Computational Depth: Concept and Applications*

Luis Antunes[1] **, Lance Fortnow[2], Dieter van Melkebeek[3]***, and N. V. Vinodchandran[4]†

[1] DCC-FC & LIACC
University of Porto, Porto, Portugal
[2] Department of Computer Science
University of Chicago, Chicago, IL, USA
[3] Department of Computer Sciences
University of Wisconsin, Madison, WI, USA
[4] Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE, USA

Torturing an uninformed witness cannot give information about the crime. *Leonid Levin* [Lev84]

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. We show that a Turing machine M runs in time polynomial on average over the time-bounded universal distribution if and only if for all inputs x, M uses time exponential in the basic computational depth of x.
- Sublinear-time 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.

* Preliminary versions of different parts of this paper appeared as [AFvM01] and [AFV03]. Much of the research for this paper occurred at the NEC Research Institute.
** Partially supported by funds granted to LIACC through the Programa de Financiamento Plurianual, FCT and Programa POSI.
*** Partially supported by NSF Career award CCR-0133693.
† Partially supported by NSF grant CCF-0430991

1 The Concept of Computational Depth

Karp and Lipton [KL80] show that if NP reduces to a sparse set then NP has polynomial-size circuits and the polynomial-time hierarchy collapses. Bennett and Gill [BG81] 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 such as 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 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. 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" or "useful" information we desire. 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.

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.

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

Basic Computational Depth looks at the difference between time-bounded Kolmogorov complexity and traditional unrestricted Kolmogorov complexity. Basic Computational Depth with its simple definition captures the intuition behind Bennett's [Ben88] rather technical notion of Logical Depth. A string 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 a Turing machine M runs in average polynomial-time if for all inputs x the Turing machine uses time exponential in the basic computational depth of x.

We develop Sublinear-time 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 [NW94], 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 [BG81]. Karp and Lipton [KL80] 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 [Sip83] and polynomial-time bounded Kolmogorov complexity. It measures the difference between recognizing a string and producing it. Fortnow and Kummer [FK96] 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.

The rest of this paper is organized as follows. In the next section, we present notation and definitions we use. We also state some related results from the literature. We consider Basic Computational Depth in Section 3, Sublinear-time Computational Depth and Shallow sets in Section 4, and Distinguishing Computational Depth in Section 5. Section 6 presents some concluding remarks.

2 Preliminaries

Most of our complexity-theoretic notation and definitions are standard and can be found in textbooks like [BDG95,Pap94]. We start with some background on Kolmogorov complexity and average case complexity theory.

2.1 Kolmogorov Complexity

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 [LV97] for more details.

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

Definition 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*

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

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

Proposition 1. *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 [Lev73] introduced a useful variant of Kolmogorov complexity weighing program size and running time.

Definition 3 (Levin). *For any strings x, y, the Levin complexity of x given y is*

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

Next we introduce Bennett's [Ben88] 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 4 (Bennett). *Let x be a string and s be a nonnegative integer. The logical depth of x at a significance level s is*

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

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 [Sip83], who used it to show that BPP is in the polynomial-time hierarchy.

Definition 5 (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 1. *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 1 can be exponential. Sipser [Sip83] proved the following theorem with the aid of a polynomially-long random string.

Theorem 1 (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 $\frac{3}{4}$th fraction of strings r of length $p(n)$*

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

Buhrman and Fortnow [BF97] showed how to eliminate r at the cost of doubling the complexity.

Theorem 2 (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 [FL98] showed that the factor of 2 can be removed for all but a small fraction of the strings.

Theorem 3 (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 extractors due to Ta-Shma [Tas96]. 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 3. Buhrman, Laplante and Miltersen [BLM00] have proved that the constant factor 2 in Theorem 2 is optimal in relativized worlds.

We need *prefix-free* Kolmogorov complexity defined using prefix free Turing machines: Turing machines with a one-way input tape (the input head can only read from left to right and crashes if it reads past the end of the input), a one-way output tape and a two-way work tape.

