

Low-Depth Witnesses are Easy to Find*

Luis Antunes
U. Porto

Lance Fortnow
Northwestern U.

Alexandre Pinto
U. Porto

André Souto
U. Porto[†]

March 17, 2009

Abstract

Kolmogorov Complexity measures the amount of information in a string by the size of the smallest program that generates that string. Antunes, Fortnow, van Melkebeek and Vinodchandran characterized the notion of useful information by computational depth, the difference between the polynomial-time-bounded Kolmogorov complexity and traditional Kolmogorov complexity.

We show unconditionally how to probabilistically find satisfying assignments for formulas that have at least one low-depth assignment. The converse holds under a standard hardness assumption though fails if $BPP = FewP = EXP$. We also prove that assuming the existence of good pseudorandom generators one cannot increase the depth of a string efficiently.

1 Introduction

Kolmogorov [Kol65], Solomonoff [Sol64] and Chaitin [Cha66] independently defined the complexity of a string x as the length of the shortest program that produces x . If one chooses a string at random this string will have near maximum Kolmogorov complexity even though one can easily create other, just as useful, random string using fresh random coins.

To capture useful information, Bennett [Ben88] formally defined the *s-significant logical depth* of an object x as the time required by a standard

*This paper previously appeared...

[†]The authors from U. Porto are partially supported by funds granted to LIACC through the Programa de Financiamento Plurianual, FCT and Programa POSI.

universal Turing machine to generate x by a program that is no more than s bits longer than the shortest descriptions of x . Later, Antunes, Fortnow, van Melkebeek and Vinodchandran [AFMV06] defined a simpler notion by taking the difference between polynomial-time Kolmogorov complexity and traditional unbounded Kolmogorov complexity. Intuitively, computational depth measures the amount of non-random or useful information in a string. We have seen a number of results about computational depth such as giving a generalization of sparse and random sets [AFMV06] as well as using depth to characterize the worst-case running time of problems that run quickly on average over all polynomial-time samplable distributions [AF05].

In this paper we continue the study of computational depth. Suppose we have a Boolean formula that has a satisfying assignment of computational depth d . We show how to probabilistically find an assignment in time exponential in d . We show that under a standard hardness assumption the converse also holds. Under the unlikely, but open case, that $\text{BPP} = \text{FewP} = \text{EXP}$, one can find formulas that have a single solution of high computational depth that can be found quickly probabilistically.

We also look at the question of whether one can increase the depth of a string efficiently. We show that under a standard hardness assumption one cannot significantly increase the depth of a string in polynomial time. Once again if $\text{BPP} = \text{EXP}$ we show examples where one can produce a string of high depth from a string of very low depth. Finally, we explore the question as to whether a triangle inequality holds for conditional depth.

The rest of this paper is organized as follows. In the next section, we present notation and definitions used. We also state some related results from the literature. In Section 3, we study the relation between computational depth of a solution for a Boolean formula and the existence of a probabilistic algorithm to solve that formula. In Section 4, we show that the computational depth cannot increase rapidly and finally in Section 5, we study some properties of computational depth.

2 Preliminaries

All the strings used are elements of $\Sigma^* = \{0, 1\}^*$. The function \log denotes \log_2 and $|\cdot|$ denotes the length of a string or the cardinality of a set, depending on the context. The letter ε always means the empty string.

The complexity class BPP consists of those problems computable in probabilistic polynomial time with two-sided error bounded away from one-half. The class EXP is the set of problems computable in time $2^{p(n)}$ for

some polynomial $p(n)$. The class FewP are those problems accepted by a polynomial-time nondeterministic Turing machine that has at most a fixed polynomial number of accepting paths for each input.

2.1 Kolmogorov Complexity

For a full understanding of Kolmogorov complexity we refer the reader to the book of Li and Vitányi [LV08]. Here, we present just the definitions and results needed. Since our results are not affected by $O(\log n)$ factors the actual model of Kolmogorov complexity is not important. For definition purposes we will use the self-delimiting Kolmogorov complexity. A set of strings A is prefix-free if there are no strings x and y in A where x is a proper prefix of y .

Definition 2.1. *A function $t : \mathbf{N} \rightarrow [0, \infty)$ is time-constructible if there exists a Turing machine that runs in time exactly $t(n)$ on every input of size n .*

All explicit resource bounds we use in this paper are time-constructible.

