Information Distance Revisited

Bruno Bauwens
National Research University Higher School of Economics, Moscow, Russia
https://www.hse.ru/en/org/persons/160550073
brbauwens@gmail.com

Abstract

We consider the notion of information distance between two objects x and y introduced by Bennett, Gács, Li, Vitanyi, and Zurek [2] as the minimal length of a program that computes x from y as well as computing y from x, and study different versions of this notion. In the above paper, it was shown that the prefix version of information distance equals $\max(\mathrm{K}(x|y), \mathrm{K}(y|x))$ up to additive logarithmic terms. It was claimed by Mahmud [13] that this equality holds up to additive $O(1)$-precision. We show that this claim is false, but does hold if the distance is at least logarithmic. This implies that the original definition provides a metric on strings that are at superlogarithmically separated.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Kolmogorov complexity, algorithmic information distance

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.46

Related Version https://arxiv.org/abs/1807.11087

Acknowledgements This work was initiated by Alexander (Sasha) Shen, who informed me about the error in [13] during a discussion of the paper [15]. Afterwards I explained the proof of Theorem 13 to Sasha. He simplified it, and he wrote all of the current manuscript and the proof of Theorem 13, which is available in the ArXiv version of this paper [1]. Later, I added Theorem 14. Only this proof is written by me, and is also available on ArXiv [1]. After this was added, Sasha decided that his contribution was no longer proportional, and decided he did no longer want to remain an author. I am especially grateful for his generous permission to publish this nicely written document, with minor modifications suggested by reviewers. I thank the reviewers for these suggestions. All errors in this document are solely my responsability. I thank Mikhail Andreev for the proof of Proposition 7 and many useful discussions. Finally, I thank Artem Grachev and the participants of the Kolmogorov seminar in Moscow state university for useful discussions.

1 Introduction

Informally speaking, Kolmogorov complexity measures the amount of information in an object (say, a bit string) in bits. The complexity $\mathrm{C}(x)$ of x is defined as the minimal bit length of a program that generates x. This definition depends on the programming language used, but one can fix an optimal language that makes the complexity function minimal up to an $O(1)$ additive term. In a similar way one can define the *conditional* Kolmogorov complexity $\mathrm{C}(x|y)$ of a string x given some other string y as a condition. Namely, we consider the minimal length of a program that transforms y to x. Informally speaking, $\mathrm{C}(x|y)$ is the amount of information in x that is missing in y, the number of bits that we should give in addition to y if we want to specify x.

The notion of *information distance* was introduced in [2] as "the length of a shortest binary program that computes x from y as well as computing y from x." It is clear that such a program cannot be shorter than $\mathrm{C}(x|y)$ or $\mathrm{C}(y|x)$ since it performs both tasks; on the other hand, it cannot be much longer than the sum of these two quantities (we can combine the programs that map x to y and vice versa with a small overhead needed to separate the two parts and to distinguish x from y). As the authors of [2] note, "being shortest, such a program should take advantage of any redundancy between the information required to go from x to y and the information required to go from y to x", and the natural question arises: to what extent is this possible? The main result of [2] gives the strongest upper bound possible and says that the information distance equals $\max(\mathrm{C}(x|y), \mathrm{C}(y|x))$ with logarithmic precision. In many applications, this characterization turned out to be useful, see [11, Section 8.4]. In fact, in [2] the prefix version of complexity, denoted by $\mathrm{K}(x|y)$, and the corresponding definition of information distance were used; see, e.g. [14] for the detailed explanation of different complexity definitions. The difference between prefix and plain versions is logarithmic in the complexity, so it does not matter whether we use plain or prefix versions if we are interested in results with logarithmic precision. However, the prefix version of the above characterization has an advantage: after adding a large enough constant, this distance satisfies the triangle inequality. The plain variant does not have this property, and this follows from Proposition 7 below. However, several inequalities that are true with logarithmic precision for plain complexity, become true with $O(1)$-precision if prefix complexity is used. So one could hope that a stronger result with $O(1)$-precision holds for prefix complexity. If this is true, then also the original definition satisfies the triangle inequality (after a constant increase). In [2, Section VII], this was conjectured to be false, and in [13] it was claimed to be true; in [12] a similar claim is made with reference to [2].[1] Unfortunately, the proof in [13] contains an error, and we show that the result is not valid for prefix complexity with $O(1)$-precision. On the other hand, it is easy to see that the original argument from [2] can be adapted for plain complexity to obtain the result with $O(1)$-precision, as noted in [15].

In this paper we try to clarify the situation and discuss the possible definitions of information distance in plain and prefix versions, and their subtle points (one of these subtle points was the source of the error in [13]). We also discuss some related notions. In Section 2 we consider the easier case of plain complexity; then in Section 3 we discuss the different definitions of prefix complexity (with prefix-free and prefix-stable machines, as well as definitions using the a priori probability) and in Section 4 we discuss their counterparts for the information distance. In Section 5 we use the game approach to show that indeed the relation between information distance (in the prefix version) and conditional prefix complexity is *not* valid with $O(1)$-precision, contrary to what is said in [13]. Finally, we show that if the information distance is at least logarithmic, then equality holds.

