New Applications of the Incompressibility Method: Part II

Harry Buhrman*
CWI

Tao Jiang†
McMaster University

Ming Li‡
University of Waterloo

Paul Vitányi§
CWI and University of Amsterdam

Abstract

The incompressibility method is an elementary yet powerful proof technique based on Kolmogorov complexity. It has been used successfully in many areas [8]. To further demonstrate its power and elegance we exhibit new simple proofs using the incompressibility method.

1 Introduction

The incompressibility of individual random objects yields a simple but powerful proof technique: *the incompressibility method*. This method is a general purpose tool that can be used to prove lower bounds on computational problems, to obtain combinatorial properties of concrete objects, and to analyze the average complexity of an algorithm. Since the early 1980's, the incompressibility method has been successfully used to solve many well-known questions that had been open for a long time and to supply new simplified proofs for known results. A survey is [8].

The purpose of this paper is pragmatic, in the same style as [9, 7]. We want to further demonstrate how easy the incompressibility method can be used, via a new collection of simple examples. The proofs we have chosen to be included in [7] and here are not difficult ones. They are from diverse topics and most of these topics are well-known. Some of our results are new (but this is not important), and some are known before. In all cases, the proofs are new, and the new proofs are much simpler than the old ones (if they exist).

*Partially supported by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556, and by NWO through NFI Project ALADDIN number NF 62-376. Address: CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: buhrman@cwi.nl

†Supported in part by the NSERC Research Grant OGP0046613 and a CGAT grant. Address: Department of Computer Science, McMaster University, Hamilton, Ont L8S 4K1, Canada. Email: jiang@maccs.mcmaster.ca

‡Supported in part by the NSERC Research Grant OGP0046506, CITO, a CGAT grant, and the Steacie Fellowship. Address: Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada. E-mail: mli@math.uwaterloo.ca

§Partially supported by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556, and by NWO through NFI Project ALADDIN number NF 62-376. Address: CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: paulv@cwi.nl

2 Kolmogorov Complexity and the Incompressibility Method

We use the following notation. Let x be a finite binary string. Then $l(x)$ denotes the *length* (number of bits) of x. In particular, $l(\epsilon) = 0$ where ϵ denotes the *empty word*.

We can map $\{0,1\}^*$ one-to-one onto the natural numbers by associating each string with its index in the length-increasing lexicographical ordering

$$(\epsilon, 0), (0, 1), (1, 2), (00, 3), (01, 4), (10, 5), (11, 6), \ldots . \tag{1}$$

This way we have a binary representation for the natural numbers that is different from the standard binary representation. It is convenient not to distinguish between the first and second element of the same pair, and call them "string" or "number" arbitrarily. As an example, we have $l(7) = 00$. Let $x, y, \in \mathcal{N}$, where $\mathcal{N}$ denotes the natural numbers. Let $T_0, T_1, \ldots$ be a standard enumeration of all Turing machines. Let $\langle \cdot, \cdot \rangle$ be a standard one-one mapping from $\mathcal{N} \times \mathcal{N}$ to $\mathcal{N}$, for technical reasons chosen so that $l(\langle x, y \rangle) = l(y) + O(l(x))$.

Informally, the Kolmogorov complexity, [10], of x is the length of the *shortest* effective description of x. That is, the *Kolmogorov complexity* $C(x)$ of a finite string x is simply the length of the shortest program, say in FORTRAN (or in Turing machine codes) encoded in binary, which prints x without any input. A similar definition holds conditionally, in the sense that $C(x|y)$ is the length of the shortest binary program which computes x on input y. Kolmogorov complexity is absolute in the sense of being independent of the programming language, up to a fixed additional constant term which depends on the programming language but not on x. We now fix one canonical programming language once and for all as reference and thereby $C()$. For the theory and applications, as well as history, see [8]. A formal definition is as follows:

Definition 1 Let U be an appropriate universal Turing machine such that

$$U(\langle \langle i, p \rangle, y \rangle) = T_i(\langle p, y \rangle)$$

for all i and $\langle p, y \rangle$. The *conditional Kolmogorov complexity* of x given y is

$$C(x|y) = \min_{p \in \{0,1\}^*} \{ l(p) : U(\langle p, y \rangle) = x \}.$$

The unconditional Kolmogorov complexity of x is defined as $C(x) := C(x|\epsilon)$.

