be the integer such that $2^{-i} \leqslant \varepsilon < 2^{-i+1}$. Then the same strategy wins $2^{-i}, \delta 2^{\alpha(i-1)}$-game.

As movement of ants is computable (given i), for every i there is an algorithm M_i such that $m_{M_i}(A)$ coincides with the limit of fraction of ants in node A for every finite set A.

The algorithm M_i depends on i. However, all M_i can be merged into one algorithm M such that

$$m_M(A) = \sum_i \frac{m_{M_i}(A)}{i(i+1)}$$

(choose i with probability $\frac{1}{i(i+1)}$ and then run M_i). For this algorithm M for infinitely many i there is A such that

$$m_M(A) \geqslant \frac{2^{-i}}{i(i+1)}, \qquad I(A) \geqslant \alpha i - c.$$

These inequalities imply that

$$\alpha H(A) \leqslant \alpha(i + \log i(i+1) + O(1)) \leqslant I(A) + O(\log I(A)). \quad \square$$

Acknowledgment

The author is sincerely grateful to Alexey Klimenko for reading the preliminary version of the paper.

References

[1] C.H. Bennett, P. Gács, M. Li, P.M.B. Vitanyi, and W.H. Zurek. “Information Distance”, IEEE Trans. on Information Theory **44** (1998), No 4, 1407–1423.

[2] B. Durand, A. Shen, and N. Vereshchagin. Descriptive Complexity of Computable Sequences. Theoretical Computer Science 171 (2001), p. 47–58;

[3] S. Fiorini and R. J. Wilson, Edge-colorings of graphs, Pitman 1977.

[4] Gregory Chaitin, ”A theory of program size formally identical to information theory,” Association for Computing Machinery Journal, vol. 22, 1975, pp. 329-340.

[5] Kolmogorov A. N., Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1–7, 1965.)

[6] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro, Computability by probabilistic machines, In: C. E. Shannon and J. McCarthey (Eds.), Automata Studies, Princeton University Press, Princeton, New Jersey, 1956, 183–212.

[7] M. Li, P. Vitanyi. An Introduction to Kolmogorov Complexity and its Applications. Second edition. Springer Verlag, 1997.

[8] D.W. Loveland. "A variant of Kolmogorov concept of Complexity", *Information and Control,* 15:510–526, 1969.

[9] D.A. Martin, Borel indeterminacy, Ann. Math. 102 (1978) 363–371.

[10] An.A. Muchnik, On basic structures of the descriptive theory of algorithms. *Soviet Math. Dokl.*, **32**, 671–674 (1985)

[11] R.J. Solomonoff. "A formal theory of inductive inference, part 1 and part 2," *Information and Control,* 7:1–22, 224-254, 1964.

[12] R.M. Solovay, In: A.I. Arruda, N.C.A. da Costa, R. Chaqui (Eds.) On Random R.E. Sets, Non-Classical Logics, Model Theory and Computability, North-Holland, Amsterdam, 1977, pp. 283–307.

[13] V.A. Uspensky, A.Kh. Shen'. "Relations between varieties of Kolmogorov complexities," *Math. Systems Theory,* 29:271–292, 1996.

[14] Nikolai K. Vereshchagin, Paul M. B. Vitanyi, A Theory of Lossy Compression for Individual Data, CoRR cs.IT/0411014: (2004)

[15] N. Vereshchagin, "Kolmogorov complexity of enumerating finite sets" Information Processing Letters 103 (2007) 34-39.

[16] R. A. Wilson, Graphs, Colorings and the Four-color Theorem, Oxford Univ. Pr. 2002.

[17] A.K. Zvonkin, L.A. Levin. "The complexity of finite objects and the development of the concepts of information and randomness by means of theory of algorithms." *Russian Math. Surveys*, 25(6):83–124, 1970.