2 Plain complexity and information distance

Let us recall the definition of plain conditional Kolmogorov complexity. Let $U(p, x)$ be a computable partial function of two string arguments; its values are also binary strings. We may think of U as an interpreter of some programming language. The first argument p is considered as a program and the second argument is an input for this program. Then we define the complexity function

$$\mathrm{C}_U(y|x) = \min\{|p| \colon U(p, x) = y\};$$

[1] The authors of [12] define (section 2.2) the function $E(x, y)$ as the prefix-free non-bipartite version of the information distance (see the discussion below in section 4.1) and then write: "the following theorem proved in [4] was a surprise: Theorem 1. $E(x, y) = \max\{\mathrm{C}(x|y), \mathrm{C}(y|x)\}$". They do not mention that in the paper they cited as [4] (it is [2] in our list) there is a logarithmic error term; in fact, they do not mention any error terms (though in other statements the constant term is written explicitly). Probably this is a typo, since more general Theorem 2 in [12] does contain a logarithmic error term.

here $|p|$ stands for the length of a binary string p, so the right hand side is the minimal length of a program that produces output y given input x. The classical Solomonoff–Kolmogorov theorem says that there exists an optimal U that makes C_U minimal up to an $O(1)$-additive term. We fix some optimal U and then denote C_U by just C. See, e.g., [11, 14] for the details.

Now we want to define the information distance between x and y. One can try the following approach: take some optimal U from the definition of conditional complexity and then define

$$E_U(x, y) = \min\{|p| \colon U(p, x) = y \text{ and } U(p, y) = x\},$$

i.e., consider the minimal length of a program that both maps x to y and y to x. However, there is a caveat, as the following simple observation shows.

▶ **Proposition 1.** *There exists some computable partial function U that makes C_U minimal up to an $O(1)$ additive term, and still $E_U(x, y)$ is infinite for some strings x and y and therefore not minimal.*

Proof. Consider an optimal function U and then define U' such that $U(\Lambda, x) = \Lambda$ where Λ is the empty string, $U'(0p, x) = 0U(p, x)$ and $U'(1p, x) = 1U(p, x)$. In other terms, U' copies the first bit of the program to the output and then applies U to the rest of the program and the input. It is easy to see that $\mathrm{C}_{U'}$ is minimal up to an $O(1)$ additive term, but $U'(q, \cdot)$ has the same first bit as q, so if x and y have different first bits, there is no q such that $U(q, x) = y$ and $U(q, y) = x$ at the same time. ◀

On the other hand, the following proposition is true (and can be proven in the same way as the existence of the optimal U for conditional complexity):

▶ **Proposition 2.** *There exists a computable partial function U that makes E_U minimal up to $O(1)$ additive term.*

Now we may define information distance for plain complexity as the minimal function E_U. Some examples. If x has small complexity, then $E_U(x, y) = \mathrm{C}(y) + O(1)$. Let x and y be n-bit strings, and let $\oplus$ denote the bitwise xor-operation. We have $E_U(x, y) \leqslant n + O(1)$, because $x \oplus (x \oplus y) = y$ and $y \oplus (y \oplus x) = x$. It turns out that the original argument from [2] can be easily adapted to show the following result (that is a special case of a more general result about several strings proven in [15]):

▶ **Theorem 3.** *The minimal function E_U equals* $\max(\mathrm{C}(x|y), \mathrm{C}(y|x)) + O(1)$.

Proof. We provide the adapted proof here for the reader's convenience. In one direction we have to prove that $\mathrm{C}(x|y) \leqslant E_U(x, y) + O(1)$, and the same for $\mathrm{C}(y|x)$. This is obvious, since the definition of E_U contains more requirements for p: it should map both x to y and y to x, while in $\mathrm{C}(x|y)$ it is enough to map y to x.

To prove the reverse inequality, consider for each n the binary relation R_n on strings (of all lengths) defined as

$$R_n(x, y) \quad \Longleftrightarrow \quad \mathrm{C}(x|y) < n \text{ and } \mathrm{C}(y|x) < n.$$

By definition, this relation is symmetric. It is easy to see that R_n is (computably) enumerable uniformly in n, since we may compute better and better upper bounds for C reaching ultimately its true value. We think of R_n as the set of edges of an undirected graph whose vertices are binary strings. Note that each vertex x of this graph has degree less than 2^n since there are less than 2^n programs of length less than n that map x to its neighbors.

For each n, we enumerate edges of this graph (i.e., pairs in R_n). We want to assign colors to the edges of R_n in such a way that edges that have a common endpoint have different colors. In other terms, we require that for every vertex x all edges of R_n adjacent to x have different colors. For that, 2^{n+1} colors are enough. Indeed, each new edge needs a color that differentiates it from less than 2^n existing edges adjacent to one its endpoint and less than 2^n edges adjacent to other endpoint.

Let us agree to use $(n+1)$-bit strings as colors for edges in R_n, and perform this coloring in parallel for all n. Now we define $U(p,x)$ for a $(n+1)$-bit string p and arbitrary string x as the string y such that the edge (x,y) has color p in the coloring of edges from R_n. Note that n can be reconstructed as $|p|-1$. The uniqueness property for colors guarantees that there is at most one y such that (x,y) has color p, so $U(p,x)$ is well defined. It is easy to see now that if $\mathrm{C}(x|y) < n$ and $\mathrm{C}(y|x) < n$, and p is the color of the edge (x,y), then $U(p,x) = y$ and $U(p,y) = x$ at the same time. This implies the reverse inequality (the $O(1)$ terms appears when we compare our U with the optimal one). ◀

▶ Remark 4. In the definition of information distance given above we look for a program p that transforms x to y and also transforms y to x. Note that we *do not tell the program which of the two transformations is requested.* A weaker definition would provide also this information to p. This modification can be done in several ways. For example, we may require in the definition of E that $U(p,0x) = y$ and $U(p,1y) = x$, using the first input bit as the direction flag. An equivalent approach is to use two computable functions U and U' in the definition and require that $U(p,x) = y$ and $U'(p,y) = x$. This corresponds to using different interpreters for both directions.