It is easy to see that there are strings that can be described by programs much shorter than themselves. For instance, the function defined by $f(1) = 2$ and $f(i) = 2^{f(i-1)}$ for $i > 1$ grows very fast, $f(k)$ is a "stack" of k twos. Yet for each k it is clear that $f(k)$ has complexity at most $C(k) + O(1)$.

By a simple counting argument one can show that whereas some strings can be enormously compressed, the majority of strings can hardly be compressed at all. For each n there are 2^n binary strings of length n, but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ possible shorter descriptions. Therefore, there is at least one binary string x of length n such that $C(x) \geq n$. We call such strings *incompressible*. It also follows that for any length n and any binary string y, there is a binary string x of length n such that $C(x|y) \geq n$.

DEFINITION 2 For each constant c we say a string x is *c-incompressible* if $C(x) \geq l(x) - c$.

Strings that are incompressible (say, c-incompressible with small c) are patternless, since a pattern could be used to reduce the description length. Intuitively, we think of such patternless sequences as being random, and we use "random sequence" synonymously with "incompressible sequence." It is possible to give a rigorous formalization of the intuitive notion of a random sequence as a sequence that passes all effective tests for randomness, see for example [8].

How many strings of length n are c-incompressible? By the same counting argument we find that the number of strings of length n that are c-incompressible is at least $2^n - 2^{n-c} + 1$. Hence there is at least one 0-incompressible string of length n, at least one-half of all strings of length n are 1-incompressible, at least three-fourths of all strings of length n are 2-incompressible, $\ldots$, and at least the $(1 - 1/2^c)$th part of all 2^n strings of length n are c-incompressible. This means that for each constant $c \geq 1$ the majority of all strings of length n (with $n > c$) is c-incompressible. We generalize this to the following simple but extremely useful *Incompressibility Lemma.*

LEMMA 1 *Let c be a positive integer. For each fixed y, every set A of cardinality m has at least $m(1 - 2^{-c}) + 1$ elements x with $C(x|y) \geq \lfloor \log m \rfloor - c$.*

PROOF. By simple counting. □

As an example, set $A = \{x : l(x) = n\}$. Then the cardinality of A is $m = 2^n$. Since it is easy to assert that $C(x) \leq n + c$ for some fixed c and all x in A, Lemma 1 demonstrates that this trivial estimate is quite sharp. The deeper reason is that since there are few short programs, there can be only few objects of low complexity.

DEFINITION 3 A *prefix set*, or prefix-free code, or prefix code, is a set of strings such that no member is a prefix of any other member. A prefix set which is the domain of a partial recursive function (set of halting programs for a Turing machine) is a special type of prefix code called a *self-delimiting* code because there is an effective procedure which reading left-to-right determines where a code word ends without reading past the last symbol. A one-to-one function with a range that is a self-delimiting code will also be called a self-delimiting code.

A simple self-delimiting code we use throughout is obtained by reserving one symbol, say 0, as a stop sign and encoding a natural number x as $1^x 0$. We can prefix an object with its length and iterate this idea to obtain ever shorter codes:

$$E_i(x) = \begin{cases} 1^x 0 & \text{for } i = 0, \\ E_{i-1}(l(x))x & \text{for } i > 0. \end{cases} \tag{2}$$

Thus, $E_1(x) = 1^{l(x)}0x$ and has length $l(E_1(x)) = 2l(x) + 1$; $E_2(x) = \lg_1(l(x))x$ and has length $l(E_2(x)) = l(x) + 2l(l(x)) + 1$. We have for example

$$l(E_3(x)) \leq l(x) + \log l(x) + 2 \log \log l(x) + 1.$$

Define the pairing function

$$\langle x, y \rangle = E_2(x)y \tag{3}$$

with inverses $\langle \cdot \rangle_1, \langle \cdot \rangle_2$. This can be iterated to $\langle \langle \cdot, \cdot \rangle, \cdot \rangle$.

