On the Logical Depth Function

L. Antunes[1] and A. Souto[2] and A. Teixeira[3] and P.M.B. Vitányi[4]

1,3 Instituto de Telecomunicações and
Faculdade de Ciências Universidade do Porto
2 Instituto de Telecomunicações and
Instituto Superior Técnico, Universidade Técnica de Lisboa
4 CWI and University of Amsterdam

Abstract

We investigate the logical depth function of finite strings. For the function associated with string x with argument b and value d we have that d is the least running time of the computation of an element of the set of b-incompressible programs for x. For a given string we consider the possible gap between two values of this function if the arguments differ by just a constant. We show that there is an infinite sequence of strings, every successor one bit longer than its predecessor, such that the widths of the associated gaps rise at least as fast as the Busy Beaver function, that is, faster than any computable function. As a consequence, the computation time of shortest programs associated with this sequence of strings plus programs that are a certain number of bits longer rises (like the Busy Beaver function) faster than any computable function.

Classification: Logical depth, Kolmogorov complexity, Information measures, Busy Beaver function.

⋆ This work was supported by FCT projects PEst-OE/EEI/LA0008/2011 and PTDC/EIA-CCO/099951/2008. The authors are also supported by the grants SFRH/BPD/76231/2011 and SFRH/BD/33234/2007 of FCT.
Adress authors 1,3: Departamento de Ciência de Computadores R.Campo Alegre, 1021/1055, 4169 - 007 Porto - Portugal. Email author 1: lfa@dcc.fc.up.pt,Email author 3: andreiasofia@ncc.pt.
Adress author 2: Departamento de Matemática IST Av. Rovisco Pais, 1, 1049-001 Lisboa - Portugal. Email: a.souto@math.ist.utl.pt.
Author 4: CWI, Science Park 123, 1098XG Amsterdam, The Netherlands. Email: Paul.Vitanyi@cwi.nl.

1 Introduction

To compute a string x from a shortest program for it may take a very long time. However, to compute the same string from a program of about length $|x|$ that essentially spells out x takes very little time, since we can simply copy. A program for x of larger length than a given program for x may decrease the computation time but certainly does not increase it. Therefore, the longest computation time is associated 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 [3]. 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 (but see Remark 1). A more formal treatment as well as an intuitive approach was given in the textbook [8], Section 7.7. In [2] the notion of *computational* depth is defined as $K^t(x) - K(x)$ (see definitions below). This is similar to the negative logarithm of the expression $Q_U^d(x)/Q_U(x)$ in Definition 3, since [7] proved in the so-called Coding Theorem that $-\log Q_U(x) = K(x)$ up to a constant additive term but requires also $-\log Q_U^t(x) = K^t(x)$ up to a small (preferable constant) additive term. The last equality is a major open problem in Kolmogorov complexity theory, see [8] Exercises 7.6.3 and 7.6.4.

1.2 Results

We prove that there is an infinite sequence of strings, each string one bit longer than its predecessor, such that for each string in the sequence the difference in logical depths associated with a fixed constant difference in significance increases exponentially (Theorem 3). We prove next (Theorem 4 and Corollary 2) for (possibly another) infinite sequence of strings that this exponential increase can be, in fact, an increase as fast as the Busy Beaver function, the first incomputable function [9]. In fact, this function is total and rises faster than any computable function. Consequently, the running times of shortest programs associated with this sequence and certain longer programs rises faster than any computable function (Corollary 3). Therefore the function such that its nth value is equal to the logical depth of the nth string in the sequence at appropriate low fixed constant significance, is incomputable.

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 3, 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 also use the binary logarithm which is denoted by "log."

Often the resource-bounds in this paper 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 also many functions that are not time constructible. An example is the Busy Beaver function $BB : \mathbb{N} \to \mathbb{N}$ (Definition 7) 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 by $time(p)$. 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 $n \in \mathbb{N}$.

2.1 Kolmogorov Complexity

We refer the reader to the textbook [8] 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 [6] 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 [8]. 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 [5].

Definition 1 *Let U be an 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*

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

Let $t : \mathbb{N} \to \mathbb{N}$ be a time-constructible function. The notation $U^t(p, y) = x$ means that $U(p, y) = x$ within $t(|x|)$ steps. The t-time-bounded prefix-free Kolmogorov complexity *$K^t(x|y)$ of x given y is defined by*

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

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. A well-known result of [4] 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, we can choose a reference optimal Turing machine U such that $U(i, p) = T_i(p)$ for all i, p. If $T_i(p)$ terminates in time $t(n)$ then $U(i, p)$ terminates in time $O(t(n) \log t(n))$. Let x be a string. Denote by x^* the first shortest program in standard enumeration such that $U(x^*) = x$.

