Computational Depth

Luis Antunes *
University of Porto

Lance Fortnow†
NEC Research Institute

Dieter van Melkebeek‡
Institute for Advanced Study

Abstract

We introduce Computational Depth*, a measure for the amount of "nonrandom" or "useful" information in a string by considering the difference of various Kolmogorov complexity measures. We investigate three instantiations of Computational Depth:*

- *Basic Computational Depth, a clean notion capturing the spirit of Bennett's Logical Depth.*
- *Time-t Computational Depth and the resulting concept of Shallow Sets, a generalization of sparse and random sets based on low depth properties of their characteristic sequences. We show that every computable set that is reducible to a shallow set has polynomial-size circuits.*
- *Distinguishing Computational Depth, measuring when strings are easier to recognize than to produce. We show that if a Boolean formula has a nonnegligible fraction of its satisfying assignments with low depth, then we can find a satisfying assignment efficiently.*

Torturing an uninformed witness cannot give information about the crime. *Leonid Levin* [10]

1 Introduction

Karp and Lipton [8] show that if NP reduces to a sparse set then NP has polynomial-size circuits and the polynomial-time hierarchy collapses. Bennett and Gill [2] show that if NP reduces to a random set then NP has polynomial-size circuits and the polynomial-time hierarchy collapses. Are these two separate results or just two specific examples of some more general principle? We show that the latter is true.

Both sparse and random sets do not contain much information about NP problems like Satisfiability or about any other fixed language for that matter. We can always simulate the effects of a random oracle by flipping coins.

Kolmogorov complexity measures the amount of information in a string. However randomly chosen strings have high Kolmogorov complexity. We need some method to measure the amount of nonrandom information in a string.

We develop *Computational Depth* to accomplish exactly this. The concept is simple: We consider the difference of two different Kolmogorov complexity measures. What remains is the nonrandom information we desire.

We define several types of Computational Depth based on this idea. There is no single best type of Computational Depth. Rather different notions have different properties and applications that one can use as appropriate.

In this paper we focus on three specific types of Computational Depth: Basic Computational Depth, Distinguishing Computational Depth, and Time-t Computational Depth.

Basic Computational Depth looks at the difference between Levin's Ct complexity and the traditional unrestricted Kolmogorov complexity. Basic Computational Depth with its rather simple definition captures the intuition behind Bennett's [3] rather technical notion of Logical Depth. A strings x has large Logical Depth if it has short programs but these programs require considerable computation time. Computational Depth has a similar property. We show that exponentially many strings of a given length have both large Logical Depth and Basic Computational Depth.

*Research done while visiting the Center for Discrete Mathematics and Computer Science (DIMACS). Supported in part by DIMACS and Fundação Calouste Gulbenkian. This author is also partially supported by funds granted to LIACC through the Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia and Programa POSI. Address: Artificial Intelligence and Computer Science Laboratory, R.Campo Alegre, 823, 4150-180 Porto, Portugal. Email: `lfa@ncc.up.pt`. Web: `http://www.ncc.up.pt/~lfa`.

†Address: 4 Independence Way, Princeton, NJ 08540 Email: `fortnow@research.nj.nec.com`. Web: `http://www.neci.nj.nec.com/homepages/fortnow`.

‡Address: School of Mathematics, Institute for Advanced Study, Einstein Drive, Princeton, NJ 08540. Email: `dieter@ias.edu`. Web: `http://www.math.ias.edu/~dieter`.

We develop Time-t Computational Depth and *Shallow Sets* to answer the question posed at the beginning of the introduction. Shallow sets are sets where the initial segments of their characteristic sequence have similar polylogarithmic time-bounded Kolmogorov complexity and traditional Kolmogorov complexity. Sparse sets and random sets are shallow.