In a typical proof using the incompressibility method, one first chooses an individually random object from the class under discussion. This object is effectively incompressible. The argument invariably says that if a desired property does not hold, then the object can be compressed. This yields the required contradiction. Then, since most objects are random, the desired property usually holds on average.

3 Complexity Often Exceeds Length

A simple counting argument shows that for every n there is at least one string x of length n such that $C(x|n) \geq n$ and a string y of length n such that $C((y) \geq n$. We show here that this can be strengthened considerably: (i) A positive fraction of all strings of length n have complexity at least n; (ii) There is a fixed constant c such that for every k there is an n with $kc \leq n < (k+1)c$ such that a positive fraction of all strings of length n have complexity exceeding n (here c, k are positive integers). Item (i) can be formulated slightly more general:

LEMMA 2 *There is a constant $d > 0$ such that for every n there are at least $\lfloor 2^n/d \rfloor$ strings x of length n with $C(x|n) \geq n$ (respectively, $C(x) \geq n$).*

PROOF. It is well-known that there is a constant $c \geq 0$ such that for every n every string x of length n has $C(x|n) \leq n + c$. Hence for every n and every x of length $l(x) \leq n - 2\log(n - l(x)) - c - 1$ we have

$$C(x|n) \leq C(x|l(x)) + 2\log(n - l(x)) < n.$$

Consequently, for some constant $c' > 0$ there are at most $2^n - 2^{n-c'}$ programs of length $< n$ available as shortest programs for the strings of length n (there are $2^n - 1$ potential programs and $2^{n-c'} - 1$ thereoff are already taken). Hence there are at least $2^{n-c'}$ strings x of length n with $C(x|n) \geq n$.

In the unconditional case the proof is simpler and for $C(x|l(x)) \leq l(x) + c$ we find that there are at least 2^{n-c} strings x of length n with $C(x) \geq n$. This is because we can dispense with the $2\log(n - l(x))$ term induced by the conditional 'n'. $\square$

REMARK 1 A similar result for the *prefix version* $K(\cdot)$ of Kolmogorov complexity (also called *self-delimiting complexity*) is given in [4] using a more complicated argument. Note that neither result implies the other.

We show that for every length n there are many strings that have complexity at least their length. How many there (provably) are depends on the constant c in the the upper bounds on the (un)conditional complexity. For particular universal machines we can determine this complexity. John Tromp has formulated a universal machine in combinatory logic, [8] pp. 201–202, such that for all x, y this reference machine achieves $C(x) \leq l(x) + 8$ and $C(x|y) \leq l(x) + 2$. Using this reference machine there are at least 2^{n-8} strings x of length n that have $C(x) \geq n$. In the conditional case can set $c = 2$ and hence must choose $l(x)$ in the proof above so that $(n - l(x)) - 2\log(n - l(x)) \geq 3$ which is the case for $n - l(x) = 10$. From this we can determine that $c' := 10$ satisfies the requirements. Hence there are at least 2^{n-10} strings x of length n such that $C(x|n) \geq n$.

Can we prove that the complexity of many strings must *exceed* their lengths? This is indeed the case. The proof uses an incompressibility argument to show *negative* incompressibility.

