Theory Comput Syst (2017) 60:280–298
DOI 10.1007/s00224-016-9672-6

Sophistication vs Logical Depth

Luís Antunes[1] · Bruno Bauwens[2] · André Souto[3,4] · Andreia Teixeira[5,6]

Published online: 19 March 2016

Abstract Sophistication and logical depth are two measures that express how complicated the structure in a string is. Sophistication is defined as the minimal complexity of a computable function that defines a two-part description for the string that is shortest within some precision; the second can be defined as the minimal computation time of a program that is shortest within some precision. We show that the Busy Beaver function of the sophistication of a string exceeds its logical depth with logarithmically bigger precision, and that logical depth exceeds the Busy Beaver function of sophistication with logarithmically bigger precision. We also show that sophistication is unstable in its precision: constant variations can change its value by a linear term in the length of the string.

✉ Luís Antunes
lfa@fc.up.pt

[1] Faculty of Sciences, University of Porto, CRACS & INESC-Porto LA, R.Campo Alegre, 1021/1055, 4169-007 Porto, Portugal

[2] Faculty of Computer Science, Research University Higher School of Economics, Kochnovskiy Proezd 3, 125319 Moscow, Russia

[3] Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, Lisboa, Portugal

[4] SQIG-Instituto de Telecomunicações, Lisboa, Portugal

[5] Health Information and Decision Sciences Department, Universidade do Porto, Porto, Portugal

[6] CINTESIS, Center for Research in Health Technologies and Information Systems, Porto, Portugal

Keywords Sophistication · Logical depth · Kolmogorov complexity · Algorithmic sufficient statistics · Busy Beaver function

1 Introduction

Solomonoff [1], Kolmogorov [2] and Chaitin [3] independently defined a measure of information contained in a bit string x as the length of a shortest program that produces x on a universal Turing machine. This measure, usually represented by $C(x)$, is called Kolmogorov complexity. Kolmogorov complexity does not express whether the string contains sophisticated structure. For example, consider for some n a randomly generated n-bit string. With high probability the complexity is about n and the string has no (complicated) structure. On the other hand, the $(2^n - 1)$-bit string representing the Halting problem for programs of length less than n has also complexity close to n but has very complicated structure. Informally, "sophistication of structure" can be measured by the minimal computation time of a program modeling the structure or by the minimal size of a program that models the structure.

The first notion is Bennett's *logical depth* [4]. At significance level c, it is defined as the minimal time to compute x by a program p that is c-incompressible on a universal prefix-free Turing machine U (of some type), i.e. $C_U(p) \geq |p| - c$. Bennett [4] showed that this measure is closely related to the minimal time for which some time-bounded version of algorithmic probability converges within a factor 2^{-c}. We will use the following simpler variant (which is closely related with the previous one, see Section 2):

> *The time required to compute x by a program no more than c bits longer than a shortest program.*

Examples of strings that are non-deep according to this definition are the random strings and the efficiently computable ones. In [5], this notion was used to show that if the complexity class **NP** reduces to a sequence for which every initial segment is not deep, up to "polylog" precision in the length of the string, then the polynomial time hierarchy collapses. In particular, it would imply a collapse if **NP** reduces to a sparse or to a random set.

Koppel [6] defined a different notion of depth for infinite sequences based on some variant of monotone Kolmogorov complexity. The class of deep sequences is defined by the ones for which the depth of initial segments is not bounded by a computable function of their length. In particular, the set of such sequences is disjoint from the set of random ones, and hence, they define a set of measure zero. Lutz [7] showed that deep sequences contain useful information in the following computational sense: the class of sequences that truth-table reduces to them has non-zero measure in the class of computable sequences.

Kolmogorov [8, 9] defined for each string the notion of structure function dividing a shortest program for a string in two parts – one part accounting for useful regularities and another accounting for the remaining information presented in the string – in such a way that this two-part description is as small as the shortest one-part

description. He represented the regularities in the string by finite sets. Later, Koppel [6, 10, 11] expressed regularities as monotone computable functions and called the minimal complexity of the function defining a shortest two-part code *sophistication*. Following Koppel's work, Li and Vitányi [12] and independently Antunes and Fortnow [13] revisited the notion of sophistication considering computable functions (that are not necessarily monotone). It was observed that there are strings with near maximum sophistication, and such strings encode the halting problem for smaller programs. Furthermore, in [13] coarse sophistication was introduced, and it was shown that it is roughly equivalent to a variation of logical depth based on the Busy Beaver function. In Section 3, we present a more detailed overview of the literature on these measures of sophistication.

In order to quantify the amount of structure in a string, we consider the length of its representation in terms of some model, for which the string is "typical". Depending on the definition, the model could be a set (set sophistication), or a density function (effective complexity), or a monotone Turing machine (etc.) One definition that does not follow this scheme is the notion of logical depth, which measures depth in terms of time, instead of program length. The main contribution of this paper is to show that by converting time to length using the Busy Beaver function, this notion of depth is closely related to sophistication and that it has the same kinds of properties as the other model-based definitions. From this, we conclude that all sophistication measures defined using Kolmogorov complexity, are equivalent in this sense. A closely related result was previously shown in Theorems 3.1.21 and 3.3.3 of [30]. Although, using a very technical but closely related scaling function based on the convergence time of Chaitin Omega numbers. We also study the stability of sophistication under changes of significance. From Theorem IV.4 in [14], one concludes that a logarithmic change of the significance can change sophistication maximally (i.e. almost $|x|$). We show this also holds for constant changes of the significance.

2 Definitions and Results

For a string x, let $|x|$ be the length of x. For each Turing machine, we associate a partial function U that maps pairs of strings to strings. We fix a reference Turing machine U that is universal in the following sense: for any other machine V, there is a string w_V such that $U(w_V p, y) = V(p, y)$ if $V(p, y)$ is defined. If y is the empty string we write $U(p)$ rather than $U(p, y)$.

The *Kolmogorov complexity* of x is defined as

$$C(x) = \min_p\{|p| : U(p) = x\}.$$

Note that changing the universal machine U affects Kolmogorov complexity by less than an additive constant.

Koppel [6], using monotone functions as a model, defined sophistication for infinite strings. Monotone complexity functions are standard in the literature, for the definition of monotone Turing machines we reefer the reader to the original paper

[6] and also to [30]. Later, Li and Vitányi [15] and Antunes and Fortnow [13] independently simplified Koppel's definition of sophistication for finite strings, using computable functions (that are not necessarily monotone).

