

Article

## Using Information Theory to Study Efficiency and Capacity of Computers and Similar Devices

**Boris Ryabko**

Institute of Computational Technology of Siberian Branch of Russian Academy of Science, Siberian State University of Telecommunications and Informatics, Novosibirsk 630000, Russia

\* Author to whom correspondence should be addressed; E-Mail: boris@ryabko.net;  
Tel.: +7 383 2698204; Fax: +7 383 2698203.

Received: 6 July 2010; in revised form: 22 July 2010 / Accepted: 5 August 2010 /

Published: 12 August 2010

---

**Abstract:** We address the problem of estimating the efficiency and capacity of computers. The main goal of our approach is to give a method for comparing the capacity of different computers, which can have different sets of instructions, different kinds of memory, a different number of cores (or processors), *etc.* We define efficiency and capacity of computers and suggest a method for their estimation, which is based on the analysis of processor instructions and their execution time. How the suggested method can be applied to estimate the computer capacity is shown. In particular, this consideration gives a new look at the organization of the memory of a computer. Obtained results can be of some interest for practical applications.

**Keywords:** computer capacity; computer efficiency; information theory; Shannon entropy; channel capacity

---

### 1. Introduction

Nowadays, computer science includes many natural approaches and models which give a possibility to estimate the power of different computational models and the complexity of different computational tasks. In particular, the field of computational complexity is concerned with studying the amount of resources (such as running time, memory usage, or program length) required for solving computational

tasks (such as decision problems, search problems, or optimization problems); see, e.g., [1] for a review. The related field of algorithmic information theory is based on a model where the task complexity is estimated by the length of a computer program [2,3].

Much less is known about how to compare the performance of real computers that can have different clock rates, different sets of instructions, different kinds of memory, different number of cores (or processors), *etc.* In fact, currently the performance of different computers (and supercomputers) is estimated only experimentally by solving benchmark sets of different computational tasks and measuring the total time of calculation.

In what follows we will consider models of real computers. One such computer is described in details by D. Knuth [4]. (It is worth noting that this computer has been implemented after it was described theoretically.)

We address the problem of what are the efficiency (or performance) and capacity of a computer and how they can be estimated. The main goal of our approach is to give a method to compare the capacity of computers. For this purpose we suggest definitions of the computer efficiency and capacity and methods of their estimation. We will mainly consider computers, but our approach can be applied to all devices which contain processors, memories and instructions. (Among those devices we mention mobile telephones and routers.) Besides, we describe a method for estimating the computer capacity and apply it to several examples which are of some theoretical and practical interest. In particular, this consideration gives a new look at the organization of computer memory.

The suggested approach is based on the concept of Shannon entropy, the capacity of a discrete noiseless channel and some other ideas of C. Shannon [5] that underly information theory.

## 2. The Efficiency and Capacity of Computers

### 2.1. The Basic Concepts and Definitions

In order to make the paper self-contained, we define a simple model of a real computer. A detailed description can be found in [4]. A computer consists of a set of instructions  $I$  and an accessible memory  $M$ . It is important to note that any instruction  $x \in I$  contains not only its name (say, JUMP), but memory addresses and indexes of registers. For example, all instructions JUMP which deal with different memory addresses are contained in  $I$  as different instructions.

We suppose that at the initial moment there is a program and data which can be considered as binary words  $P$  and  $D$ , located in the memory of a computer  $M$ . In what follows we will call the pair  $P$  and  $D$  a computer task. A computer task  $\langle P, D \rangle$  determines a certain sequence of instructions  $X(P, D) = x_1x_2x_3\dots$ ,  $x_i \in I$ . (It is supposed that an instruction may contain an address of a memory location, the index of a register, *etc.*) For example, if the program  $P$  contains a loop which will be executed ten times, then the sequence  $X$  will contain the body of this loop repeated ten times. We say that two computer tasks  $\langle P_1, D_1 \rangle$  and  $\langle P_2, D_2 \rangle$  are different, if the sequences  $X(P_1, D_1)$  and  $X(P_2, D_2)$  are different.