Using Nisan-Wigderson generators [12], we show that if a computable set A is polynomial-time Turing reducible to a shallow set then A has polynomial-size circuits generalizing this result for random sets due to Bennett and Gill [2]. Karp and Lipton [8] show that if all NP sets have polynomial-size circuits than the polynomial-time hierarchy collapses and thus we also get this collapse if all NP sets are shallow.

Distinguishing Computational Depth considers the difference between polynomial-time bounded distinguishing complexity as developed by Sipser [14] and polynomial-time bounded Kolmogorov complexity. It measures the difference between recognizing a string and producing it. Fortnow and Kummer [6] show that under reasonable assumptions, there exist strings with high Distinguishing Computational Depth.

We show that if a nonnegligible fraction of the satisfying assignments of a formula ϕ have low Distinguishing Computational Depth given ϕ then we can find a satisfying assignment in probabilistic quasipolynomial time. We also show that injective polynomial-time computable functions cannot map strings of low depth to high depth.

2 Preliminaries

We briefly introduce Kolmogorov complexity and some of its variants. We present them at the level of generality we will need. We refer to the textbook by Li and Vitányi [11] for more details.

We fix once and for all a universal (oracle) Turing machine U.

Definition 2.1 *Let x and y be strings, t a time-bound and A an oracle. The t-*time bounded Kolmogorov complexity *of x given y relative to A is*

$$\begin{aligned} C^{t,A}(x|y) &= \min_p\{|p| : U^A(p,y) \text{ halts in at most} \\ &\quad t(|x|+|y|) \text{ steps and outputs } x\}. \end{aligned}$$

The Kolmogorov complexity of x given y relative to A is $C^A(x|y) = C^{\infty,A}(x|y)$.

The default value for y is the empty string ϵ, and for A the empty oracle. We typically drop these arguments in the notation if they have their default values.

A different universal machine U may affect the program size $|p|$ by at most a constant additive factor, and the running time t by at most a logarithmic multiplicative factor. The same will hold for all other measures we will introduce.

Definition 2.2 *A string x is* incompressible *if $C(x) \geq |x|$. We also call such x algorithmically random.*

Proposition 2.3 *For all nonnegative integers n, at least half of the strings of length at most n are incompressible.*

The classical Kolmogorov complexity of a string x does not take into account the time necessary to produce the string from a description of length $C(x)$. Levin [9] introduced a useful variant of Kolmogorov complexity weighing program size and running time.

Definition 2.4 (Levin) *For any strings x, y, the Levin complexity of x given y is*

$$\begin{aligned} Ct(x|y) &= \min_{p,t}\{|p| + \log t : U(p,y) \text{ halts in at most } t \\ &\quad \text{steps and outputs } x\}. \end{aligned}$$

Next we introduce Bennett's [3] definition of logical depth. A string x is called logically deep if it takes a lot of time to generate it from any short description.

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

$$\begin{aligned} depth_s(x) &= \min_p\{t : U(p) \text{ halts in at most } t \text{ steps,} \\ &\quad \text{outputs } x, \text{ and } |p| < C(x)+s\}. \end{aligned}$$

Note that algorithmically random strings are shallow at any significance level. In particular, Chaitin's Ω is shallow.

Bennett has proved that a fast deterministic processes is unable to transform a shallow object into a deep one, and that fast probabilistic processes can do so only with small probability. This property is referred to as the *slow growth law*.

Instead of considering a shortest program that outputs a string x, we could also consider a shortest program that distinguishes x from all other strings, i.e., it accepts x and rejects every other input. In the unbounded setting the two measures coincide up to a constant, as we can run through all possible strings until we find the one accepted by the program, and print it out. In a resource-bounded setting, there seems to be a substantial difference.

Distinguishing complexity was introduced by Sipser [14], who used it to show that BPP is in the polynomial-time hierarchy.

