On the Logical Depth Function and the Running Time of Shortest Programs

L. Antunes[a,1,*], A. Souto[b,2], P.M.B. Vitányi[c,3]

[a] *Instituto de Telecomunicações and Faculdade de Ciências Universidade do Porto*
[b] *Instituto de Telecomunicações and Instituto Superior Técnico, Universidade Técnica de Lisboa*
[c] *CWI and University of Amsterdam*

Abstract

For a finite binary string x its logical depth d for significance b is the shortest running time of a program for x of length $K(x) + b$. There is another definition of logical depth. We give a new proof that the two versions are close. There is an infinite sequence of strings of consecutive lengths such that for every string there is a b such that incrementing b by 1 makes the associated depths go from incomputable to computable. The maximal gap between depths resulting from incrementing appropriate b's by 1 is incomputable. The size of this gap is upper bounded by the Busy Beaver function. Both the upper and the lower bound hold for the depth with significance 0. As a consequence, the minimal computation time of the associated shortest programs rises faster than any computable function but not so fast as the Busy Beaver function.

1. Introduction

The logical depth is related to complexity with bounded resources. Computing a string x from one of its shortest programs may take a very long time. However, computing the same string from a simple 'print(x)' program of length about $|x|$ bits takes very little time.
A program for x of larger length than a given program for x may decrease the computation time but, except for pathological cases, does not increase it.

*Corresponding author.
Email addresses: lfa@dcc.fc.up.pt (L. Antunes), a.souto@math.ist.utl.pt (A. Souto), Paul.Vitanyi@cwi.nl (P.M.B. Vitányi)

[1]This work was supported by FCT projects PEst-OE/EEI/LA0008/2011 and PTDC/EIA-CCO/099951/2008. Departamento de Ciência de Computadores R.Campo Alegre, 1021/1055, 4169 - 007 Porto - Portugal.
[2]The author is supported by by FCT projects PEst-OE/EEI/LA0008/2011, PTDC/EIA-CCO/099951/2008 and the grant SFRH/BPD/76231/2011. Departamento de Matemática IST Av. Rovisco Pais, 1, 1049-001 Lisboa - Portugal.
[3]CWI, Science Park 123, 1098XG Amsterdam, The Netherlands

Therefore, except for pathological cases we associate the longest computation time with a shortest program for x. Such a program is incompressible. There arises the question how much time can be saved by computing a given string from a b-incompressible program (a program that can be compressed by at most b bits) when b rises.

1.1. Related Work

The minimum time to compute a string by a b-incompressible program was first considered in [4]. This minimum time is called the *logical depth* at significance b of the string concerned. Definitions, variations, discussion and early results can be found in the given reference. A more formal treatment, as well as an intuitive approach, was given in the textbook [10], Section 7.7. In [1] the notion of *computational* depth is defined as $K^d(x) - K(x)$ (see definitions below). This would equal the negative logarithm of the expression $Q_U^d(x)/Q_U(x)$ in Definition 3 if the following were proved. In [9] Levin proved in the so-called Coding Theorem that $-\log Q_U(x) = K(x)$ up to a constant additive term it remains to prove that $-\log Q_U^d(x) = K^d(x)$ up to a small additive term. The last equality is a major open problem in Kolmogorov complexity theory, see [10] Exercises 7.6.3 and 7.6.4.

1.2. Results

All computations below vary by the choice of reference optimal prefix machine. We prove that there is an infinite sequence of strings, $x_1, x_2, \ldots$ with $|x_{m+1}| = |x_m| + 1$, such that for every $m > 0$ each x_m is computed by b_1^m, b_2^m-incompressible programs in d_1^m, d_2^m steps, respectively, with $b_2^m = b_1^m + 1$ while $d_1^m - d_2^m$ rises faster than any computable function but not faster than the Busy Beaver function, the first incomputable function [11] (Theorem 2 and Corollary 1). We call b_j^m the "significance" of "logical depth" d_j^m of string x_m $(j = 1, 2)$. Then we prove (Theorem 3) for this infinite sequence of strings that the running times of shortest programs associated with this sequence rise faster than any computable function but again not so fast as the Busy Beaver function. We also give a new proof (provided by a referee) of a new result showing the closeness of the two versions of logical depth (Theorem 1).
The rest of the paper is organized as follows: in Section 2, we introduce some notation, definitions and basic results needed for the paper. In Section 4, we prove the results mentioned.

