Inequalities for space-bounded Kolmogorov complexity

Bruno Bauwens, Peter Gács, Andrei Romashchenko, Alexander Shen

Inequalities for space-bounded Kolmogorov complexity*

Bruno Bauwens,† Peter Gács,‡ Andrei Romashchenko,§ Alexander Shen¶

Abstract

Finding all linear inequalities for entropies remains an important open question in information theory. For a long time the only known inequalities for entropies of tuples of random variables were Shannon (submodularity) inequalities. Only in 1998 Zhang and Yeung [18] found the first inequality that cannot be represented as a convex combination of Shannon inequalities, and several other non-Shannon inequalities were found soon after that. It turned out that the class of linear inequalities for entropies is rather fundamental, since the same class can be equivalently defined in terms of subgroup sizes or projections of multidimensional sets (Chan, Yeung [2, 3], Romashchenko, Shen, Vereshchagin [12]). The non-Shannon inequalities have interesting applications (e.g., to proofs of lower bounds for the information ratio of secret sharing schemes). Still the class of linear inequalities for entropies is not well understood, though some partial results are known (e.g., Matúš has shown in [10] that this class cannot be generated by a finite family of inequalities).

This class also appears in algorithmic information theory: the same linear inequalities are true for Shannon entropies of tuples of random variables and Kolmogorov complexities of tuples of strings (Hammer et al., [5]). This parallelism started with the Kolmogorov–Levin formula [7] for the complexity of pairs of strings with logarithmic precision. Longpré proved in [8] a version of this formula for the space-bounded complexities.

In this paper we prove a stronger version of Longpré's result with a tighter space bound, using Sipser's trick [15]. Then, using this result, we show that *every linear inequality that is true for complexities or entropies, is also true for space-bounded Kolmogorov complexities with a polynomial space overhead*, thus extending the parallelism to the space-bounded algorithmic information theory.

*Authors want to thank the members of the ESCAPE team (especially Ruslan Ishkuvatov), the participants of the Kolmogorov seminar (Moscow) and Algorithmic Randomness workshop for discussions. We are grateful to anonymous reviewers for STACS2021 conference (where the submission was rejected) who suggested many corrections and improvements.

†National Research University Higher School of Economins, ORCID 0000-0002-6138-0591. Supported by Russian Science Foundation (grant 20-11-20203)

‡Boston University, gacs@bu.edu, ORCID 0000-0003-2496-0332

§LIRMM, University of Montpellier, CNRS, Montpellier, France and IITP RAS, Moscow (on leave), https://www.lirmm.fr/~romashchen/, andrei.romashchenko@lirmm.fr, ORCID 0000-0001-7723-7880. Supported by ANR-15-CE40-0016 RaCAF and RFBR 19-01-00563 grants. Supported by ANR grant FLITTLA

¶LIRMM, University of Montpellier, CNRS, Montpellier, France and IITP RAS, Moscow (on leave), www.lirmm.fr/~ashen, alexander.shen@lirmm.fr, ORCID 0000-0001-8605-7734. Supported by ANR-15-CE40-0016 RaCAF and RFBR 19-01-00563 grants. Supported by ANR grant FLITTLA. Part of the work was done while participating in the American Institute of Mathematics Workshop on Algorithmic Randomness (August 2020)

1 Space-bounded Kolmogorov complexity

Kolmogorov in his seminal paper of 1965 [6] defined the complexity of a finite string as the minimal length of a program that produces this string:

$$\mathrm{C}_I(x) = \min\{|p| : I(p) = x\}.$$

Here I is a machine (considered as an interpreter of some programming language), p is a binary string (considered as a program without input), and $|p|$ is its length. In a similar way Kolmogorov defined $\mathrm{C}_I(y|x)$, the conditional complexity of y given x, as the minimal length of a program p that transforms x to y:

$$\mathrm{C}_I(y|x) = \min\{|p| : I(p,x) = y\}.$$

In this case the interpreter I has two arguments (considered as a program and an input for this program).

There exists an interpreter that is optimal to within an additive constant (Solomonoff, Kolmogorov). Different optimal interpreters lead to complexity functions that differ at most by an $O(1)$ additive term. So the complexity can be considered as an intrinsic property of the strings involved. Complexity measures the amount of information in individual finite objects, not random variables (distributions) as Shannon's information theory does. The relation between complexities of strings and entropies of probability distributions is well established: in particular it is shown in [5] that the same linear inequalities hold for both.

It is easy to see that complexity functions are not computable; moreover, they do not have non-trivial computable lower bounds. This fact is the basis for Chaitin's famous proof of Gödel's incompleteness theorem [1].

To information theorists, the non-computability of the complexity functions may obscure somewhat their combinatorial significance. A natural approach to the question is to consider versions of Kolmogorov complexity in which the interpreter has some resource (time, space) bounds. This makes the complexity functions computable since now for each program we can run it until it produces some result or exceeds the bound (if the latter does not happen for a long time, we know that there is a loop and the program will never terminate). However, the inequalities for these resource bounded versions become more complex, with different resource bounds on the two sides. In this paper we show a way to overcome these difficulties for the case of space bounds. We will see that each linear inequality holding for entropies holds also for many space-bounded versions of Kolmogorov complexity: the space bound can be chosen from a dense infinite hierarchy of possibilities.

From a more pragmatic point, one could add that unrestricted complexity is not only non-computable, but also irrelevant: if some string has a short program but the time needed to run this program is huge, this string for all practical purposes may be indistinguishable from an incompressible one.

Kolmogorov was aware of these issues: in the last paragraph of [6], he writes that the description complexities introduced above

> "... have one important disadvantage: They do not take into account the *difficulty* of transforming a program p and an object x into an object y.[1] Introducing necessary definitions, one can prove some mathematical statements that may be interpreted as the existence of objects that have very short programs, so their complexity is very

[1] The English translation mentions the "difficulty of preparing a program p for passing from an object x to an object y". However, the original Russian text is quite clear: Kolmogorov speaks about the complexity of *decompression*, not compression.

small, but the reconstruction of an object from the program requires an enormous amount of time. I plan to study elsewhere[2] the dependence of the necessary program complexity $K^t(x)$ on the allowed difficulty t for its transformation into x. Then the complexity $K(x)$ as defined earlier will be the minimum of $K^t(x)$ for unbounded t."[6, p. 11][3].

Defining $\mathrm{C}^r(x)$ and $\mathrm{C}^r(y|x)$ as the minimal length of the programs that generate x or transform x to y with resource bound r, we need to fix some computational model and the exact meaning of the resource bound. It is natural to consider time-bounded or space-bounded computations. The study of time-bounded complexity immediately bumps into the P vs. NP problem [8, 9], so in this paper we consider only the space-bounded version of complexity.

Usually the space used by a computation is measured up to a constant factor, but we need more precision. So we should fix a computational model carefully. For Turing machines with arbitrary tape alphabet one should take into account not only the number of cells used but also the alphabet size. If each cell may contain one of k symbols (where $k \geqslant 2$), then one should multiply the number of used cells by $\log_2 k$. This makes the Turing machine model "calibrated" in the following sense: the number of configurations with space not exceeding s, is close to 2^s. In fact, it differs from 2^s by a polynomial (in s) factor, since we have to take into account the head position (or heads positions for multitape Turing machines). The simulation between models, in our case, the emulation of multitape machines on machines with smaller number of tapes, uses $O(\log s)$ overhead for space s computations, so the space bounds do not depend on the choice of the model up to logarithmic additive terms (this precision is much better than for time bounds).

We need to specify also how the machine gets the input string (strings) and how it produces the output string. If input/output is written on the tape, then the space used by the computation cannot be less than the input/output length. To avoid this artificial restriction, one usually assumes that separate tapes are used for input and output, and make them read- and write-only (so they cannot be used for computations). If we switch from this model to the worktape-only model, we get, in addition to $O(\log s)$, also $O(\text{input/output size})$ space overhead.

For technical reasons, in this paper we use a specific and a bit unusual computation model (finite-state automaton plus two stacks, see below). The results obtained for this model remain valid for multitape Turing machines, but sometimes in a slightly weaker form, namely, with additional $O(\log s)$ terms that appear when we switch from one model to another.[4]

Our machines have

- one-sided one-directional read-only input tapes with end markers[5] (one or two tapes, depending on the number of inputs);
- one-sided one-directional write-only output tape;

[2]Unfortunately, Kolmogorov did not publish those "mathematical statements" about resource-bounded complexity (though he gave some talks on this topic), and his ideas about algorithmic statistics, as the subject is known now, were understood only much later. It turned out that the dependence of $K^t(x)$ on t (if the resource bound t is measured in "busy beaver units") gives, for every string x, some curve that can be equivalently defined in terms of Kolmogorov structure function, (α, β)-stochasticity, or two-part descriptions. See [16, 17] for a survey of algorithmic statistics.

[3]Kolmogorov used the notation $K(x)$ for complexity function; now it is usually denoted by $\mathrm{C}(x)$, while the notation $\mathrm{K}(x)$ is used for the so-called "prefix complexity". We follow this convention.

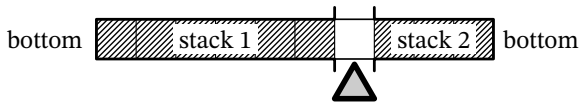
[4]Strangely enough, it seems that this specific model is essential in our proofs and we do not know how to avoid it even if we agree to have additional $O(\log s)$ terms in our results.

[5]For the program tape, the use of the end markers means that we consider plain complexity, not the prefix one (that requires that the interpreter finds by itself where the program ends). However, we allow logarithmic terms in our inequalities for complexities, so the difference between plain and prefix complexity is not important for us.

- two binary stacks (with PUSH/POP/EMPTY requests) as memory devices.

All these devices are connected to the finite-state control unit. The tape alphabet is binary, and the sum of the stack lengths is considered as the space measure.