Definition 2.6 (Sipser) *Let x and y be strings, t a time-bound and A an oracle. The t-*time bounded distinguishing complexity *of x given y relative to A, $CD^{t,A}(x|y)$, is the length of the shortest program p such that*

1. *$U^A(p,x,y)$ accepts in at most $t(|x|+|y|)$ steps, and*

2. *$U^A(p,z,y)$ rejects for all $z \neq x$.*

Again, we may drop y and A from the notation in case they have their default values.

Sipser used distinguishing complexity to answer the question of how much information is needed to distinguish a given string from all other strings in a given set.

Kolmogorov complexity gives the following answer to this question.

Lemma 2.7 *Let A be an oracle. There exists a constant c such that for all strings x of length n in A*

$$C^A(x) \leq \log|A \cap \{0,1\}^n| + c\log n.$$

The running time of the programs underlying Lemma 2.7 can be exponential. Sipser [14] proved the following theorem with the aid of a polynomially-long random string.

Theorem 2.8 (Sipser) *There is a polynomial p and a constant c such that for any oracle A, for every string x of length n in A, and for most strings r of length $p(n)$*

$$CD^{p,A}(x|r) \leq \log|A \cap \{0,1\}^n| + c\log n.$$

Buhrman and Fortnow [4] showed how to eliminate r at the cost of doubling the complexity.

Theorem 2.9 (Buhrman-Fortnow) *There is a polynomial p and a constant c such that for any oracle A, and for all strings x of length n in A*

$$CD^{p,A}(x) \leq 2\log|A \cap \{0,1\}^n| + c\log n.$$

Fortnow and Laplante [7] showed that the factor of 2 can be removed for all but a small fraction of the strings.

Theorem 2.10 (Fortnow-Laplante) *For every positive ϵ there is a polynomial p and a constant c such that for any oracle A, for any length n, and for all but an ϵ fraction of the strings x in $A \cap \{0,1\}^n$*

$$CD^{p,A}(x) \leq \log|A \cap \{0,1\}^n| + (\log\frac{n}{\epsilon})^c.$$

The proof of Fortnow and Laplante uses explicit constructions of expanders due to Ta-Shma [15]. An optimal extractor construction would allow us to replace $(\log\frac{n}{\epsilon})^c$ by $c\log\frac{n}{\epsilon}$ in the statement of Theorem 2.10. Buhrman, Laplante and Miltersen [5] have proved that the constant factor 2 in Theorem 2.9 is optimal in relativized worlds.

3 Computational Depth

Kolmogorov complexity measures the amount of information in a string x as the length of a shortest description of x. Random strings are incompressible. They are therefore deemed to contain a lot of information. However, random information may not be very useful from a computational point of view.

Intuitively, computational depth measures the amount of "nonrandom" or "useful" information in a string. Formalizing this intuitive notion is tricky. A computationally deep string x should take a lot of effort to construct from its short description. Incompressible strings are trivially constructible from their shortest description, and therefore computationally shallow.

After some attempts Bennett [3] formally defined the s-significant logical depth of an object 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. We consider logical depth as one instantiation of this general theme, and propose several other variants. Which one is most appropriate depends on the problems we want to apply it to.

The first notion of computational depth we propose is the difference between a time-bounded Kolmogorov complexity and traditional Kolmogorov complexity. From this measure we define *shallow sets* and show that, in a computational setting, shallow sets provide very little useful information: Every computable set that reduces to a shallow set also reduces to a sparse set.

Our second variant is the difference between two time bounded complexity measures namely, $C^{t_1}(x)$ and $CD^{t_2}(x)$ where t_1, t_2 are polynomials. We establish an analog to Bennett's slow growth law for this measure. We also show that if a nonnegligible fraction of the satisfying assignments of a Boolean formula ϕ have small depth then we can find a satisfying assignment of ϕ in quasipolynomial probabilistic time.

Finally, we present a third variant based on Levin's Ct complexity. We argue that it forms a closely related but more manageable version of Bennett's logical depth.

