Practical Estimation of Kolmogorov Complexity using Highly Efficient Compression Algorithms

Angel Kuri-Morales[1], Oscar Herrera[2], José Galaviz[3], Martha Ortiz-Posadas[4]

[1]Departamento de Computación. Instituto Tecnológico Autónomo de México
[2]Centro de Investigación en Computación. IPN
[3]Departamento de Matemáticas, Facultad de Ciencias. UNAM
[4]Laboratorio de Informática Médica. Departamento de Ingeniería Eléctrica.
Universidad Autónoma Metropolitana Iztapalapa.
akuri@itam.mx
jose@pateame.fciencias.unam.mx heoscar@yahoo.com posa@xanum.uam.mx

Abstract. In this paper we describe a heuristic approach to the problem of calculating the algorithmic information of message *m* by estimating Kolmogorov complexity. This is achieved by defining a basic element of information which we call a metasymbol. We show that when *m* is expressed as an ensemble (which we call M) of metasymbols it complies with the minimum message length principle. Therefore, M is the shortest way of encoding *m* and approaches Kolmogorov's bound. We discuss approaches to the problem of finding the best set of metasymbols using AI techniques.

1 Introduction

Information, in the classical theory, is defined as a function of the probability of appearance of a symbol S_i at one end of a communication channel. If the S_i are statistically independent, the information contained in a message is equal to the sum of the information of every one of its symbols, which is intuitively satisfactory. However, in spite of its simplicity, we must notice that we ignore the *meaning* of a message and focus *only* in communicating it between a *transmitter* (tx) and a *receiver* (rx) under the assumption that the universe of possible messages is known both to tx and rx. This notion of information is a measure of one's freedom when selecting a message. Given the choice of sending the full contents of this paper or the phrase "Let's eat" the information concerned is, precisely, one bit.

Another intuitive way to look at the problem is to consider that the information in a finite string is the number of bits of the shortest self-contained program that computes the string and terminates. Thus, a long sequence of *n* 0's (for *n*, say, equal to 10,000) contains little information because a program of about $\log_2(n)$ bits outputs it. Furthermore, it can be shown that all reasonable choices of programming languages lead to quantification of the amount of absolute information in individual objects that is invariant up to an additive constant. We call this quantity the *Kolmogorov complexity* (K-complexity) of the object. If an object contains regularities, then it has a shorter description than itself. We call such object *compressible*. A consequence of this is that a truly random sequence cannot be expressed in a shorter string than the enumeration of its constituents and, conversely, that a string compressed to its utmost is, for all practical purposes, random and that its length is a strict measure of its K-complexity. This second fact leads us to conclude that an algorithm achieving highly efficient compression will allow us to measure the K-complexity of an object; a fact which has been overlooked in the past.

The crux of the matter lies in defining “highly efficient compression” (HEC). In what follows we discuss a representation which provides an operative definition of HEC. It relies on the discovery of the so-called metasymbols and their proper selection. We have already proven that both problems are NP-complete and, therefore, that their practical solution must depend on heuristics. We discuss a set of algorithms which approach theoretical bounds and, therefore, offer a practical tool for the calculation of K-complexity. In part 2 we discuss the representation, in part 3 we discuss the algorithms, in part 4 we offer our conclusions.

2 Representation

Our method relies on the analysis of *m* in order to identify a set of the underlying patterns (which we call *metasymbols*). In what follows we describe one possible representation. In order to do this we must clearly define a metasymbol and the associated concepts.

2.1 Definitions

Metasymbol (Ms). It is a collection of symbols which appears embedded in *m*. Its symbols are not necessarily contiguous (they may show gaps). Its size (number of symbols) is, in general, different from the sizes of other Ms. In general, we expect that Ms_i appears more than once in *m*. The *i*-th copy of a Ms in *m* is called its *i*-th *instance*.

Position of Ms_i. It is defined by the i-th offset (the number of symbols intervening between the first symbol of Ms_i and the first symbol of Ms_{i+1}). By convention the offset of the Ms_1 is relative to the first symbol found in *m*. The position of all the symbols in Ms_i is completely specified by these offsets. Since the structure of the Ms is fixed, when we specify the offset of Ms we implicitly specify the position of all of its symbols.