Definition 2 *A string x is* c-incompressible *if*

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

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

following:

THEOREM 1 *There are at least* $2^n(1-2^{-c})+1$ *strings* $x \in \Sigma^n$ *that are* c*-incompressible.*

The definition of logical depth [3] is based on $Q_U(x)$, the so-called *a priori* probability [8] and its time bounded version. Here $U^d(p)$ means that $U(p)$ terminates in at most d steps.

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

We give two tentative definitions.

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

$$depth_\varepsilon^{(1)}(x) = \min_d \left\{ d : \frac{Q_U^d(x)}{Q_U(x)} \geq \varepsilon \right\}$$

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 Turing machine needs to compute x by a b-incompressibe program (one that can be compressed by at most b bits).

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

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

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) requires only that a weighted average of all programs for x require at most $\mathrm{depth}_{2^{-b}}^{(1)}(x)$ steps. The quantitative difference is small as the following theorem shows (a combination of Theorem 7.7.1 and Exercise 7.7.1 in [8]).

Remark 1 The originating reference [3] can be used for inspiration but is not everywhere trustworthy. For example on page 245 first paragraph it states that "Given a string x, its length n and a value of the significance parameter s ... one can compute the depth." This is false since Definition 3 is incomputable since $Q_U(x)$ is incomputable because of the halting problem, and Definition 4 is incomputable because $K(p)$ is incomputable. $\diamond$

THEOREM 2 *A string* x *satisfies* $d = depth_{2^{-b}}^{(1)}(x)$ *(*b *up to precision* $\min\{K(d), K(b)\}+$

$O(1)$*) if and only if* d *is the least number of steps needed by a* b*-incompressible program to print* x*.*

In [8] the precise form of the theorem is given as

$$\frac{1}{2^{b+\min\{K(d),K(b)\}+O(1)}} \leq \frac{Q_U^d(x)}{Q_U(x)} \leq \frac{1}{2^{b+O(1)}},$$

in the sense that the proof of right inequality shows the "if" part in the above statement of Theorem 2 while the proof of the left inequality shows the "only if" part.

Notice that by the Coding Theorem of [7] which states $-\log Q_U(x) = K(x) + O(1)$ we have:

$$\begin{aligned}\frac{2^{-K(x)+O(1)}}{2^{b+\min\{K(d),K(b)\}+O(1)}} &= \frac{1}{2^{K(x)+\min\{K(d),K(b)\}+b+O(1)}} \\ \leq Q_U^d(x) \leq \frac{2^{-K(x)+O(1)}}{2^{b+O(1)}} &= \frac{1}{2^{K(x)+b+O(1)}}\end{aligned}$$

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

Definition 5 A string x is (d, b)-*deep* if Definition 4 holds.

This means that the logical depth of a string x can for all purposes can be considered as a function. As a function it is easier to interpret.

Definition 6 The function $f_x : \mathbb{N} \to \mathbb{N}$ defined by $f_x(b) = d$, where d is the least number of steps needed by a b-incompressible program to print x, is the *functional logical depth* of x.

Remark 2 It is easy to see that $f_x(0)$ is the least number of steps to compute x from an incompressible program. For example, x^* is known to be incompressible [8]. Thus, $time(x^*) \geq f_x(0)$. (As far as the authors know it is not known that if $U(p) = x$ and p is incompressible then p is a shortest program for x.) For higher arguments f_x is monotonic nonincreasing until $f_x(|x| - K(x) + O(1)) = O(|x| \log |x|)$, the $O(1)$ term represents a program to copy the literal representation of x in $O(|x| \log |x|)$ steps. $\diamond$

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*.

To make comparisons between the logical depths of two arguments feasible, one can scale the running times for different programs. Here we use, as scaling factor, a Busy Beaver function as was first done in [1].