3.1 Time-t Depth and Shallow Sets

In this section we discuss shallow sets, sets containing little nonrandom information. We will show that computable sets reducible to shallow sets must have small circuits. In particular if NP-complete sets reduce to shallow sets then the polynomial-time hierarchy collapses.

First we give a simple notion of depth based on a time complexity function t.

Definition 3.1 *Let t be a time-bound. The time-t depth of a string x is*

$$D^t(x) = C^t(x) - C(x).$$

To define shallow sets A we will look at a depth measure on the initial segments of the characteristic sequence of a set A, i.e.,

$$A(\epsilon)A(0)A(1)A(00)\ldots$$

To properly define shallow sets we need to use Definition 3.1 for sublinear time bounds. We give a definition for C^t for sublinear time functions t by allowing the universal machine U random access to the description r of the string x (denoted as oracle access U^r) and requiring only that each bit of x be generated in the allotted time.

Definition 3.2 *Let t be a time-bound and x a string.*

$$\begin{aligned} C^t(x) \quad &= \quad \min_{p,r}\{|p| + |r| : U^r(p,i) \textit{ outputs } x_i \\ &\qquad \textit{in } t(|x|) \textit{ steps for all } 1 \le i \le |x|\}. \end{aligned}$$

This definition is essentially equivalent to Definition 2.1 for superlinear t.

We can now define shallow sets. We first define shallow strings.

Definition 3.3 *Fix a constant k. The string x is k-*shallow *if*

$$D^{\log^k}(x) \le \log^k |x|.$$

In the proof of our main result we will be interested in the characteristic sequences available to some Turing machine running in time n^j on some input of length n. The initial segment of the characteristic sequence up to length n^j has length $N = 2^{n^j+1} - 1$. In that case, $\log^k N$ is approximately n^{jk}.

We now define shallow sets.

Definition 3.4 *A set A is* shallow *if there exists a k such that almost every initial characteristic sequence of A is k-shallow.*

Every sparse set is shallow. In fact, every set that is polynomial-time reducible to a sparse set is shallow. Random sets are also shallow: A randomly chosen set is shallow with probability one.

Despite the fact that most sets are shallow, we now show that these sets have very limited computational power.

Theorem 3.5 *If we have sets A and B, A shallow, B computable and $B \in P^A$ then B is in P/poly.*

To prove Theorem 3.5 we need the following result of Nisan and Wigderson [12]:

Lemma 3.6 (Nisan-Wigderson) *For any fixed nonnegative integer d, there exists a family of generators $\{G_0, G_1, \ldots\}$ with the following properties:*

- *G_v maps strings of length u polynomial in $\log v$ to strings of length v.*
- *For any circuit D of depth d and size v, we have*

$$|\Pr_{\rho\in\{0,1\}^v}[D(\rho)] - \Pr_{\sigma\in\{0,1\}^u}[D(G_v(\sigma))]| < 1/v.$$

- *Each output bit of G_v is computable in time polynomial in $\log v$.*

Proof of Theorem 3.5. By assumption there is some Turing machine M running in time n^j for some j, such that $B = L(M^A)$, and all sufficiently long initial segments of the characteristic sequence of A are k-shallow for some nonnegative integer k.

Let a_i be the ith bit of the characteristic sequence of A. Fix some input length n. Let $z = a_1 \ldots a_{2^{n^j+1}-1}$ be the characteristic sequence of A up to strings of length n^j. Let $N = |z| = 2^{n^j+1} - 1$. We have that $C^{n^{jk}}(z) - C(z) \le n^{jk}$. Let $\ell \doteq C^{n^{jk}}(z)$, which gives us $C(z) \ge \ell - n^{jk}$. Note that $\ell \le |z| < 2^{n^j+1}$.

By Definition 3.2 there must be a p and r such that $|p| + |r| = \ell$ and $U^r(p,i)$ outputs a_i in time n^{jk} for each i, $1 \le i \le N$. Note that $C(z) \le C(\langle p, r\rangle) + O(\log n)$.