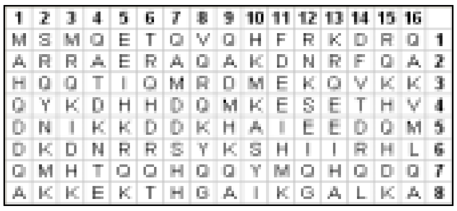
Gap. It is the number of unspecified symbols between two contiguous symbols in a Ms. There may be S symbols (not necessarily different) in Ms*i*; hence, there are S-1 gaps in it. By the *structure* of a Ms we mean the enumeration of its gaps.

Content of a metasymbol. It is the enumeration of the values of all the symbols in the Ms. In general, *m* is not fully covered by all instances of the different Ms, i.e. there are symbols which are not accounted for by a collection of Ms. The enumeration of those symbols not covered in *m* is called the *filler*.

2.2 The Method

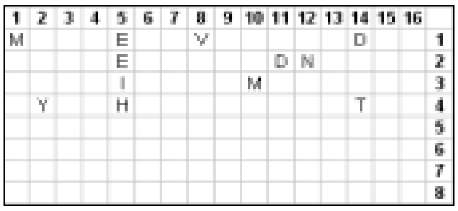
In general, the steps to express *m* as a collection of Ms [1] are: 1) Find an adequate set of Ms, 2) Express *m* as a sequence of Ms, 3) Describe the position, structure and contents of each of the Ms, 4) Describe the contents of the filler.

In what follows we illustrate the representation method with a hypothetical *m* arranged in a 16X8 matrix (see figure 1). In *m* we may identify 5 metasymbols. The first metasymbol (Ms_1) consists of 12 symbols (see figure 2), and it appears 3 times. The positions of its first symbol in the first, second and third instances are (1,1); (7,3) and (9,4). To identify the position of the next instances of the Ms_1 refer to figure 1.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
M	S	M	Q	E	T	Q	V	Q	H	F	R	K	D	R	Q	1
A	R	R	A	E	R	A	Q	A	K	D	N	R	F	Q	A	2
H	Q	Q	T	I	Q	M	R	D	M	E	K	Q	V	K	K	3
Q	Y	K	D	H	H	D	Q	M	K	E	S	E	T	H	V	4
D	N	I	K	K	D	D	K	H	A	I	E	E	D	Q	M	5
D	K	D	N	R	R	S	Y	K	S	H	I	I	R	H	L	6
Q	M	H	T	Q	Q	H	Q	Q	Y	M	Q	H	Q	D	Q	7
A	K	K	E	K	T	H	G	A	I	K	G	A	L	K	A	8

Fig. 1. Message arranged in a 16X8 matrix



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
M				E			V						D			1
				E						D	N					2
				I					M							3
	Y			H									T			4
																5
																6
																7
																8

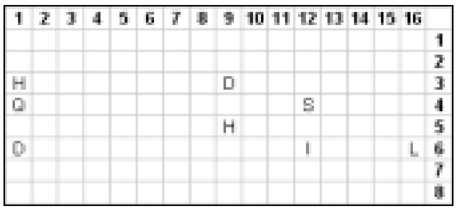
Fig. 2. First instance of Ms_1

Ms_2 consists of 8 symbols and it appears 2 times. Figure 3 shows its first instance with initial position at (1,3). The second instance appears at (15,4).

Ms_3 consists of 3 symbols and it appears 9 times. Its first instance is shown in figure 4. Observe that its initial position is (15,1). For instances 2 to 9 their initial positions are (2,2), (3,2), (6,2), (13,2), (8,3), (5,6), (6,6) and (14,6) respectively (refer to figure 1).

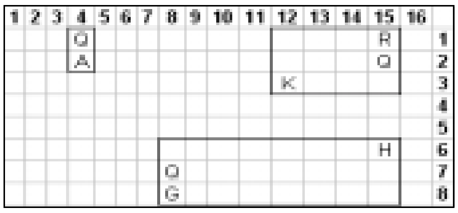
Ms_4 consists of 2 symbols and it appears 7 times. In figure 4 is shown its first instance and its position at (4,1). For instances 2 to 7 its positions are (7,1), (9,1), (16,1), (1,7), (9,7) and (16,7) respectively (refer to figure 1).

Ms_5 consists of 3 symbols and it appears 2 times. Figure 4 shows its first instance with positions at (15,6) and (3,7).