Note that two stacks are equivalent to a finite tape that can be extended (by inserting an empty cell) or contracted (by deleting a cell) near the head of the Turing machine; the stacks correspond to the parts of the tape on the left and on the right of the head. The head of such a machine knows whether it is at the first/last cell of the tape, and can insert an empty cell or delete a cell before/after the current one.[6]



Definition 1. Let I be a machine of the described type with two one-directional read-only input tapes, one one-directional write-only output tape and two binary stacks. We say that $I^s(p,x) = y$ if machine I with inputs p and x produces y and the total length of the two stacks never exceeds s during the computation. We define the conditional space-bounded Kolmogorov complexity as

$$\mathrm{C}^s_I(y|x) = \min\{|p| : I^s(p,x) = y\}.$$

The unconditional version is obtained when the condition x is the empty string.

Then, as usual, we need a version of the Kolmogorov–Solomonoff universality theorem that says that there exists an optimal machine making the complexity minimal. Now the space bounds should be taken into account, and for our model $O(1)$ additional space is enough:

Proposition 1. *There exists an optimal machine V such that for every machine P there exists a constant c such that*

$$\mathrm{C}^{s+c}_V(y|x) \leqslant \mathrm{C}^s_P(y|x) + c$$

for all x, y.

Both machines P and V are of the type we described (we consider only machines of this type if not stated otherwise). We use the same c both for the space overhead and the complexity increase, but this obviously does not matter.

Proof. Recall a usual construction of a universal machine that can simulate the behavior of an arbitrary machine M_r (an arbitrary finite-state program for the control unit) given its description r, for arbitrary inputs p, x. Here r is a binary string that describes some machine M_r.

We modify this construction to get the required optimal machine V. Let us double each bit in r and denote the result by $\bar{r}$. The machine V is defined in such a way that

$$V(\bar{r}01p,x) = M_r(p,x).$$

This guarantees that if r is the description of machine P (i.e., $M_r = P$), then

$$\mathrm{C}_V(y|x) \leqslant \mathrm{C}_P(y|x) + 2|r| + 2,$$

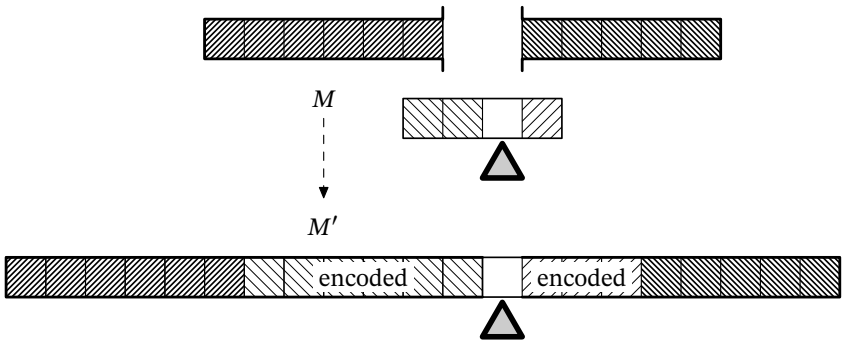
[6] To simulate such a tape (with insertions and deletions) on a standard tape, a $O(\log s)$ space overhead is needed: when inserting a cell, we need to move information along the tape to make space for the new cell, and for that we need $O(\log s)$ additional space for counters (or a special marker symbol that makes the alphabet bigger, so the overhead is even worse).

since for every P-program p we have an equivalent V-program $\bar{r}01p$. Therefore, the complexity increase when switching from P to V is $O(1)$. This would be enough if we did not care about the space bounds. But now we need to describe in more details what the simulating machine V does, and check that V uses only $O(1)$ additional space compared to M_r, where $O(1)$-constant may depend on r but not on p and x.

We start with a general remark about our computational model. Let us add an auxiliary tape (a finite read-write tape with insertions/deletions, as explained above) to the two-stack machine we described.[7]

Lemma 1. *Every machine M of this enhanced type can be simulated by a two-stack machine M' in such a way that at every moment of the computation the space used by M' is bounded by $s_1 + O(s_2)$, where s_1 is the total length of stacks of M, and s_2 is the number of cells on the tape at the same moment.*

Proof of Lemma 1. Let us encode the contents of M's tape in some way (discussed later), and put this encoding between the contents of two stacks of M (reproduced literally, without any encoding).



This will be the contents of the (insertable/deletable) tape of M', and by a *special zone* of this tape we mean the part occupied by the encoding.

The head of M' is always in the $O(1)$-neighborhood of the special zone. When M performs an operation on its stack (left or right), M' moves its head to the corresponding endpoint of the special zone and simulates the required operation. When M performs an operation on the tape, M' finds the place in the special zone that corresponds to the head position, and simulates the required operation.

We need an encoding that makes all these operations possible. For example, we may encode each bit on the tape of M by a group of three identical bits (000 or 111) on the tape of M'. Then we use three other 3-bit blocks (out of 6 remaining) as left and right endmarkers for the special zone, and as a marker that indicates the position of the M-head. The alignment information (position of the M'-head modulo 3) is kept in the finite memory of M'. Then M' can distinguish the markers from the encoded bits and find the place it needs (the endpoint or the M-head position).

The space bound for the simulation[8] is easy to check: M's stacks are copied without any overhead, and each bit on M's tape uses $O(1)$ bits in the encoding. We also use $O(1)$ bits for three markers, but this term is absorbed by $O(s_2)$. □

[7] This is equivalent to adding two more stacks, so we get a machine with four stacks of the same kind. Still in the following lemma the two new stacks are treated differently. Namely, their length is taken into account with some constant factor, so it is more convenient to speak about two stacks and one tape, even if this tape is equivalent to two other stacks.

[8] In fact, we may use better encoding and replace $O(s_2)$ by $s_2 + O(\log s_2)$, but this is not needed for our purposes.

Now we describe machine V that uses an additional tape (and then apply Lemma 1). The machine V starts by reading $\bar{r}$ and writing r on its tape, then it skips the separator 01 and leaves p on the input tape (while x is kept unchanged on the other input tape). Then V executes the program r written on its tape, reading p and x and manipulating the stacks according to r's instructions. For that V needs some additional space on the tape to keep the current state of the simulated program and other information. This space depends only on r but not on p and x. It remains to apply Lemma 1 to construct an equivalent machine with two stacks; the term $O(s_2)$ appearing in this Lemma depends only on r as required. □

Remark 1. Note that in this argument we used that the input tape is one-directional. Still the result remains valid if we write the input on a bidirectional read-only tape with two endmarkers. In this case we need to distinguish during the simulation whether the input head is inside p or not, but this can be checked by going left by $O(|r|)$ cells and coming back. Note that we have r on the work tape and there is enough space to keep the numbers of size $O(|r|)$.

The other non-standard feature of our model is that it uses two stacks instead of a normal tape. But this feature is not important. We can adapt the argument to standard Turing machines: since the size of the self-delimited block does not change during the simulation, this block can be moved along the normal tape (no cell insertions) without moving the information outside the block (this would happen if the block changed its size on a normal tape).

Remark 2. The same construction works for time bounds (instead of space bounds), but we would get a constant factor instead of an additive constant:

$$\mathrm{C}_V^{c\cdot t}(y|x) \leqslant \mathrm{C}_P^t(y|x) + c,$$

where C_V^t stands for the time-bounded complexity (defined in a similar way). Indeed, each step of a simulated machine now requires several steps of the simulating machine, and the number of these steps is bounded by a constant that depends only on r, but not on p and x.

Now we fix some optimal machine V, call the corresponding function $\mathrm{C}_V^s(y|x)$ the *space-bounded Kolmogorov conditional complexity function* and denote it by $\mathrm{C}^s(y|x)$. The unconditional space-bounded complexity $\mathrm{C}^s(x)$ can be defined then as $\mathrm{C}^s(x|\varepsilon)$ for the empty condition ε. It is easy to see that we get an equivalent definition of unconditional complexity if we consider machines V that use only one input tape. Proposition 1 guarantees that these notions are invariant (do not depend on the choice of the optimal machine) up to $O(1)$ changes in the complexity and the space bound.

2 Space-bounded complexity of pairs

The Kolmogorov–Levin theorem (formula for the complexity of pairs, [7]) says that

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

Here $\mathrm{C}(x,y)$ is the complexity of a pair of strings that is defined as the complexity of some computable encoding for it. For the unbounded complexity the choice of encoding is not important, since any computable transformation changes the unbounded complexity only by an $O(1)$ additive term. For the space-bounded version this is no longer the case, and we define the complexity $\mathrm{C}^s(x,y)$ as $\mathrm{C}^s(\bar{x}01y)$, where $\bar{x}$ is x with doubled bits. This encoding of a pair (x,y) treats x and y in different ways, so the natural question is whether the pair complexity as defined above is reasonably robust, e.g., does not change too much when we exchange x and y. The following proposition answers this question; note that the space overhead is no more a constant, but is proportional to the size of x and y.

Proposition 2.

$$\mathrm{C}^{s+O(|x|+|y|)}(y,x) \leqslant \mathrm{C}^s(x,y) + O(1)$$

for all s, x, y.

As usual, this means that there exists some c such that $\mathrm{C}^{s+c(|x|+|y|+1)}(y,x)$ is bounded by $\mathrm{C}^s(x,y)+c$ for all s, x, and y. We add here 1 to take care for the special case $|x| = |y| = 0$. In this case $x = y$ and the statement is vacuous, but in other statements it could be important. We agree that everywhere the $O(\ldots)$ notation allows $O(1)$ terms, too.

Proof. As for the unbounded case, we consider an optimal machine $V(p)$, and then transform it into a machine $\hat{V}$ that exchanges the pair elements in $V(p)$: if $V(p) = \bar{x}01y$, then $\hat{V}(p) = \bar{y}01x$. Then we apply Proposition 1 to the machine $\hat{V}$. The only thing we need is to be sure that the $\hat{V}(p)$ computation can be performed in space $s + O(|x| + |y|)$, if $V(p)$ produces $\bar{x}01y$ in space s.