Definition 2.2. *Let U be a fixed Turing machine with a prefix-free domain. For any strings $x, y \in \{0, 1\}^*$, the Kolmogorov complexity of x given y is $K(x|y) = \min_p \{|p| : U(p, y) = x\}$. For any time constructible t , the t -time-bounded Kolmogorov complexity of x given y is $K^t(x|y) = \min_p \{|p| : U(p, y) = x \text{ in at most } t(|x|) \text{ steps}\}$.*

The default value for y is the empty string ε and we typically drop this argument in the notation. We can fix a universal machine U whose program size $|p|$ is at most a constant additive factor worse, and the running time t at most a logarithmic multiplicative factor.

Definition 2.3. *A string x is incompressible if $K(x) \geq |x|$. We also call such x algorithmically random.*

2.2 Computational Depth

The Kolmogorov complexity of a string x does not take into account the time necessary to produce the string from a description of length $K(x)$. Levin [Lev73], introduced a useful variant of Kolmogorov complexity weighing program size and running time.

Definition 2.4. *For any strings x, y , the Levin complexity of x given y is
$$Kt(x|y) = \min_p \{|p| + \log t : U(p, y) \text{ halts in at most } t \text{ steps and outputs } x\}.$$*

After some attempts, Bennett [Ben88] formally defined the s -significant logical depth of a string x as the time required by a standard universal Turing machine to generate x by a program that is no more than s bits longer than the shortest descriptions of x . A string x is called logically deep if it takes a relatively large amount of time to generate it from any short description.

Definition 2.5. *Let x be a string and s be a nonnegative integer. The logical depth of x at a significance level s is*

$$\text{depth}_s(x) = \min_p \{t : U(p) \text{ halts and outputs } x \text{ in at most } t \text{ steps, and } |p| < K(x) + s\}$$

Note that algorithmically random strings are shallow at any significance level. In particular, Chaitin's Ω is shallow. Deep strings are hard to find, however they can be constructed by diagonalization, see [Ben88].

Antunes, Fortnow, van Melkebeek and Vinodchandran [AFMV06] propose a notion of *Computational Depth* as a measure of nonrandom information in a string. Intuitively strings of high depth are low Kolmogorov complexity strings (and hence nonrandom), but a resource bounded machine cannot identify this fact. Indeed, Bennett's logical depth [Ben88] can be viewed as such a measure, but its definition is rather technical. Antunes *et al* [AFMV06] suggest that the difference between two Kolmogorov complexity measures captures the intuitive notion of nonrandom information. Based on this intuition and with simplicity in mind, in this work we use the following depth measure.

Definition 2.6. *Let t be a constructible time bound. For any string $x \in \{0, 1\}^*$,*

$$\text{depth}_t(x) = K^t(x) - K(x)$$

and more generally

$$\text{depth}_t(x|y) = K^t(x|y) - K(x|y).$$

2.3 Pseudorandom generators

Pseudorandom generators are efficiently computable functions which stretch a seed into a long string so that for a random input the output looks random for a resource-bounded machine. We need some pseudorandom generators based on hard functions.

The following lemma is implicit in the work of Nisan and Wigderson [NW94].

Lemma 2.7 (Nisan-Wigderson). *Consider a set \mathcal{H} of functions from Σ^ℓ to Σ^n with n bounded by a polynomial in ℓ with the following properties.*

1. *For every ℓ , at least $3/4$ of all possible functions mapping Σ^ℓ to Σ^n are in \mathcal{H} .*
2. *For some k , there is a Σ_k^p function with oracle access to a function H which on input 1^ℓ will accept exactly when H is in \mathcal{H} .*

There is a polynomial-time computable function $H'(x, r)$ with $x \in \Sigma^\ell$ and $|r|$ polynomial in ℓ such that for at least $2/3$ of the possible r , $\hat{H}_r(x) = H'(x, r)$ is in \mathcal{H} .

Impagliazzo and Wigderson [IW97] strengthen the work of Nisan and Wigderson to show how to achieve full derandomization based on strong hardness assumptions. Klivans and van Melkebeek [KvM02] generalize Impagliazzo-Wigderson by showing the results hold for relativized worlds in a strong way.

