A Brief Introduction to Kolmogorov Complexity

Michal Koucký
MÚ AV ČR, Praha
<koucky@math.cas.cz>

May 4, 2006

Abstract

In these notes we give a brief introduction to Kolmogorov complexity. The notes are based on the two talks given at KAM Spring School, Borová Lada, 2006.

1 Introduction

The set of all finite binary strings is denoted by $\{0,1\}^*$. For $x \in \{0,1\}^*$, $|x|$ denotes the length of x.

Kolmogorov complexity tries to answer the fundamental question: "What is a random object?" Consider which of the following (decimal) strings seem to be random?

$$33333333333 \tag{1}$$
$$31415926535 \tag{2}$$
$$84354279521 \tag{3}$$

Most people would rule out the first one to be random, and they could agree that the remaining two are random. Indeed, most statisticians would agree that the latter two are random as they pass essentially all possible statistical tests. Yet, the second sequence consists of the first eleven digits of π. The third one is taken really at random.

From the perspective of probability, all three strings have the same probability of being chosen when we take a string of eleven digits fully at random namely, each of them has probability 10^{-11}. Hence, they all are equally likely to be obtained by a random process. So probability does not really explain the intuitive notion of *randomness*.

Imagine that we would extend our strings to one million digits. Then the first string would became a million times the digit three, the second one would be the first million digits of π and the last one would be 84354279521.... In fact it would take us thousand of pages to describe the last one. There is no pattern in it. It is really random.

Hence, the notion of randomness is connected to patterns in strings and to a way how we can describe them. The first two strings in our example have very short descriptions (few words) whereas the last string has very long description as it lacks any regularity. The longer the necessary description of a string the more randomness is in the string. This intuition leads to the following definition of *Kolmogorov complexity* of a string $x \in \{0,1\}^*$: the Kolmogorov complexity of x is the length of the shortest description of x. Of

course the length of the description depends on the language we use for our description—we can use Czech or French or English....

We make it formal as follows. Let $\phi : \{0,1\}^* \to \{0,1\}^*$ be a *partial recursive function*. (A partial recursive function is any function f for which there is a program that takes an input y and produces output $f(y)$. $f(y)$ may not be defined for some y and the program may not halt on such y's or to produce any output.)

Definition 1. *For a string $x \in \{0,1\}^*$, the Kolmogorov complexity of x with respect to ϕ is*

$$C_\phi(x) = \min\{|p|,\ p \in \{0,1\}^* \ \&\ \phi(p) = x\}.$$

Let us consider several examples. If ϕ_1 is the identity function $\phi_1(x) = x$ then $C_{\phi_1}(x) = |x|$. If $\phi_2(0) = 10111001011110011010110111$ and $\phi_2(1x) = x$ then $C_{\phi_2}(10111001011110011010110111) = 1$ and $C_{\phi_1}(x) = |x| + 1$ for all other strings x. So Kolmogorov complexity depends a lot on the chosen *descriptive language* ϕ. Luckily, the following Invariance Theorem brings some order into this chaos.

Theorem 2. *There exists a partial recursive function U so that for any other partial recursive function ϕ there is a constant $c > 0$ such that*

$$C_U(x) \le C_\phi(x) + c$$

for all strings x.

A machine U satisfying the preceding theorem is in some sense minimal among all machines, and we will call it *universal*.

Proof. The proof is quite simple. Let $\phi_0, \phi_1, \phi_2, \ldots$ be an enumeration of all partial recursive functions. (Every p.r.f. can be associated with a program that computes it and that program can be uniquely mapped to a (possibly huge) integer.) Let $\langle x, y\rangle : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be some simple to compute one-to-one mapping, e.g., $\langle x, y\rangle = 0^{|x|}1xy$. Then U is defined as follows: On input w, decode w into i and p such that $w = \langle i, p\rangle$ and run ϕ_i on input p. If $\phi_i(p)$ stops then output whatever ϕ_i had output.

It is easy to verify that such a U is partial recursive and that it satisfies our theorem. □

So from now on we fix some machine U which satisfies the Invariance Theorem and we will consider the Kolmogorov complexity of x to be $C_U(x)$. We will write $C(x)$ instead of $C_U(x)$ from now on.

We are ready to define a random string.

Definition 3. *A string x is Kolmogorov random if $C(x) \ge |x|$.*

This definition is not void as there is a Kolmogorov random string of every length: there are $2^n - 1$ descriptions of length less than n but there are 2^n strings of length n. Let us consider couple of strings and their Kolmogorov complexity:

- 0^n has Kolmogorov complexity $\log n + O(1)$ as we only need to specify the integer n and a short program that will reconstruct 0^n from n.
- The sequence of the first n digits of π has Kolmogorov complexity $\log n + O(1)$. The reason is the same as above. (Just download a program for π from Internet.)
- There is a Kolmogorov random string x with $C(x) \ge n$. See above.