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
																1
																2
H								D								3
Q											S					4
								H								5
D											I				L	6
																7
																8

Fig. 3. First instance of Ms_2



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
			Q											R		1
			A											Q		2
											K					3
																4
																5
														H		6
							Q									7
							G									8

Fig. 4. First instances of Ms_3, Ms_4 and Ms_5

The symbols of all instances of Ms_i, i=1, …, 5 account for 99 of the 128 symbols in *m* (77.34%). The rest of the symbols follow no pattern and make up the filler.

For clarity, we assign the Greek letters α, β, γ, δ and ε to Ms_1, ..., Ms_5 respectively; then $m = \alpha\,\delta\,\delta\,\delta\,\gamma\,\delta\,\gamma\,\gamma\,\gamma\,\gamma\,\beta\,\alpha\,\gamma\,\alpha\,\beta\,\gamma\,\gamma\,\gamma\,\varepsilon\,\delta\,\varepsilon\,\delta\,\delta$. However, without an explicit definition of the position, structure and contents of the various Ms the metasymbolic expression of the message is meaningless. Therefore, we proceed as follows.

1) We describe the positions of the instances of the metasymbols. In the message this corresponds to the sequence: 0 2 2 1 5 0 1 0 2 6 3 5 0 16 5 21 0 7 0 1 1 5 6. Every number in the sequence represents the distance between the *i*-th and (*i+1*)-th Ms. In *m*, Ms_1 starts on the position of the first symbol (1,1); then the first instance of Ms_4 is two symbols away from Ms_1. The second instance of Ms_4 appears 2 symbols away from its first instance. Likewise, the first instance of Ms_3 is 5 symbols away from the second instance of Ms_4, and so on.

2) We define the structure of all the Ms as a sequence of gaps, one for every Ms. Since the symbols are unknown, code '0' is reserved to indicate the end of the structure of a Ms. The structure of Ms_i is, thus, a collection of gaps plus a '**0**'. Furthermore, since it is possible to have a gap of size 0, every gap is encoded as its actual size +1. As an example, the structure of Ms_1 is specified by the sequence: 4 3 6 7 6 1 9 5 8 3 9 **0**. The gap between S_1 and S_2 is 3 (4-1), the gap between symbols S_2 and S_3 is 2 (3-1), and so on.

3) We proceed to define the contents of every metasymbol. The contents of the Ms in *m* are defined as follows. Ms_1: {MEVDEDNIMYHT}; Ms_2: {HDQSHDIL}; Ms_3:{RQK}; Ms_4:{QA}; Ms_5 :{HQG}.

At this point we have defined all necessary elements: order, position, structure and contents, for every Ms in *m*. However, as stated, there are symbols which are not accounted for by the collection of Ms defined. However, the exact positions of all undefined symbols in *m* are, at this point, known.

4) We enumerate the contents of undefined localities to complete the cover. The following sequence is the filler of *m*:{SMTHFRKAQKFTHDIKKAEDKSKMEKHAK}.

Under this representation *m*'s K-complexity (K) is given by

$$K = \mu(1+\sum N_i) + \omega \sum N_i + (\gamma + \lambda)\sum S_i + L - \lambda \sum S_i N_i$$

Where N_i denotes the number of instances of Ms_i; M = number of different Ms; $\mu = \lceil \log_2(M) \rceil$; $\omega = \lceil \log_2[\text{max. offset } Ms_i] \rceil$; $\gamma = \lceil \log_2[\text{max. gap } Ms_i] \rceil$; $\lambda = \lceil \log_2 |Ms_i| \rceil$; $S_i = \lceil \log_2(\text{max. symbols } Ms_i) \rceil$ and $L = |m|$ in bits.

3 Algorithms

In the preceding discussion we assumed that the Ms in *m* are known. As pointed out in the introduction, such is not the case in general. In fact, the problem of finding the Ms is NP-hard. In what follows we discuss two approaches to the solution. In 3.1 we focus on an approach where Ms are determined on-line; in 3.2 we describe an alternative where these are determined off-line. We have conducted statistical tests which allow us to affirm that both methods approach the Ms set which leads to maximum compression.