Again we use Lemma 1 and equip the machine $\hat{V}$ with an additional tape; we need only to remember that the space used on this tape is counted with some constant factor. Instead of writing the bits of $\bar{x}$ on the output tape like V does, the machine $\hat{V}$ writes them (or just bits of x without duplication) on the work tape, until the next two-bit block is 01. After that all output bits of V (i.e., bits of y) are doubled, so $\bar{y}$ is printed on the output tape. When V terminates, $\hat{V}$ prints 01 and after that copies the bits of x from the tape.

It remains to note that our transformation does not change the content of the stacks, and the additional space on the tape is $O(|x| + |y|)$ — in fact, even $O(|x|)$, since we do not need to store y. □

Remark 3. A similar argument for standard Turing machines (no insertion of cells allowed) would give additional $O(\log s)$ overhead.

Now the complexity of pairs is defined, and we would like to develop a space-bounded version of Kolmogorov–Levin formula for the complexity of pairs. This formula says that

$$\mathrm{C}(x,y) = \mathrm{C}(x) + \mathrm{C}(y|x) + O(\log n),$$

if x and y are strings of length at most n, and contains two inequalities, one in each direction. We want to provide the space-bounded counterparts for them. In one direction this is easy to do. One can get even a bit stronger bound that has term $O(\log \mathrm{C}^s(x))$ instead of $O(\log n)$; note that for the reasonable values of s we have $\mathrm{C}^s(x) = O(|x|)$.

Proposition 3.

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

Note the general structure of this statement: we consider an arbitrary bound s on the right-hand side, and on the left side we have to use a slightly bigger bound (for our model $s + O(|x| + |y|)$ is enough).

Proof. Let p and q be the minimal programs for x and for $x \mapsto y$. We need to construct the program for the pair (x,y), i.e., for the string $\bar{x}01y$, whose length will be bounded by $|p| + |q| + O(\log |p|)$. This program will work for some other decompressor $\hat{V}$, and then we use universality to replace $\hat{V}$ by V.

The program (description) for the pair (x,y) can be constructed as $\bar{l}01pq$ where l is the length of p, written in binary. The decoding machine $\hat{V}$ again uses an auxiliary tape (and then we use Lemma 1). First the machine $\hat{V}$ copies $\bar{l}01$ to the auxiliary tape. After that the machine $\hat{V}$ reads and stores p on the tape. Note that the length of p (i.e., l) is already on the tape, so $\hat{V}$ knows

when to stop reading p. Then $\hat{V}$ simulates the optimal unconditional decompressor on p, reading the bits of p from the auxiliary tape and storing the output bits (i.e., bits of x) also on the auxiliary tape. Now $\hat{V}$ is ready to simulate the computation of the optimal conditional decompressor on q, reading the bits of q from the input tape (the rest of the input) and using stored bits of x instead of input bits from its second tape. It is easy to see that we need $O(|p|+|q|+|x|)$ cells on the tape (in fact, $O(|p|+|x|)$ cells).

It is not all we need: there is a technical problem. Namely, for small s we cannot guarantee that $|p|+|q| \leqslant O(|x|+|y|)$. So, the space overhead $O(|p|+|q|+|x|))$ may not be $O(|x|+|y|)$. However, if $|p|+|q|$ significantly exceeds $|x|+|y|$, then we may use the inequality $\mathrm{C}^{O(|x|+|y|)}(x,y) \leqslant |x|+|y|+O(\log|x|)$ instead. The latter inequality is obtained if we use x and y instead of p and q in the construction above, and use trivial decompressors instead of the optimal ones. □

Remark 4. In the right hand side of Proposition 3 we may replace $O(\log \mathrm{C}^s(x))$ by $O(\log \mathrm{C}^s(x,y))$. This is not immediately obvious, because we cannot bound $\mathrm{C}^s(x)$ by $\mathrm{C}^s(x,y)$ with exactly the same s. But for the "paradoxical" case $\mathrm{C}^s(x,y) < \mathrm{C}^s(x)$ the entire inequality is obviously true.

The other direction of the Kolmogorov–Levin formula is more difficult (both for unbounded and space-bounded complexity).

Theorem 1. *For all strings x,y and for every number s we have*

$$\mathrm{C}^{s'}(x)+\mathrm{C}^{s'}(y|x) \leqslant \mathrm{C}^s(x,y)+O(\log \mathrm{C}^s(x,y)),$$

where $s' = s+O(|x|+|y|)$.

Here we use the notation s' for the space bound on the left-hand side to avoid repetitions. The exact meaning of this statement: there exists a constant c such that for all x,y and for every s we have $\mathrm{C}^{s'}(x)+\mathrm{C}^{s'}(y|x) \leqslant \mathrm{C}^s(x,y)+c\log \mathrm{C}^s(x,y)$, where $s' = s+c|x|+c|y|+c$.

This bound assumes that we use the computational model with two stacks; for ordinary Turing machines an additional $O(\log s)$ term is needed in the expression for s'.

Longpré [8, Theorem 3.13, p. 35] proved essentially[9] the same result with $2s+O(\log s)$ instead of s; in his paper he uses $3s$, but his argument gives $2s+O(\log s)$ without changes. We improve this bound using Sipser's technique from [15] with some additional refinements.

Proof. The proof is obtained by a modification of Longpré's argument which in its turn is a modification of the standard proof of the Kolmogorov–Levin formula. So we first recall the standard argument, then explain the modifications used by Longpré, and then prove the final result.

Recalling the standard argument without resource bounds The standard argument (for the complexities without resource bounds) goes as follows. Let x and y be two strings of length at most n and let m be the complexity of the pair: $\mathrm{C}(x,y) = m$. We may always assume that $m = O(n)$: for the unbounded case it is always true, since we have two strings of length at most n and their pair has complexity at most $2n+O(\log n)$. For the bounded case and very small s the complexity $\mathrm{C}^s(x,y)$ may be larger than $2n+O(\log n)$ (since even the trivial program for the pair still requires some space to run), but then the inequality is obviously true for $s' = O(n)$, since both terms on the left-hand side are bounded by $n+O(1)$ for this value of s'.

[9] His setting is slightly different: for him s is not a numerical parameter, but a function of the input size, so the exact comparison is difficult.

Consider the set S_m of all pairs $\langle x', y' \rangle$ such that $\mathrm{C}(x', y') \leqslant m$. This set can be enumerated by an algorithm (given m), and there are $O(2^m)$ of them. Our pair $\langle x, y \rangle$ is an element of this set. Count the pairs $\langle x, y' \rangle$ in this set *that have the same first coordinate* (i.e., the first coordinate x). Assume that we have about 2^k of them for some k. We may choose k in such a way that the number of those pairs is between 2^k and 2^{k+1}. Now we make two observations:

- Knowing x, we can filter the pairs in the enumeration of S_m and keep only the pairs with the first coordinate x, looking at their second coordinate. This process enumerates at most 2^{k+1} strings, and y is one of them. The string y can be reconstructed if we know x, m and the ordinal number of y in this enumeration (this requires $k + O(1)$ bits of information). In total we get $O(\log m) + k$ bits (we need to separate m and the ordinal number, and this involves some separation overhead, but this overhead can be absorbed by $O(\log m)$: we may repeat each bit of m twice and add 01 at the end). Therefore, $\mathrm{C}(y|x) \leqslant k + O(\log m)$.
- On the other hand, we can enumerate all x' such that there are at least 2^k different y' such that $\mathrm{C}(x', y') \leqslant m$; there are at most $O(2^{m-k})$ of them, since each of them produces at least 2^k pairs and the total number of pairs is $O(2^m)$. The string x appears in this enumeration. So we can specify x by the ordinal number in the enumeration ($m - k + O(1)$ bits), in addition to the values of m and k needed for the enumeration. The total number of bits is $O(\log m) + m - k$, therefore $\mathrm{C}(x) \leqslant m - k + O(\log m)$. (Note that $k \leqslant m$, so k also has a self-delimited encoding of size $O(\log m)$.)

Combining the bounds for $\mathrm{C}(x)$ and $\mathrm{C}(y|x)$ and recalling that $m = O(n)$, we get the desired result.

How to obtain a weak space bound (following Longpré) The argument for the unbounded case (as presented above) does not work as is for the space-bounded complexity. The problem is that the enumeration used in this argument needs a lot of space, since the lists of enumerated objects are exponential in m. However, another approach is possible. Recall that x and y are strings of length at most n. There are at most $O(2^{2n})$ pairs $\langle x, y \rangle$ of strings of length at most n. We may consider them in some fixed order (e.g., in the lexicographical one), and compute $\mathrm{C}^s(x, y)$ for each pair. As we have discussed, the function C^s is computable, and the following lemma shows that we do not need too much space to compute it.

Lemma 2. *The complexity* $\mathrm{C}^s(x)$ *can be computed* (*given* s *and* x *such that* $s \geqslant \Omega(|x|)$) *in space* $2s + O(\log s) + O(|x|)$.

This is a weak version of this lemma (that gives Longpré's result). We will see later that one can replace $2s + O(\log s)$ by s, and this will allow us to finish the proof of Theorem 1, but we start with a simpler bound.

Proof of Lemma 2. We know that $\mathrm{C}^{O(|x|)}(x) \leqslant |x| + O(1)$, and our assumption $s = \Omega(|x|)$ guarantees that $\mathrm{C}^s(x) \leqslant |x| + O(1)$. So it is enough to try all the programs of length at most $|x| + O(1)$ to see which of them produce x with space bound s (in the order of increasing length, so the first one found will be the shortest one). To keep track of the current program, we need $O(|x|)$ space. To simulate the program and to keep track of the space used by it, we need additional $s + O(\log |s|)$ space. The only problem is that the program that we try may never terminate. To detect these cases, we may use a counter for the number of steps. Since a machine with space bound s has at most $2^{s+O(\log s)}$ configurations, if the number of steps exceeds this $2^{s+O(\log s)}$ bound, some configuration appears twice and the program is in the infinite loop. To detect this

loop, we use a counter of size $s + O(\log s)$. In total we need $2s + O(\log s) + O(|x|)$ space to find the complexity, as claimed. □

Now the proof goes as before. We consider the set $S^s_{m,n}$ of all pairs $\langle x', y' \rangle$ such that $|x'| \leqslant n$, $|y'| \leqslant n$ and $\mathrm{C}^s(x', y') \leqslant m$. The pair $\langle x, y \rangle$ is one of its elements. Choose k in such a way that the number of pairs $\langle x, y' \rangle$ in this set (with the first coordinate x) is between 2^k and 2^{k+1}. Then

- $\mathrm{C}^{2s+O(\log s)+O(n)}(y|x) \leqslant k + O(\log s) + O(\log n)$;
- $\mathrm{C}^{2s+O(\log s)+O(n)}(x) \leqslant m - k + O(\log s) + O(\log n)$.