Definition 6. *Let U be a fixed prefix free universal Turing machine. Then for any string $x \in \{0,1\}^*$, the Kolmogorov complexity of x is,*

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

For any time constructible t, the t-time-bounded Kolmogorov complexity of x is,

$$K^t(x) = \min\{|p| : U(p) = x \text{ in at most } t(|x|) \text{ steps}\}$$

2.2 Average case complexity

In theoretical computer science we typically analyze the worst-case performance of algorithms. Many algorithms with bad worse-case performance nevertheless perform well in practice. The instances that require a large running-time rarely occur. Levin [Lev86] developed a theory of average-case complexity to capture this issue.

We give definitions from average case complexity theory necessary for our purposes. For more details readers can refer to the survey by Wang [Wan97]. In average case complexity theory, a computational problem is a pair (L, μ) where $L \subseteq \Sigma^*$ and μ is a probability distribution. The probability distribution is a function from Σ^* to the real interval $[0, 1]$ such that $\sum_{x\in\Sigma^*} \mu(x) \leq 1$. For probability distribution μ, the *distribution function*, denoted by μ^* is given by $\mu^*(x) = \sum_{y\leq x} \mu(x)$. The notion of *polynomial on average* is central to the theory of average case completeness.

Definition 7. *Let μ be a probability distribution function on $\{0,1\}^*$. A function $f : \Sigma^+ \to \mathbf{N}$ is polynomial on μ-average if there exists an $\epsilon > 0$ such that $\sum_x \frac{f(x)^\epsilon}{|x|}\mu(x) < \infty$.*

From the definition it follows that any polynomial is polynomial on μ-average for any μ. It is easy to show that if functions f and g are polynomial on μ-average, then the functions $f.g$, $f + g$, and f^k for some constant k are also polynomial on μ-average.

Definition 8. *Let μ be a probability distribution and $L \subseteq \Sigma^*$. Then the pair (L, μ) is in Average Polynomial time (denoted as Avg-*P*) if there is a Turing machine accepting L whose running time is polynomial on μ-average.*

We need the notion of *domination* for comparing distributions. The next definition formalizes this notion.

Definition 9. *Let μ and ν be two distributions on Σ^*. Then μ* dominates *ν if there is a constant c such that for all $x \in \Sigma^*$, $\mu(x) \geq \frac{1}{|x|^c}\nu(x)$. We also say ν is dominated by μ.*

Proposition 2. *If a function f is polynomial on μ-average, then for all distributions ν dominated by μ, f is also polynomial on ν-average.*

Average case analysis is, in general, sensitive to the choice of distribution. If we allow arbitrary distributions then average case complexity classes take the form of traditional worst-case complexity classes [LV92]. So it is important to restrict attention to distributions which are in some sense *simple*. Usually simple distributions are identified with the polynomial-time computable or polynomial-time samplable distributions.

Definition 10. *Let t be a time constructible function. A probability distribution function μ on $\{0,1\}^*$ is said to be t-time computable, if there is a deterministic Turing machine that on every input x and a positive integer k, runs in time $t(|x|+k)$, and outputs a fraction y such that $|\mu^*(x) - y| \leq 2^{-k}$.*

The most controversial aspect in the average case complexity theory is the association of the class of *simple* distributions with P-computable, which may seem too restrictive. Ben-David *et al.* in [BCGL92] use a wider family of natural distributions, P-samplable, consisting of distributions that can be sampled by randomized algorithms, working in time polynomial in the length of the sample generated.

Definition 11. *A probability distribution μ on $\{0,1\}^*$ is said to be* P*-samplable, if there is a probabilistic Turing machine M which on input 0^k produces a string x such that $|Pr(M(0^k) = x) - \mu(x)| \leq 2^{-k}$ and M runs in time $poly(|x|+k)$.*

Every P-computable distribution is also P-samplable, however the converse is unlikely.

Theorem 4 ([BCGL92]). *If one-way functions exists, then there is a* P*-samplable probability distribution μ which is not dominated by any polynomial-time computable probability distribution ν.*

The Kolmogorov complexity function $K(.)$ naturally defines a probability distribution on Σ^*: for any string x assign a probability of $2^{-K(x)}$. Kraft's inequality implies that this indeed is a probability distribution. This distribution is called the *universal distribution* and is denoted by $\mathbf{m}$. The universal distribution has many equivalent formulations and has many nice properties. Refer to the textbook by Li and Vitanyi [LV97] for an in-depth study on $\mathbf{m}$. The main drawback of $\mathbf{m}$ is that it is not computable. In this paper we consider a resource-bounded version of the universal distribution.