DEFINITION 7 *The* Busy Beaver function $BB : \mathbb{N} \to \mathbb{N}$ *is defined by*

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

DEFINITION 8 *The* Busy Beaver logical depth*, at significance level b, of a string x is defined by*

$$depth_b^{BB}(x) = \min_l \{l : |p| \leq K(p) + b \text{ and } U(p) = x \text{ in time at most } BB(l)\}.$$

The Busy Beaver functional *logical depth, at significance level b, of a string x is defined by*

$$f_x^{BB}(b) = \min_l \{l : |p| \leq K(p) + b \text{ and } U(p) = x \text{ in time at most } BB(l)\}.$$

If b is large enough so that we can have $p = qx^*$ with $|q| \leq b$ and q a copy program running in time polynomial in $|x|$, then there exists a p as used in this definition since $BB(l) \to \infty$ with growing l. Notice that Definition 8 rescales the logical depth of Definition 4, since $BB^{-1}(\text{depth}_b(x)) = \text{depth}_b^{BB}(x)$. From Definition 8 it follows directly that $K(\text{depth}_b^{BB}(x)) \leq K(x, b) + O(1)$. To see this, one can simulate any computation that terminates, keeping track of the number of computation steps and thus, $K(time(p)) \leq K(p) + O(1)$ which implies the inequality.

3 The graph of logical depth

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

THEOREM 3 *Let c, k_1 and k_2 be appropriate constants. For each large enough n there is a string x of length n such that $depth_{k_2}(x) \geq 2^n$ and $depth_{2c+k_1}(x) = O(n \log n)$. (In terms of the functional logical depth $f_x(k_2) \geq 2^n$ and $f_x(2c + k_1) = O(n \log n)$).*

PROOF. Assume the conditions in the theorem. Consider the set

$$A = \{x \in \Sigma^n : |p| < n - c \wedge U(p) = x \text{ in at most } 2^n \text{ steps }\}.$$

The set A is a subset of the c-compressible strings. Let $B = \Sigma^n \setminus A$. Then B contains all c-incompressible strings, and therefore $|B| > 2^n(1 - 2^{-c})$. (Note

that B also contains c-compressible strings, for instance y such that $U(y^*) = y$ in more than 2^n steps and $|y^*| < n-c$.) Let $x \in B$ be a $(c+k_1)$-incompressible string such that for all strings p with $U(p) = x$ and $|p| \leq n + O(\log n)$ holds

$$n - c - k_1 \leq K(p) \leq n - c - k_2, \tag{2}$$

where $0 < k_2 < k_1$. We defer the proof that such strings x exist to Lemma 1. Assume for now that these strings exist.

Claim 3 *$depth_{k_2}(x) \geq 2^n$.*

Proof. Assume that $\text{depth}_{k_2}(x) < 2^n$. By Definition 5 of logical depth, there exists a k_2-incompressible p with $U(p) = x$ in time less then 2^n such that (2) holds. Therefore

$$K(p) + k_2 \leq n - c - k_2 + k_2 = n - c.$$

Hence $x \in A$: contradiction. □

Claim 4 *$depth_{2c+k_1}(x) = O(n \log n)$.*

Proof. Let c be the length of a shortest prefix-free version of the program `print`. Then `print`(x) with x literal has length at most $n+c$. By Definition 5 of logical depth there exist $(2c+k_1)$-incompressible programs p with $U(p) = x$ such that by (2) we have:

$$K(p) + 2c + k_1 \geq n - c - k_1 + 2c + k_1 = n + c.$$

Let `print`(x) be $(2c+k_1)$-incompressible. That is, x is $(c+k_1)$-incompressible. Then $\text{depth}_{2c+k_1}(x)$ is at most the running time of `print`(x), which is at most $O(n \log n)$. □ □

Lemma 1 *Let $c, k_1, k_2 \geq 0$ be large enough constants $0 < k_2 < k_1$, n a large enough positive integer, and B the set described in the last proof. There exists a $(c+k_1)$-incompressible string $x \in B$ such that for all strings p with $U(p) = x$ and $|p| = n + O(\log n)$ inequality (2) holds.*

Proof. Let $a, b \geq 0$ be constants to be chosen later, n large enough, and q be the lexicographic first program (not necessarily for x) of length $n - c - a$ with the longest running time of all programs of these lengths. The *longest running time* of a a string is defined as the maximal running time of a prefix of the string for which the computation halts. Therefore q has the longest running time of all programs of length at most $n - c - a$. To see this, suppose there is a program q' with $|q'| < n - c - a$ with a longer running time than q. Then we can pad q' with nonsignificant 0's to the length of q. We write this longest

running time of q (the maximum of the running times of the halting prefixes of q) as $time'(q)$.