It is important to note that we do not suppose that all possible sequences of instructions are allowable. In principle, there can be sequences of instructions which are forbidden. For example, it is possible that some pairs of instructions are forbidden, *etc.* In other words, it is possible that sequences of

instructions should satisfy some limitations (or some rules). We define the set of all allowable sequences of instructions by  $S_C$ . So, any computer task can be presented as a sequence of instructions from  $S_C$ . Moreover, the opposite is also true: any sequence of instructions from  $S_C$  can be considered as a computer task. Indeed, using a so-called assembly language any sequence of instructions from  $S_C$  can be presented as a computer program, see, for example, [6]. (It is worth noting that some sequences can be meaningless and two different sequences of instructions can be equal. This situation is typical for any language when someone considers its capacity, because a language contains synonyms, etc.)

Let us denote the execution time of an instruction  $x$  by  $\tau(x)$ . For the sake of simplification, we suppose that all execution times  $\tau(x), x \in I$ , are integers and the greatest common divisor of  $\tau(x), x \in I$ , equals 1. (This assumption is valid for many computers if the time unit equals a so-called clock rate and there are instructions whose executed time is one unit, i.e.,  $\tau(x) = 1$ ; see [6].) In this paper this assumption gives a possibility to use  $\lim$  instead of  $\limsup$ , when the capacity will be defined.)

Naturally, the execution time  $\tau(X)$  of a sequence of instructions  $X = x_1x_2x_3\dots x_t$  is given by

$$\tau(X) = \sum_{i=1}^t \tau(x_i)$$

Denote the number of different problems, whose execution time equals  $T$ , by  $\nu(T)$  and let  $N(T)$  be the size of the set of all sequences of instructions, whose execution time equals  $T$ , i.e.,

$$N(T) = |\{X : \tau(X) = T\}| \quad (1)$$

The key observation is as follows:

$$\nu(T) = N(T) \quad (2)$$

Hence,

$$\log \nu(T) = \log N(T) \quad (3)$$

(Here and below  $T$  is an integer,  $\log x \equiv \log_2 x$  and  $|Y|$  is the number of elements of  $Y$  if  $Y$  is a set, and the length of  $Y$  if  $Y$  is a word.) In other words, the total number of computer tasks executed in time  $T$  is equal to (1). Basing on this consideration we give the following definition.

**Definition 1** Let there be a computer with a set of instructions  $I$  and let  $\tau(x)$  be the execution time of an instruction  $x \in I$ . The computer capacity  $C(I)$  is defined as follows:

$$C(I) = \lim_{T \rightarrow \infty} \frac{\log N(T)}{T} \quad (4)$$

where  $N(T)$  is defined in (1).

**Claim 1** The limit (4) exists if  $I$  is finite, execution times  $\tau(x), x \in I$  are integers and the greatest common divisor of  $\tau(x), x \in I$ , equals 1.

*Proof* is given in Appendix.

The next question to be investigated is the definition of the efficiency of computer (or performance), when a computer is used for solving problems of a certain kind. For example, one computer can be a Web server, another can be used for solving differential equations, etc. Certainly, the computer efficiency

depends on the problems the computer has to solve. In order to model this situation we suggest the following approach: there is an information source which generates a sequence of computer tasks in such a way, that the computer begins to solve each next task as soon as the previous task is finished. We will not deal with a probability distribution on the sequences of the computer tasks, but consider sequences of computer instructions, determined by sequences of the computer tasks, as a stochastic processes. In what follows we will consider the model when this stochastic process is stationary and ergodic, and we will define the computer efficiency for this case.

A natural question is the applicability of this model. The point is that modern computers are commonly used for solving several computer tasks in parallel, which is why the sequence of executed instructions is a mixture of quite a large number of subsequences. So, in some natural situations, this sequence can be modeled by a stationary ergodic source.

The definition of efficiency will be based on results and ideas of information theory, which we introduce in what follows. Let there be a stationary and ergodic process  $z = z_1, z_2, \dots$  generating letters from a finite alphabet  $A$  (the definition of stationary ergodic process can be found, for example, in [7]). The  $n$ -order Shannon entropy and the limit Shannon entropy are defined as follows:

$$h_n(z) = -\frac{1}{n+1} \sum_{u \in A^{n+1}} P_z(u) \log P_z(u)$$

$$h_\infty(z) = \lim_{n \rightarrow \infty} h_n(z) \quad (5)$$

where  $n \geq 0$ ,  $P_z(u)$  is the probability that  $z_1 z_2 \dots z_{|u|} = u$  (this limit always exists, see [5,7]). We will consider so-called i.i.d. sources. By definition, they generate independent and identically distributed random variables from some set  $A$ . Now we can define the computer efficiency.