3.1 On-Line Search

MSIM is a tool for searching metasymbols based on Vasconcelos' Genetic Algorithm (VGA)[2]. The VGA uses a population *P* of *N* individuals, a temporary population of 2*N* individuals, deterministic selection (from the best *N* individuals) and has two main operators: mutation and crossover. Assuming a binary coding of the individuals, mutation replaces a bit's value by its complement with probability *Pm*. Crossover requires two individuals and interchanges their genetic material with probability *Pc*.

The off-line algorithm (MSIM) models *m* as an array of symbols indexed from 0 to |*m*|-1. The individuals of MSIM are permutations of all the |*m*| indices of the message which lie in the interval [0, |*m*|-1]. Each index is coded in traditional weigthed binary and requires $\lceil \log_2 L \rceil$ bits. We delimit the metasymbols of an individual by reading the indices from left to right and looking for an ascending order.

As an example, in figure 5 we illustrate a possible individual for the hyptothetical *m* "xyazbcdxyez".

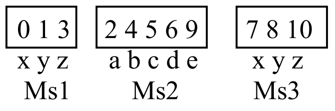
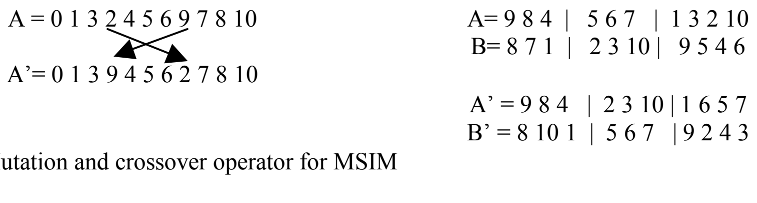


Fig. 5. An individual of VGA in MSIM

The relation between the first occurrence of a metasymbol and its repeated instances is obtained by comparing the contents of the metasymbols as well as their gaps. In the example of figure 1, Ms 3 is a repeated instance of Ms 1 because they match their contents as well as their gaps. The other symbols make up the filler. The positions of all the metasymbols can be determined from the positions of the first symbol of each metasymbol. In genetic algorithm's terminology, the measure of fitness for MSIM is *K* which is to be minimized.

VGA's mutation and crossover operations are not directly applicable in manipulating the individuals of MSIM because they act on the genotype (binary encoding) which may lead to invalid individuals (out of the feasible region), i.e., individuals with repeated indices or with indices out of the interval [0, |*m*|-1]. For this reason we redefined the mutation and crossover operators to render them appropriate.

The mutation operator consists of a permutation of two indices whereas the crossover corresponds to PMX [3]. Both of them retain the resulting individuals in the feasible region. Figure 6 illustrates this two operators.



Mutation
A = 0 1 3 2 4 5 6 9 7 8 10
A'= 0 1 3 9 4 5 6 2 7 8 10
Crossover
A= 9 8 4 | 5 6 7 | 1 3 2 10
B= 8 7 1 | 2 3 10 | 9 5 4 6
A' = 9 8 4 | 2 3 10 | 1 6 5 7
B' = 8 10 1 | 5 6 7 | 9 2 4 3

Fig. 6. Mutation and crossover operator for MSIM

MSIM's representation is slightly different to the one described in section 2.2; it encodes the position between the i-th and the j-th symbol corresponding to the next repeated instance of the i-th metasymbol. Here we do not need to explicitly express the sequences of the metasymbols since we can group the positions of the α's, β's, γ's, δ's and ε's, as shown in figure 7.

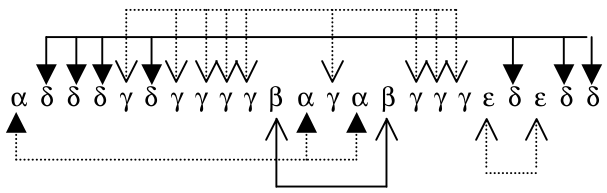


Fig. 7. Alternative coding of m

Hence, instead of the sequence of metasymbols we can indicate their positions as follows:

α: 0, 11, 2; β: 10, 4; γ: 4, 2, 1, 1, 1, 3, 3, 1, 1; δ: 1, 1, 1, 2, 14, 2,1, 1; ε: 18, 2