Definition 1 (as in [13]) Let c be integer. The *sophistication* of a string x with significance c is:

$$\mathrm{soph}_c(x) = \min_p \left\{ |p| : \begin{array}{c} U(p,d) \text{ is defined for all } d \\ \text{and there is a } d \text{ s.t. } U(p,d) = x \\ \text{and } |p| + |d| \leq C(x) + c \end{array} \right\}.$$

If no such p exists, then $\mathrm{soph}_c(x) = +\infty$.

Clearly, sophistication is non-increasing in c. For $c = |x| + O(1)$, sophistication is bounded by $O(\log |x|)$. It might happen that sophistication is finite for negative c, however one can show that finite sophistication implies $-c \leq O(\log |x|)$.

Bennett [4] defined the c-significant logical depth of an object x as the time required by a prefix-free machine to generate x with a program p that is c-incompressible (i.e. $K(p) \geq |p| - c$, where K stands for the complexity on a universal prefix-free machine). Our results are related to a more intuitive version of logical depth (also discussed in [4]). In Appendix A, we explain why this notion is sufficiently close to Bennett's definition. Let $\mathrm{time}(p)$ be the number of computation steps made by U on input x to reach a halting state.

Definition 2 (Logical depth) For any $c \geq 0$, the *logical depth* of a string x at significance level c is

$$\mathrm{depth}_c(x) = \min \{\mathrm{time}(p) : |p| \leq C(x) + c \text{ and } U(p) = x\}.$$

Note that depth is always finite (for $c \geq 0$). For $c < 0$ let $\mathrm{depth}_c(x) = +\infty$. One can, scale down the running time to program length using the inverse Busy Beaver function

$$bb(n) = \min \{|p| : U(p) \text{ halts and } \mathrm{time}(p) \geq n\}.$$

The Busy Beaver logical depth is simply the inverse Busy Beaver function of the logical depth. By definition, this equals the minimal complexity of an upper bound of the logical depth.

Definition 3 The *Busy Beaver logical depth* of x with significance c is:

$$\begin{aligned} \mathrm{depth}_c^{bb}(x) &= bb(\mathrm{depth}_c(x)) \\ &= \min_{p,q} \left\{ |q| : \begin{array}{l} |p| \leq C(x) + c \text{ and } U(p) = x \\ \text{and } \mathrm{time}(p) \leq \mathrm{time}(q) \end{array} \right\} \end{aligned}$$

From the definition it is easy to see that $\mathrm{depth}_c^{bb}(x) \leq C(x) \leq |x| + O(1)$. Clearly, $\mathrm{depth}_c(x)$ is non-increasing in c. For some machines U we have $\mathrm{depth}_c(x) \geq |x|$ for

all x, for example, if every halting program on U always scans the full input. The following lemma shows that changing two such machines changes the Busy Beaver logical depth to a function that is close.

Lemma 1 *For all universal*[1] *Turing machines U and V, there exist a constant c' such that for all c and x: $depth_{c,U}(x) \geq |x|$ [no Busy Beaver here!] implies*

$$depth^{bb}_{c+c',V}(x) \leq depth^{bb}_{c,U}(x) + c'.$$

We postpone the proof of this lemma to the Appendix A. Let the *upper graph* of a function f be $\{(n, m) : m \geq f(n)\}$. Let the distance between two points (n, m) and (n', m') be $\max(|n - n'|, |m - m'|)$.

Definition 4 Two functions f and g are *c-close* if the upper graphs of these functions are in a c-neighbourhood of each other.

If f and g are non-increasing, this is equivalent to $f(n + c) \leq g(n) + c$ and $g(n + c) \leq f(n) + c$. The previous lemma shows that the depth function of all universal machines U with $\text{depth}_{c,U}(x) \geq |x|$ are $O(1)$-close.

Sophistication and logical depth are conceptually very different since the former measures program lengths while the latter running times. In order to establish a relationship between these measures, we rescale logical depth from running time to program length using the Busy Beaver function. Our main results show that although sophistication can suffer huge drops within $O(\log(c))$ changes in significance (Theorem 2), sophistication and bb-depth cannot diverge too much within a margin of $O(\log(|x|))$ in the significance.

The first main result of the paper states that sophistication and logical depth are $O(\log |x|)$-close.

Theorem 1 *For a fixed x, the functions $depth^{bb}_c(x)$ and $soph_c(x)$ are $O(\log |x|)$-close, i.e., for some e and for all c and x with $|x| \geq e$:*

$$\begin{aligned} depth^{bb}_{c+e\log|x|}(x) &\leq soph_c(x) + e\log|x| \\ soph_{c+e\log|x|}(x) &\leq depth^{bb}_c(x) + e\log|x|. \end{aligned}$$

In Theorem 6 in Section 6 it is shown that the margin in the significance cannot be made constant, and hence, depth and sophistication are not $O(1)$-close. The second main result states that for a fixed string x, the sophistication function is unstable in its significance; more precisely, for some x and c, small changes of c can result in large changes of $\text{soph}_c(x)$.

[1] In fact the proof only requires that U and V are optimal, i.e. for all machines W there exist c_W such that $C_U(x) \leq C_W(x) + c_W$ and similarly for V.

Theorem 2 *For some e and for large c there are infinitely many x such that*[2]

$$soph_c(x) - soph_{c+e\log c}(x) \geq \frac{3}{4}|x|.$$

This theorem shows that for a fixed string "the sophistication of this string" corresponds to a *function* (of c), rather than a single number (in a similar way as Kolmogorov introduced the closely related structure function, see Section 3).

3 Related Definitions of Sophistication

We describe related notions of sophistication and present a few definitions. No definitions or results from this section are needed in later sections. For a recent overview paper, we refer to [31].

The first approach to define some notion of sophistication goes back to Kolmogorov [8, 9] (see [16]) and uses the definition of a typical string in a set.

Definition 5 A string x is *c-typical* in a finite set S containing x iff

$$C(x|S) \geq \log|S| - c.$$

For such S, a literal representation of the lexicographic index of x in S (of length $\log|S|+O(1)$) is almost a shortest description for x given S. By a counting argument, one can show that all but at most a fraction 2^{-c} of elements in a set are c-typical.[3] Kolmogorov asked whether there exist strings that are not typical in any finite set with small Kolmogorov complexity.[4]

In [18–20] a positive answer was shown, i.e., some strings are only typical in sets of complexity close to the length of the string. Kolmogorov called such strings absolutely non-stochastic, because they have high mutual information with the Halting problem. It is believed that such strings can not appear with high probability in a statistical experiment. We define the *non-stochasticity* of a string as the minimal complexity of a set in which the string is c-typical:

Definition 6 $\text{nstoch}_c(x) = \min\{C(S) : x \text{ is } c\text{-typical in } S\}$.