Let x with $|x| = n$ be the first string in lexicographic order such that every program p with $U(p) = x$ and $|p| \leq n + O(\log n)$ satisfies

$$K^{time'(q)}(p) \geq n - c - a. \tag{3}$$

(Such x exist. For example, if $|y| = n$ and $K(y) \geq n - c - a$ then (3) with r instead of p is satisfied for all strings r such that $U(r) = y$.) For the above Definition 3 we have for all such p

$$K(p) \leq |q| + b \leq n - c - a + b$$

for a constant $b \geq 0$. Namely, from q we can compute $time'(q)$ and $n - c - a$ (since it is equal to $|q|$), from this we can compute the set of p's satisfying (3). Since we are only interested in at most $n + O(\log n)$ length p's the set of relevant p's is finite. By scrutinizing this finite set and knowing n through the constant $c - a$ while we knew $n - c - a$ we can determine the string x. All this can be done (including knowing the constant $c - a$) using a b-length program with $b > 0$ a constant. Moreover, we have for every p with $U(p) = x$ and $|p| \leq n + O(\log n)$ that

$$K(p) \geq n - c - a.$$

To see this, fix p. To the contrary assume $K(p) < n - c - a$. The construction of q and $K(p) < n - c - a$ imply $time(p^*) \leq time'(q)$. Therefore, $K^{time(p^*)}(p)(= p) = K^{time'(q)}(p) < n - c - a$ contradicting (3). Since we can choose $p = x^*$ this also proves that x of length n is $(c - a)$-incompressible.

Set $a > b$ and $k_1 = a$ and $k_2 = a - b$. This proves the lemma. □

THEOREM 4 *Let c, k_1, and k_2 be appropriate constants. For every large enough n there is a string x of length n such that $depth^{BB}_{k_2}(x) \geq n$ and $depth^{BB}_{2c+k_1}(x) = O(\log n)$. (In terms of the functional logical depth $f^{BB}_x(2c+k_1) = O(\log n)$ and $f^{BB}_x(k_2) \geq n$)*

PROOF. The proof of Theorem 3 gives the intuition to understand the idea of the present proof. Consider

$$A = \{x \in \Sigma^n : |p| < n - c \ \wedge \ U(p) = x \text{ in at most } BB(n) \text{ steps}\}.$$

Taking $B = \Sigma^n \backslash A$, we know that B has at least $2^n(1-2^{-c})$ elements. Let $x \in B$ be a $(c + k_1)$-incompressible string such that for all strings p with $U(p) = x$ and $|p| \leq n + O(\log n)$ inequality (2) holds. Such x exist by Lemma 1.

Claim 5 *$depth^{BB}_{k_2}(x) \geq n$.*

PROOF. Assume that $\text{depth}^{BB}_{k_2}(x) < n$. Then, by definition of Busy Beaver logical depth there exists a k_2-incompressible p with $U(p) = x$ such that by (2) we have

$$K(p) + k_2 \leq n - c - k_2 + k_2 = n - c$$

such that $U(p) = x$ in time less than $BB(n)$. Hence $x \in A$: contradiction. □

Claim 6 *$depth^{BB}_{2c+k_1}(x) \leq O(\log n)$.*

PROOF. Let c be the length of a shortest prefix-free version of the program `print`. Then `print`(x) with x literal has length at most $n+c$. By Definition 5 of logical depth there exists a $(2c+k_1)$-incompressible program p with $U(p) = x$ such that by (2) we have

$$K(p) + 2c + k_1 \geq n - c - k_1 + 2c + k_1 = n + c.$$

Let `print`(x) be $(2c+k_1)$-incompressible. Then, the running time $\text{depth}^{BB}_{2c+k_1}(x)$ is at most $BB^{-1}(time(\texttt{print(x)})) = BB^{-1}(n \log n) = O(\log n)$ (since the Busy Beaver function grows faster than any computable function). □ □

We can adapt the argument presented above to prove that if we set

$$A = \{x \in \Sigma^n : |p| < n - g(n) \ \wedge \ U(p) = x \text{ in at most } BB(n) \text{ steps}\},$$

where $g(n)$ is a computable function $c \leq g(n) \leq n - \log n$ then the arguments go through as well. The above case corresponds to $g(n) \equiv c$. If we set $g(n) = \log n$ for instance then we obtain the result below.

COROLLARY 1 Let c, k_1 and k_2 be sufficiently large constants. For every sufficiently large n there is a string x of length n such that we have $\text{depth}^{BB}_{k_2}(x) \geq n$ and $\text{depth}^{BB}_{k_1+2\log n}(x) = O(\log n)$.

