ELSEVIER

Theoretical Computer Science 271 (2002) 97–109

Theoretical Computer Science

www.elsevier.com/locate/tcs

Conditional complexity and codes

Andrej A. Muchnik *

Nizhnyaya Radishchevskaya, Institute of New Technologies Education, Street #10, Moscow 109004, Russia

Abstract

Let x and y be binary strings. We prove that there exists a program p of size about $K(x|y)$ that maps y to x and has small complexity when x is known ($K(p|x) \approx 0$). Having in mind the parallelism between Shannon information theory and algorithmic information theory, one can say that this result is parallel to Wolf–Slepian and Körner–Csiszar–Marton theorems, see (I. Csiszar and J. Körner, Information theory, Coding Theorems for Discrete Memoryless Systems, Akadémiai Kiadó, Budapest, 1981).

We show also that for any three strings x, y, z of length at most n the length of the shortest program p that maps both y and z to x (i.e., $p(y) = p(z) = x$) equals $\max(K(x|y), K(x|z) + O(\log n)$.

Keywords: Conditional Kolmogorov complexity; Minimal program; Hashing; Expanders

1. Introduction

Let x and y be binary strings. The conditional Kolmogorov complexity $K(x|y)$ of x when y is known (complexity of x relative to y) is defined as the length of the shortest program that maps y to x. The exact value of $K(x|y)$ depends on the programming system used. However, the difference is rather small: different natural definitions (original Kolmogorov definition, prefix complexity and others, see [4]) differ at most by $O(\log n)$ for strings of length n. We ignore additive $O(\log n)$-terms, therefore we can use any of these definitions.

One may expect that the shortest program that maps y into x contains exactly the information that is present in x but is missing in y. However, this is an oversimplification, because different short programs mapping y to x may have different properties, as the following example shows: Let x and y be independent random strings of length n. Consider two programs that map y to x. The first one ignores input and prints x. The second maps t to $t \oplus p$ where $\oplus$ denotes bitwise addition modulo 2 and p is chosen

* Corresponding author. Fax: +7-0959156963.

0304-3975/02/$ - see front matter
PII: S0304-3975(01)00033-0

in such a way that $y \oplus p = x$ (i.e., $p = x \oplus y$). These two programs have nothing in common (binary strings x and p have no common information).

We prove that for any x and y there exists a program p that (1) p maps y to x, (2) p has (almost) minimal length ($|p| \approx K(x|y)$, where $|p|$ stands for the length of p) and (3) p is simple relative to x ($K(p|x) \approx 0$). The exact statement is given below. (In our example the first program has these properties but not the second one.) This statement provides a partial answer to the question posed in [1] (Remark 3.8, for details see [6]).

The same method can be applied for several conditions. We prove that for any three strings of length at most n there exists a program p of length $\max(K(x|z), K(y|z)) + O(\log n)$ such that $p(y) = x$ and $p(z) = x$. This problem was considered by Gorbunov [3] who proved that it is impossible to prove similar result with $O(\log(K(x|y) + K(x|z)))$-precision. Let us mention also that we can obtain one of the results from [1] (about the program that maps y to z and vice versa) as a corollary if we let $x = \langle y, z \rangle$.

2. Programs and codes

To avoid mentioning a specific programming system, we speak about codes instead of programs. We say that the string p is a *code for x when y is known* if $K(x|y, p) \approx 0$ (here $K(x|y, p)$ stands for the conditional complexity of x when pair $\langle y, p \rangle$ is known). It is easy to see that

- if p is a code for x when y is known then the complexity of p (and therefore the length of p) is at least $K(x|y)$;
- for any strings x and y there is a string p such that p is a code for x when y is known and $|p| \approx K(x|y)$.

The exact statements are:

- $K(p) \geqslant K(x|y) - K(x|y, p) - O(\log n)$ for any strings x, y, p of length at most n;
- for any strings x and y of length at most n there exists a string p such that $|p| = K(x|y) + O(\log n)$ and $K(x|y, p) = O(\log n)$.