Lemma 2.8 (Impagliazzo-Wigderson, Klivans-van Melkebeek). *For any oracle A , suppose that there are languages in $DTIME(2^{O(n)})$ that for some $\epsilon > 0$, cannot be computed by circuits of size $2^{\epsilon n}$ with access to an oracle for A . Then there is a k and a pseudorandom generator $g : \Sigma^{k \log n} \rightarrow \Sigma^n$ computable in time polynomial in n such that for all relativizable circuits C of size n*

$$\left| \Pr_{s \in \Sigma^{k \log n}} (C^A(g(s)) = 1) - \Pr_{r \in \Sigma^n} (C^A(r) = 1) \right| = o(1).$$

Peter Bro Miltersen [Mil01] makes the following connection to uniform hardness.

Lemma 2.9 (Miltersen). *If $DTIME(2^{O(n)})$ is not contained in $DSPACE(2^{o(n)})$ for infinitely many input lengths then for every language A in $PSPACE$ there is some $\epsilon > 0$ such that $DTIME(2^{O(n)})$ contains a language that does not have circuits of size $2^{\epsilon n}$ with access to A .*

By Lemma 2.9 under the assumption that $DTIME(2^{O(n)})$ is not contained in $DSPACE(2^{o(n)})$, the hypothesis and conclusion hold for Lemma 2.8 for any A in $PSPACE$, and in particular any A in the polynomial-time hierarchy.

3 Finding Low-Depth Witnesses

In this section we study the relation between the depth of a solution for a Boolean formula and the existence of a probabilistic algorithm to solve the formula. For this section we assume that n represents the number of variables in the Boolean formula considered.

Theorem 3.1. *Let ϕ be a Boolean formula over n variables. If ϕ has a satisfying assignment w with $\text{depth}_t(w|\phi) \leq O(\log t + \log n)$ then there exists a probabilistic algorithm that finds a satisfying assignment of ϕ in time polynomial in t and n .*

Proof. Let $m = K(w|\phi)$. Since $\text{depth}_t(w|\phi) = K^t(w|\phi) - K(w|\phi) \leq c \log t + c \log n$ we can write

$$m' = K^t(w|\phi) \leq m + c \log t + c \log n \quad (3.1)$$

Now consider the following set:

$$A = \{z | \phi(z) = \text{True} \text{ and } K^t(z) \leq m'\}$$

where we have used $\phi(z)$ to denote the value of ϕ for the assignment z . The second condition of the definition of A says that if $z \in A$, there exists a program p such that $|p| \leq m'$ and p generates z in time t . By construction, given ϕ and m' , A is a computable set and $w \in A$.

From the fact that $w \in A$, we get $m = K(w|\phi) \leq K(w|\phi, m') + K(m') \leq \log |A| + O(\log n)$. Using Inequality 3.1 we can write:

$$|A| \geq \frac{2^m}{c'n} \geq \frac{2^{m'}}{\text{poly}(n, t)}.$$

The following probabilistic algorithm M produces a satisfiable assignment for ϕ .

Input: Formula ϕ ; Output: z an assignment for ϕ ;

1. Guess m' ;
2. Generate randomly a program p of length at most m' ;
3. Run the universal Turing machine U with program p for t steps;
4. If z is a satisfiable assignment accept, otherwise reject.

The probability of this algorithm generating an assignment for ϕ is at least:

$$\frac{1}{n + c \log t + c \log n} \times \frac{|A|}{2^{m'}} \geq \frac{2^{m'}/\text{poly}(n, t)}{2^{m'}} = \frac{1}{\text{poly}(n, t)}.$$

If we run the algorithm a polynomial number of times with high probability one of those runs will produce a satisfying assignment of ϕ . \square

Consider the converse problem, i.e., if a probabilistic algorithm finds a valid assignment for a Boolean formula ϕ in time t , is there a witness w for ϕ such that $\text{depth}_t(w|\phi) \leq O(\log t + \log n)$, where n is the number of variables occurring in ϕ ?

The answer is false if we assume that $\text{BPP} = \text{FewP} = \text{EXP}$ and is true if we assume that good pseudorandom generators exist.