2. Preliminaries

We use *string* or *program* to mean a finite binary string. The alphabet is $\Sigma = \{0, 1\}$, and $\Sigma^* = \{0, 1\}^*$ is the set of all strings. Strings are denoted by the letters x, y and z. The *length* of a string x (the number of occurrences of bits in it) is denoted by $|x|$, and the *empty* string by ϵ. Thus, $|\epsilon| = 0$. We use the notation Σ^n for the set of strings of length n. We use the binary logarithm which is denoted by "log". Given two functions f and g, we say that $f \in O(g)$

if there is a constant $c > 0$, such that $f(n) \leq c \cdot g(n)$, for all but finitely many natural numbers n.

2.1. Time Bounds

Often the resource-bounds are *time constructible*. There are many definitions. For example, there is a Turing machine whose running time is exactly $t(n)$ on every input of size n, for some function $t : \mathbb{N} \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. However, in this paper there are functions that are not time constructible. An example is the Busy Beaver function $BB : \mathbb{N} \to \mathbb{N}$ (Definition 5) which is not computable (it rises faster than any computable function). In this case, and when we just mean a number of steps, we indicate in the superscript the number of steps taken, usually using d. Given a program p, we denote its running time (the number of steps taken by the reference optimal prefix Turing machine defined below) by $time(p)$.

2.2. Computability

A pair of nonnegative integers, such as (p, q) can be interpreted as the rational p/q. We assume the notion of a computable function with rational arguments and values. A function $f(x)$ with x rational is *semicomputable from below* if it is defined by a rational-valued total computable function $\phi(x, k)$ with x a rational number and k a nonnegative integer such that $\phi(x, k + 1) \geq \phi(x, k)$ for every k and $\lim_{k\to\infty} \phi(x, k) = f(x)$. This means that f (with possibly real values) can be computed in the limit from below (see [10], p. 35). A function f is *semicomputable from above* if $-f$ is semicomputable from below. If a function is both semicomputable from below and semicomputable from above then it is *computable*.

2.3. Kolmogorov Complexity

We refer the reader to the textbook [10] for details, notions, and history. We use Turing machines with a read-only one-way input tape, one or more (a finite number) of work tapes at which the computation takes place, and a one-way write-only output tape. All tapes are semi-infinite, divided into squares, and each square can contain a symbol from a given alphabet or blanks. The machine uses for all of its tapes a finite alphabet and all tapes are one-way infinite. Initially, the input tape is inscribed with a semi-infinite sequence of 0's and 1's. The other tapes are empty (contain only blanks). At the start, all tape heads scan the leftmost square on their tape. If the machine halts for a certain input then the contents of the scanned segment of input tape is called the *program* or *input*, and the contents of the output tape is called the *output*. The machine thus described is a *prefix Turing machine*. Denote it by T. If T terminates, then the program is p and the output is $T(p)$. The set $\mathcal{P} = \{p : T(p) < \infty\}$ is *prefix-free* (no element of the set is a proper prefix of another element). By the ubiquitous Kraft inequality [8] we have