It is easy to show that the optimal functions U and U' exist for this two-interpreter version of the definition. A priori we may get a smaller value of information distance in this way, because the program's task is easier when the direction is known, informally speaking. But it is not the case for the following simple reason. Obviously, this new quantity is still an upper bound for both conditional complexities $\mathrm{C}(x|y)$ and $\mathrm{C}(y|x)$ with $O(1)$ precision. Therefore Theorem 3 guarantees that this new definition of information distance coincides with the old one up to $O(1)$ additive terms. For the prefix versions of information distance such a simple argument does not work anymore, see below.

We have seen that different approaches lead to the same notion of plain information distance (up to $O(1)$ additive term). There is also a simple and natural quantitative characterization of this notion as a minimal function in a class of functions.

▶ **Theorem 5.** *Consider the class of functions E that are non-negative, symmetric, upper semicomputable, and for some c, all n and all x, satisfy*

$$\#\{y\colon E(x,y) < n\} \leqslant c2^n. \tag{*}$$

For every optimal U this class contains E_U, and for any E in this class, we have $E_U \leqslant E + O(1)$.

Recall that upper semicomputability of E means that one can compute a sequence of total upper bounds for E that converges to E. The equivalent requirement: the set of triples (x,y,n) where x,y are strings and n are natural numbers, such that $E(x,y) < n$, is (computably) enumerable.

Proof. The function $\max(\mathrm{C}(x|y), \mathrm{C}(y|x))$ is upper semicomputable and symmetric. The inequality $(*)$ is true for it since it is true for the smaller function $\mathrm{C}(y|x)$ (for $c = 1$; indeed, the number of programs of length less than n is at most 2^n).

On the other hand, if E is some symmetric upper semicomputable function that satisfies $(*)$, then one can for any given x and n enumerate all y such that $E(x, y) < n$. There are less than $c2^n$ strings y with this property, so given x, each such y can be described by a string of $n + \lceil \log c \rceil$ bits, its ordinal number in the enumeration. Note that the value of n can be reconstructed from this string by decreasing its length by $\lceil \log c \rceil$, so $\mathrm{C}(y|x) \leqslant n + O(1)$ if $E(x, y) < n$. It remains to apply the symmetry of E and Theorem 3. ◀

▶ Remark 6. The name "information distance" motivates the following question: does the plain information distance satisfy the triangle inequality? With logarithmic precision the answer is positive, because

$$\mathrm{C}(x|z) \leqslant \mathrm{C}(x|y) + \mathrm{C}(y|z) + O(\log(\mathrm{C}(x|y) + \mathrm{C}(y|z))).$$

However, if we replace the last term by an $O(1)$-term, then the triangle inequality is no more true. Indeed, for all strings x and y, the distance between the empty string Λ and x is $\mathrm{C}(x) + O(1)$, and the distance between x and some encoding of a pair (x, y) is at most $\mathrm{C}(y) + O(1)$, and the triangle inequality for distances with $O(1)$-precision would imply $\mathrm{C}(x, y) \leqslant \mathrm{C}(x) + \mathrm{C}(y) + O(1)$, and this is not true, see, e.g., [14, section 2.1].

One may ask whether a weaker statement saying that there is a maximal (up to an $O(1)$ additive term) function in the class of all symmetric non-negative functions E that satisfy both the condition $(*)$ and the triangle inequality, is true. The answer is negative, as the following proposition shows.

▶ **Proposition 7.** *There are two upper semicomputable symmetric functions E_1, E_2 that both satisfy the condition $(*)$ and the triangle inequality, such that no function that is bounded both by E_1 and E_2 can satisfy $(*)$ and the triangle inequality at the same time.*

Proof. Let us agree that $E_1(x, y)$ and $E_2(x, y)$ are infinite when x and y have different lengths. If x and y are n-bit strings, then $E_1(x, y) \leqslant k$ means that all the bits in x and y outside the first k positions are the same, and $E_2(x, y) \leqslant k$ is defined in a symmetric way for the last k positions. Both E_1 and E_2 satisfy the triangle inequality (and even the ultrametric inequality) and also satisfy the condition $(*)$, since the ball of radius k consist of strings that coincide except for the first/last k bits. If E is bounded both by $E_1 + O(1)$ and $E_2 + O(1)$ and satisfies the triangle inequality, then by changing the first k and the last l positions in a string x we get a string y such that $E(x, y) \leqslant k + l + O(1)$, and it is easy to see that the number of strings that can be obtained in this way is not $O(2^{k+l})$, but $\Theta((k + l)2^{k+l})$. ◀

3 Prefix complexity: different definitions

The notion of prefix complexity was introduced independently by Levin [6, 8, 4] and later by Chaitin [3]. There are several versions of this definition, and they all turn out to be equivalent, so people usually do not care much about technical details that are different. However, if we want to consider the counterparts of these definitions for information distance, their differences become important if we are interested in $O(1)$-precision.

Essentially there are four different definitions of prefix complexity that appear in the literature.

3.1 Prefix-free definition

A computable partial function $U(p,x)$ with two string arguments and string values is called *prefix-free* (with respect to the first argument) if $U(p,x)$ and $U(p',x)$ cannot be defined simultaneously for a string p and its prefix p' *and for the same second argument x*. In other words, for every string x the set of strings p such that $U(p,x)$ is defined is prefix-free, i.e., does not contain a string and its prefix at the same time.