Indeed, the first inequality becomes obvious if we rewrite it as follows:

$$K(x|y) \leqslant K(x|y, p) + K(p) + O(\log n)$$

(a program that produces p and a program that maps y, p to x can be combined into a program that produces x from y; the overhead while combining these programs is $O(\log n)$).

On the other hand, if p is the shortest program that maps y to x then $|p| = K(x|y)$ by definition and $K(x|y, p) = O(\log n)$ since we get x from y and p just applying p to y.

A code p for x when y is known can be easily transformed into a program that maps y to x and has approximately the same length as p. Indeed, since $K(x|y, p)$ is small, there exists a short program that maps (y, p) into x. Fix the second argument in that program being equal to p. We get a program mapping y to x whose length is about $|p|$ (most part of it is occupied by a constant p).

3. Minimal codes

Let us mention that any string x is a code for x when (any) y is known. Therefore, one can find at least one code for x when y is known that is simple relative to x. It turns out that this property can be combined with minimal length requirement: there is some code p for x when y is known that has length $K(x|y)$ and at the same time is simple relative to x.

Let us note that a weaker requirement "p is simple relative to x, y" can be satisfied easily. Try in parallel all the programs of length $K(x|y)$ and look for a program that maps y to x; the first program having this property will be the code we are looking for, since we need only $O(\log n)$ bits (the value of $K(x|y)$) to produce this program when x and y are known.

Theorem 1. *Let x and y be arbitrary strings of length less than n. Then there exists a string p of length $K(x|y)$ such that $K(p|x) = O(\log n)$ and $K(x|p, y) = O(\log n)$.*

(The constants in $O(\log n)$-notation do not depend on n, x, y.)

4. Proof sketch

The proof uses "hashing" (similar ideas are used in [2] in polynomial-time situation). Let X be the set of all strings of length less than n. Let m be the conditional complexity $K(x|y)$. Consider the set P of all strings of length m. Our hash function is of type $\chi : X \to P$. (We discuss later how to construct such a function and what properties of this function are required.) We use the hash value $p = \chi(x)$ as a code for x. Assume that the complexity of the hash function itself is small; then $p = \chi(x)$ has small complexity $K(p|x)$. To get an upper bound for $K(x|y, p)$ we need to reconstruct x from y and p. We generate all strings x' of length less than n such that $K(x'|y) \leqslant m$ (by trying in parallel all programs of length at most m and applying them to y). Let X_y be the set of all such x'. For each $x' \in X_y$ we calculate $\chi(x')$. If we are lucky and x is the only element of X_y that has hash value p, then x can be reconstructed from y, p, n, m, χ and $K(x|y, p)$ is small (we need only a small number of bits to determine n, m and χ).

However, this is only a rough sketch; to make this construction work we need to make several concessions described in the next section.

5. Modifications

5.1. Several hash functions

We consider several hash functions $\chi_1, \ldots, \chi_N$ instead of one. The number N is bounded by a polynomial in n, therefore any hash value $\chi_i(x)$ is simple relative to x. Indeed, if x is known, it is enough to specify i to determine $\chi_i(x)$ (we assume that

the family $\chi_1,\ldots,\chi_N$ is simple), and i is of size $\log N = \log \mathrm{poly}(n) = \mathrm{O}(\log n)$ bits. If there is at least one function χ_i that separates the given x from all other $x' \in X_y$ (i.e., $\chi_i(x) \neq \chi_i(x')$ for all $x' \in X_y$ such that $x' \neq x$), this is enough. Indeed, x can be reconstructed from y and $p = \chi_i(x)$ if we know m, n, i (we also need to know the family of hash functions which is assumed to be simple).

5.2. Several preimages

One more modification is the following one. In fact, we do not need to have unique $x' \in X_y$ such that $\chi_i(x') = p$. It is enough for us that the number of such elements x' is bounded by a polynomial in n. Indeed, in this case we can add the ordinal number of x (in the enumeration of all $x' \in X_y$ such that $\chi_i(x') = p$) to the information needed to reconstruct x from y and p.