$$\sum_{p\in\mathcal{P}} 2^{-|p|} \leq 1. \tag{1}$$

The same holds for a fixed conditional or auxiliary. The above unconditional case corresponds to the case where the conditional is ϵ. Among the universal prefix-free Turing machines we consider a special subclass called *optimal*, see Definition 2.0.1 in [10]. To illustrate this concept let $T_1, T_2, \ldots$ be a standard enumeration of (prefix) Turing machines, and let U_1 be one of them. If $U_1(i, pp) = T_i(p)$ for every index i and program p and outputs 0 for inputs that are not of the form pp (doubling of p), then U_1 is also universal. However, U_1 can not be used to define Kolmogorov complexity. For that we need a machine U_2 such that $U_2(i, p) = T_i(p)$ for every i, p. A machine such as U_2 is called an *optimal* prefix Turing machine. Optimal prefix Turing machines are a strict subclass of universal prefix Turing machines. The above example illustrates the strictness. To define Kolmogorov complexity we require optimal prefix Turing machines and not just universal prefix Turing machines. The term 'optimal' comes from the founding paper [7].
It is still possible that two different optimal prefix Turing machines have different computation times for the same input-output pairs or even different sets of programs. To avoid these problems we fix a reference machine. Necessarily, the reference machine has a certain number of worktapes. A well-known result of [6] states that n steps of a multiworktape prefix Turing machine can be simulated in $O(n \log n)$ steps of a two-worktape prefix Turing machine. Thus, for such a simulating optimal Turing machine U' we have $U'(i, p) = T_i(p)$ for all i, p; if $T_i(p)$ terminates in time $t(n)$ then $U'(i, p)$ terminates in time $O(t(n) \log t(n))$. Altogether, we fix such an simulating (as above) optimal prefix Turing machine U' and call it the *reference optimal prefix Turing machine* U.

DEFINITION 1. *Let U be the reference optimal prefix-free Turing machine, and x, y be strings. The* prefix-free Kolmogorov complexity *$K(x|y)$ of x given y is defined by*

$$K(x|y) = \min\{|p| : U(p, y) = x\}.$$

The notation $U^d(p, y) = x$ means that $U(p, y) = x$ within d steps. The d-time-bounded prefix-free Kolmogorov complexity *$K^d(x|y)$ of x given y is defined by*

$$K^d(x|y) = \min\{|p| : U^d(p, y) = x\}.$$

The default value for the auxiliary input y for the program p, is the empty string ϵ. To avoid overloaded notation we usually drop this argument in case it is there. Let x be a string. Denote by x^* the first shortest program in standard enumeration such that $U(x^*) = x$.

DEFINITION 2. *Let $|x| = n$. The string x is* c-incompressible *if*

$$K(x) \geq n + K(n) - c.$$

The string x given n is c-incompressible *if*

$$K(x|n) \geq n - c.$$

A simple counting argument can show the existence of c-incompressible strings (c-incompressible strings given their length) of every length for the plain complexity $C(x)$. Since $K(x) \geq C(x)$ (respectively $K(x|n) \geq C(x|n)$) we have the following:

LEMMA 1. *There are at least $2^n(1-2^{-c})+1$ strings $x \in \Sigma^n$ (given n) that are c-incompressible with respect to prefix-free Kolmogorov complexity.*

3. Logical Depth

The logical depth [4] consists of two versions. One version is based on $Q_U(x)$, the so-called *a priori* probability [10] and its time-bounded version. Here $U^d(p)$ means that $U(p)$ terminates in at most d steps.

$$Q_U(x) = \sum_{U(p)=x} 2^{-|p|}, \qquad Q_U^d(x) = \sum_{U^d(p)=x} 2^{-|p|}.$$

For convenience we drop the subscript and consider U as understood.

DEFINITION 3. *Let x be a string, b a nonnegative integer. The logical depth, tentative version 1, of x at significance level $\varepsilon = 2^{-b}$ is*

$$depth_\varepsilon^{(1)}(x) = \min\left\{d : \frac{Q^d(x)}{Q(x)} \geq \varepsilon\right\}.$$

Using a program that is longer than another program for output x can shorten the computation time. Thus, the b-significant logical depth of an object x is defined as the minimal time the reference optimal prefix Turing machine needs to compute x by a b-incompressible program (one that can be compressed by at most b bits). This leads to the following.

DEFINITION 4. *Let x be a string, b a nonnegative integer. The logical depth, tentative version 2, of x at significance level b, is:*

$$depth_b^{(2)}(x) = \min\{time(p) : |p| \leq K(p) + b \wedge U(p) = x\}.$$

*In this case we say that the string x is (d, b)-*deep.