LEMMA 3 *There are constants $c, d > 0$ such that for every large enough n there are at least $\lfloor 2^n/d \rfloor$ strings x of length $l(x)$ in the interval $[n-c, n]$ with $C(x|n) > n$ (respectively, $C(x) > n$).*

PROOF. For every n there are equally many strings of length $\leq n$ to be described and potential programs of length $\leq n$ to describe them. Since some programs do not halt for every large enough n there exists a string x of length at most n such that $n < C(x|n) \leq l(x) + c$ and a string y of length at most n such that $n < C(y) \leq l(y) + c$.

Let there be $m \geq 1$ such strings. Given m and n we can enumerate all $2^{n+1}-m-1$ strings x of length $\leq n$ and complexity $C(x|n) \leq n$ by dovetailing the running of all programs of length $\leq n$. The lexicographic first string of length $\leq n$ not in the list, say x, is described by a program p giving m in $\log m$ bits plus an $O(1)$-bit program to do the decoding of x. Therefore, $\log m + O(1) \geq C(x|n) > n$ which proves the theorem for the conditional case. The unconditional result follows similarly by padding the description of x up to length $n+c'$ for a constant c' and adding the description of c' to program p describing x. This way we can first retrieve c' from p and then retrieve n from the length of p. $\square$

So there are lots of strings x that have complexity larger than their lengths. How much larger can this excess get? In the theory of Kolmogorov complexity the laws are invariant with respect to the choice of the particular reference universal Turing machine in the definition of the Kolmogorov complexity, the excess of maximal complexity over the length depends on this choice. Since one can choose a reference universal Turing machine such that $C(x) \leq l(x) + 1$ we cannot generally prove that the excess exceeds 1.

REMARK 2 For example, we can easily choose a reference universal Turing machine that has no halting programs of odd length, or such that it has no halting programs of length $i \bmod 100$ for $i = 0, \ldots, 98$. In such a case there are many x's that have complexity at least $l(x) + 100$. In the opposite extreme, given an appropriate universal Turing machine U we can transform it into a universal Turing machine U' such that $U'(1p) = p$ and $U'(0p) = U(p)$ for all p. Taking U' as reference universal Turing machine we clearly have $C(x) \leq l(x) + 1$ for all x. Consequently, for every n the shortest programs of strings of length $< n$ have length at most n. This means that there are at most $2^{n+1} - 2^n = 2^n$ strings available of length at most n to serve as shortest programs for strings of length n.

By definition of U' the Lemma 3 means that at least $\Omega(2^n)$ strings of length n are used as shortest programs for strings of length $n-1$, while by definition of U' at least $2^n/2$ strings of length n are used as (not necessarily shortest) programs for strings of length $n-1$. Consequently, at most $2^n/2$ strings x of length n have complexity $C(x) = n$ and at least $\Omega(2^n)$ strings y of length n have complexity $C(y) = n + 1$. There are no strings z of length n that have complexity $C(z) > n + 1$.

4 Average Time for Boolean Matrix Multiplication

We begin with a simple illustration of average-case analysis using the incompressibility method. Consider the problem of multiplying two $n \times n$ boolean matrices $A = (a_{i,j})$ and $B = (b_{i,j})$. Efficient algorithms for this problem have always been a very popular topic in the theoretical computer science literature due to the wide range of applications of boolean matrix multiplication. The best worst-case time complexity obtained so far is $O(n^{2.376})$ due to Coppersmith and Winograd [5]. In 1973, O'Neil and O'Neil devised a simple algorithm described below which runs in $O(n^3)$ time in the worst case but achieves an average time complexity of $O(n^2)$ [12].

Algorithm QuickMultiply(A, B)

1. Let $C = (c_{i,j})$ denote the result of multiplying A and B.
2. For $i := 1$ to n do
3. Let $j_1 < \cdots < j_m$ be the indices such that $a_{i,j_k} = 1$, $1 \le k \le m$.
4. For $j := 1$ to n do
5. Search the list $b_{j_1,j}, \ldots, b_{j_m,j}$ sequentially for a bit 1.
6. Set $c_{i,j} = 1$ if a bit 1 is found, or $c_{i,j} = 0$ otherwise.