For a prefix-free function U we may consider the complexity function $\mathrm{C}_U(y|x)$. In this way we get a smaller class of complexity functions compared with the definition of plain complexity, and the Solomonoff–Kolmogorov theorem can be easily modified to show that there exists a minimal complexity function in this smaller class (up to $O(1)$ additive term, as usual). This function is called *prefix conditional complexity* and usually is denoted by $\mathrm{K}(y|x)$. It is greater than $\mathrm{C}(y|x)$ since the class of available functions U is more restricted; the relation between C and K is well studied, see, e.g., [14, chapter 4] and references within.

The unconditional prefix complexity $\mathrm{K}(x)$ is defined in the same way, with U that does not have a second argument. We can also define $\mathrm{K}(x)$ as $\mathrm{K}(x|y_0)$ for some fixed string y_0. This string may be chosen arbitrarily; for each choice we have $\mathrm{K}(x) = \mathrm{K}(x|y_0) + O(1)$ but the constant in the $O(1)$ bound depends on the choice of y_0.

3.2 Prefix-stable definition

The prefix-stable version of the definition considers another restriction on the function U. Namely, in this version the function U should be *prefix-stable* with respect to the first argument. This means that if $U(p,x)$ is defined, then $U(p',x)$ is defined and equal to $U(p,x)$ for all p' that are extensions of p (i.e., when p is a prefix of p'). We consider the class of all computable partial prefix-stable functions U and corresponding functions C_U, and observe that there exists an optimal prefix-stable function U that makes C_U minimal in this class.

It is rather easy to see that the prefix-stable definition leads to a version of complexity that does not exceed the prefix-free one, since each prefix-free computable function can be easily extended to a prefix-stable one. The reverse inequality is not so obvious and there is no known direct proof; the standard argument compares both versions with the forth definition of prefix complexity, (the logarithm of a maximal semimeasure, see Section 3.4 below).

Prefix-free and prefix-stable definitions correspond to the same intuitive idea: the program should be “self-delimiting”. This means that the machine gets access to an infinite sequence of bits that starts with the program and has no marker indicating the end of a program. The prefix-free and prefix-stable definitions correspond to two possible ways of accessing this sequence. The prefix-free definition corresponds to a blocking read primitive (if the machine needs one more input bit, the computation waits until this bit is provided). The prefix-stable definition corresponds to a non-blocking read primitive (the machine has access to the input bits queue and may continue computations if the queue is currently empty). We do not go into details here; the interested reader could find this discussion in [14, section 4.4].

3.3 A priori probability definition

In this approach we consider the *a priori probability* of y given x, the probability of the event “a random program maps x to y”. More precisely, consider a prefix-stable function $U(p,x)$ and an infinite sequence π of independent uniformly distributed random bits (a random variable). We say that $U(\pi,x) = y$ if $U(p,x) = y$ for some p that is a prefix of π. Since U is prefix-stable, the value $U(\pi,x)$ is well defined. For given x and y, we denote by $m_U(y|x)$ the probability of this event (the measure of the set of π such that $U(\pi,x) = y$). For each prefix-stable U we get some function m_U. It is easy to see that there exists an optimal U that makes m_U maximal (up to an $O(1)$-factor). Then we define prefix complexity $\mathrm{K}(y|x)$ as $-\log m_U(y|x)$ for this optimal U, where the logarithm has base 2.

It is also easy to see that if we use prefix-free functions U instead of prefix-stable ones, we obtain the same definition of prefix complexity. Informally speaking, if we have an infinite sequence of random bits as the first argument, we do not care whether we have blocking or non-blocking read access, the bits are always there. The non-trivial and most fundamental result about prefix complexity is that this definition, as the logarithm of the probability, is equivalent to the two previous ones. As a byproduct of this result we see that the prefix-free and prefix-stable definitions are equivalent. This proof and the detailed discussion of the difference between the definitions can be found, e.g., in [14, chapter 4].

3.4 Semimeasure definition

The semimeasure approach defines a priori probability in a different way, as a convergent series that converges as slow as possible. More precisely, a *lower semicomputable semimeasure* is a non-negative real-valued function $m(x)$ on binary strings such that $m(x)$ is a limit of an increasing sequence of rational numbers and $\sum_x m(x) \leqslant 1$ that is computable uniformly in x. There exists a lower semicomputable semimeasure $\mathsf{m}(x)$ that is maximal up to $O(1)$-factors, and its negative logarithm coincides with unconditional prefix complexity $\mathrm{K}(x)$ up to an $O(1)$ additive term.

We can define conditional prefix complexity in the same way, considering semimeasures with parameter y. Namely, we consider lower semicomputable non-negative real-valued functions $m(x, y)$ such that $\sum_x m(x, y) \leqslant 1$ for every y. Again there exists a maximal function among them, denoted by $\mathsf{m}(x|y)$, and its negative logarithm equals $\mathrm{K}(x|y)$ up to an $O(1)$ additive term.

To prove this equality, we note first that the a priori conditional probability $m_U(x|y)$ is a lower semicomputable conditional semimeasure. The lower semicomputability is easy to see: we can simulate the machine U and discover more and more programs that map y to x. The inequality $\sum_x m_U(x|y)$ also has a simple probabilistic meaning: the events "π maps y to x" for a given y and different x are disjoint, so the sum of their probabilities does not exceed 1. The other direction (starting from a semimeasure, construct a machine) is a bit more difficult, but in fact it is possible (even exactly, without additional $O(1)$-factors). See [14, chapter 4] for details.