REMARK 1. It is easy to see that $\text{depth}_0^{(2)}(x)$ is the least number of steps to compute x from an incompressible program. For example, x^* is known to be incompressible up to an additive constant [10]. Thus, $time(x^*) \geq \text{depth}_0^{(2)}(x)$. For higher b the value of $\text{depth}_b^{(2)}(x)$ is monotonic nonincreasing until

$$\text{depth}^{(2)}_{|x|-K(x)+O(1)}(x) = O(|x| \log |x|),$$

the $O(1)$ term represents a program to copy the literal representation of x in $O(|x| \log |x|)$ steps. It is the aim of this paper to study the properties the graph of f_x can have. For example, if x is random (i.e., $|x| = n$ and $K(x) \geq n + K(n)$) then always $b = O(1)$ and always $d = O(n \log n)$. These x's, but not only these, are called *shallow*. ◊

Version (2) is stronger than version (1) in that in the version (2) every individual program at significance level b must take at most $\mathrm{depth}_b^{(2)}(x)$ steps to compute x, while version (1) to be equivalent would require only that a weighted average of all programs for x require at most $\mathrm{depth}_{2^{-b}}^{(1)}(x)$ steps. The quantitative difference between the two versions of logical depth is small as the following statement shows. Compare with Theorem 7.7.1 together with Exercise 7.7.1 in [10]. The anonymous referee provided a slightly different statement and the new proof below.

THEOREM 1. *Let x be (d,b) deep. Then*

$$\frac{1}{2^{b+K(b)+O(1)}} \leq \frac{Q^d(x)}{Q(x)} < \frac{1}{2^{b+1}}.$$

PROOF.
(Right $<$) This inequality follows from the next claim:

CLAIM 1. *If for integers $d, b > 0$ the total a priori probability $Q^d(x)$ of all programs that compute x within d steps is at least $2^{-b-1}Q(x)$, then one of these programs is $(b+1)$-incompressible.*

PROOF. Indeed, if for some c all programs computing x within d steps are c-compressible, then the twice iterated reference optimal Turing machine (in its role as decompressor) computes x with probability $2^c Q^d(x) \geq 2^{c-b-1}Q(x)$ from the c-compressed versions. But $Q(x) \geq 2^{c-b-1}Q(x) + 2^{-b-1}Q(x)$. Hence $c - b - 1 < 0$ that is $c < b + 1$. Hence there is a program computing x within d steps that is $(b+1)$-incompressible. Then x may be $(d, b+1)$-deep contradicting the assumption that x is (d,b)-deep. Hence $Q^d(x) < 2^{-b-1}Q(x)$. □

(Left $\leq$) This inequality follows from the next claim:

CLAIM 2. *If for integers d, b there exists a b-incompressible program that computes x in time d, then $Q^d(x) \geq 2^{-b-K(b)-O(1)}Q(x)$.*

PROOF. Assume by way of contradiction that $Q^d(x) < 2^{-B}Q(x)$ for some B (see below the considerations about the choice of B). Consider the following lower semicomputable semiprobability (the total probability is less than 1): for each string x we enumerate all programs p that compute x in order of halting (time), and assign to each halting p the probability $2^{-|p|+B}$ until the total probability would pass $Q(x)$ with the next halting p. Since $Q(x)$ is lower semicomputable we can postpone assigning probabilities. But eventually or never for some p the total probability may pass $Q(x)$ and this p and all subsequent halting p's for x get assigned probability 0. Therefore, the total probability assigned to all halting programs for x is less than $Q(x)$. Since by assumption $Q^d(x) < 2^{-B}Q(x)$ we have $2^B Q^d(x) = \sum_{U^d(p)=x} 2^{-|p|+B} < Q(x)$. Since $Q(x) < 1$ we have $Q^d(x) < 2^{-B}$ and therefore all programs that compute x in at most d steps are $(B - O(1))$-compressible given B, and therefore $(B - K(B) - O(1))$-compressible.