Definition 12. *The t-time bounded universal distribution, $\mathbf{m}^t$ is given by $\mathbf{m}^t(x) = 2^{-K^t(x)}$.*

3 Basic Computational Depth

In this section we study Basic Computational Depth. As an application, we will show that a Turing machine M runs in average polynomial-time if for all inputs x the Turing machine uses time exponential in the basic computational depth of x. In order to obtain this characterization, we make use of the prefix-free Kolmogorov measure.

Definition 13 (Basic Computational Depth). *Let t be a constructible time bound. For any string $x \in \{0,1\}^*$,*

$$bcd^t(x) = K^t(x) - K(x).$$

Logically deep strings are not easy to identify, but can be constructed by diagonalization in time larger than 2^t for depth t [Ben88]. 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 5. *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*

$$bcd^{2^n}(x) \geq (1-\epsilon)n - c\log n.$$

Proof. Consider the set A consisting of all strings x of length n for which there exists a program p of length $\leq n-2$ such that $U(p)$ outputs x in at most 2^n steps. Since the number of programs of length at most $n-2$ is less than 2^{n-1}, we have $|A| < 2^{n-1}$. Let B denote $\{0,1\}^n \setminus A$. Hence $|B| > 2^{n-1}$ and for any $0 < \epsilon < 1$, there are $> 2^{\epsilon n}$ strings in B. Let D be the lexicographically first $2^{\epsilon n}$ strings in B. Since D is computable and any $x \in D$ can be specified by ϵn bits to describe its position in the lexicographic order in D, we have that for every $x \in D$, $K(x) \leq \epsilon n + O(\log n)$. We also have that for every $x \in D$, $K^{2^n}(x) \geq n-1$ since every program p of size $\leq n-2$ such that $U(p)$ outputs x must run for at least 2^n steps. It follows that for any $x \in D$, $bcd^{2^n}(x) \geq (1-\epsilon)n - c\log n$ for some constant c. $\diamond$

We now develop the application of basic computational depth to capture average-case complexity.

Levin gives a clean definition of *Average Polynomial Time* for a given language L and a distribution μ. Some languages may remain hard in the worst case but can be solved in Average Polynomial Time for all reasonable distributions. We give a crisp formulation of such languages using basic computational depth.

We have two results that hold for every language L.

1. If (L,μ) is in Average Polynomial Time for all P-samplable distributions μ then there exists a Turing machine M computing L and a polynomial p such that for all x, the running time of $M(x)$ is bounded by $2^{O(bcd^p(x)+\log|x|)}$.
2. If there exists a Turing machine M and a polynomial p such that M computes L and for all inputs x, the running time of $M(x)$ is bounded by $2^{O(bcd^p(x)+\log|x|)}$, then (L,μ) is in Average Polynomial Time for all P-computable distributions.

We do not get an exact characterization from these results. The first result requires P-samplable distributions and the second holds only for the smaller class of P-computable distributions. However, we can get an exact characterization by considering the time-bounded universal distribution $\mathbf{m}^t$. We show that the following are equivalent for every language L and every polynomial p:

– $(L, \mathbf{m}^p)$ is in Average Polynomial Time.
– There is some Turing machine M computing L such that for all inputs x the running time of M is bounded by $2^{O(bcd^p(x)+\log|x|)}$.

This exact characterization can be used to prove both (1) and (2).

Theorem 6. *Let T be a constructible time bound. Then for any time constructible t, the following statements are equivalent.*

1. $T(x) \in 2^{O(bcd^t(x)+\log|x|)}$.
2. *T is polynomial on $\mathbf{m}^t$-average.*