The semimeasure definition can be reformulated in terms of complexities (by taking exponents): $\mathrm{K}(x|y)$ is a minimal (up to $O(1)$ additive term) upper semicomputable non-negative integer function $k(x, y)$ such that

$$\sum_x 2^{-k(x,y)} \leqslant 1$$

for all y. A similar characterization of plain complexity would use a weaker requirement

$$\#\{x \colon k(x, y) < n\} < c2^n$$

for some c and all y. (We discussed a similar result for information distance where the additional symmetry requirement was used, but the proof is the same.)

3.5 Warning

There exists a definition of plain conditional complexity that does *not* have a prefix-version counterpart. Namely, the plain conditional complexity $\mathrm{C}(x|y)$ can be equivalently defined as the *minimal unconditional plain complexity of a program that maps y to x*. In this way we do not need the programming language used to map y to x to be optimal; it is enough to assume that we can computably translate programs in other languages into our language; this property, sometimes called *s-m-n-theorem* or *Gödel property of a computable numbering*, is true for almost all reasonable programming languages. Of course, we still assume that the language used in the definition of unconditional Kolmogorov complexity is optimal.

One may hope that $\mathrm{K}(x|y)$ can be similarly defined as the minimal unconditional prefix complexity of a program that maps y to x. The following proposition shows that it is not the case.

▶ **Proposition 8.** *The prefix complexity* $\mathrm{K}(x|y)$ *does not exceed the minimal prefix complexity of a program that maps y to x; however, the difference between these two quantities is not bounded.*

Proof. To prove the first part, assume that $U_1(p)$ is a prefix-stable function of one argument that makes the complexity function

$$\mathrm{C}_{U_1}(q) = \min\{|p|\colon U(p) = q\}$$

minimal. Then $\mathrm{C}_U(q) = \mathrm{K}(q) + O(1)$. (We still need an $O(1)$ term since the choice of an optimal prefix-stable function is arbitrary). Then consider the function

$$U_2(p, x) = [U_1(p)](x)$$

where $[q](x)$ denotes the output of a program q on input x. Then U_2 is a prefix-stable function from the definition of conditional prefix complexity, and

$$\mathrm{C}_{U_2}(y|x) \leqslant \mathrm{C}_{U_1}(q)$$

for any program q that maps x to y (i.e., $[q](x) = y$). This gives the inequality mentioned in the proposition. Now we have to show that this inequality is not an equality with $O(1)$-precision.

Note that $\mathrm{K}(x|n) \leqslant n + O(1)$ for every binary string x of length n. Indeed, a prefix-stable (or prefix-free) machine that gets n as input can copy n first bits of its program to the output. (The prefix-free machine should check that there are exactly n input bits.) In this way we get n-bit programs for all strings of length n.

Now assume that the two quantities coincide up to an $O(1)$ additive term. Then for every string x there exists a program q_x that maps $|x|$ to x and $\mathrm{K}(q_x) \leqslant |x| + c$ for all x and some c. Note that q_x may be equal to q_y for $x \neq y$, but this may happen only if x and y have different lengths. Consider now the set Q of all q_x for all strings x, and the series

$$\sum_{q\in Q} 2^{-\mathrm{K}(q)}. \qquad (**)$$

This sum does not exceed 1 (it is a part of a similar sum for all q that is at most 1, see above). On the other hand, we have at least 2^n different programs q_x for all n-bit strings x, and they correspond to different terms in $(**)$; each of these terms is at least 2^{-n-c}. We get a converging series that contains, for every n, at least 2^n terms of size at least 2^{-n-c}. It is easy to see that such a series does not exist. Indeed, each tail of this series should be at least 2^{-c-1} (consider these 2^n terms for large n when at least half of these terms are in the tail), and this is incompatible with convergence. ◀

Why do we get a bigger quantity when considering the prefix complexity of a program that maps y to x? The reason is that the prefix-freeness (or prefix-stability) requirement for the function $U(p, x)$ is formulated separately for each x: the decision where to stop reading the program p *may depend on its input* x. This is not possible for a prefix-free description of a program that maps x to y. It is easy to overlook this problem when we informally describe prefix complexity $\mathrm{K}(x|y)$ as "the minimal length of a program, written in a self-delimiting language, that maps y to x", because the words "self-delimiting language" implicitly assume that we can determine where the program ends while reading the program text (and before we know its input), and this is a wrong assumption.

3.6 Historical digression

Let us comment a bit on the history of prefix complexity. It appeared first in 1971 in Levin's PhD thesis [6]; Kolmogorov was his thesis advisor. Levin used essentially the semimeasure definition (formulated a bit differently). This thesis was in Russian and remained unpublished for a very long time. In 1974 Gács' paper [4] appeared where the formula for the prefix complexity of a pair was proven. This paper mentioned prefix complexity as "introduced by Levin in [4], [5]" ([7] and [8] in our numbering). The first of these two papers does not say anything about prefix complexity explicitly, but defines the monotone complexity of sequences of natural numbers, and prefix complexity can be considered as a special case when the sequence has length 1 (this is equivalent to the prefix-stable definition of prefix complexity). The second paper has a comment "(to appear)" in Gács' paper. We discuss it later in this section.

Gács does not reproduce the definition of prefix complexity, saying only that it is "defined as the complexity of specifying x on a machine on which it is impossible to indicate the endpoint[2] of a master program: an infinite sequence of binary symbols enters the machine and the machine must itself decide how many binary symbols are required for its computation". This description is not completely clear, but it looks more like a prefix-free definition if we understand it in such a way that the program is written on a one-directional tape and the machine decides where to stop reading. Gács also notes that prefix complexity (he denotes it by $KP(x)$) "is equal to the [negative] base two logarithm of a universal semicomputable probability measure that can be defined on the countable set of all words".