**Definition 2** *Let there be a computer with a set of instructions  $I$  and let  $\tau(x)$  be the execution time of an instruction  $x \in I$ . Let this computer be used for solving such a randomly generated sequence of computer tasks, that the corresponding sequence of the instructions  $z = z_1 z_2 \dots, z_i \in I$ , is a stationary ergodic stochastic process. Then the efficiency is defined as follows:*

$$c(I, z) = h_\infty(z) / \sum_{x \in I} P_z(x) \tau(x) \quad (6)$$

where  $P_z(x)$  is the probability that  $z_1 = x, x \in I$ .

Informally, the Shannon entropy is a quantity of information (per letter), which can be transmitted and the denominator in (6) is the average execution time of an instruction.

More formally, if we take a large integer  $T$  and consider all  $T$ -letter sequences  $z_1 \dots z_T$ , then, for large  $T$ , the number of “typical” sequences will be approximately  $2^{Th_\infty(z)}$ , whereas the total execution time of the sequence will be approximately  $T \sum_{x \in I} P_z(x) \tau(x)$ . (By definition of a typical sequence, the frequency of any word  $u$  in it is close to the probability  $P_z(u)$ . The total probability of the set of all typical sequences is close to 1.) So, the ratio of  $\log(2^{Th_\infty(z)})$  and the average execution time will be asymptotically equal to (6), if  $T \rightarrow \infty$ . A rigorous proof can be obtained basing on methods of information theory; see [7]. We do not give it, because definitions do not need to be proven, but mention that there are many results about channels which transmit letters of unequal duration [8].

## 2.2. Methods for Estimating the Computer Capacity

Now we consider the question of estimating the computer capacity and efficiency defined above. The efficiency, in principle, can be estimated basing on statistical data, which can be obtained by observing a computer which solves tasks of a certain kind.

The computer capacity  $C(I)$  can be estimated in different situations by different methods. In particular, a stream of instructions generated by different computer tasks can be described as a sequence of words created by a formal language, or the dependence between sequentially executed instructions can be modeled by Markov chains, *etc.* Seemingly the most general approach is to define the set of admissible sequences of instructions as a certain subset of all possible sequences. More precisely, the set of admissible sequences  $G$  is defined as a subset  $G \subset A^\infty$ , where  $A^\infty$  is the set of one-side infinite words over the alphabet  $A$ :  $A^\infty = \{x : x = x_1x_2\dots\}$ ,  $x_i \in A, i = 1, 2, \dots$ . In this case the the capacity of  $G$  is deeply connected with the topological entropy and Hausdorff dimension; for definitions and examples see [9–12] and references therein. We do not consider this approach in details, because it seems to be difficult to use it for solving applied problems which require a finite description of the channels.

The simplest estimate of computer capacity can be obtained if we suppose that all sequences of the instructions are admissible. In other words, we consider the set of instructions  $I$  as an alphabet and suppose that all sequences of letters (instructions) can be executed. In this case the method of calculation of the lossless channel capacity, given by C. Shannon in [5], can be used. It is important to note that this method can be used for upper-bounding the computer capacity for all other models, because for any computer the set of admissible sequences of instructions is a subset of all words over the “alphabet”  $I$ .

Let, as before, there be a computer with a set of instructions  $I$  whose execution time is  $\tau(x), x \in I$ , and all sequences of instructions are allowed. In other words, if we consider the set  $I$  as an alphabet, then all possible words over this alphabet can be considered as admissible sequences of instructions for the computer. The question we consider now is how one can calculate (or estimate) the capacity (4) for this case. The solution is suggested by C. Shannon [5] who showed that the capacity  $C(I)$  is equal to the logarithm of the largest real solution  $X_0$  of the following equation:

$$X^{-\tau(x_1)} + X^{-\tau(x_2)} + \dots + X^{-\tau(x_s)} = 1 \quad (7)$$

where  $I = \{x_1, \dots, x_s\}$ . In other words,  $C(I) = \log X_0$ .