Proof. $(1 \Rightarrow 2)$. We will show that the statement 1 implies that $T(x)$ is polynomial on $\mathbf{m}^t$-average. Let $T(x) \in 2^{O(bcd^t(x)+\log|x|)}$. Because of the closure properties of functions which are polynomial on average, it is enough to show that the function $T'(x) = 2^{bcd^t(x)}$ is polynomial on $\mathbf{m}^t$-average. This essentially follows from the definitions and Kraft's inequality. The details are as follows. Consider the sum

$$\begin{aligned}\sum_{x\in\Sigma^*}\frac{T'(x)}{|x|}\mathbf{m}^t(x) &= \sum_{x\in\Sigma^*}\frac{2^{bcd^t(x)}}{|x|}2^{-K^t(x)}\\ &= \sum_{x\in\Sigma^*}\frac{2^{K^t(x)-K(x)}}{|x|}2^{-K^t(x)}\\ &= \sum_{x\in\Sigma^*}\frac{2^{-K(x)}}{|x|} \quad < \quad \sum_{x\in\Sigma^*}2^{-K(x)} \quad < \quad 1\end{aligned}$$

The last inequality is the Kraft's inequality.

$(2 \Rightarrow 1)$ Let $T(x)$ be a time constructible function which is polynomial on $\mathbf{m}^t$-average. Then for some $\epsilon > 0$ we have

$$\sum_{x\in\Sigma^*}\frac{T(x)^\epsilon}{|x|}\mathbf{m}^t(x) < 1$$

Define $S_{i,j,n} = \{x \in \Sigma^n | 2^i \le T(x) < 2^{i+1} \text{ and } K^t(x) = j\}$. Let 2^r be the approximate size of $S_{i,j,n}$. Then the Kolmogorov complexity of elements in $S_{i,j,n}$ is r up to an additive $\log n$ factor. The following claim states this fact more formally.

Claim. For $i, j \le n^2$, let $2^r \le |S_{i,j,n}| < 2^{r+1}$. Then for any $x \in S_{i,j,n}$, $K(x) \le r + O(\log n)$.

Consider the above sum restricted to elements in $S_{i,j,n}$. Then we have

$$\sum_{x\in S_{i,j,n}}\frac{T(x)^\epsilon}{|x|}\mathbf{m}^t(x) < 1$$

$T(x) \geq 2^i$, $\mathbf{m}^t(x) = 2^{-j}$ and there are at least 2^r elements in the above sum. Hence the above sum is lower-bounded by the expression $\frac{2^r \cdot 2^{i\epsilon} \cdot 2^{-j}}{|x|^c}$ for some constant c. This gives us

$$\begin{aligned} 1 &> \sum_{x \in S_{i,j,n}} \frac{T(x)^\epsilon}{|x|} \mathbf{m}^t(x) \\ &\geq \frac{2^r \cdot 2^{i\epsilon} \cdot 2^{-j}}{|x|^c} = 2^{i\epsilon + r - j - c\log n} \end{aligned}$$

That is $i\epsilon + r - j - c\log n < 1$. From the *Claim*, it follows that there is a constant d, such that for all $x \in S_{i,j,n}$, $i\epsilon \leq bcd^t(x) + d\log|x|$. Hence $T(x) \leq 2^{i+1} \leq 2^{\frac{d}{\epsilon}(bcd^t(x) + \log|x|)}$. ◇

The above theorem has an interesting connection to a result due to Li and Vitányi [LV92] connecting the average-case complexity and the worst-case complexity. Li and Vitányi [LV92] showed that when the inputs to any algorithm are distributed according to the universal distribution, the algorithm's average case complexity is of the same order of magnitude as its worst case complexity. Rephrasing this result in the setting of average polynomial time we can make the following statement.

Theorem 7 (Li-Vitányi). *Let T be a constructible time bound. The following statements are equivalent*

1. *$T(x)$ is bounded by a polynomial in $|x|$.*
2. *T is polynomial on $\mathbf{m}$-average.*

Theorem 6 could be viewed as a time-bounded version of Li and Vitányi's result. In Theorem 6, as $t \to \infty$, K^t approaches K. So bcd^t approaches 0 and $\mathbf{m^t}$ approaches $\mathbf{m}$. Hence Theorem 6 is a generalization of Li and Vitányi's theorem. This directly addresses the issue raised by Miltersen [Mil93] of relating a time-bounded version of the above theorem with Levin's average-case complexity.

We will use Theorem 6 to prove our results on average polynomial time. We need the following domination property of time bounded universal distributions.