5.3. Not all $x \in X_y$ are good

As we have seen, it is enough to construct a simple family $\chi_1,\ldots,\chi_N$ of hash functions such that N is bounded by a polynomial in n and for any $x \in X_y$ (recall that X_y is the set of all strings x such that $|x| < n$ and $K(x|y) \leqslant m$) there exists i such that the cardinality of the set

$$\{x' \in X_y \mid \chi_i(x') = \chi_i(x)\}$$

is bounded by a polynomial.

Our last concession is the following one: we do not need that such i exists for all $x \in X_y$. It is enough if such i exists for all $x \in X_y$ except for a small fraction. Indeed, consider the set of all "bad" values of x (that share the same hash value with many other elements for any hash function χ_i). Bad values of x can be enumerated if y, m, n are given (and the hash family is known). Therefore, if there are only few of them, all bad values have small complexity relative to y (their complexity relative to y is less than m). Since $K(x|y) = m$ by assumption (here x is the string we started with), the string x is "good".

6. How to construct hash functions

After all these concessions we need to show that a family of hash functions with required properties does exist. It is convenient to represent this family by a bipartite graph that has strings from X on the left and strings from P on the right. Each string $x \in X$ is connected by edges with N strings $\chi_1(x),\ldots,\chi_N(x)$. (If some values coincide, there are several edges connecting two vertices.)

We are interested in the restriction of this graph onto X_y; all other left vertices can be removed. Which vertices in X_y are good (in the sense described above)? If some vertex x has a right neighbor p that has few left neighbors (in the restricted graph) then x is guaranteed to be good, because x shares hash value p with small number

of other elements of X_y. (For simplicity we do not care which hash function gives p. In fact there is nothing bad if $\chi_i(x)=\chi_j(x')$ for $i\neq j$, but we do not take that into account.)

Let us summarize: a right vertex p is "bad" if it has many left neighbors and "good" otherwise. A left vertex is "safe" (is guaranteed to be good) if it has at least one good right neighbor. If it is not the case (x has no good neighbors) we say that x is "dangerous". (As we have said, the dangerous vertex can be in fact good, but we do not take it into account.)

Now assume that the (unrestricted) graph has the following expander-like property: for any set $S\subset X$ of size slightly smaller than 2^m the set S' of all right neighbors of all vertices from S has at least $|S|$ elements. This property remains true if we restrict graph to $X_y\times P$. It guarantees that if the number of bad vertices on the right is small then the number of dangerous vertices on the left is also small (recall that all neighbors of dangerous vertices are bad). The number of bad right-hand side vertices does not exceed the ratio

$$\frac{\text{number of edges in the restricted graph}}{\text{degree of a bad vertex}}.$$

The total number of edges in the restricted graph is $|X_y|\times N$ (each left-hand side vertex has N outgoing edges); the degree of any bad vertex is big by definition.

The only remaining problem is to construct the family of hash functions having expander-like property. This is done in the next section. We prove (by a probabilistic argument) that such a family exists. Then we can try all families (for given n and m) until we find a family with the required property. The first family is determined by m and n and therefore has complexity $O(\log n)$.

7. Estimating probabilities

Lemma 1. *Let m and n be positive integers and $m\leqslant n$. Let X and P be finite sets of cardinalities 2^n-1 and 2^m. Let positive numbers N and u satisfy the inequality*

$$\frac{u}{2^m}<2^{-(m+n)/N}.$$

Then there exists a family of N functions

$$\chi_1,\ldots,\chi_N : X\to P$$

with the following property: for any set $U\subset X$ having cardinality u the set of all $\chi_i(u)$ for all $u\in U$ and all $i\in\{1,\ldots,N\}$ contains at least u elements.

Proof. Let us prove that randomly chosen functions $\chi_1,\ldots,\chi_N$ have the required property with positive probability.

There is at most 2^{nu} possibilities for the set U since there are $(2^n-1)^u$ sequences of length u whose elements belong to X. (A stronger bound would take into account

that different sequences can lead to the same set U but this bound is enough for our purposes.)