Kolmogorov also considered a more restrictive class of "good" set-models for a string x. To understand this criterion, consider the *structure set*, which is the set of all pairs (i, j) for which there is an x-containing set of complexity at most i and cardinality at most 2^j, see Fig. 1. Ignoring $O(\log|x|)$-terms, the set contains the

[2] For any $\varepsilon > 0$, we can replace the term $\frac{3}{4}|x|$ by $(1-\varepsilon)|x|$ if the significance of the second sophistication term is replaced by $c + O(\log(c/\varepsilon))$.

[3] On the other hand, any set must have non-typical elements unless the set contains a lot of mutual information with the Halting problem [17].

[4] The Kolmogorov complexity of a set is the length of a shortest program that prints all its elements and halts.

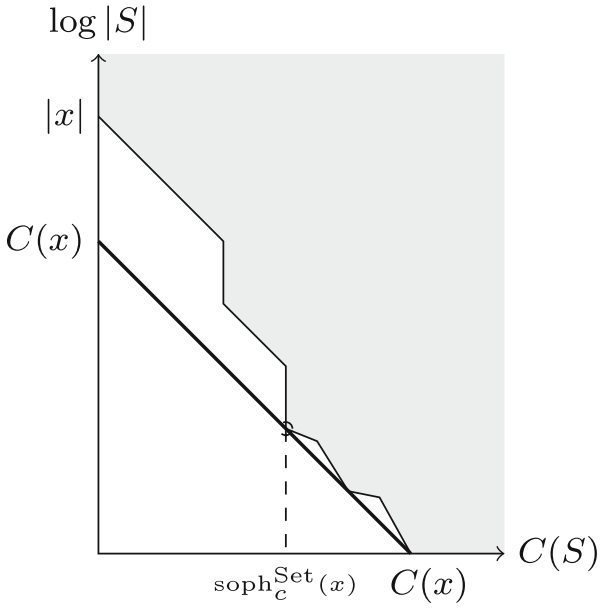

Fig. 1 The structure set of a string x is the set of all pairs (i, j) for which there exists an x-containing set of complexity at most i and cardinality at most 2^j. Such a set is schematically represented above in gray.

points $(C(x), 0)$ and $(0, |x|)$ witnessed by the set $\{x\}$ and the set of all strings of length $|x|$. Note that if the set contains a point (i, j), it also contains the points $(i + k, j - k)$ for all $k \leq j$.[5] Hence, the lower border of the set, called structure function, decreases with at least slope one (still ignoring $O(\log |x|)$ terms). No point appears below the line $i + j = C(x)$, otherwise the corresponding set could be used to construct a program for x of size less then $C(x)$. Cover [21] (see also Section 14.12 of the reference [22] and Section 5.5.1 [12]) mentioned explicitly the left-most place where the set approaches this line, which we call *set sophistication* of x:

Definition 7 $\text{soph}_c^{\text{Set}}(x) = \min\{C(S) : x \in S \wedge C(S) + \log |S| \leq C(x) + c\}$.

By the following theorem and lemma, sophistication, set-sophistication and non-stochasticity for a string x are all $O(\log |x|)$-close.

Theorem 3 ([14]) *For all x, the functions $nstoch_c(x)$ and $soph_c^{Set}(x)$ are $O(\log |x|)$-close.*

Lemma 2 ([23]) *For all x, the functions $soph_c(x)$ and $soph_c^{Set}(x)$ are $O(\log |x|)$-close.*

[5] Partition the set in subsets of size at most 2^{j-k}, this increases the complexity of the x-containing set by at most k.

All these sophistication functions are unstable: increasing the parameter c by $O(\log|x|)$, the function values can drop from maximal value $|x| - O(\log|x|)$ to $O(\log|x|)$.

Corollary 1 (of Theorem IV.4 in [14]) *There exists e such that for all c there exist infinitely many x such that*[6]

$$soph_c^{Set}(x) \geq |x| - e\log|x| \quad \textit{and} \quad soph_{c+e\log|x|}^{Set}(x) \leq e\log|x|.$$

In [20] it was shown that a sufficient set of almost minimal complexity of a string x can be computed from an initial segment of the binary code of the number of halting programs of length $C(x)$. Hence, such a set contains high mutual information with the Halting problem for short programs (see [24]). In [25, 26] it is argued that in some cases this statistic can hardly be interpreted as a denoised version of x. In fact, compared to x, a sufficient two part-code (S, z) (where z is the lexicographic index of x in S) can contain different computational information from x, although $C(x|S, z)$ and $C(S, z|x)$ are both small. The proposed solution was to impose the existence of a computable bijection of small complexity between x and (S, z). This is equivalent to the requirement that there exists a short total program computing S from x. In [26] it was shown that this version of sophistication can be much larger than set-sophistication. In fact, the result shows that strings with large such sophistication can appear with non-negligible probability in some statistical experiments.

Until now, we considered two model types in the definitions of sophistication. In Definition 1, we used computable functions that are *c-sufficient* for x, i.e., functions f for which a string d exists such that $f(d) = x$ and $C(f) + |d| \leq C(x) + c$. In Definition 7, we considered c-sufficient sets for x, i.e., sets S containing x for which $C(S) + \log|S| \leq C(x) + c$. Another popular model type are computable probability density functions P. Such a P is c-sufficient for x if $C(P) + \log(1/P(x)) \leq C(x) + c$ [20].[7] In a similar way, probabilistic sophistication at significance level c is defined as the probability density function of minimal complexity that is c-sufficient. By Lemmas 7.1 and 7.2 of [23] all these variants of sophistication are $O(\log|x|)$-close.

In order to generalize the notion of sophistication for (infinite) sequences, Koppel [6, 10, 11] considered monotone computable functions f as models. The sufficiency criterion for the two-stage code for x is the existence of a string d such that $f(d) = x$ and $Km(f) + |d| \leq H(x) + c$ where $H(x)$ is the minimal length of a two-part description for x on some special monotone machine and $Km(x)$ denotes

[6] Theorem IV.4 in [14] states that every decreasing function f is $(C(f) + O(\log|x|))$-close to the function

$$\lambda_x(k) = \min\{C(S) + \log|S| : S \ni x \wedge C(x) \leq k\}$$

of some string of length $f(0)$. (We use plain complexity in the definition of λ_x, because all results hold up to $O(\log|x|)$ terms). For fixed x, the function λ_x is the inverse of $soph_x^{\mathrm{Set}}$.