It is easy to see that the efficiency (6) is maximal, if the sequence of instructions  $x_1x_2\dots, x_i \in I$  is generated by an i.i.d. source with probabilities  $p^*(x) = X_0^{-\tau(x)}$ , where  $X_0$  is the largest real solution to the Equation (7),  $x \in I$ . Indeed, having taken into account that  $h_\infty(z) = h_0(z)$  for i.i.d. source [7] and the definition of entropy (5), the direct calculation of  $c(I, p^*)$  in (6) shows that  $c(I, p^*) = \log X_0$  and, hence,  $c(I, p^*) = C(I)$ .

It will be convenient to combine all the results about computer capacity and efficiency in the following statement:

**Theorem 1** *Let there be a computer with a set of instructions  $I$  and let  $\tau(x)$  be the execution time of  $x \in I$ . Suppose that all sequences of instructions are admissible computer programs. Then the following equalities are valid:*

- i) The alphabet capacity  $C(I)$  (4) equals  $\log X_0$ , where  $X_0$  is the largest real solution to the Equation (7).
- ii) The efficiency (6) is maximal if the sequences of instructions are generated by an i.i.d. source with probabilities  $p^*(x) = X_0^{-\tau(x)}$ ,  $x \in I$ .

### 3. MIX and MMIX

As an example we briefly consider the MMIX and MIX computers suggested by D. Knuth [4,13]. The point is that those computers are described in details and MMIX can be considered as a model of a modern computer, whereas MIX can be considered as a model of computers produced in the 1970th. The purpose of this consideration is to investigate the given definitions and to look at how various characteristics of a computer influence its capacity, therefore we give some details of the description of MIX and MMIX.

We consider a binary version of MIX [13], whose instructions are represented by 31-bit words. Each machine instruction occupies one word in the memory, and consists of 4 parts: the address (12 bits and the sign of the word) in memory to read or write; an index specification (1 byte, describing which register to use) to add to the address; a modification (1 byte) that specifies which parts of the register or memory location will be read or altered; and the operation code (1 byte). So, almost all 31-bit words can be considered as possible instructions and the upper bound of the number of the set of instructions  $I$  (and letters of the “computer alphabet”) is  $2^{31}$ . Each MIX instruction has an associated execution time, given in arbitrary units. For example, the instruction *JMP* (jump) has the execution time 1 unit, the execution times of *MUL* and *DIV* (multiplication and division) are 10 units and 12 units, correspondingly. There are special instructions whose execution time is not constant. For example, the instruction *MOVE* is intended to copy information from several cells of memory and the execution time equals  $1 + 2F$ , where  $F$  is the number of cells.

From the description of MIX instructions and Theorem 1 we obtain the following equation for calculating the upper bound of the capacity of MIX:

$$\frac{2^{28}}{X} + \frac{2^{26}}{X^2} + \frac{2^{26}}{X^{10}} + \frac{2^{25}}{X^{12}} + \sum_{F=0}^{2^{25}} \frac{2^{25}}{X^{1+2F}} = 1 \quad (8)$$

Here the first summand corresponds to operations with execution time 1, etc. It is easy to see that the last sum can be estimated as follows:  $\sum_{F=0}^{2^{25}} \frac{2^{25}}{X^{1+2F}} < \frac{2^{25}}{X} \frac{X^2}{X^2-1}$ . Having taken into account this inequality and (8), we can obtain by direct calculation that the MIX capacity is approximately 28 bits per time unit.

The MMIX computer has 256 general-purpose registers, 32 special-purpose ones and  $2^{64}$  bytes of virtual memory [4]. The MMIX instructions are presented as 32-bit words and in this case the “computer alphabet” consists of almost  $2^{32}$  words (almost, because some combinations of bits do not make sense). In [4] the execution (or running) time is assigned to each instruction in such a way that each instruction takes an integer number of  $v$ , where  $v$  is a unit that represents the clock cycle time. Besides, it is assumed that the running time depends on the number of memory references (*mems*) that a program uses. For example, it is assumed that the execution time of each of the *LOAD* instructions is  $v + \mu$ , where  $\mu$  is an average time of memory reference [4]. If we consider  $v$  as the time unit and define  $\hat{\mu} = \mu/v$ , we

obtain from the description of MMIX [4] and (7) the following equation for finding an upper bound on the MMIX capacity:

$$2^{24} \left( \frac{139}{X} + \frac{32}{X^2} + \frac{5}{X^3} + \frac{17}{X^4} + \frac{3}{X^5} + \frac{4}{X^{10}} + \frac{2}{X^{40}} + \frac{4}{X^{60}} \right) + 2^8 2^{64} \left( \frac{46}{X^{1+\hat{\mu}}} + \frac{2}{X^{1+20\hat{\mu}}} + \frac{46}{X^{2+2\hat{\mu}}} \right) = 1 \quad (9)$$