Theorem 3.2. *If $\text{BPP} = \text{FewP} = \text{EXP}$ then we can probabilistically find a witness for any satisfiable formula quickly but for every polynomial q there exists infinitely many Boolean formulas ϕ and some $\epsilon > 0$ such that*

$$\text{depth}_q(w|\phi) \geq n^\epsilon$$

for all satisfying assignments w of ϕ .

Proof. For any formula ϕ we can find a satisfying assignment in exponential time and since $\text{EXP} = \text{BPP}$ we can find a witness probabilistically quickly.

Fix a complete language L for EXP and since L is in FewP , let M be an NP machine accepting L with at most $p(n)$ accepting paths on each input.

Let x be in L with $|x| = n$. By Cook's reduction of the NP-completeness of SAT, we can compute a ϕ that has the same number of satisfying assignments as $M(x)$ has accepting computations. Let w be any witness of ϕ . We have

- $K(w|\phi) = O(\log n)$, since we can search for the satisfying assignments w of ϕ and identify one of them by its index.
- Let q be any polynomial. If $K^q(w|\phi) \leq n^{o(1)}$ then we can compute whether x is in L in deterministic time $2^{n^{o(1)}}$ by trying all small programs and seeing if any of them produce a witness.

By the time hierarchy theorem EXP is not contained in deterministic time $2^{n^{o(1)}}$ so there must be infinitely many ϕ with $K^q(w|\phi) \geq n^\delta$ for some $\delta > 0$. Thus we get $\text{depth}_q(w|\phi) \geq n^\epsilon$ for any $\epsilon < \delta$. \square

We now prove that if good pseudorandom generators exist then the converse proposition holds:

Proposition 3.3. *Let ϕ be a Boolean formula over n variables. Under the hypothesis of Lemma 2.9 there exists a probabilistic algorithm that produces a witness w of ϕ in time $t(n)$, then $\text{depth}_t(w|\phi) \leq O(\log n + \log t)$.*

Proof. Since there exists a probabilistic algorithm that finds an assignment for ϕ in time $t(n)$, then there exists a random string r such that $|r| \leq t(n)$ and $\text{depth}_t(w|\phi, r) = O(1)$. Now, given a seed of length $O(\log r + \log t)$ we can derandomize the algorithm in BPP by using the pseudorandom generator. So $K^t(w|\phi) \leq O(\log r + \log t) \leq O(\log n + \log t)$ and then $\text{depth}_t(w|\phi) \leq O(\log n + \log t)$. \square

While we don't believe that $\text{BPP} = \text{FewP} = \text{EXP}$, one cannot have a relativizable proof of separation. By combining the constructions of relativized worlds where $\text{UP} = \text{EXP}$ and $\text{BPP} = \text{EXP}$, we get a relativized world where $\text{UP} = \text{FewP} = \text{BPP} = \text{EXP}$.

*** NEED REFERENCES ***

4 Depth Cannot Increase Rapidly

In this section we show that if f is an honest polynomial time computable function then it can not significantly increase the depth of its argument, i.e., deep objects are not quickly produced from shallow ones. A function f is honest if for some polynomial p , $p(|f(x)|) \geq |x|$ for all x .

We start by showing that relative to a random oracle for every honest efficiently computable f , the depth of $f(x)$ cannot be much greater than the depth of x . Later on we will replace the random oracle with a pseudorandom generator.

Lemma 4.1. *Let $f : \Sigma^* \rightarrow \Sigma^*$ be an honest polynomial time computable function and $x \in \Sigma^n$. If $y = f(x)$, then, relative to a random oracle, $\text{depth}_{t'}(y) \leq \text{depth}_t(x) + O(\log n)$, for any polynomial $t(n)$ and some polynomial $t'(n)$ depending on $t(n)$.*

Proof. Let $y = f(x)$ and consider the following set

$$A_y = \{z : f(z) = y \text{ and } K^t(z) \leq K^t(x)\}$$

By construction, x is in A_y and since f is computable in polynomial time, A_y can be computed by enumerating all programs of size up to $K^t(x)$,

running them up to time t and keeping the outputs z that satisfy $f(z) = y$. So,

$$K(x|y) \leq K(A_y|y) + \log |A_y| = \log |A_y| + O(\log n)$$

which implies $|A_y| \geq 2^{K(x|y) - O(\log n)}$. By symmetry of information we have:

$$K(x|y) = K(y|x) + K(x) - K(y) \pm O(\log n).$$

As $y = f(x)$ and f is known, we have that $K(y|x) = O(1)$ and so

$$K(x|y) = K^t(x) - \text{depth}_t(x) - K(y) \pm O(\log n)$$

implying that

$$|A_y| \geq 2^{K(x|y) - O(\log n)} = \frac{2^{K^t(x)}}{2^{\text{depth}_t(x) + K(y) + c \log n}}$$

for some constant c . If we randomly pick a program of size smaller or equal than $K^t(x)$, the probability that it is in A_y is at least

$$\frac{1}{2^{\text{depth}_t(x) + K(y) + c \log n}}.$$

Now let R be a random oracle. We can use R as a function $\Sigma^k \rightarrow \Sigma^{\leq K^t(x)}$ where $k = \text{depth}_t(x) + K(y) + c' \log n$ and then

$$\begin{aligned} \Pr[\exists w : R(w) \in A_y] &= 1 - \Pr[\forall w : R(w) \notin A_y] \\ &\geq 1 - \left(1 - \frac{1}{2^{\text{depth}_t(x) + K(y) + c' \log n}}\right)^{2^{\text{depth}_t(x) + K(y) + c' \log n}} \\ &= 1 - e^{-2^{(c'-c) \log n}} = 1 - e^{-n^{c'-c}} \geq 1 - 2^{-n^{c'-c}} \end{aligned}$$

Then

$$\begin{aligned} \Pr[\forall y \exists w : R(w) \in A_y] &= 1 - \Pr[\exists y \forall w_y : R(w_y) \notin A_y] \\ &\geq 1 - 2^n 2^{-n^{c'-c}} = 1 - 2^{-n^{c'-c-1}} \end{aligned}$$

For the appropriate choice of c' we can find a w_y for every y , with high probability, such that $R(w_y)$ is in A_y , i.e., $f(R(w_y)) = y$. So for some polynomial t' we have

$$K^{t'}(y) \leq |w_y| + O(1) = \text{depth}_t(x) + K(y) + O(\log n).$$

From this we conclude that

$$\text{depth}_{t'}(y) \leq \text{depth}_t(x) + O(\log n).$$

□

We now improve the previous result to more general terms by using the composition of the pseudorandom generator in Lemma 2.8 with the pseudorandom generator in Lemma 2.7, as done by Antunes and Fortnow [AF05].

Theorem 4.2. *Let $f : \Sigma^* \rightarrow \Sigma^*$ be a polynomial time computable function and $x \in \Sigma^n$. Under the hypothesis of Lemma 2.9, if $y = f(x)$ then $\text{depth}_{t'}(y) \leq \text{depth}_t(x) + O(\log n)$ for any polynomial $t(n)$ and some polynomial $t'(n)$ depending on $t(n)$.*

Proof. Continuing the proof of Lemma 4.1, consider the set \mathcal{H} of functions h such that for all y there is a w_y such that $f(h(w_y)) = y$. The set \mathcal{H} fulfills the requirements of Lemma 2.7 so we can use a polynomially-long random seed to describe an h in \mathcal{H} . Given a good pseudorandom generator (Lemma 2.8) we can use an $O(\log n)$ bit random string to generate the seed for the first generator. We call the result of the composition of the two pseudorandom generators G .

Composing this procedure with the procedure of the previous theorem, we have a way to describe y by the following program for a fixed y :

Input: a seed s and a witness w_y

Output: y

1. Compute $s' = G(s)$.
2. Compute $h = R(s')$ where R is a procedure that computes a function h from the oracle s' as per Lemma 4.1. The result is a function that maps a seed of length $\text{depth}_t(x) + K(y) + O(\log n)$ into a program of size at most m .
3. Compute $p = h(w_y)$, giving a program in the set B_y .
4. Run the universal Turing Machine with program p and call the output z .
5. Compute $f(z)$. (By the construction in the Lemma 4.1, this is y .)
6. Output y .

Since the several constructions are independent of any given instance, we have a description for y requiring only the description of s and w_y that can be computed in time $t'(n)$, a polynomial depending on the running time of the function h , depending on t , and the running time of p , again depending on t . Therefore,

$$\begin{aligned}
K^{t'}(y) &\leq |s| + |w_y| + O(1) \\
&= O(\log n) + \text{depth}_t(x) + K(y)
\end{aligned}$$

So, $\text{depth}_{t'}(y) \leq \text{depth}_t(x) + O(\log n)$. □

However if $\text{BPP} = \text{EXP}$ the previous result does not hold.