Theorem 8 ([LV97]). $\mathbf{m}^t$ *dominates any t/n-time computable distribution.*

Proof. Let μ be a t/n-time computable distribution and let μ^* denote the distribution of μ. We will show that for any $x \in \Sigma^n$, $K^t(x) \leq -\log(\mu(x)) + C_\mu$ for a constant C_μ which depends on μ. Let $B_i = \{x \in \Sigma^n | 2^{-(i+1)} \leq \mu(x) < 2^{-i}\}$. Since for any x in B_i, $\mu(x) \geq 2^{-(i+1)}$, we have that $|B_i| \leq 2^i$. Consider the real interval $[0,1]$. Divide it into intervals of size 2^{-i}. Since $\mu(x) \geq 2^{-i}$, we have for any $j, 0 \leq j \leq 2^i$, the j^{th} interval $[j2^{-i}, (j+1)2^{-i}]$ will have at most one $x \in B_i$ such that $\mu(x) \in [j2^{-i}, (j+1)2^{-i}]$. Since μ is t/n-computable, for any $x \in B_i$, given j, we can do a binary search to output the unique x satisfying $\mu(x) \in [j2^{-i}, (j+1)2^{-i}]$. This involves computing μ^* correct up to $2^{-(i+1)}$. So the total running time of the process will be bounded by $O((t/n)n)$. Hence we have the theorem. $\diamond$

In the proof of Theorem 8, $\mathbf{m}^t$ very strongly dominates t/n-time computable distributions, in the sense that $\mathbf{m}^t(x) \geq \frac{1}{2^{C_\mu}}\mu(x)$. The definition of domination that we follow only needs $\mathbf{m}^t$ to dominate μ within a polynomial.

Corollary 1. *Let M be a deterministic Turing machine whose running time is bounded by $2^{O(bcd^t(x)+\log|x|)}$, for some polynomial t. Then for any t/n-computable distribution μ, the pair $(L(M), \mu)$ is in Avg-*P*.*

Proof. Let M be a Turing machine and let $L(M)$ denote the language accepted by M. Let T_M denote its running time. If $T_M(x) \in 2^{O(bcd^t(x)+\log|x|)}$ then from the implication ($1 \Rightarrow 2$) of Theorem 6 and Theorem 8 we have that $(L(M), \mu)$ is in Avg-P for any μ which is computable in time t/n. $\diamond$

Hence a sufficient condition for a language L (accepted by M) to be in Avg-P with respect to all polynomial-time computable distributions is that the running time of M is bounded by an exponential in bcd^t, for all polynomials t. An obvious question that arises is whether this condition is necessary. We partially answer this question.

We show that if (L, μ) is in Average Polynomial Time for all P-samplable distributions μ then there exists a Turing machine M computing L and a polynomial p such that for all x, the running time of $M(x)$ is bounded by an exponential in the depth. We need the following polynomial time samplable distribution that dominates $\mathbf{m}^t$.

Theorem 9. *For any polynomial t, there is a* P*-samplable distribution μ which dominates $\mathbf{m}^t$.*

Proof. We will define a samplable distribution μ_t by prescribing a sampling algorithm for μ_t as follows. Let U be the universal machine.

Sample $n \in \mathbf{N}$ with probability $\frac{1}{n^2}$
Sample $1 \leq j \leq n$ with probability $1/n$
Sample uniformally $y \in \Sigma^j$
Run $U(y)$ for t steps. If U stops and outputs a string $x \in \Sigma^n$, output x.

For any string x of length n, $K^t(x) \leq n$. Hence it is clear that the probability that x is at least $\frac{1}{n^3}2^{-K^t(x)}$. $\diamond$

Corollary 2. *Let M be a machine which runs in time T_M. Suppose for all distributions μ that are* P*-samplable, T_M is polynomial on μ-average, then $T_M(x) \in 2^{O(bcd^t(x)+\log|x|)}$, for some polynomial t.*

Proof. ¿From Theorem 9 if a machine M runs in time polynomial on average for all P-samplable distributions, it also runs in time polynomial on average with respect to $\mathbf{m}^t$. From Theorem 6 it follows that M runs in time $2^{O(bcd^t(x)+\log|x|)}$. $\diamond$

It is natural to ask whether there are polynomial-time computable distributions dominating $\mathbf{m}^t$. This will improve the above corollary from samplable distributions to computable distributions. Schuler [Sch99] showed that if such a distribution exists then no polynomially secure pseudo-random generators exists. Hence it is unlikely that there are polynomial-time computable distributions dominating universal distributions.