[7] This probabilistic sufficiency criterion was defined in [20] in terms of prefix-free complexity, because $2^{-K(x|P)}$ defines a probability distribution and hence, it is natural to compare it with $P(x)$. Prefix complexity and plain complexity differ by at most $O(\log|x|)$ [12], and this precision is sufficient for our discussion.

the monotone Kolmogorov complexity relatively to the machine considered. It is not hard to show that $H(x) = C(x) + O(\log|x|)$ and that this notion of sophistication is $O(\log|x|)$-close to the aforementioned notions.[8] On this model, Koppel defined sophistication and depth for sequences in two variants, and for each variant he showed that sophistication and depth are equal up to constants.

The last variant of sophistication is effective complexity [27, 28]. This notion uses a probability density function P. Inspired by an information-theoretic solution of the problem of Maxwell's Demon, total entropy of P has been defined as $C(P) + H(P)$, where $H(P) = \sum_x P(x)\log_2(1/P(x))$ denotes the Shannon entropy of P.[9] A probability density function P is a *c-good* model for x if $C(P) + H(P) \leq C(x) + c$ and $\log(1/P(x)) \leq H(P) + c$.[10] The c-effective complexity is the minimal complexity of a c-good model. In Lemma 21 of [29], it is shown that effective complexity is $O(\log|x|)$-close to set-sophistication.[11] In Theorem 18 of [29] it was also shown that strings with high effective complexity have high computational depth. Moreover, the proof shows that effective complexity is upper bounded by the Busy Beaver logical depth with slightly bigger significance. Our Theorem 1 implies also the other direction, i.e., that effective complexity is $O(\log|x|)$-close to Busy Beaver logical depth.

4 Sophistication and Busy Beaver Logical Depth are Close

Koppel [6] proved an equivalence between logical depth and sophistication for infinite sequences. For such sequences, and for fixed significance, depth is defined as the minimal complexity of a total function rather than the minimal complexity of an upper bound for a number. In this section we show that sophistication and Busy Beaver logical depth of a string x are $O(\log|x|)$-close functions.

Theorem 4 *The functions $depth^{bb}_c(x)$ and $soph_c(x)$ are $O(\log|x|)$-close, i.e., for some e and for all c and x with $|x| \geq e$:*

$$depth^{bb}_{c+e\log|x|}(x) \leq soph_c(x) + e\log|x|$$
$$soph_{c+e\log|x|}(x) \leq depth^{bb}_c(x) + e\log|x|.$$

[8] It is unclear whether $H(x) = Km(x) + O(1)$.

[9] The definition of total entropy used in [27, 28] is $K(P) + H(P)$. Notice that plain and prefix complexity are close ($|K(P) - C(P)| \leq O(\log C(P))$. See also footnote 7.

[10] In fact, in [27] the precision for which these inequalities should hold is not discussed. Also, the authors suggest that the computation time of a program for P is bounded by some computable function. In [29] the first requirement $c = \delta|x|$ is chosen for some $\delta > 0$ and in the second requirement a different parameter is chosen. Furthermore, P should be computable as a real function and no restrictions on the computation time are considered. Also, $K(P)$ was replaced by $K(P, H(P))$.

[11] Indeed, if P is c-good then it is $(2c)$-sufficient. For the other direction, note that at most $2^{H(P)+c+1}$ elements satisfy $\log(1/P(x)) \leq \lceil H(P)\rceil + c$, and these elements can be computed given P and $\lceil H(x)\rceil \leq C(x) + c \leq |x| + c + O(1)$. Hence a c-good model defines a $(c + O(\log|x|))$-sufficient set.

In the Appendix we provide an alternative and more technical proof of this result involving Chaitin Ω-numbers that might be of interest for people with background in the theory of algorithmic randomness.

Proof To prove the first inequality, we consider $c \leq |x| + O(1)$; otherwise the theorem follows directly. Consider p and d such that

$A1$. the function $U(p, \cdot)$ is total,
$A2$. $U(p, d) = x$,
$A3$. $|p| + |d| \leq C(x) + c$.

For later use, note that by assumption on c we have that $|p|$ and $|d|$ are bounded by $2|x| + O(1)$. We need to construct q and r such that

1. $U(q) = x$ and $|q| \leq |p| + |d| + O(\log |x|)$,
2. $\text{time}(q) \leq \text{time}(r)$,
3. $|r| \leq |p| + c + O(\log |x|)$.

The idea of the construction is to let r be a shortest program for the maximal computation time needed to simulate $U(p, e)$ for some e of length $|d|$. Let us define this quantity more formally.

Construction of q. For a string y, let $\overline{y}$ be a computable prefix-free encoding of length $|y| + 2\log|y|$. (For example $\overline{y} = b_1 0 b_2 0 \ldots b_{\log|y|} 1 y$ where b is $|y|$ in binary.) Let V be a machine such that $V(\overline{y}e) = U(y, e)$ if $U(y, e)$ is defined. Thus $U(w\overline{y}e) = V(\overline{y}e) = U(y, e)$ for some w and all y, e. Let $q = w\overline{p}d$. Thus, $U(w\overline{p}d) = U(p, d) = x$. Recall that $|p| \leq 2|x| + O(1)$, hence, $|q|$ satisfies condition 1:

$$|q| \leq O(1) + (|p| + O(\log|p|)) + |d| \leq |p| + |d| + O(\log|x|)\,.$$

Construction of r. Let

$$t = \max_e \{\text{time}(w\overline{p}e) : |e| = |d|\}.$$

The program r is a shortest program printing a string containing t zeros. Clearly, the running time of this program is at least t and by construction this exceeds $\text{time}(q) \geq \text{time}(w\overline{p}d)$, which verifies condition 2. For condition 3 notice that to compute t, we only need to know p and $|d| \leq 2|x| + O(1)$, hence,

$$|r| \leq |\overline{p}| + O(\log|d|) \leq |p| + O(\log|x|).$$

This concludes the proof of the first inequality.

Now we prove the second inequality. For each k, l such that $l \leq k$ consider a sequence of strings x_i and markers

$$S_{l,k} = x_1, x_2, \ldots, x_i, \square, x_{i+1}, \ldots, x_j, \square, x_{j+1}, \ldots$$

that can be enumerated as follows: dovetail all programs of length l and k, and enumerate the output of the k-bit programs in order of computation time. Each time a program of length l halts, also append a marker to the series (if k-bit programs with the same computation time appear, append the marker last). One easily observes that:

1. the sequence $S_{l,k}$ can be enumerated from k, l,

2. there are at most 2^l markers, and at most 2^k strings,
3. if a program of length k outputs x in at most $BB(l) = \max\{\text{time}(p) : |p| \leq l\}$ steps, then x appears in the sequence $S_{l,k}$ before its last marker.

The second inequality of the theorem follows from the following lemma.