Levin's 1974 paper [8] says that "the quantity $KP(x)$ has been investigated in details in [6,7]". Here [7] in Levin's numbering is Gács paper cited above ([4] is our numbering) and has the comment "in press", and [6] in Levin's numbering is cited as [Levin L.A., On different version of algorithmic complexity of finite objects, to appear]. Levin does not have a paper with exactly this title, but the closest approximation is his 1976 paper [9], where prefix complexity is defined as the logarithm of a maximal semimeasure. Except for these references, [8] describes the prefix complexity in terms of prefix-stable functions: "It differs from the Kolmogorov complexity measure $\langle \ldots \rangle$ in that the decoding algorithm A has the following "prefix" attribute: if $A(p_1)$ and $A(p_2)$ are defined and distinct, then p_1 cannot be a beginning fragment of p_2".

The prefix-free and a priori probability definitions were given independently by Chaitin in [3] (in different notation) together with the proof of their equivalence, so [3] was the first publication containing this (important) proof.

Now it seems that the most popular definition of prefix complexity is the prefix-free one, for example, it is given as the main definition in [11].

[2] The English translation says "halting" instead of "endpoint" but this is an obvious translation error.

4 Prefix complexity and information distance

4.1 Four versions of prefix information distance

Both the prefix-free and prefix-stable versions of prefix complexity have their counterparts for the information distance.

Let $U(p,x)$ be a partial computable prefix-free [respectively, prefix-stable] function of two string arguments having string values. Consider the function

$$E_U(x,y) = \min\{|p| \colon U(p,x) = y \text{ and } U(p,y) = x\}.$$

As before, one can easily prove that there exists a minimal (up to $O(1)$) function among all functions E_U of the class considered. It will be called *prefix-free* [respectively *prefix-stable*] *information distance* function. We clarify the difference between these variants.

Note that only the cases when $U(p,x) = y$ and also $U(p,y) = x$ matter for E_U. So we may assume without loss of generality that $U(p,x) = y \Leftrightarrow U(p,y) = x$ waiting until both equalities are true before finalizing the values of U. Then for every p we have some matching M_p on the set of all strings: an edge x–y is in M_p if $U(p,x) = y$ and $U(p,y) = x$. This is indeed a matching: for every x only $U(p,x)$ may be connected with x.

The set M_p is enumerable uniformly in p. In the prefix-free version the matchings M_p and M_q are disjoint (have no common vertices) for two compatible strings p and q (one is an extension of the other). For the prefix-stable version M_p increases when p increases (and remains a matching). It is easy to see that a family M_p that has these properties, always corresponds to some function U, and this statement holds both in the prefix-free and prefix-stable version.

There is another way in which this definition could be modified. As we have discussed for plain complexity, we may consider two different functions U and U' and consider the distance function

$$E_{U,U'}(x,y) = \min\{|p| \colon U(p,x) = y \text{ and } U'(p,y) = x\}.$$

Intuitively this means that we know the transformation direction in addition to the input string. This corresponds to matchings in a bipartite graph where both parts consist of all binary strings; the edge x–y is in the matching M_p if $U(p,x) = y$ and $U'(p,y) = x$. Again instead of the pair (U,U') we may consider the family of matchings that are disjoint (for compatible p, in the prefix-free version) or monotone (for the prefix-stable version). In this way we get two other versions of information distance that could be called *bipartite prefix-free* and *bipartite prefix-stable* information distances.

In [2] the information distance is defined as the prefix-free information distance with the same function U for both directions, not two different ones. The definition in section III considers the minimal function among all E_U. This minimal function is denoted by $E_0(x,y)$, while $\max(\mathrm{K}(x|y),\mathrm{K}(y|x))$ is denoted by $E_1(x,y)$, see section I of the same paper. The inequality $E_1 \leqslant E_0$ is obvious, and the reverse inequality with logarithmic precision is proven in [2] as Theorem 3.3.

Which of the four versions of prefix information distance is the most natural? Are they really different? It is easy to see that the prefix-stable version (bipartite or not) does not exceed the corresponding prefix-free version, since every prefix-free function has a prefix-stable extension. Also each bipartite version (prefix-free or prefix-stable) does not exceed the corresponding non-bipartite version for obvious reasons: one may take $U = U'$. It is hard to say which version is most natural, and the question whether some of them coincide or all four are different, remains open. But as we will see in Theorem 13, the smallest of all four, the prefix-stable bipartite version, is still bigger than E_1 (the maximum of conditional complexities), and the difference is unbounded. Hence, for all four versions, including the prefix-free non-bipartite version used both in [2, 12, 13], the equality with $O(1)$-precision is not true, contrary to what is said in [13]. This was conjectured in [2, Section VII].

However, before going to this negative result, we prove some positive results about the definition of information distance that is a counterpart of the a priori probability definition of prefix complexity.

4.2 A priori probability of going back and forth

Fix some prefix-free function $U(p, x)$. The conditional a priori probability $m_U(y|x)$ is defined as

$$\Pr_\pi[U(\pi, x) = y],$$

where π is an infinite uniformly randomnly generated bitsequence, and $U(\pi, x) = y$ means that $U(p, x) = y$ for some p that is a prefix of π. As we discussed, there exists a maximal function among all m_U, and its negative logarithm equals the conditional prefix complexity $\mathrm{K}(y|x)$.