- What about some string of complexity about $\sqrt{n}$? Sure, there is one. Consider $y \in \{0,1\}^{\sqrt{n}}$ that is Kolmogorov random. Then $x = y0^{n-\sqrt{n}}$ has Kolmogorov complexity about $\sqrt{n} + O(1)$. Why? If it would have a description much shorter than $|y|$ we could describe y using such description: first produce x and then output only its first $\sqrt{|x|}$ bits. On the other hand, description of y is a good description of x: produce y and then append $|y|^2 - |y|$ zeroes. So there are strings of essentially all possible complexities.

- Every string of length n has Kolmogorov complexity at most $n + O(1)$. Why?

- How many ones and zeroes has a Kolmogorov random string of length n? About a half, but exactly? There are $\binom{n}{n/2}$ strings which have the same number of ones and zeroes. Call the set of such strings $S^n_{n/2}$. These string can be easily identified and enumerated by a program. Hence, given n and i, we can find the i-th string s_i in $S^n_{n/2}$ and output it. Thus, $C(s_i)$ is at most the size of a description of i, plus the size of a description of n, plus some constant for the program described above. $|S^n_{n/2}| < \binom{n}{n/2} < c2^n/\sqrt{n}$ for some constant c, hence to specify i we only need at most $n - 1/2 \log n + \log c$ bits. n does not really have to be specified as it can be deduced from the length of the description of i. Thus, all strings $s_i \in S^n_{n/2}$ have Kolmogorov complexity at most $n - 1/2 \log n + O(1)$.

  It turns out that Kolmogorov random strings of length n have $n/2 \pm c\sqrt{n}$ zeroes. By Chernoff bound there are relatively few strings of length n that have the number of ones farther from $n/2$ than $c\sqrt{n}$, and by extending the argument above there are also relatively few strings that have the number of ones closer to $n/2$ than $c\sqrt{n}$. Since these strings are few and easy to identify, they have small Kolmogorov complexity. (In fact, the deviation from $n/2$ in the number of ones have to be Kolmogorov random by itself.) The following proposition generalizes this argument.

Proposition 4. *Let A be a recursive (recursively enumerable) set and n be an integer. Let $A_n = A \cap \{0,1\}^n$. For all strings x in A_n it holds, $C(x) \leq \log |A_n| + 2 \log n + O(1)$.*

Often the term $2 \log n$ can be omitted as n can be deduced from the length of the description.

Proof. The proof is straightforward. Since A is recursive (recursively enumerable) we can design a program that given i and n prints the i-th string of A_n in some enumeration. Hence, all strings in A_n can be described by giving i, n and the program for enumerating A. The description of i, n and the program has to be concatenated into one string in such a way that i, n and the program can be recovered from the string. One can use the pairing function from the proof of Theorem 2 for doing that. The factor two in the logarithmic term comes from there. □

It is useful to note that the set of strings that are *not* Kolmogorov random is recursively enumerable—given a string x we can run all programs of length shorter than x in parallel and see if any one of them ever outputs x. If that happens we accept x.

This brings us to the fact that the number of strings of length n that are Kolmogorov random is Kolmogorov random by itself. It is about $2^n/c$ for some constant $c > 1$. If that were not the case, we could find all strings of length n that are not Kolmogorov random, and then print the first one which should be random. Program for such a computation would only need to know the number of non-random strings of length n. The number of non-random strings is 2^n minus the number of random strings, i.e., we can easily compute one from the other one. Since the above program prints out a Kolmogorov random string, both the numbers of random and non-random strings must require close to n bits to specify. Hence, they are both about $2^n/c$.

Proposition 5. *It is uncomputable (undecidable) whether a string is Kolmogorov random.*

We have seen that non-random strings are recursively enumerable. This proposition thus implies that Kolmogorov random strings are not recursively enumerable as otherwise we could decide about a string whether it is Kolmogorov random or not.

Proof. We give two proofs. The first one is very simple, the second one is more complex but it shows that deciding Kolmogorov randomness is as hard as deciding the Halting Problem.

1. Assume we can decide whether a string is Kolmogorov random by some program P. We can then specify the lexicographically first Kolmogorov random string of length n using $\log n + O(1)$ bits: run program P on all strings of length n in the lexicographical order until you find a string that is Kolmogorov random; print out the Kolmogorov random string. This only requires to specify the program P and n. Hence, no such P can exist.