Lemma 3 *Every string that appears before the last marker in a sequence $S_{l,k}$ satisfying properties 1 and 2 above, satisfies*

$$soph_{k-C(x)+O(\log k)}(x) \leq l + O(\log k).$$

We show that this lemma and Property 3 implies the inequality. Assume $c \leq |x| + O(1)$, otherwise the inequality holds for trivial reasons. Let $l = \text{depth}_c^{bb}(x)$. There is a program p for x with $|p| \leq C(x) + c$ that runs in at most $BB(l)$ steps (and in more than $BB(l-1)$ steps). Let $k = |p|$, thus $l \leq k$ (if $|p| < l$, the running time of p would be at most $BB(l-1)$). Enumerate a sequence $S_{l,k}$ as described above with parameters l and k. Notice that x appears in $S_{l,k}$ before the last marker. By the claim, we have

$$\text{soph}_{c+O(\log k)}(x) \leq l + O(\log k).$$

The inequality follows from this and $k \leq C(x) + c \leq 2|x| + O(1)$. □

To complete the proof of Theorem 1 we prove Lemma 3.

Proof of Lemma 3 For any computable function f, let $C(f)$ denote the minimal length of a program that computes f. For any x as in the Lemma, we need to show that there is a computable function f such that:

1. $C(f) \leq l + O(\log k)$
2. $C(f) + |d| \leq k + O(\log k)$ for some $d \in f^{-1}(x)$.

Consider a segment of strings $x_{i+1}, \ldots, x_j$ in the sequence, separated by two markers □ that contains x. We associate a function f to this segment that maps the lexicographic first $j - i$ strings to $x_{i+1}, \ldots, x_j$ and all other strings to the empty one. Notice that f is computable, and can be computed from k, l and the number of markers that precede the defining segment (which is at most 2^l). This implies $C(f) \leq l + O(\log kl) = l + O(\log k)$, i.e., condition 1.

It remains to show condition 2. Let $\delta = \log(j - i)$, i.e. the logarithm of the size of the segment. Observe that at most $2^{k-\delta}$ segments in the sequence have size at least 2^δ (by assumption 2). Hence, $C(f) \leq k - \delta + O(\log kl\delta)$. Since the segment contains x, there is a d such that $f(d) = x$, and by construction $|d| \leq \delta$. Hence $C(f) + |d| \leq (k - \delta) + \delta + O(\log(kl\delta))$, i.e. condition 2. □

5 Sophistication is Unstable

In [13] the authors conjectured that Koppel's definition of sophistication might not be stable, in the sense that small changes in the significance c level could drastically change the value of $\text{soph}_c(x)$. To avoid this problem, they proposed a different

sophistication measure where they incorporated the significance level as a penalty in the formula obtaining a robust measure, called coarse sophistication. However, one can argue that this measure is not robust in the sense that drastic changes can happen for slight changes of the weight of the penalty function [25].

In Section 3, we used Theorem IV.4 of [14] to show that (most variants of) sophistication are unstable if the significance is increased by $O(\log |x|)$. With the same proof technique, one can show that also sophistication when defined with prefix complexity is unstable with constant changes of the significance.

One might ask whether sophistication functions defined with plain complexity are also unstable with constant changes in the precision? We provide a positive answer to this question.

Theorem 5 *For some e and for large c there are infinitely many x such that*

$$soph_c(x) - soph_{c+e\log c}(x) \geq \frac{3}{4}|x|.$$

The proof also uses a technique inspired by the proof of Theorem IV.4 in [14]. However, some technical difficulties appear because we are using plain machines. Let us explain the problem. In the definition of sophistication of x, we consider pairs of strings (p, d) such that $U(p, d) = x$. For some k there are 2^k strings of length k, but there are $(k + 1)2^k$ pairs (p, d) with $|p| + |d| = k$. In [14] self-delimiting programs are used and the combinatorial part of the argument uses that the amount of two-part codes of length k is at most 2^k. In this paper we do not use self-delimiting machines, and therefore the combinatorial argument needs a bit more care.

Lemma 4 *For some c' and for all k and x such that $k + \log k \leq |x|$ we have*

$$soph_{|x|-C(x)-\log k+c'}(x) \leq k + c'.$$

Recall that sophistication is defined for negative significance. This lemma even proves that sophistication can be negative for all random strings, i.e., strings x for which $C(x) \geq |x|$.

Proof Let $n = |x|$. In order to prove the lemma it is sufficient to show that there is a two-part description (p, d) for x satisfying $|p| + |d| \leq n - \log k + O(1)$ and $|p| \leq k + O(1)$. The idea to prove it is to use the length of $|p|$ to encode the last $\log k - 1$ bits of x.

Let i be the index of the last $\log k - 1$ bits of x in the lexicographic order of strings; (i.e., $x_{n-\log k+2} \dots x_n$ is the i-th string in the sequence $\varepsilon, 0, 1, 00, 01, \dots$). Notice that $i < k$.

Let p be the program that on input d first prints $x_1 \dots x_i$, subsequently prints d, and finally prints $x_{n-\log k+2} \dots x_n$. Clearly, the above description defines a total function. Moreover, only the information in $x_1 \dots x_i$ is needed to evaluate this function, since the last part of the output can be computed from i. Hence, we can construct p such that $|p| = i + O(1) \leq k + O(1)$.

Furthermore, for $d = x_{i+1} \dots x_{n-\log k+1}$ we have $U(p, d) = x$ and $|p| + |d| \leq (i + O(1)) + (n - i - \log k) \leq n - \log k + O(1)$. □

Proof of Theorem 5. It is sufficient to show that for all k, c there is a string x of length $k + \log k + 2$ such that $\text{soph}_{c-O(\log c)}(x) \geq k$ and $\text{soph}_{c+O(1)}(x) \leq k/8 + O(1)$.

Our construction of x implies that $C(x) \geq k - c$. Hence, applying Lemma 4 with $k \leftarrow k/8$ implies $\text{soph}_{c+O(1)}(x) \leq k/8 + O(1)$; indeed, the significance is

$$|x| - C(x) - \log(k/8) + c' \leq (k + \log k + 2) - (k - c) - \log k + 3 + c' = c + O(1).$$

The inequality $\text{soph}_{c-O(\log c)}(x) \geq k - c$ follows by the requirements that $C(x) \leq k - c + O(\log c)$ and that there exist no pairs (p, d) such that

1. $U(p, d) = x$ and $|p| + |d| < k$,
2. $|p| < k - c$ and $U(p, y)$ is defined for all y such that $|p| + |y| < k$.

Let us summerize the properties needed in the construction of x (of length $k + \log k + 2$). The complexity should be

$$k - c \leq C(x) \leq k - c + O(\log c),$$

and there should not exists pairs (p, d) satisfying conditions 1 and 2 above.