Now consider the set T consisting of all pairs (q,s) such that

- $|q| + |s| = \ell$,
- For $1 \le i \le N$, $U^s(q,i)$ halts in time n^{jk} and outputs some value f_i.
- For each string y of length n, y is in B iff $M^F(y)$ accepts where F is the oracle whose characteristic sequence is $f_1 f_2 \ldots f_N 000\ldots$.

The set T has some nice properties:

- (p,r) is in T.
- T can be computed by a constant depth circuit of size $2^{n^{O(1)}}$, namely as follows. For each string y of length n, we have to verify that the oracle machine M accepts y when its oracle queries, say about the value of f_i, are answered by running $U^s(q,i)$ for n^{jk} steps. Since M runs in time n^j, for any fixed y, this process can be viewed as a computation running in time $n^j \cdot n^{jk}$ with random access to (q,s). Such a computation can be expressed as an OR of $2^{n^{j(k+1)}}$ AND's of size $n^{j(k+1)}$ each over the input (q,s). AND'ing all these circuits together for all y's of length n yields a depth 3 circuit of size $2^n \cdot 2^{n^{j(k+1)}} \cdot n^{j(k+1)}$ deciding T. Call this circuit D.
- For each pair (q,s) in T, $C(\langle q,s\rangle) \le \log|T| + O(\log n)$ since B is computable.

By the third item we have

$$\begin{aligned}\log|T| &\geq C(\langle p,r\rangle) - O(\log n)\\ &\geq C(z) - O(\log n)\\ &\geq \ell - n^{jk} - O(\log n).\end{aligned}$$

This gives us $|T| \geq 2^\ell/2^{n^c}$ for some constant c.

Let v be the max of 2^{n^c} and the size of the circuit D describing T. Let G_v be the Nisan-Wigderson generator from Lemma 3.6. We have

$$|\Pr_{\rho\in\{0,1\}^v}[D(\rho)] - \Pr_{\sigma\in\{0,1\}^u}[D(G_v(\sigma))]| < 1/v.$$

Since D picks (q,s) uniformly from the initial bits of $\{0,1\}^v$ we have

$$\Pr_{\rho\in\{0,1\}^v}[D(\rho)] = \Pr_{|q|+|s|=\ell}[(q,s)\in T] \geq |T|/2^\ell \geq 1/v.$$

So we have

$$\Pr_{\sigma\in\{0,1\}^u}[D(G_v(\sigma))] > 0.$$

In particular, there is some σ such that $D(G_v(\sigma))$ is true. This σ has length polynomial in $\log v$ which is polynomial in n. We let this σ, v, $|q|$ and $|s|$ be our advice. From the advice we can efficiently compute every bit of a pair (q,s) in T which we can use to determine membership in B on strings of length n. $\diamond$

Karp and Lipton [8] show that if NP-complete languages have polynomial-size circuits then the polynomial-time hierarchy collapses to the second level. This gives us the following corollary.

Corollary 3.7 *If any NP-complete language is Turing-reducible to a shallow set then the polynomial-time hierarchy collapses to Σ_2^p.*

Balcázar, Díaz and Gabarró [1] showed the following characterization of $PSPACE/\mathrm{poly}$.

Theorem 3.8 *$A \in PSPACE/\mathrm{poly}$ if and only if for every n the characteristic sequence of A of strings up to length n has logarithmic Kolmogorov complexity by machines using polynomial space.*

We can use shallow sets to prove a similar result to characterize the computable sets in P/poly. Hartmanis argued that his approach could not be used to characterize P/poly because of the time needed for writing the output. We feel Definition 3.2 handles these issues well.

Corollary 3.9 *Let C be computable. C is in P/poly iff C is shallow.*

3.2 Distinguishing Computational Depth

In this section we introduce another variant of computational depth based on the difference between time bounded Kolmogorov complexity and time bounded distinguishing complexity. We prove a close analog of Bennett's slow growth law and also show how to find in quasipolynomial probabilistic time a satisfying assignment to any satisfiable Boolean formula for which a significant fraction of the satisfying assignments has logarithmic depth.