An analysis of the average-case time complexity of QuickMultiply is given in [12] using probabilitistic arguments. Here we give an analysis using the incompressibility method.

Theorem 1 *Suppose that the elements of A and B are drawn uniformly and independently. Algorithm QuickMultiply runs in $O(n^2)$ time on the average.*

Proof. Let n be a sufficiently large integer. Observe that the average time of QuickMultiply is trivially bounded between $O(n^2)$ and $O(n^3)$. By the Incompressibility Lemma, out of the 2^{2n^2} pairs of $n \times n$ boolean matrices, at least $(n-1)2^{2n^2}/n$ of them are $\log n$-incompressible. Hence, it suffices to consider $\log n$-incompressible boolean matrices.

Take a $\log n$-incompressible binary string x of length $2n^2$, and form two $n \times n$ boolean matrices A and B straightforwardly so that the first half of x corresponds to the row-major listing of the elements of A and the second half of x corresponds to the row-major listing of the elements of B. We show that QuickMultiply spends $O(n^2)$ time on A and B.

Consider an arbitrary i, where $1 \le i \le n$. It suffices to show that the n sequential searches done in Steps 4 – 6 of QuickMultiply take a total of $O(n)$ time. By the statistical results on various blocks in incompressible strings given in Section 2.6 of [8], we know that at least $n/2 - O(\sqrt{n \log n})$ of these searches find a 1 in the first step, at least $n/4 - O(\sqrt{n \log n})$ searches find a 1 in two steps, at least $n/8 - O(\sqrt{n \log n})$ searches find a 1 in three steps, and so on. Moreover, we claim that none of these searches take more than $4 \log n$ steps. To see this, suppose that for some j, $1 \le j \le n$, $b_{j_1,j} = \cdots = b_{j_{4\log n},j} = 0$. Then we can encode x by listing the following items in a self-delimiting manner:

1. A description of the above discussion.

2. The value of i.

3. The value of j.

4. All bits of x except the bits $b_{j_1,j}, \ldots, b_{j_{4\log n},j}$.

This encoding takes at most

$$O(1) + 2\log n + 2n^2 - 4\log n + O(\log\log n) < 2n^2 - \log n$$

bits for sufficiently large n, which contradicts the assumption that x is $\log n$-incompressible.

Hence, the n searches take at most a total of

$$\begin{aligned}
& (\sum_{k=1}^{\log n} (n/2^k - O(\sqrt{n\log n})) \cdot k) + (\log n) \cdot O(\sqrt{n\log n}) \cdot (4\log n) \\
< \; & (\sum_{k=1}^{\log n} kn/2^k + O(\log^2 n\sqrt{n\log n}) \\
= \; & O(n) + O(\log^2 n\sqrt{n\log n}) \\
= \; & O(n)
\end{aligned}$$

steps. This completes the proof. □

5 Average Complexity of Finding the Majority

Let $x = x_1 \cdots x_n$ be a binary string. The *majority bit* (or simply, the *majority*) of x is the bit (0 or 1) that occurs more than $\lfloor n/2 \rfloor$ times in x. The majority problem is that, given a binary string x, determine a position i such that x_i is the majority bit of x, using only bit comparisons. When x has no majority, we must report so.

The time complexity for finding the majority has been well studied in the literature (see, *e.g.* [1, 2, 3, 6, 13]). It is known that, in the worst case, $n - \nu(n)$ bit comparisons are necessary and sufficient [2, 13], where $\nu(n)$ is the number of occurrences of bit 1 in the binary representation of number n. Recently, Alonso, Reingold and Schott [3] studied the average complexity of finding the majority assuming the uniform probability distribution model. Using quite sophisticated arguments based on decision trees, they showed that on the average finding the majority requires at most $2n/3 - \sqrt{8n/9\pi} + O(\log n)$ comparisons and at least $2n/3 - \sqrt{8n/9\pi} + \Theta(1)$ comparisons.