Construction of x. We keep a list of all strings of length $k + \log k + 2$. At each stage we mark some strings and the lexicographic first string without a mark is the current candidate for x. At each stage, marks are given as follows: we dovetail all programs p, and if a program of length less than $k - c$ halts with an output in the list, then that output is marked. Clearly, there are less than 2^{k-c} strings that are marked in this way. Secondly, if a program p is found satisfying condition 2, i.e., for which the computations $U(p, y)$ terminate for all y such that $|y| + |p| < k$, then all strings $U(p, y)$ in the list are simultaneously marked. These marks appear in less than 2^{k-c} different stages, and the number of such marks is less than $\sum_{i=0}^{k} 2^i 2^{k-i} < (k+1)2^k$. Hence, the total number of marked strings is less than $(k + 1)2^{k+1} \leq 2^n$ which means there is always a candidate for x and at some stage the new candidate remains permanent. By construction, $C(x) \geq k - c$ and there is no pair (p, d) for which both conditions 1 and 2 are satisfied.

Now we have to prove that $C(x) \leq k - c + O(\log c)$. x can be computed from k, c and the total number N of replacements of the candidate for x. Since there are less than $2^{k-c} + 2^{k-c}$ stages where new marks are given, we have $N < 2^{k-c+1}$ times and hence $C(x|k, c) \leq k - c + O(1)$. In fact, if N is represented in binary with $k - c$ bits, we can compute k from c and the length of this representation. Thus $C(x|c) \leq k - c + O(1)$ and hence $C(x) \leq k - c + O(\log c)$. □

6 Sophistication and Busy Beaver Logical Depth are not $O(1)$-Close

In this section we investigate whether there exists an $O(1)$-close relation between sophistication and logical depth. More precisely, for every c can we find an e such

that

$$\mathrm{soph}_{c+e}(x) \leq \mathrm{depth}^{bb}_{c}(x) + e \quad \text{and} \quad \mathrm{depth}^{bb}_{c+e}(x) \leq \mathrm{soph}_{c}(x) + e?$$

The following theorem provides a negative answer:

Theorem 6 *For all large l there exist infinitely many strings x such that $depth^{bb}_{l}(x) \geq |x| - O(l)$ and $soph_0(x) \leq O(l^2 2^l)$.*

We explain informally why an equivalence with $O(1)$ precision fails. In the definition of sophistication of x, we consider pairs of strings (p, d) such that $U(p, d) = x$. As noted before, for all k there are 2^k strings of length k, but there are $(k+1)2^k$ pairs (p, d) with $|p| + |d| = k$. This suggests that strings might exist that have a two-part code (p, d) for which $|p| + |d|$ is smaller than $C(x)$. In other words, this suggests that sophistication can be finite even for negative significance. For an explicit example, choose a string x for which $C(x) \geq |x|$ and apply Lemma 4 in Section 5. For all random and almost random strings, sophistication with negative significance can still be small. There exist x that are only compressible by a small amount and for which the logical depth is high for small significance. Such x are almost random, and hence can have small sophistication even with negative significance.

We now prove Theorem 6 by combining Lemma 4 in Section 5 with the following lemma.

Lemma 5 *For some c, for all d and for all $n > d$ there is x of length n such that*

$$C(x) \geq n - d\,,$$

$$depth^{bb}_{d-2\log d-c}(x) \geq n - d\,.$$

Proof We prove the existence of such strings for $n > d$, since for the other case is trivial.

Let x be the lexicographically first string of length n which is incompressible in time $BB(n - d)$, i.e. there is no program strictly shorter than n computes x in $BB(n - d)$ steps.

To show the inequalities in the statement of the lemma, it is sufficient to show that

$$n - d < C(x) \leq n - d + 2\log d + O(1).$$

For the right inequality, notice that we can compute x from $BB(n - d)$ and n. Furthermore, with $O(1)$ bits of information, n can be computed from d and the length of a witnessing program for $BB(n - d)$ (notice that a program witnessing $BB(n - d)$ has length $n - d + O(1)$). Hence x has a program of length $n - d + 2\log d + O(1)$.

For the left inequality, notice that by the right inequality we have $C(x) < n$ for large d. By choice of x, any program producing x of length at most $n - 1$ must do it in time longer than $BB(n - d)$, and by definition of $BB(n - d)$ this program must be strictly longer than $n - d$. □

Proof of Theorem 1 Let c' be the constant from Lemma 4. For any large k we apply Lemma 5 with $d = \log k - c'$ to obtain a string x of complexity $C(x) \geq |x| - \log k + c'$. Apply this bound to Lemma 4; the significance of the sophistication is at most $|x| - (|x| - \log k + c') - \log k + c' = 0$ and we conclude that $\text{soph}_0(x) \leq k + c' \leq O(2^d)$.

At the same time x satisfies $\text{depth}_{d-2\log d-c}(x) \geq |x| - d$. Hence setting $l = d - 2\log d - c$ the equations of the Theorem 1 are satisfied. Since k can be any large number, also d and l can be any large number, completing the proof. □

If sophistication can be finite for negative significance, it would be fair to compare $\text{depth}_{O(1)}(x)$ to $\text{soph}^{bb}_{\ell}(x)$ where ℓ equals the minimal value of the significance for which sophistication is finite. This value is $-\log C(x) + O(1)$ for every x. The following lemma implies that even with such a correction we can not have a correspondence with sublogarithmic terms in the significance.

Lemma 6 *There exists an e such that for all x and $c \geq 0$: $\text{soph}_{-2c-e}(x) \geq 2^c$.*

If f is a sublogarithmic function, this lemma implies that $\text{soph}_{-\log|x|+f(|x|)}(x)$ is at least proportional to $\sqrt{|x|}$ for large x. (And is finite for x such that $C(x) \geq |x|$.) On the other hand, $\text{depth}_{O(1)}(x) \leq bb(|x|+O(1))$ for all random x. Hence, this approach does also not provide a close correspondence between depth and sophistication.