Indeed, y can be reconstructed if we know the ordinal number of y in the enumeration of all y' such that $\langle x, y' \rangle \in S^s_{m,n}$, and this set can be enumerated (in the lexicographical order) when x, m, n and s are known. Lemma 2 guarantees that this can be done in space $2s + O(\log s) + O(n)$; recall also that $m = O(n)$ according to our assumption. On the other hand, x can be enumerated together with the other $O(2^{m-k})$ strings x' of length at most n such that there are at least 2^k strings y' of length at most n with $\mathrm{C}^s(x', y') \leqslant m$. We can check whether x' has the required property trying all y' sequentially and counting them in $O(n)$ space. The ordinal number of x in the enumeration requires $m - k$ bits, all other parameters require $O(\log n) + O(\log s)$ bits, and the space used in the process is still $2s + O(\log s) + O(n)$.

Combining these two inequalities, we get

$$\mathrm{C}^{s'}(y|x) + \mathrm{C}^{s'}(x) \leqslant \mathrm{C}^s(x, y) + O(\log s) + O(\log n),$$

where $s' = 2s + O(\log s) + O(n)$.

This result is weaker than the claim we need to prove in three aspects. First, we need to replace $2s + O(\log s)$ by s in the expression for s'. Second, we proved the inequality with $O(\log s)$ in the right hand side that should not be there. Note that this term makes the statement vacuous if s is exponential in n, and does not allow us to get the unbounded Kolmogorov–Levin theorem as a corollary of the bounded version when $s \to \infty$. Finally, we would like to replace n (the length of the strings) in the last term $O(\log n)$ by the complexity of the pair, so the last term would be $O(\log \mathrm{C}^s(x, y))$.

How to eliminate factor 2 **(following Sipser)** First let us explain how the factor 2 can be avoided using the following result that goes back to [15]:

Proposition 4 (Sipser)**.** *Let M be a machine. Then there is a machine $\overline{M}$ that decides, given a string x and number s, whether M terminates on input x in space s or not. Machine $\overline{M}$ uses at most $s + O(|x|)$ space working on pair x, s.*

In this statement we assume that the two-stack computation model is used; as a corollary, we get the same result with the additional term $O(\log s)$ for other standard models, e.g., multitape Turing machines.

Proof. We start by proving a weaker statement with a looser space bound $s + O(\log s) + O(|x|)$. For this bound, there is no problem with keeping x, the number of input bits already read by M, and the binary representation of s in the memory.

We may assume without loss of generality that machine M clears its stacks when terminates, and reads its input completely. For that the old final state is transformed into a cleaning state that pops elements until both stacks are empty, and reads the input until its end.

Let us consider all configurations of M that use space at most s. We include in the configuration the contents of the stacks, the state of the machine, and the position of the input head on x

(the input string for which we want to check the termination). We may ignore the output tape: it is write-only, so operations with the output tape do not affect termination. These configurations are considered as vertices of a directed graph. Namely, for every vertex (configuration) v of that kind, draw an edge that goes from v to the next configuration (after one computation step is performed), or no outgoing edges if v is final or if the next computation step violates the space bound. According to our assumption, the final configuration is unique. Let us denote it by f. We need to check whether a (unique) path starting from the initial configuration gets into f.

The graph may have cycles (the machine may go into a loop). However, the connected component of the final configuration, i.e., the set of vertices v such that there is a path from v to f, is a tree where edges go from a vertex to its parent. Indeed, the outgoing path is unique (the machine is deterministic), so the vertices of any loop cannot have a path to v. The root of this tree is f. The termination question can now be reformulated as follows: is the initial configuration in the tree?

To answer this question, one can traverse the tree using depth-first search. Note that the standard (non-recursive) algorithm for this (see, e.g., the textbook [13, Chapter 3]) does not use any additional memory, and the basic operations can be performed with $O(1)$ space. More precisely, let us order siblings (sons of the same parent) arbitrarily (but consistently). This induces a natural ordering on the leaves. We can traverse the tree, visiting all the leaves in this order. In this process we make three types of moves: from a vertex to (a) its first child, (b) its parent and (c) its next sibling (in the chosen order). All non-leaf vertices are visited twice: first on the path from the root to leaves, the second time on the way back to the root. The tree-traversing algorithm at every step keeps the current vertex and one bit that says whether we are on the way to the leaves or back. The basic operation of the tree-traversing algorithm are the following:

- *Checking whether the given vertex v has children, and if yes, finding the first child of v.* In our case this means that the current configuration can be obtained from some other configuration; if yes, we should find the first among those predecessor configurations (children).

- *Checking whether the given vertex v is the last sibling in the ordering we have on v's siblings; finding the next sibling of v if it exists.* In our case we should consider all the configurations that have the same successor, and find the next one in the chosen ordering (if our configuration is not the last one).

- *Checking whether the given vertex v is the root, and finding the parent of v if v is not the root.* In our terms it means that we have to check whether the configuration is final, and find the successor configuration if it is not.

We also need to keep track of the configuration size (since we do not consider configurations that require more than s space), but this can be done in $O(\log s)$ memory. All other checks are local (require $O(1)$ additional memory), since only the immediate neighborhood of the head ($O(1)$ top elements of the stacks) needs to be taken into account.

We need also to keep track of the position of the input head in x (and keep x in the memory), but this is easy to do with $O(|x|)$ overhead. This finishes the argument for $s + O(\log s) + O(|x|)$ bound.

To get rid of $O(\log s)$ in this bound (as promised), we need additional (and rather strange) tricks. The machine M has two stacks, as well as the machine $\overline{M}$ that we need to construct. However, it is convenient to use Lemma 1 and add an auxiliary tape to $\overline{M}$; the space used on this tape is taken into account with some constant factor.

We keep x (and the input head position in x) on the auxiliary tape; this requires $O(|x|)$ space and is not a problem. We use the stacks of $\overline{M}$ to keep (literally) the contents of M's stacks in the current position (i.e., the current vertex considered by the tree traversal algorithm). The basic operations listed above are local and do not require memory (except for x and the input position, already taken into account). However, we need to check whether the modified position of M still uses space at most s, i.e., that the total size of two stacks still does not exceeds s after possible increase in the stack sizes. Before, having $O(\log s)$ additional space, we could keep the value of s and the current lengths of stacks, and make these checks. What can we do now? The following idea helps: let us remember s all the time, but in an indirect way: we keep on the auxiliary tape the *difference between s and the total length of two stacks* (of M or M', they are the same). This difference is enough to check whether the possible neighbor in the tree is valid (has total stack length at most s). When the total length approaches s, the difference counter is small and requires only $O(1)$ bits. When stacks are short, the counter is big and may require $O(\log s)$ bits — but since we measure the total length of the stacks and the tape, these $O(\log s)$ additional bits are not a problem (it is easy to see that $k + O(\log(s-k)) \leqslant s + O(1)$ for all $k < s$). This finishes the proof of the Proposition 4 (in its strong form, without $O(\log s)$ term). □

Sipser's trick allows us to prove the following stronger version of Lemma 2:

Lemma 3. *The complexity* $\mathrm{C}^s(x)$ *can be computed* (*given s and x such that* $s \geqslant \Omega(|x|)$) *in space* $s + O(|x|)$.

Proof. In the proof of Lemma 2 we need to keep s and test all the possible programs to check whether they produce x within space bound s. For that, we first check that a program terminates in space s using Proposition 4, and if yes, apply the interpreter to the program (now being sure that we do not violate the space bound) and compare the output with x. Again, we can keep s indirectly during both phases, by keeping the difference between s and the total length of the stacks. Then, if this length comes close to s, the counter is small, and when the stacks are short, we may use a lot of space for the counter. □

This immediately gives us a better bound: the inequality

$$\mathrm{C}^{s'}(y|x) + \mathrm{C}^{s'}(x) \leqslant \mathrm{C}^s(x,y) + O(\log s) + O(\log n)$$

is now proven for $s' = s + O(n)$, (now we have s instead of $2s + O(\log s)$). The $O(\log s)$ additive term *in the right hand side* is still there. It was used to remember the space bound s, so the configurations of size greater than s could be discarded. Still we can avoid this $O(\log s)$ term if we change the enumeration order.

Eliminating $O(\log s)$ term in the right hand side Instead of enumerating for some fixed s all pairs $\langle x', y' \rangle$ with $|x'|, |y'| \leqslant n$ such that $\mathrm{C}^s(x', y') \leqslant m$, we enumerate the pairs such that $\mathrm{C}^u(x', y') \leqslant m$ (and $|x'|, |y'| \leqslant n$) sequentially for $u = 1, 2, 3, \ldots$ So every pair with $|x'|, |y'| \leqslant n$ and (unbounded) complexity $\mathrm{C}(x', y') \leqslant m$ will be enumerated at some stage, but the space bound (and also the amount of space used for the enumeration) increases with time. Using some additional precautions, we may guarantee that this enumeration will be without repetitions (no pair is enumerated twice). Indeed, after we find some x', y' with $\mathrm{C}^u(x', y') \leqslant m$ for the current u, we check whether the same is true for bound $u-1$, and if yes, skip the pair. Note that it can be done without increasing the space usage, since we may reuse the same space for both bounds u and $u-1$.