In this section, we consider the average complexity of finding the majority and prove an upper bound tight up to the first major term, using simple incompressibility arguments.

The following standard tournament algorithm is needed.

Algorithm Tournament($x = x_1 \cdots x_n$)

1. If $n = 0$ then return "no majority."

2. If $n = 1$ then return x_1 as the majority.

3. Elseif $n = 2$ then
4. If $x_1 = x_2$ then return x_1 as the majority.
5. Else return "no majority".
6. Elseif $n = 3$ then
7. If $x_1 = x_2$ then return x_1 as the majority.
8. Else return x_3 as the majority.
9. Let $y = \epsilon$.
10. For $i := 1$ to $\lfloor n/2 \rfloor$ do
11. If $x_{2i-1} = x_{2i}$ then append the bit x_{2i} to y.
12. If n is odd and $\lfloor n/2 \rfloor$ is even then append the bit x_n to y.
13. Call Tournament(y).

THEOREM 2 *On the average, algorithm Tournament requires at most* $2n/3 + O(\sqrt{n})$ *comparisons.*

PROOF. Let n be a sufficiently large number. Again, since algorithm Tournament makes at most n comparisons on any string of length n, by the Incompressibility Lemma, it suffices to consider running time of Tournament on δ-incompressible strings, where $\delta \leq \log n$. Consider an arbitrary $\delta \leq \log n$ and let $x = x_1 \cdots x_n$ be a fixed δ-incompressible binary string. For any integer $m \leq n$, let $\sigma(m)$ denote the maximum number of comparisons required by algorithm Tournament on any δ-incompressible string of length m.

We know from [8] that among the $\lfloor n/2 \rfloor$ bit pairs $(x_1, x_2), \ldots, (x_{2\lfloor n/2 \rfloor - 1}, x_{2\lfloor n/2 \rfloor})$ that are compared in step 10 of Tournament, there are at least $n/4 - O(\sqrt{n\delta})$ pairs consisting of complementary bits. Clearly, the new string y obtained at the end of step 11 should satisfy

$$C(y) \geq l(y) - \delta - O(1)$$

Hence, we have the following recurrence relation for $\sigma(m)$:

$$\sigma(m) \leq \lfloor m/2 \rfloor + \sigma(m/4 + O(\sqrt{m\delta}))$$

By straightforward expansion, we obtain that

$$\begin{aligned}
\sigma(n) &\leq \lfloor n/2 \rfloor + \sigma(n/4 + O(\sqrt{n\delta})) \\
&\leq n/2 + \sigma(n/4 + O(\sqrt{n\delta})) \\
&\leq n/2 + (n/8 + O(\sqrt{n\delta})/2) + \sigma(n/16 + O(\sqrt{n\delta})/4 + O(\sqrt{(n\delta)/4})) \\
&= n/2 + (n/8 + O(\sqrt{n\delta})/2) + \sigma(n/16 + (3/4) \cdot O(\sqrt{n\delta})) \\
&\leq \cdots \\
&\leq 2n/3 + O(\sqrt{n\delta})
\end{aligned}$$

Using the Incompressibility Lemma, we can calculate the average complexity of algorithm Tournament as:

$$\sum_{\delta=1}^{\log n} \frac{1}{2^\delta}(2n/3 + O(\sqrt{n\delta})) + \frac{1}{n}n = 2n/3 + O(\sqrt{n})$$

□

6 Communication Complexity

Consider the following communication complexity problem (for definitions see the book by Kushilevitz and Nisan [11]). Initially, Alice has a string $x = x_1, \ldots, x_n$ and Bob has a string $y = y_1, \ldots, y_n$ with $x, y \in \{0,1\}^n$. Alice and Bob use an agreed-upon protocol to compute the inner product of x and y modulo 2

$$f(x,y) = \sum_{i=1}^{n} x_i . y_i \bmod 2$$

with Alice ending up with the result. We are interested in the minimal possible number of bits used in communication between Alice and Bob in such a protocol. Here we prove a lower bound of n, which is tight since the trivial protocol where Bob sends all his n bits to Alice achieves this bound. In [11] the same lower bound is obtained by a different method. We also show an $n - O(1)$ lower bound for the average-case complexity.