Proof of Lemma 6 Let e be a large enough constant that will be determined later. Suppose that $\text{soph}_{-2c-e}(x) < 2^c$ for some x and $c \geq 0$. Let p and d be such that $U(p, d) = x$ with $|p| < 2^c$ and

$$|pd| \leq C(x) - 2c - e.$$

Let $\overline{p}$ be a self-delimiting encoding of p of length at most $2\log|p| + |p| \leq 2c + |p|$. This code can be concatenated to d to get a program for x and this implies that $C(x) < 2c + |p| + |d| + e$ for some large enough e. By assumption on $|pd|$ this implies $C(x) < C(x)$, a contradiction. □

To study the relationship between depth and sophistication with more precision, one can avoid the pathology of two part codes by using self-delimiting programs for the total functions. Such programs can be concatenated with an argument without blank between both strings. This implies that one also needs to use self-delimiting programs for x, or otherwise again pathological examples can be constructed. More formally, one uses prefix-free Turing machines, which are machines for which the set of halting programs is a prefix-free set. There exists a universal such machine and we denote Kolmogorov complexity, sophistication and Busy Beaver logical depth of x on such a machine as $K(x)$, $\text{soph}^K_c(x)$ and $\text{depth}^K_c(x)$. It was shown in Theorem 3.2.2 in [30] that with these definitions sophistication and logical depth are still not $O(1)$-close.[12]

[12] The formulation of Theorem 3.2.2 in [30] uses $I(x; H) = K(x) - K^H(x)$ with $K^H(x)$ the Kolmogorov complexity on a machine that has an oracle for the Halting problem. To obtain Theorem 7 from this, use the folklore result: $\text{depth}_0(x) \geq I(x; H) + O(\log I(x; H))$.

Theorem 7 ([30]) *For all c and e there exist infinitely many x such that*

$$soph_c^K(x) \geq \left(depth_0^K(x)\right)^e.$$

For all c there exist $\varepsilon > 0$ and infinitely many x such that

$$soph_c^K(x) \geq \varepsilon|x| + depth_0^K(x).$$

Acknowledgments The authors are grateful to the anonymous reviewers and to to Alexander Shen and Nikolay Vereshchagin for useful comments and discussions. This work was partially supported by the national science foundation: Fundação para a Ciência e Tecnologia, through the scholarships SFRH/BPD/76231/2011, SFRH/BPD/75129/2010 and SFRH/BD/33234/2007, and grants of Instituto de Telecomunicações. The first author is partially funded by the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT as part of project UID/EEA/50014/2013. The second author was also supported by NAFIT ANR-08-EMER-008-01 project.

Appendix A: Machine Invariance of Logical Depth

Lemma 7 *For all universal Turing machines U and V, there exist a constant c' such that for all c and x: $depth_{c,U}(x) \geq |x|$ [no Busy Beaver here!] implies*

$$depth^{bb}_{c+c',V}(x) \leq depth^{bb}_{c,U}(x) + c'.$$

Note that for some universal machines there exist a string w such that $U(wx) = x$ for all x and the computation requires at most $O(1)$ steps. For such machines U we have $\text{depth}_{|wx|}(x) \leq O(1)$ and hence $\text{depth}^{bb}_{|wx|}(x) \leq O(1)$. Other universal machines always scan the input, and on such machines we have $\text{depth}^{bb}_{|wx|}(x) \geq bb(|x|) - O(1)$ for all x. Hence, the assumption in the lemma is necessary.

Proof Let w_V be the prefix such that $V(w_V p)$ simulates $U(p)$ for all p. Our result would follow easily if we assume that for any halting programs p, q on U such that $\text{time}(p) \leq \text{time}(q)$ we have $\text{time}(w_V p) \leq \text{time}(w_V q)$ on V; i.e. simulating U on V preserves the order of computation time. Indeed, any pair (p, q) usable in the definition of depth on U defines a pair $(w_V p, w_V q)$ that can be used in the definition of depth on V. The program $w_V p$ is minimal on V within $c + |w_V| + |w_U|$ error (where w_U is the string that allows to simulate U on V). Hence, the pair $(w_V p, w_V q)$ witnesses an increase of sophistication by at most $|w_V|$ for an increase of the significance of at most $c + |w_V| + |w_U|$.

In the case where the assumption is not true, we need to find c' and a program of length at most $|q| + c'$ on V that computes longer than $\text{time}(w_V p)$ (where c' does not depend on p, q, c). Consider the following algorithm on input q: determine all programs p that have running time at most $\text{time}(q)$ on U, determine for all these p's the maximal running time T of a program $w_V p$ on V (assume for now that for finite $\text{time}(q)$ there are finitely many such p), and finally print a string of length T. For (p, q) usable in the definition of $\text{depth}_{c,U}(x)$, the algorithm with input q produces an output longer than $\text{time}(w_V p)$, and by universality there is a program of length $|q| + c'$ on V that prints this string and hence computes longer than T.

Above, we have assumed that only finitely many programs on U have a halting time at most time(q) for halting q. This assumption is not true in general, but by the additional assumption of the lemma: $\text{depth}_{c,U}(x) \geq |x|$, it suffices to consider only a finite subset of candidates: we only need the pairs (p, q) on U such that $|p| \leq |x| + O(1)$ and $|x| \leq \text{time}(q)$, which implies $|p| \leq \text{time}(q) + O(1)$. The proof finishes by modifying the above algorithm such that it only considers programs p for which $|p| \leq \text{time}(q) + O(1)$. □

Recall that Bennett's definition of logical depth is the minimal computation time of a program on a prefix-free machine W (of some type) that is c-incompressible. We show that when scaled by the inverse Busy Beaver function, both notions of depth are $O(\log |x|)$-close. On a prefix-free machine W, both (unscaled) depths are closely related: Bennett's logical depth of x at significance c is at most $\text{depth}_{c+O(1),W}(x)$, because any c-shortest program p for x is $c + O(1)$-incompressible on W. On the other hand, by Lemma 5.3 of [7] (attributed to Bennett [4]), $\text{depth}_{c+O(1)}(x)$ is bounded by a computable function of Bennett's logical depth of x with significance c. Hence, after rescaling with the inverse Busy Beaver function, both notions are $O(1)$-close. Exchanging prefix-free machine by a plain machine, both depth notions are $O(\log |x|)$-close; indeed this follows by the same argument as Lemma 7 for $W = V$ and replacing $|w_V|$ by $O(\log |x|)$-terms in the proof (since $|K_W(x) - C_U(x)| \leq O(\log |x|)$).

Appendix B: Alternative Proof of Theorem 1

An alternative proof for the second inequality in Theorem 1 is given: there exists e such that for all c and x with $|x| \geq e$ we have

$$\text{soph}_{c+e\log|x|}(x) \leq \text{depth}^{bb}_c(x) + e\log|x|\,.$$

A *prefix stable* machine V is a plain machine such that for all strings p and extensions q of p: if $p \in \text{Dom}V$ then $q \in \text{Dom}V$ and $V(p) = V(q)$. For (infinite) sequences ω let $V(\omega)$ be $V(p)$ if a prefix p of ω exists such that $V(p)$ is defined, and undefined otherwise. For any string or sequence ω, let $0.\omega$ be the real $\sum_i \omega_i 2^{-i}$. A prefix stable machine is *left computable* [17] if for p such that $V(p)$ is defined and for all q such that $0.q \leq 0.p$, also $V(q)$ is defined. There are universal prefix stable machines that are left computable (just rearrange the programs on a universal machine). Let $\Omega = \sup\{0.p : V(p)\text{is defined}\}$.