The value  $\hat{\mu}$  depends on the realization of MMIX and is larger than 1 for modern computers [4]. Now we can estimate the capacity for different  $\hat{\mu}$ . For example, if  $\hat{\mu}$  is small (say,  $\hat{\mu} = 1$ ), the capacity is large (about 35 bits). If  $\hat{\mu}$  is large ( $\hat{\mu} = 5$ ), the capacity is around 31.5 bits.

Those examples show that the Equation (7) can be used for estimation of the computer capacity and the capacity of computers is mainly determined by the subsets of instructions whose execution time is minimal.

Theorem 1 gives a possibility to estimate frequencies of the instructions, if the computer performance efficiency is maximal (and equals its capacity). First, the frequencies of instructions with equal running time have to be equal. In turn, it means that all memory cells should be used equally often. Second, the frequency of instructions exponentially decreases as their running time increases. It is interesting that in the modern computer MMIX the share of fast commands is larger than in the old computer MIX and, hence, the efficiency of MMIX is larger. It is reached due to the usage of registers instead of the (slow) memory.

#### 4. Possible Applications

It is natural to use estimations of the computer capacity at the design stage. We consider examples of such estimations that are intended to illustrate some possibilities of the suggested approach.

First we consider a computer, whose design is close to the MMIX computer. Suppose a designer has decided to use the MMIX set of registers. Suppose further, that he/she has a possibility to use two different kinds of memory, such that the time of one reference to the memory and the cost of one cell are  $\tau_1, c_1$  and  $\tau_2, c_2$ , correspondingly. It is natural to suppose that the total price of the memory is required not to exceed a certain bound  $C$ . As in the example with MMIX we define  $\hat{\mu}_1 = \tau_1/v$ ,  $\hat{\mu}_2 = \tau_2/v$ , where, as before,  $v$  is a unit that represents the clock cycle time.

As in the case of MMIX, we suppose that there are instructions for writing and reading information from a register to a cell. The set of these instructions coincides with the corresponding set of the MMIX computer. If we denote the number of the memory cells by  $S$ , then the number of the instructions which can be used for reading and writing, is proportional to  $S$ . Having taken into account that MMIX has  $2^8$  registers and the Equation (9), we can see that the designer should consider two following equations

$$\left( 2^{24} \left( \frac{139}{X} + \frac{32}{X^2} + \frac{5}{X^3} + \frac{17}{X^4} + \frac{3}{X^5} + \frac{4}{X^{10}} + \frac{2}{X^{40}} + \frac{4}{X^{60}} \right) \right) + 2^8 S_i \left( \frac{46}{X^{1+\hat{\mu}_i}} + \frac{2}{X^{1+20\hat{\mu}_i}} + \frac{46}{X^{2+2\hat{\mu}_i}} \right) = 1 \quad (10)$$

for  $i = 1, 2$ , where  $S_i = C/c_i$ , i.e.,  $S_i$  is the number of cells of the  $i$ -th kind of memory,  $i = 1, 2$ . The designer can calculate the maximal roots for each equation ( $i = 1, 2$ ) and then he/she can choose that

kind of memory for which the solution is larger. It will mean that the computer capacity will be larger for the chosen kind of memory. For example, suppose that the total price should not exceed 1 ( $C = 1$ ), the prices of one cell of memory are  $c_1 = 2^{-30}$  and  $c_2 = 2^{-34}$ , whereas  $\hat{\mu}_1 = 1.2$ ,  $\hat{\mu}_2 = 1.4$ . The direct calculation of the Equation (10) for  $S_1 = 2^{30}$  and  $S_2 = 2^{34}$  shows that the former is preferable, because the computer capacity is larger for the first kind of memory.