The number of bits of the re-expressed *m* can be calculated by grouping the bits required for each metasymbol, thus

$$K' = \sum \left[\psi_i N_i + (\gamma + \lambda) S_i\right] + L - \lambda \sum S_i M_i$$

where ψ_i is the number of bits required to encode the positions of all the instances of the i-th Ms. The term of the first summation represents the number of bits required to encode the i-th metasymbol and is denoted by A_i, then

$$A_i = \psi N_i + (\gamma + \lambda) S_i$$

The number of bits used by the symbols of the i-th Ms in *m* is B_i and is given by

$$B_i = |M_i| f_i$$

Notice that, although we have derived two different expressions (K and K') for Kolmogorov complexity we have shown that, in practical applications, both are equivalent [4]. In fact, there is an arbitrary number of possible representations and they are all computationally equivalent. The latter is interesting in terms of MSIM's implementation.

We now define the discriminant d_i as

$$d_i = \frac{A_i}{B_i}$$

Any value of d_i smaller than 1 implies that we may achieve *m*'s compression and it does not depend on the rest of the Ms. Therefore, we can iteratively search for metasymbols whenever we are able to find an associated d_i smaller than 1. Of course, we are looking for Ms such that they minimize d_i. The Ms in the set are encoded as individuals of MSIM and introduced to the population via a special operator called the *catastrophe operator*. This operator corresponds, roughly, to forcing a large number of mutations on the individuals, hence its name. It relies on the iterative search for Ms with minimum d_i value and it will never submit individuals in which of *K'* is larger than $|m|$.

3.2 Off-Line Search

In the previous section we have introduced a method that attempts to find an optimal subset of patterns as the patterns are found. This on-line approach has the advantage that the time used to determine a good subset of patterns, is amortized by the time consumed in the pattern discovery procedure. But it has the disadvantage that some good patterns can be disregarded. If, at some stage of the algorithm, the frequency and *order* (number of defined symbols) of two patterns are alike then one of these will be selected by the algorithm, while the other is dropped even if its potentiality could emerge later in the process.

An alternative approach is to generate a set of frequent patterns without any evaluation of its goodness, and then proceed to determine the best subset of patterns to build the *a posteriori* dictionary. Evidently this approach is more expensive than the previous one, but the risk that higher performance patterns will be ignored is reduced.

The process implied in the implementation of this approach is divided in two phases:

1. The search for frequent patterns. The result obtained at the end of this phase is a set *P* of frequent patterns.
2. The search for a subset $B \subseteq P$ (*B* is assumed as the dictionary or *set of metasymbols*) such that the amount of bits required for the re-expression of the *m* together with the dictionary is minimum.

Given the kind of patterns we are looking for in *m*, the problem related to phase 1 is very complex. All the algorithms reported have an exponential complexity on $|m|$ [5]. However if some restrictions are imposed on the frequency and order of the patterns, there exist algorithms with better performance [6].

Concerning phase 2, we have proved that the search for an optimum subset of patterns if it is adopted as the dictionary, is an NP-complete problem [7], therefore heuristics are needed in order to approximate such subset.

The optimality of some pattern is given by its frequency and its order: the higher the order and frequency, the better is the pattern. But there is a trade-off between these two quantities. In one end we have that single symbols are patterns with highest frequency and minimum order; on the other we have that the pattern encompassing *m* has maximum order and minimum frequency. Hence, we need to find patterns which maximize both quantities simultaneously. Therefore, we require a measure for pattern quality including frequency and order.

The cost to include some pattern in a dictionary is better amortized if such pattern appears frequently in *m* and there is a large amount of symbols covered by all these appearances. Therefore, the product of pattern frequency and pattern order, (which we call the pattern *coverage*), can roughly provide us with an estimate the pattern contribution. However, it may happen that some symbols covered in one occurrence of a pattern overlap with those covered in another. The amount of symbols covered at least one time for a pattern is called the *effective coverage*, which is a more precise measure for the pattern's quality. The concept of effective coverage, can be generalized in the following way: given a set of patterns B, and a pattern p, the *exclusive coverage* of p is the number of symbols in the sample covered at least one time by all the occurrences of p, and not already covered by some other pattern in B. This is the final criterion we use to evaluate the patterns.

The second phase of the procedure sketched above is, therefore, accomplished by a heuristic based on the concept of exclusive coverage (which is why we have called it the *coverage-based heuristic*). The output of this algorithm may be considered as a proposed subset that could be further optimized by using some local optimization procedure. This is attempted by two local optimizers which are, actually, two different kinds of hill climbers that we will describe below.