2. Define the Halting Problem by $H = \{x;\ \text{program } x \text{ halts on the input } 0\}$. Assume we can decide which strings are Kolmogorov random by some program P. We can then decide for any string x whether the program x halts on the input 0 or not as follows: Let $n = |x|$. Using P decide for each string y of length $2n$ whether it is Kolmogorov random or not. For each string y that is not random find some program p_y of length less than n that prints it out. Let t_y be the number of steps that it takes to p_y to output y. Set $t_x = \max_y t_y$. Run x on the input 0 for t_x steps and if it accepts within t_x steps then output $x \in H$ otherwise output $x \notin H$.

The reason why the above program would decide H correctly is that if $x \in H$ but the running time of x on the input 0 is more than t_x then the actual running time of x can be used as an upper bound for the running time of all p_y's. As the running time of x can be specified using $n + O(1)$ bits (namely by specifying x) we could specify all non-random strings of length $2n$ using only $n + 2\log n + O(1)$ bits. (Run all programs of length less than $2n$ for t_x steps and see what they output.) Hence, we could describe the lexicographically first Kolmogorov random string of length $2n$ using only $n + 2\log n + O(1)$ bits. Thus the running time of x must be smaller than t_x.

Since our program could correctly decide the Halting Problem, Kolmogorov randomness of strings must be undecidable. □

2 Applications

We give here several applications of Kolmogorov complexity.

2.1 Graph labelings

We start with an example related to the talk of Arnaud Labourel on graph labelings. For a (finite) class of graphs $\mathcal{G}$ a *labeling scheme of label length* ℓ is a function $A : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ together with a labeling $l_G : V(G) \to \{0,1\}^\ell$ of every graph $G \in \mathcal{G}$ so that for all $x, y \in V(G)$, $(x,y) \in E(G)$ iff $A(l(x), l(y)) = 1$. We have already seen in the talk of Arnaud that the class of all the trees on n vertices has labeling scheme with labels of length $\log n + O(1)$. The natural question is how large labels are needed to label the class of all the graphs on n vertices. We claim that this length is $n/2 + O(\log n)$.

First, we show that labels of length $n/2 + \log n$ are sufficient. This is due to Jiří Sgall. Each vertex is going to be labeled by its vertex number plus a bit-vector of length $n/2$ which specifies to which of the next

$n/2$ vertices under a cyclic ordering of vertices the vertex is connected. Given two vertex labels at least one of the labels contains the required adjacency information.

Using Kolmogorov complexity we want to show that $n/2$ bits are needed. First notice that by exhaustive search we can actually find the best labeling scheme for graphs on n vertices. In fact we can write a program that will find it. This program will produce the function A. Each graph G on n vertices can be fully described by listing labels of its vertices $l_G(1), l_G(2), \ldots, l_G(n)$ in the optimal labeling scheme. Such a description requires ℓn bits. Hence, every graph can be described by $\ell n + 2\log n + O(1)$ bits, by providing the vertex labels, n and the program to compute A. On the other hand, a graph on n vertices may contain $\binom{n}{2} = n(n-1)/2$ different edges. Hence, there are at least $2^{n^2/2-n}$ different graphs on n vertices and each of them is uniquely described by a description of length $\ell n + 2\log n + O(1)$. Thus $\ell n + 2\log n + O(1) \geq n^2/2 - n$, i.e., $\ell \geq n/2 - 2$, for n large enough. Thus a labeling scheme for graphs on n vertices requires labels of length about $n/2$.

2.2 Prime Number Theorem

We provide another application of Kolmogorov complexity to number theory. Let p_i denote the i-th prime number. We will show the following theorem:

Theorem 6 (Weak Prime Number Theorem)**.** *There is a constant c such that for infinitely many i, $p_i < c \cdot i \cdot \log^2 i$.*

This theorem is a weak version of the usual Prime Number Theorem that $p_i/i \ln i \to 1$ as $i \to \infty$.

Proof. For a positive integer n let $1\widehat{n}$ be its binary representation with $\widehat{n} \in \{0,1\}^*$. Clearly $2^{|\widehat{n}|} \leq n < 2^{|\widehat{n}|+1}$. Fix a large enough integer x with Kolmogorov random $\widehat{x}$. We will make several observations regarding x.

1. $x = p^e y$, for some p, e and y, where p is a prime and $p^e > \log x / \log\log x$. If all maximal prime-power factors of x were at most $\log x/\log\log x$, then $x \leq (\log x/\log\log x)! < \log x^{\log x/\log\log x} = x$.