In order to prove the result aforementioned, it is sufficient to show that $\text{soph}_{c+2\log|x|,U}(x) \leq \text{depth}^{bb}_{c,W}(x) + 2\log|x|$ for large x, where W is a universal left computable machine. Indeed, there exists a universal plain machine U such that

$$\text{depth}^{bb}_{c+2\log|x|,W}(x) \leq \text{depth}^{bb}_{c,U}(x) + O(\log|x|).$$

(translating plain programs to self-delimiting ones can happen by affecting program sizes by at most $O(\log|p|)$ and computation time by a computable function of $|p|$ and the halting time).

Let p be a program satisfying the conditions in the definition of $\text{depth}_c^{bb}(x)$. We show that the initial segment where p and Ω are equal defines a computable function that satisfies the conditions in the definition of sophistication. More precisely, let i be the length of the common initial segment, then $F(d) = V(\Omega_1 \dots \Omega_i 0d)$ satisfies the conditions. Note that, $p_1 \dots p_i = \Omega_1 \dots \Omega_{i-1} 0$ and $\Omega_i = 1$ by construction of i. Thus $F(p_{i+2} \dots p_{|p|}) = x$. For any d we have $0.\Omega_1 \dots \Omega_i 0d < \Omega$ and by left computability this is in $\text{Dom}V$, thus F is computable. It remains to show that $C(F) \leq \text{depth}_{c,W}(x) + O(\log \text{depth}_{c,W}(x))$. We show that $C(\Omega_1 \dots \Omega_{i-1}) \leq \text{depth}_{c,W}(x) + O(\log i)$. In fact, given i and a t that exceeds the computation time of p, we can search for the maximal value $0.w$ for a program w that halts in t computation steps. We know that $0.p \leq 0.w \leq \Omega$, hence we can compute the first $i-1$ bits of Ω which completes the proof.

References

1. Solomonoff, R.: A formal theory of inductive inference. Part I. Inf. Control. **7**(1), 1–22 (1964)
2. Kolmogorov, A.: Three approaches to the quantitative definition of information. Problemy Peredachi Informatsii **1**(1), 3–11 (1965)
3. Chaitin, G.: On the length of programs for computing finite binary sequences. J. ACM **13**(4), 547–569 (1966). ACM Press
4. Bennett, C.: Logical Depth and Physical Complexity, pp. 227–257. Oxford University Press, Inc., New York (1988)
5. Antunes, L., Fortnow, L., van Melkebeek, D., Vinodchandran, N.V.: Computational depth: concept and applications. Theor. Comput. Sci. **354**(3), 391–404 (2006)
6. Koppel, M.: Complexity, depth, and sophistication. Complex Systems **1**, 1087–1091 (1987)
7. Juedes, D., Lathrop, J., Lutz, J.: Computational depth and reducibility. Theor. Comput. Sci. **132**, 37–70 (1994)
8. Kolmogorov, A.N.: Talk in Information Theory Symposium. In: Tallinn, Estonia (1973)
9. Kolmogorov, A.N.: Complexity of algorithms and objective definition of randomness. Uspekhi Mat. Nauk **29**(4), 155 (1974)
10. Koppel, M.: Structure. In: Herken, R. (ed.) The Universal Turing Machine: a Half-Century Survey, 2nd edition, pp. 403–419. Springer (1995)
11. Koppel, M., Atlan, H.: An almost machine-independent theory of program-length complexity, sophistication, and induction. Inf. Sci. **56**(1-3), 23–33 (1991). Elsevier Science Publishers Ltd.
12. Li, M.ing., Vitányi, P.: An Introduction to Kolmogorov Complexity and its Applications. Springer (2008)
13. Antunes, L., Fortnow, L.: Sophistication revisited. Theory of Computing Systems **45**(1), 150–161 (2009). Springer
14. Vereshchagin, N., Vitanyi, P.: Kolmogorov's structure functions and model selection. IEEE Trans. Inf. Theory **50**, 3265–3290 (2004). Computer Society
15. Li, M.ing., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications. Springer (1997)
16. Cover, T., Gacs, P., Gray, R.M.: Kolmogorov's contributions to information theory and algorithmic complexity. Ann. Probab. **17**(3), 840–865 (1989)
17. Epstein, S., Levin, L.: On sets of high complexity strings. CoRR, arXiv:abs/1107.1458 (2011)
18. Shen, A.: The concept of (alpha, beta)-stochasticity in the kolmogorov sense and its properties. Soviet Mathematics Doklady **28**(1), 295–299 (1983)
19. Vyugin, V.: On the defect of randomness of a finite object with respect to measures with given complexity bounds. Theory Prob. Appl. **32**(3), 508–512 (1987)
20. Gács, P., Tromp, J., Vitányi, P.: Algorithmic statistics. IEEE Trans. Inform. Theory **47**(6), 2443–2463 (2001)

21. Cover, T.: The Impact of Processing Techniques on Communications. chapter Kolmogorov Complexity, Data Compression and Inference., pages 23–33. J. Skwyrzynski. Martinus Nijhoff Publishers (1985)
22. Cover, T., Joy, T.: Elements of Information Theory. Wiley (1991)
23. Vitányi, P.: Meaningful information. IEEE Trans. Inf. Theory **52**(10), 4617–4626 (2006)
24. Shen, A.: Algorithmic statistics: main results. In preparation (2013)
25. Vereshchagin, N.: Algorithmic minimal sufficient statistic revisited. Mathematical Theory and Computational Practice, 478–487 (2009)
26. Verschagin, N.: On Algorithmic Strong Sufficient Statistics. In: Proceedings of Computability in Europe (2013)
27. Gell-Mann, M., Lloyd, S.: Information measures, effective complexity, and total information. Complexity **2**(1), 44–52 (1998)
28. Gell-Mann, M., Lloyd, S.: Effective complexity. Nonextensive entropy, 387–398 (2004)
29. Nihat, A., Muller, M., Szkola, A.: Effective complexity and its relation to logical depth. IEEE Trans. Inf. Theory **56**(9), 4593–4607 (2010)
30. Bauwens, B.: Computability in statistical hypotheses testing, and characterizations of independence and directed influences in time series using Kolmogorov complexity. PhD thesis, Ugent (2010)
31. Vereshchagin, N., Shen, A.: Algorithmic statistics revisited (2015). arXiv preprint arXiv:1504.04950