Kolmogorov Complexity and Computational Complexity*

Lance Fortnow
University of Chicago

Abstract

We describe the properties of various notions of time-bounded Kolmogorov complexity and other connections between Kolmogorov complexity and computational complexity.

1 Introduction

When one thinks of using Kolmogorov complexity for computational complexity one usually thinks about the incompressibility method. Here one proves a statement by taking some Kolmogorovly random object and assuming the statement is false shows a short description for that object contradicting its randomness. We have many examples of this technique directly using Kolmogorov complexity (see [LV97, Chapter 6]) or under the guise of a counting argument (for example see [Raz95, Appendix E.4]) or an entropy argument (for example [JKS03]).

We will not look at the incompressibility method in this survey, rather focusing on several other areas where Kolmogorov Complexity and Computational Complexity collide.

- We catalog many of the time-bounded variants of Kolmogorov complexity.
- We look at *instance complexity* that allows us to look at the complexity of a string in relation to a set and how it compares to time-bounded traditional and distinguishing Kolmogorov complexity.
- We show how to use time-bounded Kolmogorov complexity to characterize the sizes of efficiently computable sets.
- We show the amazing power of random strings that can help us compute difficult sets as well as derandomize.
- We look at Levin's universal distributions both enumerable and time-bounded and show its relationship to polynomial-time on average problems.
- We also look at Levin's universal search from the Kolmogorov perspective. This idea gives an optimal (up to horrendously large constants) algorithm for problems where we can efficiently check the solution.
- Finally we show how one can use notions of Kolmogorov complexity to characterize some complexity classes, in particular the P-printable sets and P/poly.

Kolmogorov complexity gives us a framework that often help us understand not only computation but efficient computation and often allows us to put quite different and complex concepts in a common framework.

*Dedicated to the memory of Andrei Kolmogorov in recognition of the 100th anniversary of his birth on April 25, 1903.

2 Kolmogorov Complexity and its Resource-Bounded Variations

We assume a basic knowledge of computational complexity such as found in Homer and Selman [HS01]. Information about complexity classes can be found in The Complexity Zoo [Aar]. We define all the notions of Kolmogorov complexity needed in this paper and we recommend Li and Vitányi [LV97] for an introduction and in-depth treatment of the area. We follow much of their notation in this survey.

Fix a nice Turing universal machine U. We define the Kolmogorov function, $\mathrm{C}(x)$ as the length of the smallest program generating x.

Definition 2.1

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

The Kolmogorov function has a beautiful theory that we will for the most part ignore in this survey though we list a few results that prove very useful in complexity.

Theorem 2.2

1. *The choice of the universal machine only affects* $\mathrm{C}(x)$ *by a constant additive factor.*
2. *For every* x, $\mathrm{C}(x) \leq |x| + c$.
3. *For every* n, *there is some* x *of length* n *such that* $\mathrm{C}(x) \geq n$.
4. *For at least a* $1 - 2^{-k}$ *fraction of the* x *of length* n, $\mathrm{C}(x) \geq n - k$.

Of course this definition of Kolmogorov complexity makes any complexity theorist cringe: what good is a small program x if it takes the life of the universe to produce x. We wish to give resource-bounded variant to Kolmogorov complexity but many such definitions exist. These distinctions usually do not make significant different in the standard Kolmogorov complexity but for resource-bounded they can make a dramatic difference and have very different applications.

The first definition is perhaps the most natural as it just limits the time to create the string.

Definition 2.3

$$\mathrm{C}^t(x) = \min_p\{|p| : U(p) \text{ outputs } x \text{ in } t(|x|) \text{ steps}\}$$

Typically in complexity we measure the amount of time of a program as a function of its input. Here we measure time as a function of the output. Why? Consider $x = 0^n$ and a program p of length $m = O(\log n)$ that just outputs n zeros. It makes more sense to consider p as running in linear time as opposed to exponential (in m) time.

Instead of producing the string, we can also measure the size of the smallest program to distinguish the string.

Definition 2.4 (Sisper [Sip83])

$$\mathrm{CD}^t(x) = \min_p \left\{ |p| \;:\; \begin{array}{l} (1)\ U(p,x) \text{ accepts.} \\ (2)\ U(p,z) \text{ rejects for all } z \neq x. \\ (3)\ U(p,z) \text{ runs in at most } t(|z|) \text{ steps for all } z \in \Sigma^*. \end{array} \right\}$$

Reminiscent of the P versus NP problem, we generally believe it easier to determine whether a string has a property than finding such such a string.

Buhrman, Fortnow and Laplante [BFL02] develop a nondeterministic version CND^t with the same definition except we allow U to be nondeterministic.

Levin [Lev73b] has a variation that takes the time as part of the Kolmogorov measure.

Definition 2.5 (Levin)

$$\mathrm{Ct}(x) = \min_p\{|p| + \log t : U(p) = x \text{ in } t \text{ steps.}\}$$

Levin uses the logarithm of the time. Allender [All01] considers the entire time. This definition will often focus on sublinear time so we need to modify how U produces the string.

Definition 2.6 (Allender)

$$\mathrm{CT}(x) = \min_p\{|p| + t : \text{ for all } i,\ 1 \leq i \leq |x|,\ U(p,i) = \text{ the } i\text{th bit of } x \text{ in } t \text{ steps.}\}$$

Usually Kolmogorov Complexity focuses solely on strings. Instance complexity focuses on strings relative to a set.