By assumption there exists a b-incompressible program from which x can be computed in d steps. We obtain a contradiction with this for $B-K(B)-O(1) \geq b$. This is the case if we set $B = b + K(b) + c$ for a large enough constant $c > 0$. Namely $B-K(B)-O(1) = b+K(b)+c-K(b+K(b)+c)-O(1) \geq b+K(b)+c-K(b+K(b))-K(c)-O(1) > b$ (note that $K(b+K(b)) \leq K(b,K(b))+O(1) \leq K(b)+O(1)$ by some easy argument [10] and $K(c) = O(\log c) < c/2$ for large c). Therefore, $Q^d(x) < 2^{-B}Q(x)$ is contradicted. Hence $Q^d(x) \geq 2^{-B}Q(x)$ which implies $Q^d(x) \geq 2^{-b-K(b)-O(1)}Q(x)$. By assumption x is (d,b)-deep satisfying the condition of this case. □

□

REMARK 2. We can replace $K(b)$ by $K(d)$ by changing the construction of the semiprobability: knowing d we generate all programs that compute x within d steps and let the semiprobabilities be proportional to $2^{-|p|}$ and the sum be at most $Q(x)$. In this way $K(b)$ in Theorem 1 is substituted by $\min\{K(b), K(d)\}$. ◊

Theorem 1 shows that the quantitative difference between the two versions of logical depth is small. We choose version 2 as our final definition of logical depth.

DEFINITION 5. *The* Busy Beaver function $BB : \mathbb{N} \to \mathbb{N}$ *is defined by*

$$BB(n) = \max_{p:|p|\leq n} \{\text{running time of } U(p) < \infty\}$$

4. The graph of logical depth

Even slight changes of the significance level b can cause large changes in logical depth.

LEMMA 2. *Let n be large enough. There exist strings x of length n such that the running time of a computation from a shortest program to x is incomputable.*

PROOF. By [5] we have $K(K(x)|x) \geq \log n - 2\log\log n - O(1)$. (This was improved to the optimal $K(K(x)|x) \geq \log n - O(1)$ recently in [3].) Hence there is no computable function $\phi(x) = \min\{d : U^d(p) = x,\ |p| = K(x)\}$. If there were, then we could run U for d steps on any program of length $n + O(\log n)$. Among the programs that halt within d steps we select the ones which output x. Subsequently, we select from this set a program of minimum length. This is a shortest program for x of length $K(x)$. Therefore, the assumption that ϕ is computable implies that $K(K(x)|x) = O(1)$ and hence a contradiction. □

The following result was mentioned informally in [4].

LEMMA 3. *The running time of a program p is at most $BB(|p| + O(\log |p|))$. The running time of a shortest program for a string x of length n is at most $BB(n + O(\log n))$.*

PROOF. The first statement of the lemma follows from Definition 5. For the second statement we use the notion of a simple prefix-code called a *self-delimiting* code. This is obtained by reserving one symbol, say 0, as a stop sign and encoding a string x as 1^x0. We can prefix an object with its length and iterate this idea to obtain ever shorter codes: $\bar{x} = 1^{|x|}0x$ with length $|\bar{x}| = 2|x|+1$, and $x' = \overline{|x|}x$ of length $|x| + 2||x|| + 1 = |x| + O(\log |x|)$ bits. From this code x is readily extracted. The second statement follows since $K(x) \leq |x| + O(\log |x|)$. □

THEOREM 2. *Let n be large enough. There is a string x of every length n such that $\text{depth}_i(x)$ is incomputable and $\text{depth}_{i+1}(x)$ is computable for some i that satisfies $0 \leq i \leq n + O(\log n)$.*

PROOF. Let x be a string of length n as in Lemma 2. Then $\text{depth}_0(x)$ is incomputable. However, $\text{depth}_{n-K(x)+O(\log n)}(x) = O(n \log n)$ and therefore computable. Namely, a self-delimiting encoding of x can be done in $n+O(\log n)$ bits. Let q be such an encoding with $q = 1^{||x||}0|x|x$ (where $||x||$ is the length of $|x|$). Let r be a self-delimiting program of $O(1)$ bits which prints the encoded string in the next self-delimiting string. Consider the program rq. Since x can be compressed to length $K(x)$, the running time $\text{depth}_{n-K(x)+O(\log n)}(x)$ is at most the running time of rq which is $O(n \log n)$.
Consider the sequence $\text{depth}_0(x), \ldots, \text{depth}_{n-K(x)+O(\log n)}(x)$. Since $\text{depth}_0(x)$ is incomputable and $\text{depth}_{n-K(x)+O(\log n)}(x)$ is computable, there must be an i satisfying $0 \leq i < n - K(x) + O(\log n)$ such that $\text{depth}_i(x)$ is incomputable and $\text{depth}_{i+1}(x)$ is computable. Since the set of all such i's is finite and discrete it has a maximum. □

Let x be as in Theorem 2 and $i_{\max}$ be the i which reaches the maximum of $\text{depth}_i(x) - \text{depth}_{i+1}(x)$, $0 \leq i \leq n + O(\log n)$.