THEOREM 3 *Assume the discussion above. Every protocol computing the inner product function requires at least n bits of communication.*

PROOF. Fix a communication protocol P that computes the inner product. Let A be an algorithm that we describe later. Let z be a string of length $2n$ such that $C(z|A, P, n) \geq 2n$. Let $z = x_1 \ldots x_n y_1 \ldots y_n$ Let Alice's input be $x = x_1 \ldots x_n$ and Bob's input be $y_1 \ldots y_n$. Assume without loss of generality that $f(x,y) = 0$ (the innerproduct of x and y is 0 modulo 2). [1] Run the communication protocol P between Alice and Bob ending in a state where Alice outputs that $f(x,y)$ is 0. Let C be the sequence of bits sent back and forth. Note that P can be viewed as a tree with C a path in this tree [11]. Hence C is self-delimiting. Consider the set S defined by

$$S := \{a : \exists b \text{ such that } P(a,b) = 0 \text{ and induces conversation C }, a, b \in \{0,1\}^n\}.$$

Given n, P and C, we can compute S. Let the cardinality of S be l. The strings in S form a matrix M over GF(2) with the ith row of M corresponding to the ith string in S (say in lexicographic ordering). Since for every $a \in S$ it holds that $f(a,y) = 0$ it follows that y is an element of the Null space of M ($y \in \text{Null}(M)$). Application of the Null space Theorem from linear algebra yields:

$$\text{rank}(M) + \dim(\text{Null(M)}) = n. \tag{4}$$

[1] If this is not the case, then we can simply use a different universal TM U' that does precisely what the current universal TM U does except it flips the first bit of every output. Thus z with first bit flipped would have Kolmogorov complexity $2n$ under U'.

Since the cardinality of S is l and we are working over GF(2) it follows that the rank of M is at least $\log(l)$ and by (4) it follows that $\dim(\text{Null}(M)) \leq n - \log(l)$. The following is an effective description of z given n and the reconstructive algorithm A explained below:

1. C;
2. the index of $x \in S$ using $\log(l)$ bits; and
3. the index of $y \in \text{Null}(M)$ with $n - \log(l)$ bits.

The three items above can be concatenated without delimiters. Namely, C itself is self-delimiting, while from C one can generate S and hence compute l. From the latter item one can compute the binary length of the index for $x \in S$, and the remaining suffix of the binary description is the index for $y \in \text{Null}(M)$. From the given description and P, n the algorithm A reconstructs x and y and outputs $z = xy$. Consequently, $C(z|A,P,n) \leq l(C) + \log l + (n - \log l)$. Since we have assumed $C(z|A,P,n) \geq 2n$ it follows that $l(C) \geq n$. $\square$

Approximately the same lower bound holds for the average-case communication complexity of computing the inner product of two n-bit strings:

THEOREM 4 *The average communication complexity of computing the inner product of two n-bit strings is at least $n - O(1)$ bits.*

PROOF. There are exactly 2^{2n-1} strings z of length $2n$ such that $z = xy$, $l(x) = l(y) = n$ and inner product of x and y modulo 2 equals 1. For such z define $\bar{z} = \bar{x}\bar{y}$ as z but with the first bit of x and the first bit of y changed so that the inner product of the resulting strings $\bar{x}$ and $\bar{y}$ equals 0. $C(\bar{w}|n) = C(w|n) + O(1)$ for every $w \in \{z, x, y\}$. Let $\delta(n)$ be a function and choose in the proof above $C(z|A,P,n) \geq 2n - \delta(n)$. By simple counting there are at least

$$2^{2n}(1 - 1/2^{\delta(n)}) \tag{5}$$

such z's. If the above inner product associated with z equals 1 then the inner product associated with $\bar{z}$ equals 0 and $C(\bar{z}|A,P,n) \geq 2n - \delta(n) - O(1)$

