Two Theorems on Time Bounded Kolmogorov-Chaitin Complexity

David Schweizer
and
Yaser Abu-Mostafa

Department of Computer Science
California Institute of Technology

5205:TR:85

Two Theorems on Time Bounded Kolmogorov–Chaitin Complexity

David Schweizer and Yaser Abu-Mostafa

Department of Computer Science
California Institute of Technology

5205:TR:85

November 1985

This work was supported by Caltech's Program in Advanced Technologies, sponsored by Aerojet General, General Motors, GTE, and TRW.

Two Theorems on Time Bounded Kolmogorov–Chaitin Complexity

David Schweizer and Yaser Abu-Mostafa
Computer Science 256-80
California Institute of Technology
Pasadena, CA 91125

Abstract

An obvious extension of the Kolmogorov–Chaitin notion of complexity is to require that the program which generates a string terminate within a prespecified time bound. We show that given a computable bound on the amount of time allowed for the production of a string from the program which generates it, there exist strings of arbitrarily low Kolmogorov–Chaitin complexity which appear maximally random. That is, given a notion of fast, we show that there are strings which are generated by extremely short programs, but which are not generated by any fast programs shorter than the strings themselves. We show by enumeration that if we consider generating strings from programs some constant number of bits shorter than the strings themselves then these apparently random strings are significant (i.e are a proper fraction of all strings of a given length).

Notation and Definitions

We will be working over the input alphabet $\Sigma = \{0, 1\}$, and we write $\#$ for the blank symbol. We write $\mathbf{s}$ to denote a string in Σ^* and λ to denote the empty string. If $\mathbf{s} \in \Sigma^n$ (i.e. $\mathbf{s}$ is of length n) we write $|\mathbf{s}| = n$.

We take $\mathbf{U}$ to be a particular implementation of the Universal Turing Machine with input alphabet Σ and states $\{q_0, q_1, \ldots, q_K\}$, where q_0 is the start state, and q_1 is the halt state. $\mathbf{U}$ operates on *encodings* of Turing Machines. We write $\rho(\mathbf{M})$ to denote the encoding of the Turing Machine $\mathbf{M}$ and $\rho(\mathbf{M})\mathbf{w}$ to denote the binary string $\mathbf{w}$ concatenated onto the encoding of $\mathbf{M}$. Given $\rho(\mathbf{M})\mathbf{w}$ as input, $\mathbf{U}$ halts iff $\mathbf{M}$ halts on the input $\mathbf{w}$, and leaves as output precisely what $\mathbf{M}$ leaves. $\mathbf{U}$ is further defined not to halt if its input is not a syntactically correct encoding of a Turing Machine. It is convenient to think of $\mathbf{U}$ as having an input tape, several work tapes, and an output tape; we remind the reader that this is a convenience which changes nothing, as multi-tape machines are formally equivalent to single-tape machines.

We call a string of the form $\rho(\mathbf{M})\mathbf{w}$ a *program*. For fixed $\rho(\mathbf{M})$, we refer to the various extensions of the encoding to full programs as *instances* of the encoding. (Note that $\rho(\mathbf{M}) = \rho(\mathbf{M})\lambda =$ the instance of $\rho(\mathbf{M})$ with the empty string as input.)

Note that $\mathbf{U}$ is a Turing Machine, and therefore has an encoding. Further, that encoding can be incorporated into other machines. We use the term *simulation* to describe the situation of a Turing Machine calculating the action of another Turing Machine on a particular string. Simulations of the action of $\mathbf{U}$ are of great importance. We denote the machine which copies its input to its output tape and then halts by $\mathbf{E}$ (for Everhalting). The existence of this machine guarantees that every finite binary string is generated by some program.