For a fixed set U let us estimate the probability of the following event: all values $\chi_i(u)$ for all $u \in U$ and for all i belong to a fixed set $V \subset P$ of cardinality $|u-1|$. This probability is less than $(u/2^m)^{Nu}$ since Nu random values are independent and each belongs to V with probability less than $u/2^m$. The number of different sets V is at most 2^{mu}. Therefore, the total probability of the bad outcome (for some U all Nu values belong to some set V of cardinality $u-1$) is less than

$$2^{mu} \cdot 2^{nu} \cdot \left(\frac{u}{2^m}\right)^{Nu}$$

and we need this bound to be less than 1. It is easy to see that this requirement is equivalent to our assumption. □

We use Lemma 1 with

$$\frac{u}{2^m} < 1/n.$$

In this case we may let $N = 2\lceil n/\log n\rceil$, and each hash function χ_i can be specified by $O(\log N) = O(\log n)$ bits (it is enough to specify n, m, and i).

8. Estimating the number of neighbors

In this section we provide bounds for the number of bad and dangerous vertices. Recall that we consider strings x and y of length less than n and $K(x|y) = m$. By X we denote the set of all strings of length less than n; by P we denote the set of all strings of length m (used as hash values).

The set X contains $2^n - 1$ elements; the set P contains 2^m elements. We apply Lemma 1 with $N = 2\lceil n/\log n\rceil$ and some value of u that will be specified later and get a bipartite graph connecting strings with their hash values.

Let x and y be fixed. A hash value $p \in P$ is "bad" if it has more than n^c neighbors in the set X_y that contains all strings $x' \in X$ such that $K(x'|y) \leqslant m$. (The exact value of a constant c will be specified later.) As we have seen, the number of bad hash values is less than

$$|X_y| \cdot N/n^c < 2 \cdot 2^m \cdot 2n/n^c = \left(\frac{4}{n^{c-1}}\right) \cdot 2^m$$

(recall that $|X_y| < 2 \cdot 2^m$ since the number of programs of length at most m is less than $2 \cdot 2^m$, and $N < 2n$). Let u be the (integer part of the) right-hand side of this inequality. We assume that $c \geqslant 4$, therefore $u/2^m < 1/n$ and we can apply Lemma 1 with $N = \lceil 2n/\log n\rceil$. Lemma guarantees that the number of dangerous vertices in X_y is less than u (otherwise u dangerous vertices would have less than u neighbors, since the number of bad vertices is less than u).

Therefore, the complexity of any dangerous vertex when y is known does not exceed

$$\mathrm{O}(\log n) + \log\left(\frac{4}{n^{c-1}} \cdot 2^m\right) = \mathrm{O}(\log n) + m - (c-1)\log n.$$

Indeed, to specify a dangerous vertex it is enough to fix m and n (after that the family of hash function is known) and then specify its ordinal number in the enumeration of all dangerous vertices (for given y). We see that if c is big enough (in fact $c=4$ suffices) and n is big enough the complexity of any dangerous vertex is less than m. Therefore, the initial vertex x is not dangerous and has a good neighbor p. Then $K(p|x)=\mathrm{O}(\log n)$ (to get p from x we specify i such that $\chi_i(x)=p$) and $K(x|y,p)=\mathrm{O}(\log n)$ (since x is one of at most n^c neighbors of p in X_y, we can specify x by its ordinal number in the list of all neighbors which can be generated if we know y, p, n, m; recall that n and m contain $\mathrm{O}(\log n)$-bits.)

Theorem 1 is proved. □

Remark. The code p is not defined uniquely by the properties mentioned in Theorem 1. Indeed, assume that y and z are random independent strings of the same length and $x=yz$. Then both y and $y\oplus z$ can be used as p but have no mutual information.