Obviously, this model can be generalized for different set of instructions and different kinds of memory. In such a case the considered problem can be described as follows. We suppose that there are instructions  $\mu_i^w(n)$  for writing information from a special register to  $n$ -th cell of  $i$ -th kind of memory ( $n = 0, \dots, n_i - 1$ ,  $i = 1, \dots, k$ , and similar instructions  $\mu_i^r(n)$  for reading. Moreover, it is supposed that all other instructions cannot directly read or write to the memory of those kinds, *i.e.*, they can write to and read from the registers only. (It is worth noting that this model is quite close to some real computers.) Denote the execution time of the instructions  $\mu_i^w(n)$  and  $\mu_i^r(n)$  by  $\hat{\tau}_i$ ,  $i = 1, \dots, k$ .

In order to get an upper bound of the computer capacity for the described model we, as before, consider the set of instructions as an alphabet and estimate its capacity applying Theorem 1. From (7) we obtain that the capacity is  $\log X_0$ , where  $X_0$  is the largest real solution of the following equation:

$$\sum_{x \in I^*} X^{-\tau(x)} + R \left( \frac{2n_1}{X^{\hat{\tau}_1}} + \frac{2n_2}{X^{\hat{\tau}_2}} + \dots + \frac{2n_k}{X^{\hat{\tau}_k}} \right) = 1 \tag{11}$$

where  $I^*$  contains all instructions except  $\mu_i^r(n)$  and  $\mu_i^w(n)$ ,  $i = 1, \dots, k$ ,  $R$  is a number of registers. (The summand  $\frac{2n_i}{X^{\hat{\tau}_i}}$  corresponds to the instructions  $\mu_i^w(n)$  and  $\mu_i^r(n)$ .)

Let us suppose that a price of one cell of  $i$ th kind of memory is  $c_i$  whereas the total cost of memory is limited by  $C$ . Then, from the previous equation we obtain the following optimization problem:

$$\log X_0 \rightarrow \text{maximum}$$

where  $X_0$  is the maximal real solution of the Equation (11) and

$$c_1 n_1 + c_2 n_2 + \dots + c_k n_k \leq C$$

$n_i \geq 0$ ,  $i = 1, \dots, k$ . The solution of this problem can be found using standard methods and used by computer designers.

The suggested approach can be applied to optimization of different parameters of computers including the structure of the set of instructions, *etc.*

## 5. Conclusions

We have suggested a definition of the computer capacity and its efficiency as well as a method for their estimation. It can be suggested that this approach may be useful on the design stage when developing computers and similar devices.

It would be interesting to analyze the “evolution” of computers from the point of view of their capacity. The preliminary analysis shows that the development of the RISC processors, the increase in quantity of the registers and some other innovations, lead to the increase of the capacity of computers. Moreover, such methods as using cache memory can be interpreted as an attempt to increase the efficiency of a computer.

It is worth noting that the suggested approach in general can be extended to multi-core processors and special kinds of cache memory.

## Acknowledgment

Research was supported by Russian Foundation for Basic Research (grant no. 09-07-00005).

## References

1. Papadimitriou, C.H. *Computational Complexity*; Addison-Wesley: New York, NY, USA, 1996.
2. Kolmogorov, A.N. Three approaches to the quantitative definition of information. *Probl. Inform. Transm.* **1965**, *1*, 3–11.
3. Li, M.; Vitanyi, P. *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd edition; Springer-Verlag: New York, NY, USA, 1997.
4. Knuth, D.E. MMIX: A RISC Computer for the New Millennium. In *The Art of Computer Programming*; Addison-Wesley: New York, NY, USA, 2005; Volume 1, Fascicle 1.
5. Shannon, C.E. A mathematical theory of communication. *Bell Sys. Tech. J.* **1948**, *27*, 379–423, 623–656.
6. Tanenbaum, A.S. *Structured Computer Organization*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2005.
7. Cover, T.M.; Thomas, J.A. *Elements of Information Theory*; Wiley: Hoboken, NJ, USA, 2006 .
8. Csiszár, I. Simple proofs of some theorems on noiseless channels. *Inform. Contr.* **1969**, *14*, 285–298.
9. Doty, D. Dimension extractors and optimal decompression. *Theory Comput. Syst.* **2008**, *43*, 425–463.
10. Fortnow, L.; Lutz, J.H. Prediction and dimension *J. Comput. Syst. Sci.* **2005**, *70*, 570–589.
11. Lind, D.; Marcus, B. *An Introduction to Symbolic Dynamics and Coding*; Cambridge University Press: Cambridge, UK, 1996.
12. Ryabko, B.Ya. Noiseless coding of combinatorial sources, Hausdorff dimension, and Kolmogorov complexity. *Prob. Inform. Transm.* **1986**, *22*, 170–179.
13. Knuth, D.E. Fundamental Algorithms. In *The Art of Computer Programming*; Addison-Wesley: New York, NY, USA, 1968; Volume 1.
14. Krichevsky, R. *Universal Compression and Retrieval*; Kluwer Academic Publishers: Dordrecht, Netherlands, 1993.