The status of a Turing Machine computation at any moment can be represented by a quadruple called a *snapshot*. The quadruple $(q, \mathbf{v}, \sigma, \mathbf{w})$ indicates that the machine is in state q with the string $\mathbf{v}$ to the left of its head, the string $\mathbf{w}$ to the right of its head, the symbol σ currently being

scanned, and the rest of the tape blank. If the machine $\mathbf{M}$ goes from one snapshot to another in exactly one transition, we write

$$(q,\mathbf{v},\sigma,\mathbf{w}) \longrightarrow_{\mathbf{M}} (q',\mathbf{v}',\sigma',\mathbf{w}'),$$

and we define $\longrightarrow^{*}_{\mathbf{M}}$ to be the reflexive, transitive closure of $\longrightarrow_{\mathbf{M}}$. The *Kolmogorov–Chaitin complexity* of a string $\mathbf{s}$ is defined as:

$$K(\mathbf{s}) = \min\{\, |\mathbf{p}| : (q_0,\lambda,\#,\mathbf{p}) \longrightarrow^{*}_{\mathbf{U}} (q_1,\lambda,\#,\mathbf{s})\,\}.$$

We now define the relation $\longrightarrow^{t}_{\mathbf{M}}$ recursively as follows:

(i) $$(q_i,\mathbf{u},\sigma_j,\mathbf{v}) \longrightarrow^{0}_{\mathbf{M}} (q_i,\mathbf{u},\sigma_j,\mathbf{v}),$$

(ii) $$(q_{i_1},\mathbf{u_1},\sigma_{j_1},\mathbf{v_1}) \longrightarrow^{t}_{\mathbf{M}} (q_{i_2},\mathbf{u_2},\sigma_{j_2},\mathbf{v_2})$$

iff $\exists\ (q_i,\mathbf{u},\sigma_j,\mathbf{v})$ such that

$$(q_{i_1},\mathbf{u_1},\sigma_{j_1},\mathbf{v_1}) \longrightarrow^{t-1}_{\mathbf{M}} (q_i,\mathbf{u},\sigma_j,\mathbf{v})$$

and

$$(q_i,\mathbf{u},\sigma_j,\mathbf{v}) \longrightarrow_{\mathbf{M}} (q_{i_2},\mathbf{u_2},\sigma_{j_2},\mathbf{v_2}).$$

Finally, we define the *time bounded Kolmogorov–Chaitin complexity*:

$$K_\tau(\mathbf{s}) = \min\{\, |\mathbf{p}| : (q_0,\lambda,\#,\mathbf{p}) \longrightarrow^{\tau(|\mathbf{s}|)}_{\mathbf{U}} (q_1,\lambda,\#,\mathbf{s})\,\}.$$

Or,

$$K_\tau(\mathbf{s}) = \min\{\, |\mathbf{p}| : \mathbf{U} \text{ given } \mathbf{p} \text{ halts within } \tau(|\mathbf{s}|) \text{ steps leaving } \mathbf{s} \text{ as output }\}.$$

We always assume that τ is chosen large enough for $\mathbf{U}$ to reduce $\rho(\mathbf{E})\mathbf{s}$ to $\mathbf{s}$ within $\tau(|\mathbf{s}|)$ steps.

Observations

Given n, the set of all binary strings of length n can be generated in lexicographic order:

$$\underbrace{\overbrace{000\cdots0}^{n \text{ bits}},\ 000\cdots1,\ \ldots,\ 111\cdots0,\ 111\cdots1}_{2^n \text{ strings}}.$$

These strings can be used to specify numbers between 1 and 2^n in an obvious fashion. The set of binary strings of length $\leq n$ can also be generated:

$$\underbrace{\lambda,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ \ldots,\ \overbrace{111\cdots1}^{n \text{ bits}}}_{2^{n+1}-1 \text{ strings}}.$$

$\mathrm{K}_\tau(\mathbf{s})$ is computable for any computable τ. We simply simulate the operation of $\mathbf{U}$ on each binary string of length $\leq |\mathbf{s}| + |\rho(\mathbf{E})|$ for $\tau(|\mathbf{s}|)$ steps of computation and then leave as output the length of the shortest program which generated $\mathbf{s}$.

Theorem 1

Let τ and σ be any computable functions, with $\sigma(n) > n$. Then there exist strings $\mathbf{s}$ with $|\mathbf{s}| = \sigma(t)$ such that $K(\mathbf{s}) \leq t$ but $K_\tau(\mathbf{s}) \geq \sigma(t)$.

Proof

We construct a Turing Machine **M**. Call $|\rho(\mathbf{M})| = r$. We will consider all instances of $\rho(\mathbf{M})$ of length t ($t > r$, obviously).