Definition 3.10 *Let x, y be strings, and t_1, t_2 be time-bounds. The (t_1, t_2)-*distinguishing computational depth *of x given y is*

$$D^{t_1,t_2}(x|y) = C^{t_1}(x|y) - CD^{t_2}(x|y).$$

It is clear that the distinguishing computational depth is always nonnegative. The exact difference between $C^t(x|y)$ and $CD^t(x|y)$ is not known. It is conceivable that both measures are always very close, in which case the notion of distinguishing depth would become trivial. However, this is unlikely because Fortnow and Kummer [6] showed that in that case the promise problem $(1SAT, SAT)$ can be solved in polynomial time.

Recall that $(1SAT, SAT) \in P$ if there is a deterministic polynomial-time algorithm which accepts all Boolean formulas with a unique satisfying assignment, and rejects all Boolean formulas which are not satisfiable. $(1SAT, SAT) \in P$ implies $NP = RP$ and $UP = P$, so in particular factoring is in P.

Theorem 3.11 (Fortnow-Kummer) $(1SAT, SAT) \in P$ *iff for every polynomial p_1 there is a polynomial p_2 and a constant c such that for any string x of length n, and any string y*

$$C^{p_2}(x|y) \leq CD^{p_1}(x|y) + c\log n.$$

We now start working towards the analog of Bennett's slow growth law for honest efficiently computable functions with few inverses. A function f is honest if for some polynomial p, $p(|f(x)|) \geq |x|$ for all x.

We will need the following lemma.

Lemma 3.12 *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function that is at most m to 1. For every polynomial p_1 there exists a polynomial p_2 such that for any string x of length n and any string y*

$$CD^{p_2}(x|y) \leq CD^{p_1}(f(x)|y) + 2\log m + O(\log n).$$

If f is a one-to-one function we have

$$CD^{p_2}(x|y) \leq CD^{p_1}(f(x)|y) + O(1).$$

Proof. Let p' be the program that distinguishes $f(x)$ given y. We create a program that on input $\langle z, y\rangle$ accepts only if $z = x$ as follows:

1. Simulate p' on $\langle f(z), y\rangle$ and reject if p' rejects. Otherwise we have $f(z) = f(x)$. If f is one-to-one we have $x = z$ and p' just accepts.

2. If $m > 1$, run a program that recognizes x among the at most m other strings that map to $f(x)$.

The first step takes polynomial time. If f is one-to-one we immediately get Lemma 3.12.

For the second step we can apply Theorem 2.9 to the set $A \doteq \{u : f(u) = f(x)\}$ given both y and $f(x)$. Note that $|A| \leq m$. Since f is polynomial-time computable, and we are given $f(x)$, we can simulate the queries to A in time polynomial in n. Therefore, we have that

$$CD^p(x|y, f(x)) \leq 2\log m + c\log n$$

for any sufficiently large polynomial p and constant c.

All together we get that for any large enough polynomial p_2

$$\begin{aligned} CD^{p_2}(x|y) &\leq CD^{p_1}(f(x)|y) \\ &\quad + CD^p(x|y, f(x)) + O(\log n) \\ &\leq CD^{p_1}(f(x)|y) \\ &\quad + 2\log m + O(\log n). \end{aligned}$$

$\diamond$

The analog to Bennett's slow growth law for logical depth reads as follows for distinguishing computational depth.