COROLLARY 1. There exists an infinite sequence of strings $x_1, x_2, \ldots$ with $|x_{m+1}| = |x_m| + 1$, each x_m with an $i_{m,\max}$ $(0 \leq i < |x_m| - K(x_m) + O(\log |x_m|))$, and $h(m) = \text{depth}_{i_{m,\max}}(x_m) - \text{depth}_{i_{m+1,\max}}(x_m)$ is the maximal gap in the logical depths of which the significance differs by 1. The function $h(m)$ rises faster than any computable function but is not faster than $BB(|x_m| + O(\log |x_m|)$ by Lemma 3.

THEOREM 3. *There is an infinite sequence $x_1, x_2, \ldots$ with $|x_{m+1}| = |x_m| + 1$ such that $h(m) \leq \text{depth}_0(x_m) \leq BB(|x_m| + O(\log |x_m|))$. (Note that $\text{depth}_0(x_m)$ is the shortest time of a computation of a shortest program for x by U.)*

PROOF. The logical depth function $\text{depth}_b(x)$ is monotonic nonincreasing in the significance argument b for all strings x by its Definition 4. By Lemma 3 and Corrollary 1 the theorem follows. □

5. Conclusion

We studied the behavior of the logical depth function associated with a string x of length n. This function is monotonic nonincreasing. For argument 0 the logical depth is the minimum running time of the computation from a shortest program for x to x. The function decreases to $O(n \log n)$ for the argument $|x|-K(x)+O(\log |x|)$. We show that there is an infinite sequence of strings such that the difference in logical depths of some significance levels differing by one rises faster than any computable function, that is, incomputably fast, but not more than the Busy Beaver function. This shows that logical depth can increase tremendously for only an incremental difference in significance. Moreover, there is an infinite sequence of strings such that the minimal computation times of associated shortest programs rises incomputably fast but not so fast as the Busy Beaver function with as argument twice the length of the string plus an additive logarithmic term in the length.

Acknowledgments

We thanks Bruno Bauwens for helpful discussions and comments, and the anonymous referee for additional comments and the new proof of Theorem 1.

References

[1] L. Antunes, L. Fortnow, D. van Melkebeek, and N. Vinodchandran. Computational depth: concept and applications. *Theoretical Computer Science*, 354(3):391–404, Elsevier Science Publishers Ltd., 2006.

[2] J.M. Barzdin, Complexity of programs to determine whether natural numbers not greater than n belong to a recursively enumerable set, Sov. Mat. Dokl., 9(1968), 1251–1254.

[3] B. Bauwens, A. Shen, Complexity of complexity and maximal plain versus prefix-free Kolmogorov complexity. *J. Symb. Logic, 2013*, To appear.

[4] C. Bennett. *Logical depth and physical complexity*, pages 227–257. Oxford University Press, Inc., New York, NY, USA, 1988.

[5] P. Gács. On the symmetry of algorithmic information, *Soviet Math. Dokl.*, 15:1477–1480, 1974. Correction, Ibid., 15:1480, 1974.

[6] F.C. Hennie and R.E. Stearns. Two tape simulation of multitape Turing machines. *J. Assoc. Comput. Mach.*, 4:533–546, 1966.

[7] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1–7, 1965.

[8] L.G. Kraft. A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, Dept. of Electrical Engineering, M.I.T., Cambridge, Mass., 1949.

[9] L. Levin. Laws of information conservation (non-growth) and aspects of the foundation of probability theory. *Problems Information Transmission*, 10:206–210, Russian Academy of Sciences, 1974.

[10] M. Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 2008.

[11] T. Rado. On non-computable functions. *Bell System Tech. J.*, XX:877–884, 1962.