In other words, we enumerate all the pairs $\langle x', y' \rangle$ with $|x'|, |y'| \leqslant n$ and $\mathrm{C}(x', y') \leqslant m$ in the following order: we compare the minimal space u needed to establish the inequality $\mathrm{C}^u(x', y') \leqslant$

m, and for the same u we use some standard ordering on pairs. In this way every pair is enumerated only once without the need to keep the list of the pairs already enumerated.

Now we choose the value of k, like we did in the in the proof of the Kolmogorov–Levin formula for unbounded complexity. For that we consider the pairs such that $\mathrm{C}^s(x,y') \leqslant m$ for given x and arbitrary y' (such that $|y'| \leqslant n$) *for the given value of* s. There exists some k such that the number of these pairs is between 2^k and 2^{k+1}.

Let us check that $\mathrm{C}^{s'}(y|x) \leqslant k + O(\log n)$ for $s' = s + O(n)$. Knowing n and m, we can perform the enumeration described above; knowing x, we can restrict the enumeration to pairs $\langle x,y'\rangle$ with the first component x. The pair $\langle x,y\rangle$ is among them; moreover, we know that its ordinal number in this restricted enumeration is at most 2^{k+1}, so we need $k+O(1)$ bits to specify this number (in addition to n, m and x). Performing the enumeration until that many pairs with first component x appear, we use only $s+O(n)$ bits, since we stop the enumeration after the required number of pairs are found. This gives the inequality we wanted (recall that $m = O(n)$, so we can specify m and n by $O(\log n)$ bits).

Now we need to show that $\mathrm{C}^{s'}(x) \leqslant m - k + O(\log n)$ for the same value of s'. For that we enumerate elements x' that have large "vertical sections" (have many y' such that $\mathrm{C}^s(x',y') \leqslant m$). Again we do it sequentially for $u = 1,2,3,\ldots$ For each u we run a loop over all x' with $|x'| \leqslant n$. For each of them we count all y' such that $|y'| \leqslant n$ and $\mathrm{C}^u(x',y') \leqslant m$. This is done sequentially (and we reuse the space at every step). If there are more than 2^k different strings y' found, we include x' in the enumeration of the elements that have large vertical sections. To avoid repetitions, we use the same trick: we check whether the size of the vertical section was not large enough for the previous value of u, repeating all the computations with this value. In this way we enumerate all x' that have large sections for unbounded complexity, using more and more space in the process. Note that we keep the current value of u all the time, but indirectly, as a combination of current stacks' length and the counter (and the counter is short when the space is tight).

This enumeration will include our x at some stage $u \leqslant s$. The ordinal number of x in the enumeration is at most $2^{m-k+O(1)}$ for the same reason as before (for complexities without space bounds). At this stage the space used by the computation is $s+O(n)$, and we stop the enumeration after a required number of strings are enumerated. To start the enumeration we need to know n, m and k, all three can be specified by $O(\log n)$ bits, in total we get $m-k+O(\log n)$ bits. We see that $\mathrm{C}^{s'}(x) \leqslant m-k+O(\log n)$ and may combine this inequality with the bound for $\mathrm{C}^{s'}(y|x)$ obtained earlier, thus eliminating the term $O(\log s)$ in the right hand side as promised.

Replacing $O(\log n)$ by $O(\log \mathrm{C}^s(x,y))$ We have proven the inequality

$$\mathrm{C}^{s'}(x) + \mathrm{C}^{s'}(y|x) \leqslant \mathrm{C}^s(x,y) + O(\log(|x|+|y|)) \qquad (*)$$

for $s' = s + O(|x|+|y|)$, (in our notation n was the maximal of the lengths of $|x|$ and $|y|$). The last step is to replace $|x|+|y|$ in the right hand side by $\mathrm{C}^s(x,y)$. This means that in our argument we do not have n as a parameter of the enumeration process and may use only m.

The idea is to replace the strings by their shortest programs. For Kolmogorov complexity with unbounded resources, a string x is "interchangeable" with one of its shortest programs p in the following sense:

$$\mathrm{C}(x|p) = O(\log m) \quad \text{and} \quad \mathrm{C}(p|x) = O(\log m),$$

where $m = \mathrm{C}(x)$. The first part is obvious for *every* program p (even with $O(1)$ instead of $O(\log m)$): we apply the optimal interpreter to p and obtain x. The second part is also pretty

simple: given x and m, we run the optimal interpreter on all programs of length m in parallel and take the first one that produces x.

A similar property is true for the space bounded Kolmogorov complexity. If $\mathrm{C}^s(x) = m$, then there is a program p of length m such that

$$\mathrm{C}^{s+O(1)}(x|p) = O(\log m) \quad \text{and} \quad \mathrm{C}^{s+O(|x|)}(p|x) = O(\log m).$$

This p is one of the programs of length m that produce x in space s. For such a program p the first part is trivial: we simulate the universal interpreter on p and obtain x, with $O(1)$ space overhead for the simulation. For the second part, we show how to find a program p of length m for x (that works in space s) given x and m. Following the argument we already used, we try all the programs of length m giving them more and more space ($s' = 1, 2, 3, \ldots$) until one of them produces x. For keeping the space overhead in this process small, we keep the value of the current space bound s' indirectly, as a difference between s' and current length of the stacks. We need also to keep x, thus the $O(|x|)$ overhead in the space bound.

Using this property, we replace x and y by some p_x and p_y whose lengths are $O(\mathrm{C}^s(x,y))$. A small technicality is that $\mathrm{C}^s(x)$ may not be bounded by $\mathrm{C}^s(x,y)$, since extracting x from the pair requires some overhead. In fact, $O(1)$ overhead is enough: $\mathrm{C}^{s+O(1)}(x) \leqslant \mathrm{C}^s(x,y)$ (but $O(|x|+|y|)$ overhead would work too). Applying the previous remark to this bound, we find p_x of length at most $\mathrm{C}^s(x,y)$ such that

$$\mathrm{C}^{s+O(1)}(x|p_x) = O(\log \mathrm{C}^s(x,y)) \quad \text{and} \quad \mathrm{C}^{s+O(|x|)}(p_x|x) = O(\log \mathrm{C}^s(x,y)).$$

The same can be done for y to get a replacement string p_y with similar properties. Then we apply the previous form of the inequality (with $O(\log n)$) to p_x and p_y, and note that replacing x by p_x in expressions with conditional or unconditional complexity changes the complexity bound by $O(\log \mathrm{C}^s(x,y))$ and the overhead by $O(|x|+|y|)$.

This finishes the proof of Theorem 1. □

3 Basic inequalities: space-bounded version

We have defined space-bounded complexity for pairs. In the same way (and with the same precision) one can define the complexity of triples, and, in general, m-tuples for every fixed m. In the section we prove space-bounded versions of the so-called *basic inequalities* for Kolmogorov complexity.

The basic inequality involves complexities of triples and says (in the unbounded version) that

$$\mathrm{C}(x) + \mathrm{C}(x,y,z) \leqslant \mathrm{C}(x,y) + \mathrm{C}(x,z) + O(\log n)$$

if x, y, z are strings of length at most n. Usually it is proved by considering conditional complexities:

$$\begin{aligned}
\mathrm{C}(x,y) &= \mathrm{C}(x) + \mathrm{C}(y|x) + O(\log n),\\
\mathrm{C}(x,z) &= \mathrm{C}(x) + \mathrm{C}(z|x) + O(\log n),\\
\mathrm{C}(x,y,z) &= \mathrm{C}(x) + \mathrm{C}(y,z|x) + O(\log n).
\end{aligned}$$

Using these equalities, we rewrite the inequality as

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

and this is a relativized version of the inequality for the complexity of pairs:

$$\mathrm{C}(y,z) = \mathrm{C}(y) + \mathrm{C}(z|y) + O(\log n) \leqslant \mathrm{C}(y) + \mathrm{C}(z) + O(\log n);$$

adding y as a condition may only decrease the complexity of z. In computability theory relativization is usually understood as adding an oracle access to some set to all the computations; almost all results of general computability theory remain valid after relativization. In algorithmic information theory a slightly different notion of relativization is also used: instead of adding a set as an oracle, we add some string as a condition in all the complexity expressions. Almost all results (and their proofs) remain valid after that.[10]

Let us do this in more detail to see how the space-bounded version can be proven. We have

$$\mathrm{C}^{s'}(x) + \mathrm{C}^{s'}(y|x) \leqslant \mathrm{C}^{s}(x,y) + O(\log n),$$
$$\mathrm{C}^{s'}(x) + \mathrm{C}^{s'}(z|x) \leqslant \mathrm{C}^{s}(x,z) + O(\log n),$$

for some s' slightly larger than s (by that we mean that $s' = s + O(n)$). Therefore,

$$2\mathrm{C}^{s'}(x) + \mathrm{C}^{s'}(y|x) + \mathrm{C}^{s'}(z|x) \leqslant \mathrm{C}^{s}(x,y) + \mathrm{C}^{s}(x,z) + O(\log n).$$

From this we conclude that

$$2\mathrm{C}^{s'}(x) + \mathrm{C}^{s''}(y,z|x) \leqslant 2\mathrm{C}^{s'}(x) + \mathrm{C}^{s'}(y|x) + \mathrm{C}^{s'}(z|x) + O(\log n) \leqslant \mathrm{C}^{s}(x,y) + \mathrm{C}^{s}(x,z) + O(\log n),$$

for some s'' slightly exceeding s', using the relativized inequality for the complexity of a pair:

$$\mathrm{C}^{s''}(y,z|x) \leqslant \mathrm{C}^{s'}(y|x) + \mathrm{C}^{s'}(z|x) + O(\log n).$$

Here $s'' = s' + O(n) = s + O(n)$ absorbs the increase $O(n)$ caused by the length of the condition x that is needed for the relativization. Now we recall that

$$\mathrm{C}^{s'''}(x,y,z) \leqslant \mathrm{C}^{s'}(x) + \mathrm{C}^{s''}(y,z|x) + O(\log n)$$

(the easy direction of the Kolmogorov–Levin formula) for some s''' slightly greater than s' and s'', and get

$$\mathrm{C}^{s'}(x) + \mathrm{C}^{s'''}(x,y,z) \leqslant \mathrm{C}^{s}(x,y) + \mathrm{C}^{s}(x,z) + O(\log n).$$

For uniformity we can replace s' by s''' on the left-hand side. Here s''' is the third iteration of adding overhead, so still $s''' = s + O(n)$, and we get the following space-bounded version of basic inequality:

Theorem 2 (Space-bounded basic inequality)**.**

$$\mathrm{C}^{s'}(x) + \mathrm{C}^{s'}(x,y,z) \leqslant \mathrm{C}^{s}(x,y) + \mathrm{C}^{s}(x,z) + O(\log n)$$

for all n, for all strings x,y,z of length at most n, for all numbers s, and for $s' = s + O(n)$.