## Appendix

*Proof of the claim 1.* We will consider stationary subsets [14], because their capacity (or combinatorial entropy [14]) can be defined analogously (4). Let  $G \subset A^\infty$ , where  $A$  is an alphabet. The definition of a stationary subset  $G$  is as follows: take a word from  $G$  and cross its first letter out. If the word obtained remains an element of  $G$ , then  $G$  is called stationary. The following proposition due to R. Krichevsky [14]:

**Proposition 1** Let  $G$  be a stationary set and  $G(s)$  be the set of all  $s$ -letter prefixes of all words from  $G$ , where  $s$  is an integer. Then  $\lim_{s \rightarrow \infty} \log |G(s)|/s$  exists.

The proof can be found in [14].

Let, as before,  $S_C$  be a set of all allowable sequences of instructions from  $I$ . We denote the subset of all infinite sequences of  $S_C$  by  $S_C^\infty$ . Let us represent a set of all instructions as  $I = \{x_1, x_2, \dots, x_N\}$  and define a new alphabet  $I^*$  as follows:  $I^* = \{x_1^1, x_1^2, \dots, x_1^{\tau(x_1)}, x_2^1, x_2^2, \dots, x_2^{\tau(x_2)}, \dots, x_N^1, x_N^2, \dots, x_N^{\tau(x_N)}\}$ . We suppose that the length of all letters from  $I^*$  equals 1 and define a subset of allowable sequences  $S_C^*$  as follows:  $x_{i_1}^1 x_{i_1}^2 \dots x_{i_1}^{\tau(x_{i_1})} x_{i_2}^1 x_{i_2}^2 \dots x_{i_2}^{\tau(x_{i_2})} \dots \in S_C^*$  if and only if  $x_{i_1} x_{i_2} \dots \in S_C$ . In words, any letter (or instructions) from  $I$ , whose length is  $\tau(x)$ , is presented as a sequence of  $\tau(x)$  letters, whose length is 1. Having taken into account that  $I$  is finite, it is easy to see that if the  $\lim_{T \rightarrow \infty} \log |S_C^*(T)|/T$  exists, then the limit (4) exists, too.

Let  $A$  be an alphabet and  $G$  is a subset of one-side infinite words over  $A$ , i.e.,  $G \subset A^\infty$ . We define the set  $G^{-1}$  as follows: take all words from  $G$  and cross their first letters out. The obtained set of words is  $G^{-1}$  and, by definition,  $G^{-s} = (G^{-(s-1)})^{-1}$ ,  $G^0 = G$ ,  $s > 0$ . Let  $\tau^* = \prod_{x \in I} \tau(x)$  and  $\hat{S}_C^* = \bigcup_{j=0}^{\tau^*} (S_C^*)^{-j}$ . From those definitions we immediately obtain that  $\hat{S}_C^*$  is a stationary set and, according to Proposition, there exists  $\lim_{T \rightarrow \infty} |\log \hat{S}_C^*(T)|/T$ . Taking into account the definitions of  $\tau^*$  and  $\hat{S}_C^*$ , we can see that for any integer  $T$

$$\log |\hat{S}_C^*(T)| \leq \log |S_C^*(T)| \leq \log(\tau^* |\hat{S}_C^*(T)|).$$

The  $\lim_{T \rightarrow \infty} |\log \hat{S}_C^*(T)|/T$  exists, hence,  $\lim_{T \rightarrow \infty} |\log(\tau^* |\hat{S}_C^*(T)|)/T$  exists. From this and last inequalities we can see that  $\lim_{T \rightarrow \infty} \log |S_C^*(T)|/T$  exists. As we mentioned above, it means that the limit (4) exists, too.

*Claim is proven.*

© 2010 by the authors; licensee MDPI, Basel, Switzerland. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>.)