9. Codes for several conditions

The construction of codes using hash functions has other applications. Now we use it to construct a code that works for several conditions. Assume that we have strings x, y, z of length less than n and $K(x|y)=K(x|z)=m$. It turns out that there exists a string p of length m such that

$$K(p|x) = \mathrm{O}(\log n),$$

$$K(x|p,y) = \mathrm{O}(\log n),$$

$$K(x|p,z) = \mathrm{O}(\log n).$$

This p is a code for x when y is known and at the same time a code for x when z is known. (Moreover, p is simple relative to x, but even without this requirement it is not trivial to find such a string p.)

The proof is given below (for a more general situation); let us explain informally how it works.

We use the family of hash functions

$$\chi_1, \ldots, \chi_N : X \to P,$$

where X is the set of all strings of length less than n and $P=\mathbb{B}^m$ is the set of all strings of length m. The code p (that we are looking for) is $\chi_i(x)$ for some i. The family of hash functions is represented as a bipartite graph on $X\times P$. This graph is the same for both y and z. However, the sets X_y and X_z (that contain strings of complexity

at most m with respect to y and z) are different, so we have to consider two restrictions of that graph (onto $X_y \times P$ and $X_z \times P$). Therefore, we get two types of "bad" vertices (having many neighbors in X_y and X_z). The proof of Theorem 1 shows that we can find y-good neighbor of x as well as z-good neighbor, but if these neighbors are different, we are in trouble.

To overcome this difficulty we prove that most neighbors of x are y-good and that most neighbors of x are z-good. Then we conclude that there exists a neighbor p that is both y- and z-good. This p is the code we are looking for.

To prove that most neighbors are good we need to change the definition of a dangerous vertex. We say that a vertex in X is y-dangerous (z-dangerous) if the fraction of y-good (z-good) vertices among its neighbors does not exceed $\frac{1}{2}$. We shall prove that x is neither y-dangerous nor z-dangerous.

Expander-like property should be modified accordingly: we require now that for any set $Z \subset P$ of cardinality u the set

$$\{x \in X \mid \text{at least half of the neighbors of } x \text{ belong to } Z\}$$

has less than u elements. (The requirements for u will be discussed later.) Note that this requirement is stronger than the previous one: replacing "at least half of" by "all" we get the previous requirement.

The last adjustment is needed when complexities $K(x|y)$ and $K(x|z)$ differ. Assume that $K(x|y) = a$ and $K(x|z) = b$ and $a > b$. Then one can find two codes p (for x when y is known) and q (for x when z is known) such that $|p| = a$, $|q| = b$ and q is a prefix of p. (Both p and q are simple when x is known.)

To prove that p and q exist we consider hash functions with values in $\mathbb{B}^a$ as well as their shortened versions that contain only first b bits. We require that both the "long" and "shortened" families have expander-like properties. (In fact, it is convenient to have expander-like properties for prefixes of any length, see the next section.)

Using codes p and q one can easily construct a program M that maps both y and z into x and has length

$$\max(K(x|y), K(x|z)) + O(\log n).$$

Program M finds out which of the strings y and z is given as input (since y and z differ at some place, we can distinguish between them if we know this place; this knowledge requires $O(\log n)$ bits). Then M applies one of the programs that maps p, y or q, z into x. (Strings p and q are "compiled-in" constants in M; it is enough to have the longer one in full and the length of the shorter one.)

10. Probability estimates for several codes

To construct the common code for several conditions we need to generalize Lemma 1. We denote by $\mathbb{B}^t$ the set of all binary strings of length t. If $x \in \mathbb{B}^t$ and $s \leqslant t$ then we denote by $[x]_s$ the prefix of x having length s.

Lemma 2. *Let n and N be positive integers and ε be a positive real number. Assume that*

$$n\,2^{N+2n+1}\varepsilon^{N/2} < 1.$$

Then there exists a family of functions

$$\chi_1,\ldots,\chi_N : \mathbb{B}^n \to \mathbb{B}^n$$

with the following property: for any $m \in \{1,\ldots,n\}$ and for any set $Q \subset \mathbb{B}^m$ having at most $\varepsilon 2^m$ elements the set of all $x \in \mathbb{B}^n$ such that

$$[\chi_i(x)]_m \in Q \quad \textit{for at least half of values of } i \in \{1,\ldots,N\}$$

contains less than $|Q|$ elements (where $|Q|$ stands for the cardinality of Q).