More general inequalities (called also basic inequalities) are obtained if we replace x,y,z by tuples of strings; they are easy corollaries of Theorem 2 (converting the tuples into their string encoding and vice versa can be done in $O(n)$ space for strings of size at most n).

[10] For the space-bounded complexity additional precautions are needed: if we add x as a condition, it may be necessary to add $O(|x|)$ to the space bound.

4 Shannon inequalities: iterations

Fix some integer $k \geqslant 1$; let $x_1, \ldots, x_k$ be some strings. For each $I \subset \{1, \ldots, k\}$ we consider the tuple x_I made of strings x_i with $i \in I$. In this notation, the basic inequalities mentioned above can be written as

$$\mathrm{C}^{s'}(x_{I\cap J}) + \mathrm{C}^{s'}(x_{I\cup J}) \leqslant \mathrm{C}^{s}(x_I) + \mathrm{C}^{s}(x_J) + O(\log n),$$

if all $x_1, \ldots, x_k$ are strings of length at most n and $s' = s + O(n)$. (The constants in the O-notation may depend on k, I, J, but not on n, $x_1, \ldots, x_k$, s.)

Taking the sum of several basic inequalities (for the same k, but for different I and J), we may get other linear inequalities for the complexities of tuples, i.e., inequalities of the type

$$\sum_{I\subset\{1,\ldots,k\}} \lambda_I \mathrm{C}(x_I) \geqslant 0,$$

where λ_I are some real coefficients. This is a well known procedure for unbounded Kolmogorov complexity [14, Chapter 10]; the resulting linear inequalities are called *Shannon inequalities*. Not all linear inequalities that are true with logarithmic precision are Shannon inequalities (an important discovery made in [18]).

In this section we show that *every Shannon inequality has a space-bounded version*. This space-bounded version is constructed as follows. We start by separating the positive and negative coefficients in the linear inequality. The corresponding groups are denoted by L and R; their elements are subsets of the set $\{1, \ldots, k\}$, and we assume that $L \cap R = \varnothing$. Now the general form of a linear inequality for complexities of tuples is

$$\sum_{I\in L} \lambda_I \mathrm{C}(x_I) \leqslant \sum_{J\in R} \mu_J \mathrm{C}(x_J) \tag{1}$$

where all λ_I and μ_J are non-negative. The following theorem says that each Shannon inequality has a space-bounded counterpart of the same form as for the basic inequalities (but with slightly weaker space bound).

Theorem 3. *Consider a linear inequality of the form* (1) *that is a linear combination of basic inequalities* (*is a Shannon inequality*)*. Then the following space-bounded version of this inequality is true:*

$$\sum_{I\in L} \lambda_I \mathrm{C}^{s'}(x_I) \leqslant \sum_{J\in R} \mu_J \mathrm{C}^{s}(x_J) + O(\log n), \tag{2}$$

if $x_1, \ldots, x_k$ are strings of length at most n, and $s' = s + O(n^2)$.

Here the constants in the O-notation depend on the inequality (more precisely, on k and the coefficients λ_I and μ_j)[11], but neither on n nor on $x_1, \ldots, x_k$. Note that the overhead is worse than for the basic inequalities: we have $O(n^2)$ instead of $O(n)$.

Proof. Consider the basic inequalities whose sum is the inequality (1). For each of them consider the space-bounded version (from Theorem 2). The sum of these space-bounded inequalities does not give (2) directly: the resulting inequality may have terms $\mathrm{C}(x_I)$ with the same I in the left and right hand sides. In other words, we get an inequality of type (1), but the sets L and R are not necessarily disjoint. For the unbounded complexities, these terms just cancel each other (partially or completely), and we get the desired Shannon inequality. Now, when adding the space-bounded versions of the same basic inequality, we get an inequality where the complexity

[11] The constant in the last line depends only on k, as the proof shows.

of the same tuple may appear with the same coefficient on both sides, but with different space bounds. We can rewrite is as

$$\sum_{I\in L}\lambda_I \mathrm{C}^{s'}(x_I)+\sum_{K\in C}\sigma_K \mathrm{C}^{s'}(x_K)\leqslant \sum_{J\in R}\mu_J \mathrm{C}^{s}(x_J)+\sum_{K\in C}\sigma_K \mathrm{C}^{s}(x_K)+O(\log n).$$

Here x_K are tuples that appear on both sides in the terms that are canceled in the unbounded version (partially or completely). Some $K\in C$ may also appear in L or R (the part that is not canceled), but not in both: the sets L and R are disjoint. We would like to cancel the complexities of x_K for $K\in C$, but now the complexities are different. They have bound s' on the left-hand side and s on the right-hand side, and cannot be canceled anymore.

The following trick helps. Let $f(s)=s+O(n)$ be the function from Theorem 2 that transforms the right-hand side bound s to the left-hand side bound s' (here s is a variable, while n and the constants in the $O(\cdot)$-notation are fixed). Consider the sequence of space bounds

$$u_0=s, u_1=f(u_0),\ldots,u_N=f(u_{N-1})$$

for some large N. All tuple complexities can only decrease if we increase the space bound from u_t to u_{t+1}. Therefore, for a large enough N, namely, $N=O(n)$ with a large enough constant, we guarantee the existence of t such that all complexities of tuples are the same with bounds u_t and u_{t+1}. Then we can add the space-bounded inequalities and cancel the common terms as we did for the unbounded version. More precisely, we know that

$$\sum_{I\in L}\lambda_I \mathrm{C}^{u_{t+1}}(x_I)+\sum_{K\in C}\sigma_K \mathrm{C}^{u_{t+1}}(x_K)\leqslant \sum_{J\in R}\mu_J \mathrm{C}^{u_t}(x_J)+\sum_{K\in C}\sigma_K \mathrm{C}^{u_t}(x_K)+O(\log n),$$

and on both sides u_t can be replaced by u_{t+1} due to our assumption. So we can cancel the common terms. We cannot compute t for which there is no change in the complexities, but its existence is guaranteed. Then we can replace the bound on the left-hand side by u_N, and on the right-hand side by $s=u_0$.

It remains to note that for $N=O(n)$ we add the $O(n)$ term $N=O(n)$ times, so the final value after N iterations is $s+O(n^2)$.

In fact, we can improve the bound in Theorem 3. For that we may note that it is not needed to have exactly the same complexities with space bounds u_{t+1} and u_t. It is enough that the difference between them is $O(\log n)$, since we have $O(\log n)$ term in the right hand side anyway. Therefore, $N=n/\log n$ iterations are enough, and in this way we replace $O(n^2)$ by $O(n^2/\log n)$, getting a bit stronger version of Theorem 3. □

Remark 5. This argument relies on the good space bounds in the left hand side of Theorem 1. If we used (instead of Theorem 1) the bound with factor 2, the n-th iteration would give an exponential factor 2^n, so we wouldn't get a polynomial (in n) space bound.[12]

Remark 6. It may happen that for some Shannon inequality the cancellation problem does not arise. This indeed happens for some natural Shannon inequalities, e.g., for

$$2\,\mathrm{C}^{s'}(A,B,C)\leqslant \mathrm{C}^{s}(A,B)+\mathrm{C}^{s}(A,C)+\mathrm{C}^{s}(B,C)+O(\log n),$$

that is therefore true for $s'=s+O(n)$. However, it is not clear whether this can be done for arbitrary Shannon inequalities.

[12] In the previous version of this paper (still available in `arxiv`, [4]) we had $f(s)=s+O(\log s)+O(n)$, and then we estimated the iterations of f by a simple but boring argument. With a better bound $f(s)=s+O(n)$ this is no more needed.

5 General result

In this section we use a similar technique to prove a more general result that covers not only Shannon inequalities but all true linear inequalities for Kolmogorov complexity. Recall that a theorem from Hammer et al. ([5], see [14, Chapter 10] for the detailed exposition) says that the same linear inequalities are true for complexities (with logarithmic precision) and for Shannon entropies. In this section we want to show that all inequalities in this class have space-bounded counterparts. For that, we need to modify the original proof from [5, 14] using the tools we developed. Let us first formulate this result in a form similar to Theorem 3.

Let us fix some integer $k \geqslant 1$.

Theorem 4 (Inequality with two space bounds)**.** *Assume that a linear inequality for unbounded complexities with non-negative coefficients λ_I and μ_J,*

$$\sum_{I\in L} \lambda_I \mathrm{C}(x_I) \leqslant \sum_{J\in R} \mu_J \mathrm{C}(x_J) + O(\log n), \tag{3}$$

is true for all n and for all strings $x_1,\dots,x_k$ of length at most n. Then its space-bounded version

$$\sum_{I\in L} \lambda_I \mathrm{C}^{s'}(x_I) \leqslant \sum_{J\in R} \mu_J \mathrm{C}^{s}(x_J) + O(\log n) \tag{4}$$

holds for all n, s, for all strings $x_1,\dots,x_k$ of length at most n and for $s' = s + O(n^2)$.

We will derive this result from a different statement that does not require separating positive and negative coefficients:

Theorem 5 (Existence of a common space bound)**.** *Assume that a linear inequality for unbounded complexities*

$$\sum_{I} \lambda_I \mathrm{C}(x_I) \leqslant O(\log n), \tag{5}$$

with coefficients λ_I that can be positive or negative, is true for all n and for all strings $x_1,\dots,x_k$ of length at most n. Then for every n, s and for every $x_1,\dots,x_k$ of length at most n its space-bounded version

$$\sum_{I\in L} \lambda_I \mathrm{C}^{s'}(x_I) \leqslant O(\log n) \tag{6}$$

holds for some $s' \in [s, s + O(n^2)]$.