The coverage-based heuristic proceeds as follows:

- **Input:** A set P of frequent patterns found in a sample S.
- **Output:** A subset $D \subseteq P$, assumed to be an approximation to the metasymbol dictionary.

```
1. Let $D = \varnothing$.
2. While the total coverage of patterns in $D$ is less than $| m |$ and $P$ has patterns with
   exclusive coverage, respect to $D$, greater than zero.
   2.1. Chose the pattern $p$ in $P$ with the highest exclusive coverage respect to $D$.
   2.2. Add $p$ to $D$, and remove $p$ from $P$.
3. Return $D$ as the dictionary.
```

The resulting dictionary can now be optimized by local search procedures. We have defined two of these mechanisms which are applied consecutively.

- Minimum Step Hill Climber (MSHC). Given the code of the subset of patterns (a binary vector of size $| P |$, where 1 in the i-th position means that the i-th pattern of P is included in the subset), the hill climber proceed to flip every bit in the vector. Every change is evaluated and, if the size obtained with the change improves the previous size, the changed code is the current subset.
- Minimum Replacement Hill Climber (MRHC). MRHC proceeds to exchange systematically, the position of every bit with value 1 with every other position whose value is 0.

Both of these climbers can be executed until no improvement is found or a fixed number of evaluations have been performed.

4 Conclusions

We have shown that the best metasymbolic re-expression and further post-encoding of message m yields the most compact representation of m. This is true provided that we are able to find an adequate set of metasymbols in m. Furthermore, since this process is lossless, the original information in m remains unaltered. The most economic representation of m is possible only when enough information is provided in order for the decoder to infer the structure, position and contents of the metasymbols. A formalization of these facts may be found in the *Invariance Theorem* which embodies (most of) the theoretical foundation of K-complexity. We now state it without proof (but see [8]).

There is a universal partial recursive function (prf) φ_0 *for the class of prf's to compute x given y. Formally this says that* $C_{\varphi_0}(x|y) \leq C_{\varphi}(x|y) + c_{\varphi}$ *for all prf's* φ *and all x and y, where* c_{φ} *is a constant depending on* φ *but not on x or y.*

For the purposes of this paper this theorem says that we will be able to decode message x from its metasymbolic representation y and that φ_0 is the prf (embodying our method of representation) allowing us to do so. Usually K-complexity is defined in terms of a Universal Turing Machine and its calculation is, in practice, very difficult at best. Our representation and the related algorithms are a practical approach which simply consists of finding those sets which minimize c_{φ} and, therefore, correspond to the smallest possible constant involved.

As discussed above, the computational cost of achieving such goal is justifiable as long as high order compression methods (other than ours) are not found.

References

1. Kuri A, "Pattern based lossless data compression", WSEAS Transactions on Communications, Issue 1, Vol. 3, 2004, pp. 22-29.
2. Kuri A, "Solution of Simultaneous Non-Linear Equations using Genetic Algorithms", *WSEAS Transactions on Systems*, Issue 1, Vol. 2, 2003, pp. 44-51.
3. Goldberg, D., (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company.
4. Kuri A and Herrera O, "Efficient lossless data compression for nonergodic sources using advanced search operators and genetic algorithms",*Advances in Artificial Intelligence, Computing Science and Computer Engineering*, Nazuno J, Gelbukh A, Yañez C, et. al. (editors), ISBN: 970-36-0194-4, ISSN: 1665-9899, Vol. 10, 2004, pp. 243-251.
5. Vilo, J., *Pattern Discovery from Biosequences*, PhD Thesis, Technical Report A-2002-3, Department of Computer Science, University of Helsinki, 2002.
6. Rigoutsos, I. and A. Floratos, "Combinatorial Pattern Discovery in biological sequences: The Teiresias Algorithm", *Bioinformatics*, Vol. 14, No. 1, 1998, pp. 55-67.
7. Kuri, A. and J. Galaviz, "Pattern-based Data Compression", in *MICAI 2004: Advances in Artificial Intelligence*, Springer Verlag, Lecture Notes in Artificial Intelligence 2972, 2004, pp. 1-10.
8. Li M and Vitányi P, *An introduction to Kolmogorov complexity and its applications*, Springer Verlag, New York, 2nd Ed, 1997.