Theorem 3.13 *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable honest function that is at most m to 1. For all polynomials p_1, p_2 there exist a polynomials q_1, q_2 such that for any string x of length n and any string y*

$$D^{q_1,p_2}(f(x)|y) \leq D^{p_1,q_2}(x|y) + 2\log m + O(\log n).$$

If f is one-to-one we have

$$D^{q_1,p_2}(f(x)|y) \leq D^{p_1,q_2}(x|y) + O(1).$$

Proof. In order to produce $f(x)$, we can first produce x and then run a program for f on x. Since f is polynomial-time computable and honest, we have that for any polynomial p_1 there exists a polynomial q_1 such that

$$C^{q_1}(f(x)|y) \leq C^{p_1}(x|y) + O(1). \qquad (1)$$

Lemma 3.12 tells us that for any polynomial p_2 there exists a polynomial q_2 such that

$$CD^{p_2}(f(x)|y) \geq CD^{q_2}(x|y) - 2\log m - O(\log n) \quad (2)$$

for $m > 2$ and

$$CD^{p_2}(f(x)|y) \geq CD^{q_2}(x|y) - O(1) \qquad (3)$$

if f is one-to-one.

Subtracting (2) or (3) from (1) as appropriate finishes the proof of the theorem. $\diamond$

We next prove that if the depth of a nonnegligible fraction of the satisfying assignments of a Boolean formula is small then we can find a satisfying assignment in quasipolynomial probabilistic time.

Theorem 3.14 *For all functions $q(n) = 2^{\log^d n}$, there exist a polynomial p and a probabilistic quasipolynomial-time algorithm that given any satisfiable Boolean formula ϕ of size n such that at least a $1/q(n)$ fraction of the satisfying assignments x to ϕ have*

$$D^{q,p}(x|\phi) \leq \log^d n,$$

the algorithm outputs an assignment of ϕ with high probability.

Proof. Fix a satisfiable Boolean formula ϕ of length n and let A denote the set of satisfying assignments of ϕ.

¿From Theorem 2.10 we know that there exists a polynomial p and constant c such that all but a $\frac{1}{2q(n)}$ fraction of the satisfying assignments x to ϕ have

$$CD^p(x|\phi) \leq \log|A| + \log^c n.$$

By hypothesis, at least a fraction $\frac{1}{2q(n)}$ of the x in A must also satisfy

$$C^q(x|\phi) \leq CD^p(x|\phi) + \log^d n,$$

so we have

$$C^q(x|\phi) \leq \log|A| + \log^b n$$

for some constant b.

Now we randomly chose a program of length at most $k + \log^b(n)$, for every $1 \leq k \leq n$. Giving these programs as input to the universal Turing machine U we can produce in quasipolynomial time a satisfying assignment of ϕ with probability at least

$$\frac{\frac{|A|}{2q(n)}}{2^{\log|A| + \log^b(n) + 1}} \geq 2^{-\log^a n}$$

for some positive constant a. Repeating this procedure a quasipolynomial number of times will produce a satisfying assignment of ϕ with high probability. $\diamond$

3.3 Basic Computational Depth

By considering $Ct(x) - C(x)$ we get a variation of Bennett's logical depth.

Definition 3.15 *For any string x the basic computational depth of x is*

$$\begin{aligned} bcd(x) &= Ct(x) - C(x) \\ &= \min_p\{\log(t) + |p| - C(x) : \\ &\qquad U(p) \textit{ outputs } x \textit{ in } t \textit{ steps}\}. \end{aligned}$$

Basic computational depth has the advantage over logical depth that we do not need a significance level parameter. In fact, we are adding to (the logarithm of) the running time of the program a penalty $|p| - C(x)$ which can be viewed as a significance level.

Logically deep strings are not easy to identify, but can be constructed by diagonalization in time larger than 2^t for depth t [3]. We prove that there are an exponential number of strings with large basic computational depth. The result holds for Bennett's notion of logical depth as well.