Theorem 10 (Schuler). *If there exists a polynomial time computable distribution that dominates $\mathbf{m}^t$ then pseudo-random generators do not exist.*

4 Sublinear-time 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 14. *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 14 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 15. *Let t be a time-bound and x a string.*

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

This definition is essentially equivalent to Definition 1 for superlinear t.

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

Definition 16. *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 17. *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 11. *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 11 we need the following result of Nisan and Wigderson [NW94]:

Lemma 2 (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 11. 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 $C^{n^{jk}}(z) - C(z) \leq n^{jk}$. Let $\ell \doteq C^{n^{jk}}(z)$, which gives us $C(z) \geq \ell - n^{jk}$. Note that $\ell \leq |z| < 2^{n^j+1}$.

By Definition 15 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, $1 \leq i \leq N$. Note that $C(z) \leq C(\langle p, r\rangle) + O(\log n)$.

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

- $|q| + |s| = \ell$,
- For $1 \leq i \leq N$, $U^s(q,i)$ 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

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

This gives us $|T| \ge 2^\ell/2^{n^c}$ 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 2. We have

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

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

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

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 [KL80] 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. *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 [BDG86] showed the following characterization of $PSPACE$/poly.

Theorem 12. *$A \in PSPACE/\mathrm{poly}$ 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 15 handles these issues well.

Corollary 4. *Let C be computable. C is in P/poly iff C is shallow.*

5 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 18. *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 [FK96] 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 13 (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*

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

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, $p(|f(x)|) \ge |x|$ for all x.

We will need the following lemma.

Lemma 3. *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.

For the second step we can apply Theorem 2 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 14. *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

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

Lemma 3 tells us that for any polynomial p_2 there exists a polynomial q_2 such that

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

for $m > 2$ and

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

if f is one-to-one.

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

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 15. *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 3 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. ◇

6 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 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 3 that we use in the proof of Theorem 15. 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 an 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 [Pag99]).

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.

References

[AFvM01] L. Antunes, L. Fortnow, and D. van Melkebeek. Computational depth. In *Proceedings of the 16th IEEE Conference on Computational Complexity*, pages 266–273, 2001.

[AFV03] L. Antunes, L. Fortnow, and N. V. Vinodchandran. Using depth to capture average-case complexity. In *14th International Symposium on Fundamentals of Computation Theory*, volume 2751 of *Lecture Notes in Computer Science*, pages 303-310. Springer, Berlin, 2003.

[BCGL92] S. Ben-David, B. Chor, O. Goldreich and M. Luby. On the theory of average case complexity. *J. Computer System Sci.*, 44(2):193–219, 1992.

[BDG86] 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.

[BDG95] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1995.

[Ben88] 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.

[BF97] 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.

[BG81] 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.

[BLM00] H. Buhrman, S. Laplante, and P. Bro Miltersen. New bounds for the language compression problem. In *Proceedings of the 15th IEEE Conference on Computational Complexity*, New York, 2000. IEEE.

[FK96] L. Fortnow and M. Kummer. On resource-bounded instance complexity. *Theoretical Computer Science*, 161:123–140, 1996.

[FL98] 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.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, August 1999.

[KL80] 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.

[Lev73] L. A. Levin. Universal search problems. *Problems Inform. Transmission*, 9:265–266, 1973.

[Lev86] L. A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[Lev84] L. A. Levin. Randomness conservation inequalities: information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[LV92] M. Li and P. Vitanyi. Average case complexity under the universal distribution equals worst-case complexity. *Information Processing Letters*, 42(3):145–149, May 1992.

[LV97] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2nd edition, 1997.

[Mil93] P. Bro Miltersen. The complexity of malign measures. In *SIAM Journal on Computing*, 22(1):147–156, 1993.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[Pag99] 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.

[Sch99] R. Schuler. Universal distributions and time-bounded Kolmogorov complexity. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 434—443, 1999.

[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.

[Tas96] 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.

[Wan97] J. Wang. Average-case computational complexity theory. In *Alan L. Selman, Editor, Complexity Theory Retrospective*, volume 2. 1997.