Hence we can apply the proof of the previous theorem for all z with randomness deficiency at most $\delta(n)$ as follows:

- If z has an associated inner product 0 then the proof as above yielding $l(C_z) \geq n - \delta(n)$ where C_z is the communication sequence associated with the computation with input z.
- If z has associated inner product 1 then the proof as above to $\bar{z}$ yielding $l(C_{\bar{z}}) \geq n - \delta(n) - O(1)$.

There are at least $2^{2n}(1 - 1/2^{\delta(n)})$ strings z of length $2n$ with $C(z|A,P,n) \geq 2n - \delta(n)$. Altogether we obtain that the average communication complexity is

$$\sum_{z \in \{0,1\}^{2n}} \Pr(z) l(C_z) = 2^{-2n} \sum_{z \in \{0,1\}^{2n}} l(C_z)$$

$$
\begin{aligned}
&\geq 2^{-2n} \sum_{\delta(n)=1}^{n} \sum_{z \in \{0,1\}^n \& C(z|A,P,n)=2n-\delta(n)} l(C_z) \\
&\geq 2^{-2n} \sum_{\delta(n)=1}^{n} 2^{2n} \frac{1}{2^{\delta(n)}} (n - \delta(n) - O(1)) \\
&= \sum_{\delta(n)=1}^{n} \frac{n}{2^{\delta(n)}} - \sum_{\delta(n)=1}^{n} \frac{\delta(n) + O(1)}{2^{\delta(n)}} \\
&\geq n - O(1).
\end{aligned}
$$

This proves the theorem. □

7 Acknowledgements

We thank Bill Gasarch for comments, Ian Munro for discussions on related subjects, and Bill Smyth for introducing us to the paper [3].

References

[1] L. Alexanderson, L.F. Klosinski and L.C. Larson, *The William Lowell Putnam Mathematical Competition, Problems and Solutions: 1965-1984*, Mathematical Association of America, Washington, DC, 1985.

[2] L. Alonso, E. Reingold and R. Schott, Determining the majority, *Information Processing Letters* 47, 1993, pp. 253-255.

[3] L. Alonso, E. Reingold and R. Schott, The average-case complexity of determining the majority, *SIAM Journal on Computing* 26-1, 1997, pp. 1-14.

[4] G.J. Chaitin, On the number of N-bit strings with maximum complexity, *Applied Mathematics and Computation*, 59(1993), 97–100.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Proc. of 19th ACM Symp. on Theory of Computing*, 1987, pp. 1-6.

[6] D.H. Greene and D.E. Knuth, *Mathematics for the Analysis of Algorithms*, 3rd ed., Birkhäuser, Boston, MA, 1990.

[7] T. Jiang, M. Li, and P.M.B. Vitányi, New applications of the incompressibility method: Part I, to appear in *The Computer Journal.*

[8] M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, New York, 2nd Edition, 1997.

[9] M. Li and P.M.B. Vitányi, Kolmogorov complexity arguments in combinatorics, *J. Comb. Th., Series A*, 66:2(1994), 226-236. Errata, *Ibid.*, 69(1995), 183.

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

[11] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, 1997.

[12] P. O'Neil and E. O'Neil. A fast expected time algorithm for boolean matrix multiplication and transitive closure. *Information and Control* 22, 1973, pp. 132-138.

[13] M.E. Saks and M. Werman, On computing majority by comparisons, *Combinatorica* 11, 1991, pp. 383-387.