2. $e = 1$. Note, $2^{e|\widehat{p}|} \cdot 2^{|\widehat{y}|} < p^e y = x$. Hence,

$$e|\widehat{p}| + |\widehat{y}| \leq |\widehat{x}| \leq C(\widehat{x}).$$

At the same time, x can be specified by giving e, p, and y. Hence, $\widehat{x}$ can be given by some encoding of $\widehat{e}$, $\widehat{p}$ and $\widehat{y}$ into one binary string so that we would be able to tell apart all three of them. The pairing function used in the proof of Theorem 2 is too inefficient for our purposes. We can use the following pairing function: $\langle u, v\rangle = l_u 01 uv$, where l_u is the binary representation of $|u|$ in which each digit is doubled. Thus $|\langle u, v\rangle| \leq |u| + |v| + 2\log|u| + 2$. Using this pairing function we can describe $\widehat{e}$, $\widehat{p}$ and $\widehat{y}$ to obtain:

$$C(\widehat{x}) \leq |\widehat{e}| + |\widehat{p}| + |\widehat{y}| + 2\log|\widehat{e}| + 2\log|\widehat{p}| + O(1).$$

But this implies that $e = 1$. (If $e > 1$ is small then p must be large and hence $(e-1)|\widehat{p}|$ outweighs $|\widehat{e}| + 2\log|\widehat{e}| + 2\log|\widehat{p}| + O(1)$. If e is large then $(e-1)|\widehat{p}|$ also outweighs the additional terms.)

From 1. and 2. we can deduce that $x = py$ for some prime $p > \log x/\log\log x$. Let i be such that $p = p_i$. Prime p_i can be described by giving its index i plus a short program that will reconstruct p_i from i. Hence,

$$C(\widehat{x}) \leq |\widehat{i}| + |\widehat{y}| + 2\log|\widehat{i}| + O(1).$$

Together with the above lower bound on $C(\widehat{x})$ we get

$$|\widehat{p_i}| \leq \widehat{|i|} + 2\log\widehat{|i|} + O(1).$$

Using the relationship between n and $\widehat{n}$, we conclude $p_i \leq c \cdot i \log^2 i$, for some constant c independent of i. From the fact that this is true for arbitrarily large constant x and $p_i > \log x / \log\log x$ we conclude the theorem. □

2.3 Gödel Incompleteness Theorem

Let T be a sound logical theory over a countable language with recursively enumerable axioms. If T is rich enough to describe computation of Turing machines then for some constant c_T and integer x, the formula "$C(\overline{x}) \geq \overline{c_T}$" is true but unprovable, where $\overline{x}$ is the constant describing x. (This is a Π_1 formula saying that for all programs p smaller than $\overline{c_T}$ and for all computations τ, if τ is a computation of p then the output of τ is not $\overline{x}$.) If for all x and all c_T such formula were provable whenever it would be true then by enumerating. all proofs, for given c_T we could find the first *large* x with $C(x) \geq c_T$. But if we choose c_T with succinct representation (very low Kolmogorov complexity), then we will be able to produce x of high Kolmogorov complexity merely from the description of c_T, the description of T and some small program. Of course, that is impossible. So "$C(\overline{x}) \geq \overline{c_T}$" cannot be provable for any large enough Kolmogorov non-random c_T and any x although it is true for many x and c_T.

2.4 Universal search procedure

The problem SAT$= \{\psi;\ \psi$ is a satisfiable Boolean formula$\}$ is a well known NP-complete problem. A related problem is SAT-search where given a satisfiable Boolean formula ψ we want to output an assignment a to ψ such that a satisfies ψ. The computational complexities of SAT and SAT-search are closely related. If SAT has an efficient algorithm then SAT-search has one as well: perform a binary search for a satisfying assignment of ψ by choosing the assignment bit by bit. On the other hand if SAT-search has an efficient algorithm (and we know its running time) then SAT has an efficient algorithm as well: run algorithm for SAT-search on ψ and if it produces an assignment within its allowed running time and the assignment satisfies ψ then ψ belongs to SAT. We will present an (almost) optimal algorithm for SAT-search. We will need the following definition.

Let $\langle\cdot,\cdot\rangle$ be a pairing function. Levin defines the time-bounded Kolmogorov complexity of a string x relative to a string y by:

$$C_t(x|y) = \min\{|p| + \log t,\ p \in \{0,1\}^* \ \&\ U(\langle p, y\rangle) = x \text{ in } t \text{ steps}\}.$$

The algorithm for SAT-search works as follows: on input formula ψ, for $i = 1, 2, \ldots$ try all strings a with $C_t(a|\psi) = i$, and see if any of them satisfies ψ. If yes, output such an a.

We leave implementation details of this algorithm to the interested reader. If p is the optimal algorithm for SAT-search and t is its running time on formula ψ then the satisfying assignment for ψ will be found in time about $2^{|p|}t^2$ by our algorithm. Hence, our algorithm for SAT-search is at most quadratically slower than the best algorithm for SAT-search. The only thing that stands in our way towards $\$1,000,000$ is that we do not have a good estimate on the running time of our SAT-search algorithm.