Theorem 4.3. *Assume $\text{BPP} = \text{EXP}$. There exists a polynomial-time computable function f such that for all polynomials t and t' there exists a polynomial q such that for every natural number n there exist strings y and x with $|y| = n$ and $|x| = q(n)$ such that*

1. $y = f(x)$,
2. $\text{depth}_t(y) \geq n - O(\log n)$, and
3. $\text{depth}_{t'}(x) \leq O(\log n)$.

Proof. Fix n . Let y be the lexicographically least string such that $K^t(y) \geq n$. We can compute y so $K(y) \leq O(\log n)$ and $\text{depth}_t(y) \geq n - O(\log n)$.

We can find y in time $2^{O(n)}$ given n and $t(n)$ so by the hypothesis $\text{BPP} = \text{EXP}$ there is a probabilistic algorithm A that will efficiently compute y . Note that the running time of A is independent of t' .

Let $m = q(n)$ be the number of random bits used by A . Let $f(r)$ simulate A using r as the random coins. Let x be Kolmogorov random of length m .

We have $f(x) = y$ since the set of strings that cause f to give the wrong answer will be small and all such strings will have low Kolmogorov complexity.

Finally we have $\text{depth}_{t'}(x) \leq O(\log n)$ because x is random. □

5 Properties of Conditional Depth

Bennett [Ben88] noted that the impossibility of rapid growth of depth can not be extended to a transitive law relative to shallowness, i.e., if x is shallow relative to y and y is shallow relative to z , this does not necessarily imply that x is shallow relative to z . Bennett considered z to be a Kolmogorov random string of size n , $y = 0^n$ and $x = z \oplus d$ where d is some deep string. Bennett's example does not work for computational depth. It is easy to see

that $K(z \oplus d|z)$ is small however when looking at $K^t(z \oplus d|z) = K^t(d|z)$, if $BPP = EXP$ we may use z to find d .

We conjecture that the transitive law relative to shallowness does not hold. However, assuming that pseudorandom generators exist, we show that depth satisfies an analog of a triangular inequality.

Theorem 5.1. *Let $x, y, z \in \{0, 1\}^*$ and $n = \max(|x|, |y|, |z|)$. Under the hypothesis of Lemma 2.9, given a polynomial $t(n)$ there exists a polynomial $t'(n)$ such that*

$$\text{depth}_{t'}(y|z) \leq \text{depth}_t(x|z) + \text{depth}_t(y|x, z) + O(\log n)$$

Proof. Define $a = K^t(x|z)$, $b = K^t(y|x, z)$, $\text{depth}_t(x|z) = r$ and $\text{depth}_t(y|x, z) = s$. Then,

$$K(x|z) = a - r \text{ and } K(y|x, z) = b - s.$$

Consider the set A consisting of all w such that there exists a $|u| \leq a$ and $|v| \leq b$ with $U^t(u, z) = w$ and $U^t(v, w) = y$.

By construction, x is an element of A and A is computable given y and z . Then, $K(x|y, z) \leq \log |A| + c$, i.e., A has at least $2^{K(x|y, z) - c}$ elements. By symmetry of information, we have that

$$\begin{aligned} K(x|y, z) &\geq K(y|x, z) + K(x|z) - K(y|z) - O(\log n) \\ &= b - s + a - r - K(y|z) - O(\log n) \end{aligned}$$

Thus

$$|A| \geq \frac{2^{a+b}}{2^{r+s+K(y|z)+c' \log n}}$$

for some constant c' . The probability of a random program p of size smaller than $a + b + O(\log n)$ generating y is at least $\frac{1}{2^{r+s+K(y|z)+c' \log n}}$. Using a similar construction of Theorem 4.2 we can find a seed of size $r + s + K(y|z) + O(\log n)$ that generates a program producing y within polynomial time t' . So,

$$K^{t'}(y|z) \leq K(y|z) + r + s + O(\log n)$$

and then

$$\text{depth}_{t'}(y|z) \leq r + s + O(\log n).$$

□

Corollary 5.2. *Let $x, y \in \{0, 1\}^*$ and $n = \max(|x|, |y|)$ and fix some polynomial $t(n)$. Then, assuming the Hypothesis of Lemma 2.9, there exists a polynomial $t'(n)$ such that*

$$\text{depth}_{t'}(y) \leq \text{depth}_t(x) + \text{depth}_t(y|x) + O(\log n).$$

However if we replace the pseudorandom generators assumption by the assumption $\text{BPP} = \text{EXP}$ the previous results do not hold.