Proof. We show that for randomly chosen functions $\chi_1,\ldots,\chi_N$ (values of $\chi_i(x)$ are independent for all i and for all x and are uniformly distributed in $\mathbb{B}^n$) the required property is violated with probability less than 1. To estimate this probability let us fix some $m \leqslant n$, some $u \leqslant \varepsilon 2^m$ and some sets $P \subset \mathbb{B}^n$ and $Q \subset \mathbb{B}^m$ such that $|P| = |Q| = u$. Consider the following event: for any $x \in P$ at least half of the values

$$\chi_1(x),\ldots,\chi_N(x)$$

belongs to Q. The probability of this event is bounded by $(2^N\varepsilon^{N/2})^u$ since for any $x \in P$ for any of at most 2^N subsets T of the set $\{1,\ldots,N\}$ with $|T| = \lceil N/2 \rceil$ the probablity of the event "$\chi_i(x) \in Q$ for all $i \in T$" does not exceed $\varepsilon^{N/2}$ (m-prefixes of uniformly distributed independent strings are independent and uniformly distributed).

Summing over all T, we get (for a given x) the bound $2^N\varepsilon^{N/2}$. Events for different x are independent, and we get the bound mentioned above.

It remains to show that

$$\sum_{m=1}^{n}\sum_{u=1}^{\varepsilon 2^m}\sum_{P\subset\mathbb{B}^n,|P|=u}\sum_{Q\subset\mathbb{B}^m,|Q|=u}(2^N\varepsilon^{N/2})^u < 1.$$

The number of different sets P does not exceed 2^{nu} (the number of sequences of length u composed from elements of $\mathbb{B}^n$); the number of different sets Q does not exceed 2^{mu}. Therefore, it is enough to show that

$$\sum_{m=1}^{n}\sum_{u=1}^{\varepsilon 2^m} 2^{nu}\,2^{mu}\,(2^N\varepsilon^{N/2})^u < 1.$$

The internal sum is a geometric progression. Lemma's assumption guarantees that the quotient of this progression is less than $\frac{1}{2}$, therefore its sum is bounded by $2\cdot$(the first term). Now we get rid of u; remaining sum has n terms and (since $2^{mu} \leqslant 2^{nu}$) it is enough to prove that

$$n\,2\cdot 2^{2n}\,2^N\,\varepsilon^{N/2} < 1$$

which is exactly our assumption. Lemma 2 is proved. □

We use this lemma for $\varepsilon = 1/n$. In this case the condition can be rewritten as

$$n2^{N+2n+1} < n^{N/2}$$

or

$$\log n + N + 2n + 1 < (N/2)\log n.$$

We see that for big enough constant d and for $N = \lceil dn/\log n \rceil$ this condition is satisfied for all n.

11. Codes that are prefixes of each other

Theorem 2. *Let x, y, z be strings of length less than n. Then there exist strings p and q such that*

$$|p| = K(x|y), \qquad |q| = K(x|z),$$

the shorter of strings p and q is a prefix of the longer one,

$$K(p|x) = \mathrm{O}(\log n), \qquad K(q|x) = \mathrm{O}(\log n),$$

$$K(x|p, y) = \mathrm{O}(\log n), \qquad K(x|q, z) = \mathrm{O}(\log n).$$

(As usual, constants in $\mathrm{O}(\log n)$-notation do not depend on n.)

Proof. Using Lemma 2, we construct N hash functions

$$\chi_1, \ldots, \chi_N : X \to \mathbb{B}^n,$$

where X is the set of all strings of length less that n, $\varepsilon = 1/n$ and $N = \mathrm{O}(n/\log n)$.

We may assume that the complexity of the family $\chi_1, \ldots, \chi_N$ is $\mathrm{O}(\log n)$ (the first family satisfying the required property can be effectively found for a given n if we try all families in some prescribed ordering).