The Turing Machine **M**:

- Computes $\sigma(t)$ and $\tau(\sigma(t))$. We note that t does not need to be specified as a parameter to the computation: the program remains available on the input tape of **U**.
- Generates a list $\mathcal{L}$ of all binary strings of length $\sigma(t)$ in lexicographic order.
- Simulates the operation of **U** on each binary string of length $\leq \sigma(t) - 1$ until it either halts or has computed for $\tau(\sigma(t))$ steps without halting.
- Removes from the list $\mathcal{L}$ every string of length exactly $\sigma(t)$ produced by a simulation. (A simulation must halt to be considered to have produced an output.)
- Prints the $\mathbf{w}^{\text{th}}$ remaining string. We show below that there are at least $2^{|\mathbf{w}|} + 1$ strings in $\mathcal{L}$ not produced by any simulation; hence the $\mathbf{w}^{\text{th}}$ remaining string exists.
- Halts.

We note that we are simulating **U** on $2^0 + 2^1 + \cdots + 2^{\sigma(t)-1} = 2^{(\sigma(t)-1)+1} - 1 = 2^{\sigma(t)} - 1$ possible programs. Further, the $2^{|\mathbf{w}|}$ instances of $\rho(\mathbf{M})$ do not halt within $\tau(\sigma(t))$ steps. Thus each instance of $\rho(M)$ of length t will produce a distinct string of length $\sigma(t)$.

We now have

$$(q_0, \lambda, \#, \rho(\mathbf{M})\mathbf{w}) \longrightarrow_{\mathbf{U}}^{*} (q_1, \lambda, \#, \mathbf{s})$$

where $|\mathbf{s}| = \sigma(t)$. Thus $K(\mathbf{s}) \leq t$. But by the operation of **M**, **s** is not produced within $\tau(\sigma(t))$ steps by any program of length $\leq \sigma(t) - 1$. Thus $K_\tau(\mathbf{s}) \geq \sigma(t)$. ∎

Theorem 2

For any fixed k and any computable function τ, there is an n_0 such that a constant fraction of all strings of length $n > n_0$ have $K(\mathbf{s}) \leq n - k$ but $K_\tau(\mathbf{s}) \geq n$.

Proof

Let $\sigma(m) = m + k$. We again call $|\rho(\mathbf{M})| = r$. Since $|\mathbf{w}| = t - r$, we have 2^{t-r} instances of the encoding $\rho(\mathbf{M})$, each of which produces a string of length $t + k$. Thus $2^{-(r+k)}$ — a fraction independent of n — of all strings of sufficient length can be compressed by k bits if we are patient, but appear maximally random if we require that generating programs run under the time constraint τ. (We require that $n_0 = r$. To see this suffices, take $\sigma(m) = m + 1$, and consider the string produced by $\rho(\mathbf{M})\lambda$.) ∎

Further Observations

Assume that τ and σ can be computed quickly. Then **M** runs in time $\sim 2^{\sigma(t)}\tau(\sigma(t))$. Thus we can construct **M'** with $\rho(\mathbf{M}')$ only slightly longer than $\rho(\mathbf{M})$ which will perform simulations for enough steps that all instances of $\rho(\mathbf{M})$ will halt. This machine prints strings which are even worse than those found by **M**, and prints almost as many strings as **M**.

Conclusions

All "real" computing is done under time constraints. This result says that there are strings which have a very strong pattern, and that that structure is in principle accessible, but which have such bad time complexity that they seem totally random to human beings. These strings

have hidden order, but the order is so well hidden that no amount of computation limited in advance is sufficient to determine that they are not random.

Acknowledgments

The first author would like to thank Young-il Choo for many, many helpful discussions.

Bibliography

[1] Chaitin, G. J., "A Theory of Program Size Formally Identical to Information Theory," *Journal of the ACM* **22** (1975), pp. 329–340.

[2] Davis, M. D. and E. J. Weyuker, *Computability, Complexity, and Languages*, New York: Academic Press, 1983.

[3] Kolmogorov, A. N., "Three Approaches to the Quantitative Definition of Information," *Problemy Peredachi Informatsii* **1** (1965), pp. 3–11.

[4] Lewis, H. R. and C. H. Papadimitriou, *Elements of the Theory of Computation*, Englewood Cliffs, New Jersey: Prentice–Hall, 1981.