This statement is purely existential (and a bit weird): it says that *there exists some s'* between s and $s + O(n^2)$ (depending on $x_1,\dots,x_k$) for which the inequality is true. Still it is easy to see that Theorem 4 immediately follows from Theorem 5: if the inequality (6) is true for some value of s', we may separate positive and negative coefficients as in (4) and then replace s' by s in the right hand side, and by $s + O(n^2)$ in the left hand side, due to the monotonicity. So it remains to prove Theorem 5.

Proof. We adapt the arguments used in [5, 14] to prove the connection between (unbounded) complexity and entropy inequalities.

Step 1. First of all, we convert our assumption into the language of Shannon's information theory and note that

$$\sum_{I} \lambda_I H(\xi_I) \leqslant 0$$

for arbitrary random variables $\xi_1,\dots,\xi_k$. Indeed, it is well known (see, e.g., [14, chapter 7]) that if ρ is an arbitrary random variable that has finite range, and $\rho^1,\dots,\rho^N$ are independent identically distributed copies of ρ, then the expected Kolmogorov complexity of the finite object $(\rho^1,\dots,\rho^N)$ is $NH(\rho)+O(\log N)$. Then, for a large N, we take N independent copies of the tuple $\xi_1,\dots,\xi_k$. For every $I\subset\{1,\dots,k\}$ we have

$$H(\xi_I)=\frac{\mathsf{E}[\mathrm{C}(\xi_I^1,\dots,\xi_I^N)]}{N}+\frac{O(\log N)}{N}.$$

The matrix ξ_i^j can be considered as a k-tuple of its columns (for each column i is fixed and j ranges from 1 to N), and the inequality for complexities can be applied to these columns. It guarantees that

$$\sum_I \lambda_I H(\xi_I)\leqslant\frac{O(\log N)}{N},$$

and we get the required inequality since the left hand side does not depend on N.

Step 2. For a given tuple $x_1,\dots,x_k$ whose elements are strings of length at most n, and for some $s'\geqslant s$ consider the set X of all the tuples $y_1,\dots,y_k$ of strings of length at most n such that

$$\mathrm{C}^{s'}(y_I|y_J)\leqslant\mathrm{C}^{s'}(x_I|x_J)$$

for all sets $I,J\subset\{1,\dots,k\}$. The log-size of X does not exceed $\mathrm{C}^{s'}(x_1,\dots,x_k)$, since one of the inequalities requires that $\mathrm{C}^{s'}(y_1,\dots,y_k)\leqslant\mathrm{C}^{s'}(x_1,\dots,x_k)$ (for empty J and for $I=\{1,\dots,k\}$). The following lemma provides a lower bound for its size:

Lemma 4. *The log-size of X is at least $\mathrm{C}^{s''}(x_1,\dots,x_k)-O(\log n)$, where $s''=s'+O(n)$.*

Proof of Lemma 4. The set of all $y_1,\dots,y_k$ of length at most n that satisfy all the inequalities

$$\mathrm{C}(y_I|y_J)\leqslant\mathrm{C}^{s'}(x_I|x_J)$$

(with unbounded complexity in the left side) can be enumerated if n and all the complexities in the right side on the inequalities are given. So the information needed to start the enumeration is of size $O(\log n)$. The tuple $x_1,\dots,x_k$ belongs to X, and can be reconstructed if its ordinal number in the enumeration is given. Therefore,

$$\mathrm{C}(x_1,\dots,x_k)\leqslant\log|X|+O(\log n).$$

Let us strengthen this inequality by using bounded complexity in the left-hand side:

$$\mathrm{C}^{s'+O(n)}(x_1,\dots,x_k)\leqslant\log|X|+O(\log n).$$

Indeed, the enumeration can be performed sequentially with increasing space bounds $1,2,3,\dots$, using Lemma 3 to compute space-bounded complexities. As before, we ensure the enumeration without repetitions by checking for every tuple $y_1,\dots,y_k$ whether it already appeared for the previous value of the space bound. In this enumeration the tuple $x_1,\dots,x_k$ appears when the space bound is s' (or less). Stopping the enumeration at this time (knowing the number of tuples that should be enumerated), we use space $s''=s'+O(n)$. As in the proof of Theorem 1, we keep the current value of the space bound all the time, but in such a way (as the difference between this value and stacks' size) that the used space never exceeds $s'+O(n)$.

Lemma 4 is proven. □

Step 3. As in the proof of Theorem 3, consider the sequence of bounds $s, f(s), f(f(s)), \ldots$ where $f(s) = s + O(n)$ is the bound from Lemma 4. When the bound s' increases, all the complexities $\mathrm{C}^{s'}(x_I|x_J)$ may only decrease. Recall that the parameter k is fixed; we have only $O(1)$-many decreasing complexities (for all pairs $I, J \subset \{1,2,\ldots,k\}$), and the initial value of these complexities is $O(n)$. Therefore, there are at most $O(n)$ steps when some complexity decreases, and it is enough to make $O(n)$ iterations to come to an iteration step when all complexities do not change. The total increase of the space bound during $O(n)$ iterations is $O(n^2)$. So we come to the following statement:

Lemma 5. *There exists some $s' \in [s, s + O(n^2)]$ such that*

$$\mathrm{C}^{f(s')}(x_I|x_J) = \mathrm{C}^{s'}(x_I|x_J)$$

for all $I, J \subset \{1,2,\ldots,k\}$.

Combining Lemma 5 with Lemma 4, we conclude that there exists $s' \in [s, s + O(n^2)]$ such that the set X of all $y_1, \ldots, y_k$ of length at most n such that

$$\mathrm{C}^{s'}(y_I|y_J) \leqslant \mathrm{C}^{s'}(x_I|x_J)$$

(for all I, J) has log-size $\mathrm{C}^{s'}(x_1, \ldots, x_k) + O(\log n)$: the upper bound for $\log |X|$ is obvious, and the lower bound is provided by Lemma 4, where we can replace s'' by s' due to the choice of s'.

Now consider a tuple of random variables $\xi_1, \ldots, \xi_k$ uniformly distributed in the set X. Its entropy is $\log |X| = \mathrm{C}^{s'}(x_1, \ldots, x_k) + O(\log n)$. The following lemma shows that the same connection between entropies and complexities is true for an arbitrary subset of indices. By ξ_I we denote the tuple of random variables ξ_i for $i \in I$.

Lemma 6.

$$H(\xi_I) = \log \mathrm{C}^{s'}(x_I) + O(\log n).$$

for every $I \subset \{1, \ldots, k\}$.

This lemma finishes the proof of Theorem 5. Indeed, if some inequality is true for (unbounded) complexities with logarithmic precision, it is true for entropies. In particular, it is true for entropies of subsets of $\xi_1, \ldots, \xi_k$, and these entropies coincide with bounded-space complexities of corresponding subsets of $x_1, \ldots, x_k$ with logarithmic precision. Therefore the inequality is also true for bounded-space complexities (for some s' in the interval $[s, s + O(n^2)]$, provided by Lemma 5). It remains to prove Lemma 6.

Proof of Lemma 6. Let I be some subset of $\{1, \ldots, k\}$, and J be its complement: $J = \{1, \ldots, k\} \setminus I$. We know that

$$H(\xi_1, \ldots, \xi_k) = H(\xi_I) + H(\xi_J|\xi_I).$$

All values of ξ_I are among tuples y_I for $y \in X$, and therefore $\mathrm{C}^{s'}(y_I) \leqslant \mathrm{C}^{s'}(x_I)$. So the range of ξ_I has log-size at most $\mathrm{C}^{s'}(x_I) + O(1)$, and the entropy of a random variable does not exceed the log-size of its range:

$$H(\xi_I) \leqslant \mathrm{C}^{s'}(x_I) + O(1).$$

For similar reasons we have

$$H(\xi_J|\xi_I) \leqslant \mathrm{C}^{s'}(x_J|x_I) + O(1).$$

Indeed, for every $y_1, \ldots, y_k$ in X we have $\mathrm{C}^{s'}(y_J|y_I) \leqslant \mathrm{C}^{s'}(x_J|x_I)$, so for every value of ξ_I the set of possible values of ξ_J has log-size at most $\mathrm{C}^{s'}(x_J|x_I) + O(1)$. The choice of s' guarantees that

the complexities of x_J given x_I with bound s' coincide with the same complexities with bound $s'' = s' + O(n)$. So we can write a chain of inequalities with precision $O(\log n)$:

$$H(\xi_1,\dots,\xi_k) = H(\xi_I) + H(\xi_J|\xi_I) \leqslant \mathrm{C}^{s'}(x_I) + \mathrm{C}^{s'}(x_J|x_I) = \mathrm{C}^{s''}(x_I) + \mathrm{C}^{s''}(x_J|x_I) \leqslant \mathrm{C}^{s'}(x_1,\dots,x_k).$$

(the last inequality is due to Theorem 1). We know that the leftmost and rightmost terms of this inequality coincide (with $O(\log n)$ precision, as for the other parts), so all the inequalities that appear in this chain are equalities with $O(\log n)$ precision. In particular, $H(\xi_I) = \mathrm{C}^{s'}(x_I) + O(\log n)$. Lemma 6 is proven. □

This finishes the proof of Theorem 5 (and its corollary, Theorem 4). □

Remark 7. Again, we do not need the complexities in Lemma 5 with bounds s and s' to be *exactly* the same; all the arguments remain valid if we make them differ by $O(\log n)$. In this way we may use $O(n/\log n)$ steps instead of $O(n)$, and get a slightly better bound $O(s) + O(n^2/\log n)$ in Theorems 4 and 5.