Assume that $K(x|y) = a$ and $K(x|z) = b$. If we take into account only first a (or b) bits of hash values, we get N mappings of type $X \to \mathbb{B}^a$ (or $X \to \mathbb{B}^b$). These mappings determine a bipartite graph on $X \times \mathbb{B}^a$ (and $X \times \mathbb{B}^b$). Each vertex on the left has n adjacent edges (some of them may lead to the same vertex on the right). We need the restrictions of these graphs onto $X_y \times \mathbb{B}^a$ and $X_z \times \mathbb{B}^b$, where X_y is the set of all strings x' of length less than n such that $K(x'|y) \leqslant a$ and X_z is the set of strings x' of length less than n such that $K(x'|z) \leqslant b$.

A vertex in $\mathbb{B}^a$ is good if it has at most n^c neighbors in X_y; a vertex in $\mathbb{B}^b$ is good if it has at most n^c vertices in X_z. (The exact value of the constant c which should be large enough will be specified later.)

The number of bad vertices in $\mathbb{B}^a$ (in $\mathbb{B}^b$) does not exceed

$$2N \cdot 2^a/n^c \quad [2N \cdot 2^b/n^c],$$

since any bad vertex has degree at least n^c and the number of edges in the restricted graph is $|X_y| \cdot N$ [$|X_z| \cdot N$] and $|X_y| < 2 \cdot 2^a$ [$|X_z| < 2 \cdot 2^b$]. (Recall that the number of programs of length at most m is $2 \cdot 2^m - 1$.)

A vertex in X_y is dangerous if at least half of its neighbors in $\mathbb{B}^a$ are bad. Lemma 2 guarantees that the number of dangerous vertices is less than

$$2N \cdot 2^a/n^c$$

(we assume that c is large enough, therefore the number of bad vertices is less than $\varepsilon 2^a$, where $\varepsilon = 1/n$, so Lemma 2 can be applied). Since we can enumerate dangerous vertices effectively (if n, a, and y are known), the complexity of any dangerous vertex with respect to y does not exceed

$$\log(2N \cdot 2^a/n^c) + \mathrm{O}(\log n) \leqslant a - c \log n + \mathrm{O}(\log n)$$

(recall that $N = \mathrm{O}(n/\log n)$ where hidden constant in O-notation does not depend on c). If c is large enough, all dangerous vertices have complexity (relative to y) less than a, therefore the initial vertex x is not dangerous. Therefore, more than $N/2$ of hash values of x have prefixes of length a that are good elements of $\mathbb{B}^a$.

For the same reasons more than $N/2$ hash values of x have good prefixes of length b. Therefore, there is a hash value $\chi_i(x)$ that has good prefixes of length a and b. Let $p = [\chi_i(x)]_a$ and $q = [\chi_i(x)]_b$. Then p is a prefix of q or vice versa. Moreover, $K(p|x) = \mathrm{O}(\log n)$ and $K(q|x) = \mathrm{O}(\log n)$ since p and q are determined by x, i, a, b. Finally, $K(x|p, y) = \mathrm{O}(\log n)$, because for any p and y we can enumerate all neighbors of p in X_y (note that we know $a = |p|$) and to specify x it is enough to give its ordinal number in this enumeration that is at most n^c and is $\mathrm{O}(\log n)$-long. For the same reasons $K(x|q, z) = \mathrm{O}(\log n)$.

Theorem 2 is proved. □

12. Number of hash functions

The following theorem shows that the number of hash-functions used in Theorem 1 (and therefore in Theorem 2) cannot be reduced significantly.

Theorem 3. *For any positive real c, any string x and any set P of cardinality less than $K(x)/c \log K(x)$ whose elements are strings of length $\lceil K(x)/2 \rceil$ there exists a string y such that*:

(i) $K(y) < \mathrm{O}(K(x))$,
(ii) $K(x|y) < K(x)/2$,
(iii) $(\forall p \in P) K(x|y, p) > (c - \mathrm{O}(1)) \log K(x)$.