Now let us consider the counterpart of this construction for the information distance. The natural way to do this is to consider the function

$$e_U(x, y) = \Pr_\pi[U(\pi, x) = y \text{ and } U(\pi, y) = x].$$

Note that in this definition the prefixes of π used for both computations are not necessarily the same. It is easy to show, as usual, that there exists an *optimal* machine U that makes e_U maximal. Fixing some optimal U, we get some function $\mathsf{e}(x, y)$. Note that different optimal U lead to functions that differ only by $O(1)$-factor. The negative logarithm of this function coincides with E_1 from [2] with $O(1)$-precision, as the following result says.

▶ **Theorem 9.**

$$-\log \mathsf{e}(x, y) = \max(\mathrm{K}(x|y), \mathrm{K}(y|x)) + O(1).$$

Proof. Rewriting the right-hand side in the exponential scale, we need to prove that

$$\mathsf{e}(x, y) = \min(\mathsf{m}(x|y), \mathsf{m}(y|x))$$

up to $O(1)$-factors. One direction is obvious: $\mathsf{e}(x, y)$ is smaller than $\mathsf{m}(x|y)$ since the set of π in the definition of e is a subset of the corresponding set for m, if we use the probabilistic definition of $\mathsf{m} = m_U$. The same is true for $\mathsf{m}(y|x)$.

The non-trivial part of the statement is the reverse inequality. Here we need to construct a machine U such that

$$e_U(x, y) \geqslant \min(\mathsf{m}(x|y), \mathsf{m}(y|x))$$

up to $O(1)$-factors.

Let us denote the right-hand side by $u(x, y)$. The function u is symmetric, lower semicomputable and $\sum_y u(x, y) \leqslant 1$ for all x (due to the symmetry, we do not need the other inequality where y is fixed). This is all we need to construct U with the desired properties; in fact $e_U(x, y)$ will be at least $0.5u(x, y)$, (and the factor 0.5 is important for the proof).

Every machine U has a "dual" representation: for every pair (x, y) one may consider the subset $U_{x,y}$ of the Cantor space that consists of all π such that $U(\pi, x) = y$ and $U(\pi, y) = x$. These sets are effectively open (i.e., are computably enumerable unions of intervals in the Cantor space) uniformly in x, y, are symmetric ($U_{x,y} = U_{y,x}$) and have the following property: for a fixed x, all sets $U_{x,y}$ for all y (including $y = x$) are disjoint.

What is important to us is that this correspondence works in both directions. If we have some family $U_{x,y}$ of uniformly effectively open sets that is symmetric and has the disjointness property mentioned above, there exists a prefix-free machine U that generates these sets as described above. This machine works as follows: given some x, it enumerates the intervals that form $U_{x,y}$ for all y (it is possible since the sets $U_{x,y}$ are effectively open uniformly in x, y). One may assume without loss of generality that all the intervals in the enumeration are disjoint. Indeed, every effectively open set can be represented as a union of a computable sequence of disjoint intervals (to make intervals disjoint, we represent the set difference between the last interval and previously generated intervals as a a finite union of intervals). Note also that for different values of y the sets $U_{x,y}$ are disjoint by the assumption. If the enumeration for $U_{x,y}$ contains the interval $[p]$ (the set of all extensions of some bit string p), then we let $U(p, x) = y$ and $U(p, y) = x$ (we assume that the same enumeration is used for $U_{x,y}$ and $U_{y,x}$). Since all intervals are disjoint, the function $U(p, x)$ is prefix-free.

Now it remains (and this is the main part of the proof) to construct the family $U_{x,y}$ with the required properties in such a way that the measure of $U_{x,y}$ is at least $0.5u(x, y)$. In our construction it will be *exactly* $0.5u(x, y)$. For that we use the same idea as in [2] but in the continuous setting. Since $u(x, y)$ is lower semicomputable, we may consider the increasing sequence $u'(x, y)$ of approximations from below (that increase with time, though we do not explicitly mention time in the notation) that converge to $u(x, y)$. We assume that at each step one of the values $u'(x, y)$ increases by a dyadic rational number r. In response to that increase, we add to $U_{x,y}$ one or several intervals that have total measure $r/2$ and do not intersect $U_{x,z}$ and $U_{z,y}$ for any z. For that we consider the unions of all already chosen parts of $U_{x,z}$ and of all chosen parts of $U_{z,y}$. The measure of the first union is bounded by $0.5 \sum_z u'(x, z)$ and the measure of the second union is bounded by $0.5 \sum_z u'(z, y)$ where u' is the lower bound for u before the r-increase. Since the sums remain bounded by 1 after the r-increase, we may select a subset of measure $r/2$ outside both unions. (We may even select a subset of measure r, but this will destroy the construction at the following steps, so we add only $r/2$ to $U_{x,y}$.) ◀

▶ Remark 10. As for the other settings, we may consider two functions U and U' and the probability of the event

$$e_{U,U'}(x, y) = \Pr_\pi[U(\pi, x) = y \text{ and } U'(\pi, y) = x]$$

for those U, U' that make this probability maximal. The equality of Theorem 9 remains valid for this version. Indeed, the easy part can be proven in the same way, and for the difficult direction we have proven a stronger statement with additional requirement $U = U'$.

One can also describe the function e as a maximal function in some class, therefore getting a quantitative definition of E_0. This is essentially the statement of theorem 4.2 in [2]. In terms of semimeasures it can be reformulated as follows.