Theorem 5.3. *If $\text{BPP} = \text{EXP}$, then there exist $x, y \in \{0, 1\}^*$ such that $\text{depth}_t(x)$ and $\text{depth}_t(y|x)$ are small but $\text{depth}_t(y)$ is big, with $n = \max(|x|, |y|)$ and t a polynomial in n .*

Proof. Let x be a Kolmogorov random string, and y the least lexicographic string with high $K^t(y)$. Since all random strings are shallow, $\text{depth}_t(x)$ is small.

The string y can be computed by the following procedure: enumerate all binary strings in lexicographic order and for each string w , compute $K^t(w)$. If this number is bigger than a certain threshold, then output w and stop. Then, $K(y) = O(1)$. On the other hand, by construction $K^t(y)$ is high, so $\text{depth}_t(y)$ is large.

Now we address $\text{depth}_t(y|x)$. Since $K(y)$ is small, then $K(y|x)$ is also small. Define L as the language consisting exactly of the string y . The program for y given above enumerates at most $2^{|y|}$ strings before outputting a result, and for each of them it executes up to a polynomial number of steps no more than $2^{K(y)} \leq 2^{|y|}$ programs. Thus, $L \in \text{EXP}$. Since by assumption $\text{BPP} = \text{EXP}$, there is a probabilistic algorithm that takes a certain random input, runs in polynomial time and outputs y . We let this random input be x . Taking the size of the probabilistic algorithm to be a constant, $K^t(y) = O(1)$ and thus $\text{depth}_t(y|x) = O(1)$. □

Acknowledgments

We thank Harry Buhrman and Aram Harrow for helpful discussions. We thank the anonymous Complexity reviewers for many useful comments, particularly for pointing out that in Theorem 3.2 we could use FewP instead of UP.

References

- [AFV01] L. Antunes and L. Fortnow. Sophistication Revisited. *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of Lecture Notes in Computer Science, pages 267-277. Springer, 2003.
- [AF05] L. Antunes and L. Fortnow. Time-bounded universal distributions. Technical Report TR05-144, Electronic Colloquium on Computational Complexity, 2005.
- [AFMV06] L. Antunes and L. Fortnow and D. van Melkebeek and N. V. Vinodchandran. Computational depth: concept and applications. *Theor. Comput. Sci.*, 354 (3): 391–404, 2006.
- [Ben88] Bennett, C. Logical Depth and Physical Complexity *The Universal Turing Machine, A Half-Century Survey*, pages: 227-257. Oxford University Press, 1988.
- [Cha66] G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13(4):145–149, 1966.
- [Has86] J. Hastad. Computational limitations of small-depth circuits. *Ph. D. Thesis*, MIT Press, 1986.
- [IW97] R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: derandomizing the XOR lemma. In *Proceedings of the 29th STOC*, pages 220–229, 1997.
- [KvM02] A. Klivans and D. van Melkebeek. Graph Nonisomorphism Has Subexponential Size Proofs Unless The Polynomial-Time Hierarchy Collapses. *SIAM Journal on Computing*, 31: 1501-1526, 2002.
- [Kol65] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1–7, 1965.
- [Lev73] L. Levin. Universal Search Problems. *Problems Inform. Transmission*, 9(1973), 265-266.
- [LV08] Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Verlag, 3rd edition, 2008.
- [Mil01] P. Miltersen. Derandomizing complexity classes. In *Handbook of Randomized Computing*. Kluwer, 2001.

- [NW94] N. Nisan and A. Wigderson. Hardness vs. randomness *J. Comput. Syst. Sci*, 49 (2):149–167, 1994
- [Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.
- [Sol64] R. Solomonoff. A formal theory of inductive inference, part I. *Information and Control*, 7(1):1–22, 1964.