Proof. Assume that $P = \{p_1, \ldots, p_j\}$ and $j < K(x)/c \log K(x)$. By v_i we denote a prefix of p_i of length $\lfloor c \log K(x)/3 \rfloor$. Let w be the concatenation of strings $v_1, \ldots, v_j$. Then

$$|w| < j \lfloor c \log K(x)/3 \rfloor < K(x)/3.$$

Consider the value $K(x|w,z)$ for different prefixes z of string x. As $|z|$ changes by 1, the value of $K(x|w,z)$ changes by O(1). We have

$$K(x|w,\Lambda) > K(x) - K(w) - \mathrm{O}(\log K(x)) > K(x) - |w| - \mathrm{O}(\log K(x)).$$

Taking into account the estimate for $|w|$, we get $K(x|w,\Lambda) > K(x)/2$ for sufficiently large $K(x)$. On the other hand, $K(x|w,x) < \mathrm{O}(1)$. Therefore, there is a prefix z_0 of the string x such that $K(x)/2 > K(x|w,z_0) > K(x)/2 - \mathrm{O}(1)$. Let y be the pair $\langle w,z_0\rangle$ (its code). Let us check (i):

$$K(y) < K(w) + K(z_0) + \mathrm{O}(\log K(x)) < K(x)/3 + K(x) + \mathrm{O}(\log K(x)).$$

Now let us check (ii):

$$K(x|y) = K(x|w,z_0) < K(x)/2.$$

Finally, we have to check (iii). For each i the string y contains "significant" information about p_i:

$$K(p_i|y) < K(x)/2 - c\log K(x)/3 + \mathrm{O}(\log K(x)).$$

(When w is known, it is enough to specify the position of the substring v_i in the string w and the remaining part of p_i after v_i to specify p_i.) This inequality implies (for every i) that

$$\begin{aligned} K(x|y,p_i) &> K(x|y) - K(p_i|y) - \mathrm{O}(\log K(x)) \\ &> (K(x)/2 - \mathrm{O}(1)) - (K(x)/2 - c\log K(x)/3 + \mathrm{O}(\log K(x))) - \mathrm{O}(\log K(x)) \\ &= (c - \mathrm{O}(1))\log K(x). \end{aligned}$$

Theorem is proved. □

13. Concluding remarks

- Theorem 2 and its proof could be easily generalized to the case of three (or polynomially many) conditions instead of two.
- Theorems 1 and 2 give us a code p of string x when y (or z) is known. We assumed that all the three strings x, y, z have length less than n. However, our proof does not use any assumption about y and z. Moreover, one can replace the assumption $|x| < n$ by $K(x) < n$.

Acknowledgements

The proofs of Theorems 1–3 were explained on the Kolmogorov seminar (September 1999). The preliminary short version of this paper was published in [5]. Author is grateful to Alexander Shen for his help in elaborating it; this paper follows his suggestions.

References

[1] C.H. Bennett, P. Gács, M. Li, P.M.B. Vitanyi, W.H. Zurek, Information distance, IEEE Trans. Inform. Theory 44 (3) (1998) 1407–1423.
[2] L. Fortnow, S. Laplante, Nearly optimal language compression using extractors, 15th Ann. Symp. Theoretical Aspects of Computer Science, Paris, France, 25–27 February 1998, Lecture Notes in Computer Science, Vol. 1373, Springer, Berlin, 1998, pp. 84–93.
[3] K.Yu. Gorbunov, On a complexity of the formula $((A \vee B) \rightarrow C)$, Theoret. Comput. Sci. 207 (1998) 383–386.
[4] M. Li, P. Vitányi, An introduction to Kolmogorov Complexity and Its Applications, 2nd Edition, Springer, Berlin, 1997.
[5] A. Muchnik, A. Semenov, Multi-conditional descriptions and codes in kolmogorov complexity, Electronic Colloquium on Computational Complexity, Report No. 15, January 27, 2000.
[6] N.K. Vereshchagin, M.V. Vyugin, Independent minimum length programs to translate between given strings, Theoret. Comput. Sci., this issue.