PROOF. (Sketch) The proof is equal to the previous one with the following adaptations:

$$A = \{x \in \Sigma^n : |p| < n - \log n \ \wedge U(p) = x \ \text{ in at most } BB(n) \text{ steps.}\}$$

The complement set B has at least $|B| \geq 2^n(1 - 2^{-\log n}) = 2^n - 2^n/n$ elements. With similar reasoning as Lemma 1 we can show the existence of a string in the complement of A satisfying $n - k_1 - \log n \leq K(x) \leq n - k_2 - \log n$. □

COROLLARY 2 Let s and f be large enough constants with $s < f$. (Here s corresponds to k_2 above and f corresponds to $2c + k_1$.) There exists an infinite sequence of strings $x_1, x_2, \ldots$ with $|x_{i+1}| = |x_i| + 1$, such that for all programs p_i^1 with $|p_i^1| \leq K(p_i^1) + s$ with $U(p_i^1) = x_i$ in d_i^1 steps and all programs p_i^2 with $|p_i^2| \leq K(p_i^2) + f$ with $U(p_i^2) = x_i$ in d_i^2 steps we have

$d_i^2 - d_i^1 \geq BB(i-1) - BB(\log^2 i)$ $(i \geq 1)$. Hence the gap in logical depths with a fixed constant difference in significance of strings x_i in this sequence rises at least as $BB(i-1) - BB(\log^2 i)$. This gap rises faster than any computable function since $BB(i-1)/2$ does and $BB(i-1)/2 > BB(i-1) - BB(\log^2 i)$ except possibly for i very small.

COROLLARY 3 For every string x by definition $f_x(0) \geq time(x^*)$ since x^* is a 0-incompressible string. Moreover, the function f_x is monotonic nonincreasing for all strings x. By the Corollary 2 there is a sequence $x_1, x_2, \ldots$ such that the values of f_{x_i} for certain arguments differ at least by $BB(i-1)/2$. Therefore, certainly the values of $f_{x_i}(0) > BB(i-1)$. Hence, there exists an infinite sequence of strings $x_1, x_2, \ldots$ with $|x_{i+1}| = |x_i| + 1$, such that the associated sequence of shortest programs $x_1^*, x_2^*, \ldots$ and programs of length $|x_1^*| + k, |x_2^*| + k, \ldots$ $(0 \leq k \leq k_2)$ have a computation times in excess of $BB(0)/2, BB(1)/2, \ldots$ (except for possibly a small initial segment). Hence the times to compute x_i from x_i^*, and from programs up to $|x_i^*| + k_2$, is in excess of $BB(i-1)/2$ $(i = 1, 2, \ldots)$, a function that is incomputable and rises faster than any computable function for i is not too small.

COROLLARY 4 There is nothing that we used from the Busy Beaver function for the result in Theorem 4 and Corollaries 1 through 3 except for the fact that it is well-defined and details about the print instruction. The same results hold for other functions that are well-defined as the Busy Beaver function and grow still faster. The proof is similar. For example functions $h_1(n) = BB(n)^{BB(n)}$ and

$$h_2(n) = BB(n)^{\overset{\overset{BB(n)}{\vdots}}{\Big\}\, BB(n) \text{ times,}}}$$

an exponential stack of $BB(n)$'s of height $BB(n)$.

4 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 at most the running time of the computation from a shortest program x^* to x. The function decreases to $O(n \log n)$ for the argument $|x| - K(x)$. We show that there is an infinite sequence of strings such that the difference in logical depths for a fixed constant difference in arguments rises faster than any computable function, that is, incomputably fast as the Busy Beaver function. This shows that logical depth can increase tremendously for only a constant difference in arguments. Moreover, there is an infinite sequence of strings such that the minimal computation times of associated shortest programs and even shortest programs plus a certain number of bits rises at

least as fast as the Busy Beaver function, that is, faster than any computable function. This seems a shot at the determining the possibly extreme running times of shortest programs and programs that are a certain number of bits longer.

Acknowledgments

We thanks Bruno Bauwens for helpful discussions and comments, and a referee for noticing that one of the definitions was not correct.

References

[1] L. Antunes and L. Fortnow. Sophistication revisited. *Theory of Computing Systems*, 45(1):150–161, Springer–Verlag, 2009.

[2] 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.

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

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

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

[6] 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.

[7] 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.

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

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