Theorem 3.16 *There exists a constant c such that for any $0 < \epsilon < 1$ there are at least $2^{\epsilon n}$ strings x of length n satisfying*

$$bcd(x) \geq (1-\epsilon)n - c\log n.$$

Proof. Consider the set A consisting of all strings x of length n for which there exists a program p of length less than n such that $U(p)$ outputs x in at most 2^n steps. Let B denote $\{0,1\}^n \setminus A$ and D the lexicographically first $2^{\epsilon n}$ strings in B. The set D exists and we know that for every $x \in D$, $C(x) \leq \epsilon n + O(\log n)$. So for all $x \in D$ every program p of size less than n such that $U(p)$ outputs x must run for at least 2^n steps. It follows that for any $x \in D$, $bcd(x) \geq (1-\epsilon)n - c\log n$ for some constant c. $\diamond$

4 Concluding Remarks

We introduced three variants of computational depth and proved some interesting results about them. We mention some open problems suggested by our results. The obvious one is to improve some of our results, such as

Conjecture 4.1 *For all polynomials q there exist a polynomial p and a probabilistic polynomial-time algorithm that given any satisfiable Boolean formula ϕ of size n such that at least a fraction $1/q(n)$ of the satisfying assignments x to ϕ have*

$$D^{q,p}(x|\phi) \leq \log q(n),$$

the algorithm outputs a satisfying assignment of ϕ with high probability.

One obvious way to prove this conjecture is trying to improve Theorem 2.10 that we use in the proof of Theorem 3.14. We stress that explicit constructions of optimal extractors would prove the conjecture.

Let $\alpha(n)$ be a function such that $0 \leq \alpha(n) \leq 1/2$ for all n. Consider a oracle A chosen at random by putting x in A independently with probability $\alpha(n)$. Is A shallow with high probability? If $\alpha(n) = 1/2$ we can simply use the characteristic sequence for the encoding. For $\alpha(n) < 1/2$ the question remains open. We basically need an efficiently decodable encoding of any set $S \subseteq \{0,1\}^n$ of size $\alpha(n)2^n$ using $\log\binom{2^n}{\alpha(n)2^n} + n^{O(1)}$ bits. This is related to the static dictionary problem (see [13]).

Of course, other variants of computational depth and their applications would also be interesting.

Acknowledgments

We thank Harry Buhrman, Alex Samorodnitsky, Venkatesh Srinivasan, and Paul Vitányi for helpful discussions and comments. We thank Vinod Variyam and the two anonymous reviewers for many helpful suggestions on the presentation.

References

[1] J. L. Balcázar, J. Díaz, and J. Gabarró. On non-uniform polynomial space. In A. L. Selman, editor, *Structure in Complexity Theory*. Springer-Verlag, 1986.

[2] C. Bennett and J. Gill. Relative to a random oracle, $P^A \neq NP^A \neq co-NP^A$ with probability one. *SIAM Journal on Computing*, 10:96–113, 1981.

[3] C. H. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 227–257. Oxford University Press, 1988.

[4] H. Buhrman and L. Fortnow. Resource-bounded Kolmogorov complexity revisited. In *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science*, pages 105–116, Berlin, 1997. Springer.

[5] H. Buhrman, S. Laplante, and P. Miltersen. New bounds for the language compression problem. In *Proceedings of the 15th IEEE Conference on Computational Complexity*, New York, 2000. IEEE.

[6] L. Fortnow and M. Kummer. On resource-bounded instance complexity. *Theoretical Computer Science*, 161:123–140, 1996.

[7] L. Fortnow and S. Laplante. Nearly optimal language compression using extractors. In *Proceedings of the 15th Symposium on Theoretical Aspects of Computer Science*, pages 84–93, Berlin, 1998. Springer.

[8] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on the Theory of Computing*, pages 302–309. ACM, New York, 1980.

[9] L. A. Levin. Universal search problems. *Problems Inform. Transmission*, 9:265–266, 1973.

[10] L. A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[11] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

[12] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[13] R. Pagh. Low redundancy in static dictionaries with $O(1)$ lookup time. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 595–604. Springer, Berlin, 1999.

[14] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[15] A. Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 276–285, 22–24 May 1996.