Remark 8. We may consider a more general class of linear inequalities in Theorem 5 that include all conditional complexities:

$$\sum \lambda_{I,J}\, \mathrm{C}(x_I|x_J) \leqslant 0.$$

Theorem 5 remains valid, and the proof is essentially the same; we need to show in Lemma 6 that

$$H(\xi_I|\xi_J) = \log \mathrm{C}^{s'}(x_I|x_J) + O(\log n)$$

for all $I, J \subset \{1,\dots,k\}$. This is done by a similar argument. First let us assume that I and J are disjoint. Let R be the set of indices that are not in I and not in J. Then we write the following chain of inequalities with $O(\log n)$ precision:

$$\begin{aligned} H(\xi_1,\dots,\xi_k) = H(\xi_J) + H(\xi_I|\xi_J) + H(\xi_R|\xi_{I\cup J}) \leqslant \mathrm{C}^{s'}(x_J) + \mathrm{C}^{s'}(x_I|x_J) + \mathrm{C}^{s'}(x_R|x_{I\cup J}) = \\ = \mathrm{C}^{s''}(x_J) + \mathrm{C}^{s''}(x_I|x_J) + \mathrm{C}^{s''}(x_R|x_{I\cup J}) \leqslant \mathrm{C}^{s'}(x_1,\dots,x_k), \end{aligned}$$

and use the same argument as before. The difference is that here we need to use the bounded-space Kolmogorov–Levin formula for triples:

$$\mathrm{C}^{s''}(x) + \mathrm{C}^{s''}(y|x) + \mathrm{C}^{s''}(z|x,y) \leqslant \mathrm{C}^{s'}(x,y,z)$$

which can be obtained by using the formula for pairs twice; recall that $O(n)$ overhead, appearing twice, is still $O(n)$.

As before, Theorem 5 implies Theorem 4.

For unbounded complexities it makes no sense to include conditional complexities in the inequalities, since Kolmogorov–Levin formula reduces them to unconditional ones. However, for space-bounded complexities this reduction will change the bounds, so we may wish to allow them to appear explicitly.

Remark 9.

In Theorems 4 and 5 we may also replace the $O(\log n)$ additive term by $O(\log \mathrm{C}^s(x_1,\dots,x_k))$.

For Theorem 4 we repeat the argument used to finish the proof of theorem 1. We noted there that for all s and x, there exists a program p of length $m = \mathrm{C}^s(x)$ such that $\mathrm{C}^{s+O(|x|)}(p|x) \leqslant O(\log m)$ and $\mathrm{C}^{s+O(1)}(x|p) \leqslant O(1)$. Hence, the better precision follows by replacing $x_1,\dots,x_k$ by the programs $p_1,\dots,p_k$ of length $|p_i| \leqslant \mathrm{C}^{s+c}(x_i)$ where the constant c should be large enough to guarantee that $\mathrm{C}^{s+c}(x_i) \leqslant \mathrm{C}^s(x_1,\dots,x_k) + O(1)$.

For Theorem 5 we use the same idea, but first we have to look at the proof of this theorem and notice that in fact we proved the following statement:

if for some strings $x_1, \ldots, x_k$ of length at most n the complexities $\mathrm{C}^s(x_I|x_J)$ change at most by d when s is increased up to $s+cn$ (here c is a large enough constant), *then the inequality* (6) *is valid for $s' = s$ with additional term $O(d)$ in the right hand size.*

Now the argument goes as follows. We have strings $x_1, \ldots, x_k$ of length at most n and look at the complexities $\mathrm{C}^{s'}(x_I|x_J)$ as a function of s'. As before, we can find a interval of length $c'n$ inside $[s, s+O(n^2)]$ where all these complexities do not change. This can be done for arbitrary large constant c' (and the constant in $O(n^2)$ depends on c'). Let $[u,v]$ be this interval. Then we have $\mathrm{C}^u(x_i) \leqslant \mathrm{C}^u(x_1, \ldots, x_k) + O(1)$, since $\mathrm{C}^{s'}(x_i)$ is the same for all $s' \in [u,v]$.

Now we apply our replacement argument and find p_i such that conditional complexities $\mathrm{C}^{u+O(n)}(x_i|p_i)$ and $\mathrm{C}^{u+O(n)}(p_i|x_i)$ are at most $O(\log m)$, and the lengths of all p_i are $O(m)$, where $m = O(\mathrm{C}^u(x_1, \ldots, x_k))$. Therefore, if we increase the left endpoint u of the interval for the space bounds by $O(n)$, in this smaller interval $[u+O(n), v]$ all the values $\mathrm{C}^{s'}(p_I|p_J)$ differ from corresponding $\mathrm{C}^{s'}(x_I|x_J)$ by at most $O(\log m)$ and therefore change (when s' is in $[u+O(n), v]$) at most by $O(\log m)$, since $\mathrm{C}^{s'}(x_I|x_J)$ do not change at all. It remains to apply the result quoted earlier to $p_1, \ldots, p_k$. Note that the lengths of $p_1, \ldots, p_k$ are $O(m)$, that $\mathrm{C}^{s'}(x_I|x_J)$ are $O(\log m)$ close to $\mathrm{C}^{s'}(p_I|p_J)$, and that the remaining interval $[u+O(n), v]$ has length at least cn for any constant c if c' is large enough.

6 Discussion

Increasing the density. Theorem 5 says that the space-bounded version of the inequality (that is true in the unbounded version) is valid for the sequence of space bounds s_j that is not very sparse: $s_{j+1} \leqslant s_j + O(n^2)$. Is it possible to improve this result and show that the inequality in question is true for "more dense" sequence of space bounds?

Space-bounded versions of other results. Our results are part of the space-bounded version of algorithmic information theory. In general, one could take some notion or theorem of algorithmic information theory and look for its space-bounded counterpart. For Muchnik's conditional codes theorem this was done by Musatov (see [11] and references therein).

However, there are many problems in this approach. For example, if we define mutual information with space bound s in a natural way as

$$I^s(a:b) = \mathrm{C}^s(a) - \mathrm{C}^s(a|b),$$

this notion is not monotone; a priori the mutual information can oscillate when s increases. It would be interesting to understand what kinds of oscillations are possible. Is it possible that two strings are mutually independent for some space bound, then dependent for some larger bound, then again independent, and so on? Also the relations between $I^s(a:b)$, $I^s(b:a)$ and the symmetric expression $\mathrm{C}^s(a) + \mathrm{C}^s(b) - \mathrm{C}^s(a,b)$ are unclear.

Time-bounded versions. We can try a similar approach for time bounds (instead of space bounds). It also works, but the natural bound in the formula for complexity of pairs multiplies the time complexity by $2^{O(n)}$; also the simulation would increase time significantly (for a one-tape machine the simulation of t steps needs more than t^2 time). When we iterate these bounds $O(n)$ times, we get ridiculously high time bounds. It is just good luck that Sipser's trick for space bounds allows us to get some reasonable space bounds, and for time bounds things are much worse. Still one can have some versions of our results with computable (though ridiculously large) time bounds.

References

[1] G. Chaitin, Computational Complexity and Gödel's incompleteness theorem, *SIGACT News*, **9** (April 1971), 11–12.

[2] T.H. Chan, A combinatorial approach to information inequalities, *Communications in Informations and Systems*, **1**(3), 241–254 (September 2001, preliminary version in 1999)

[3] T.H. Chan and R.W. Yeung, On a relation between information inequalities and group theory, *IEEE Transactions on Information Theory*, **IT-48**(7), 1992–1995 (July 2002, preliminary version in 1999)

[4] P. Gacs, A. Romashchenko, A. Shen, *Inequalities for space-bounded Kolmogorov complexity*, `https://arxiv.org/abs/2010.10221`

[5] D. Hammer, A. Romashchenko, A. Shen and N. Vereshchagin, Inequalities for Shannon Entropies and Kolmogorov Complexities, in: *Proceedings 12th IEEE conference on Computational Complexity*, Ulm, 1997, 13–23. Final version: *Journal of Computer and System Sciences*, **60**, 442–464.

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

[7] A.N. Kolmogorov, Logical basis for information theory and probability theory, *IEEE Transaction on Information Theory*, **14**, 662–664 (1968)

[8] L. Longpré, Resource bounded Kolmogorov complexity, a link between computational complexity and information theory, Ph. D. thesis, TR 86-776 (1986), 115 pp. Dept. of Computer Science, Cornell University, Ithaca, NY 14583, `https://ecommons.cornell.edu/handle/1813/6616`

[9] L. Longpré and S. Mocas, Symmetry of information and one-way functions, *Information processing letters*, **46**(2), 95–100 (1993)

[10] F. Matúš, Infinitely many information inequalities. In Proc. *IEEE International Symposium on Information Theory*, 41–44 (2007)

[11] D. Musatov, Improving the space-bounded version of Muchnik's conditional complexity theory via naive derandomization, *Theory of Computing Systems*, **55**, 299–312 (2014), see also `https://arxiv.org/abs/1009.5108`

[12] A. Romashchenko, A. Shen and N. Vereshchagin, Combinatorial interpretation of Kolmogorov complexity, in: *Proceedings 15th Annual IEEE Conference on Computational Complexity*, Florence, Italy, 2000, 131–137, `https://doi.org/10.1109/CCC.2000.856743`. Final version: *Theoretical Computer Science*, **271**(1–2), 111-123 (2002).

[13] A. Shen, *Algorithms and programming: problems and solutions*, 2nd ed., Springer, 2010.

[14] A. Shen, V.A. Uspensky, N. Vereshchagin, *Kolmogorov complexity and algorithmic randomness*, AMS, 2018.

[15] M. Sipser, Halting space-bounded computations, *Theoretical Computer Science*, **10** (1980), 335–338.

[16] N. Vereshchagin and A. Shen, Algorithmic statistics revisited, in: *Measures of Complexity. Festschrift for Alexey Chervonenkis.* Springer, 2015, 235–252.

[17] N. Vereshchagin and A. Shen, Algorithmic statistics: forty year later, in: *Computability and Complexity. Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*. Springer, 2017, 669–737.

[18] Z. Zhang and R.W. Yeung, On characterization of entropy function via information inequalities, *IEEE Transactions in Information Theory*, **IT-44**(4), 1440–1452 (1998)