▶ **Proposition 11.** *Consider the class of symmetric lower semicomputable functions $u(x, y)$ with string arguments and non-negative real values such that $\sum_y u(x, y) \leqslant 1$ for all x. This class has a maximal function that coincides with $\min(\mathsf{m}(x|y), \mathsf{m}(y|x))$ up to an $O(1)$ factor.*

Indeed, we have already seen that this minimum has the required properties; if some other function $u(x, y)$ in this class is given, we compare it with conditional semimeasures $\mathsf{m}(x|y)$ and $\mathsf{m}(y|x)$ and conclude that u does not exceed both of them.

In logarithmic scale this statement can be reformulated as follows: *the class of upper semicomputable symmetric functions $D(x, y)$ with string arguments and real values such that $\sum_y 2^{-D(x,y)} \leqslant 1$ for each x, has a minimal element that coincides with* $\max(\mathrm{K}(x|y), \mathrm{K}(y|x))$ *up to an $O(1)$ additive term.* Theorem 4.2 in [2] says the same with the additional condition for D: it should satisfy the triangle inequality. This restriction makes the class smaller and could increase the minimal element in the class, but this does not happen since the function

$$\max(\mathrm{K}(x|y), \mathrm{K}(y|x)) + c$$

satisfies the triangle inequality for large enough c. This follows from the inequality $\mathrm{K}(x|z) \leqslant \mathrm{K}(x|y) + \mathrm{K}(y|z) + O(1)$ since the left hand size increases by c and the right hand size increases by $2c$ when K is increased by c.

▶ Remark 12. To be pedantic, we have to note that in [2] an additional condition $D(x, x) = 0$ is required for the functions in the class; to make this possible, one has to exclude the term $2^{-D(x,x)}$ in the sum (now this term equals 1) and require that $\sum_{y \neq x} 2^{-D(x,y)} \leqslant 1$ (p. 1414, the last inequality). Note that the triangle inequality remains valid if we change D and let $D(x, x) = 0$ for all x.

5 A counterexample

In this section we present the main negative (and most technically difficult) result of this paper that shows that none of the four prefix distances coincides with

$$E_1(x, y) = \max(\mathrm{K}(x|y), \mathrm{K}(y|x)).$$

▶ **Theorem 13.** *The bipartite prefix-stable information distance exceeds $E_1(x, y)$ more than by a constant: the difference is unbounded.*

As we have mentioned, the other three versions of the information distance are even bigger, so the same result is true for all of them. We will explain the proof for the non-bipartite prefix-stable version (it is a bit easier and less notation is needed) and then explain the changes needed for the bipartite prefix-stable version. Our proof also provides a lower bound in terms of the length: for strings of length n, the difference can be as large as

$$\log\log n - O(\log\log\log n).$$

The proof can be found in the ArXiv version of this paper [1]. It uses game approach: We first explain the game rules, then show that a computable winning strategy in the game implies that the difference is unbounded, and finally, present that computable winning strategy. Both the game and the winning strategy has similarities with the game in [5].

6 Equality if the distance is superlogarithmic

Given the previous result, all distances become equal for pairs of strings of equal length, provided their distance is not too small.

▶ **Theorem 14.** *If $|x| = |y|$ and $E_1(x, y) \geqslant 6 \log |x|$, then all four prefix information distances are equal to $E_1(x, y) + O(1)$.*

This seems to be the first equality in information theory whose precision becomes smaller if the quantity becomes larger. The proof can be found in the ArXiv version of this paper [1].

References

1 Bruno Bauwens and Alexander Shen. Information distance revisited. *arXiv preprint*, 2018. `arXiv:1807.11087`.

2 C. H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W. H. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4), 1998.

3 G.J. Chaitin. A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach.*, 22(3):329–340, 1975. `doi:10.1145/321892.321894`.

4 P. Gács. On the symmetry of algorithmic information. *Soviet Math. Dokl.*, 15(5):1477–1480, 1974.

5 P. Gács. On the relation between descriptional complexity and algorithmic probability. *Theor. Comput. Sci.*, 22:71–93, 1983.

6 Leonid A Levin. *Some theorems on the algorithmic approach to probability theory and information theory*. PhD thesis, BostonUniversity, MA, UnitedStates, 1971. Dissertation directed by A. N. Kolmogorov; turned down as required by the Soviet authorities despite unanimously positive reviews. Translated in English in [10].

7 Leonid A Levin. On the notion of a random sequence. *Soviet Mathematics-Doklady*, 14:1413–1416, 1973.

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

9 Leonid A Levin. Various measures of complexity for finite objects (axiomatic description). *Soviet Mathematics Doklady*, 17(2):522–526, 1976.

10 Leonid A Levin. Some theorems on the algorithmic approach to probability theory and information theory. *Annals of Pureand Applied Logic*, 162:224–235, 2010. 1971 dissertation directed by A. N. Kolmogorov; turned down as required by the Soviet authorities despite unanimously positive reviews.

11 Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th edition*. Springer, 2019. 1 ed., 1993; 2 ed., 1997, 3 ed 2008,.

12 Chong Long, Xiaoyan Zhu, Ming Li, and Bin Ma. Information shared by many objects. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1213–1220. ACM, 2008.

13 MM Hassan Mahmud. On universal transfer learning. *Theoretical Computer Science*, 410(19):1826–1846, April 2009.

14 Alexander Shen, Vladimir A Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*, volume 220. Mathematical Surveys and Monographs, volume 220, xviii+511 pages. American Mathematical Society American Mathematical Soc., 2017. Draft version: `http://www.lirmm.fr/~ashen/kolmbook-eng.pdf`.

15 Paul M.B. Vitányi. Exact expression for information distance. *IEEE Transactions on Information Theory*, 63:4725–4728, 2017. `arXiv:1410.7328`.