Instance complexity was introduced by Orponen, Ko, Schöning and Watanabe [OKSW94]. Intuitively, the t-bounded instance complexity of x with respect to A is the length of the shortest program which runs in time $t(n)$, correctly decides whether x is in A, and does not make mistakes on any other input (where it is allowed to output $\perp$ for "don't know").

Definition 2.7 *For any time bound t the t-bounded instance complexity of $x \in \Sigma^*$ with respect to A is defined as*

$$\mathrm{ic}^t(x : A) = \min_p \left\{ |p| \ : \begin{array}{l} (1)\ \textit{For all } y,\ U(p,y) \textit{ runs in time } t(|y|), \\ (2)\ \textit{For all } y,\ \textit{if } U(p,y) = 1 \textit{ then } y \in A, \\ (3)\ \textit{For all } y,\ \textit{if } U(p,y) = 0 \textit{ then } y \notin A, \textit{ and} \\ (3)\ U(p,x) \neq \perp \end{array} \right\}$$

For all of these definitions we can alter them in similar ways:

Prefix-Free: We require the set of legitimate programs to form an easily-computable prefix-free set. This allow us to determine a program that is an initial sequence of some longer, perhaps infinite, string. We follow the Li-Vitányi approach of using "K" instead of "C" for prefix-free complexity, for example $\mathrm{K}^t(x)$.
Advice: We allow the program to have access to additional information. We add the notation "$|y$" to indicate the program p gets the string y as an auxiliary input, for example $\mathrm{C}^t(x|y)$. The running time t is a function of $|y|$ as well as $|x|$.
Oracles: We allow the program p to have access to a set A as an oracle indicated by a superscript "A" as in $\mathrm{C}^{t,A}(x)$. The running time does not depend on A.
Space: We measure the resource as space instead of time, using "s" instead of "t", for example $\mathrm{C}^s(x)$. One can also bound both time and space, i.e., $\mathrm{C}^{t,s}(x)$.

We use bracket notation to define a set of strings with similar Kolmogorov complexity. For example,

- $\mathrm{C}^t[\leq f(n), t(n)]$ is the set of strings x such that $\mathrm{C}^t(x) \leq f(|x|)$.
- $\mathrm{C}^t[\geq f(n), t(n)]$ is the set of strings x such that $\mathrm{C}^t(x) \geq f(|x|)$.

3 Relationships of C, CD and Instance Complexity

Any program that can produce x can also distinguish x.

Theorem 3.1 *For all x,*

$$\mathrm{CD}^t(x) \leq \mathrm{C}^t(x) + O(1).$$

Let us focus on t a polynomial. If P = NP then given a program that distinguishes x, we can find x. Since a program will distinguish a single x, we can use a weaker assumption.

Definition 3.2 Unique-SAT is easy *if there exists a polynomial-time algorithm A that on formula ϕ with exactly one satisfying assignment, $A(\phi)$ outputs that assignment.*

Unique-SAT is easy implies NP = RP [VV86] and P = UP. There exist relativized worlds where Unique-SAT is easy and P $\neq$ NP.

Theorem 3.3 *If Unique-SAT is easy then for all polynomials p there is a polynomial q and a constant c such that for all x,*

$$\mathrm{C}^q(x) \leq \mathrm{CD}^p(x) + c.$$

Fortnow and Kummer [FK96] show a tight connection for a slightly weaker statement.

Theorem 3.4 *The following are equivalent:*

1. *Unique-SAT is easy.*
2. *For all polynomials p there is a polynomial q and a constant c such that for all x and y,*

$$\mathrm{C}^q(x|y) \leq \mathrm{CD}^p(x|y) + c.$$

The instance complexity is bounded by the CD complexity hardwiring the answer for x and answering $\perp$ on all other inputs.

Theorem 3.5 *For all sets A and inputs x,*

$$\mathrm{ic}^t(x : A) \leq \mathrm{CD}^t(x) + c$$

where c is a constant independent of x and A.

Theorem 3.5 is not tight. If A is polynomial-time computable then we need only give a program for A and the polynomial-time instance complexity is bounded by a constant depending only on A. Orponen et. al. [OKSW94] show that a converse also holds.

Theorem 3.6 (Orponen et. al.) *For any set A, A is polynomial-time computable if and only if there exists a polynomial p and a constant c such that for all x,*

$$\mathrm{ic}^p(x : A) \leq c$$

Informally a P-hard instance x for a set A has its instance complexity at least the C^t complexity of A. By Theorems 3.1 and 3.5 this can only happen for inputs x where $\mathrm{CD}(x)$ and $\mathrm{C}(x)$ are basically the same.

Definition 3.7 *A set A has* p-hard instances *if for every polynomial p there exists a polynomial q and a constant c such that for infinitely many x, $ic^q(x : A) \geq C^p(x) - c$.*

Orponen et. al. [OKSW94] and Fortnow and Kummer [FK96] give some examples of sets with P-hard instances.

Theorem 3.8 (Orponen et. al., Fortnow-Kummer) *A recursive set A not in* P *has P-hard instances if at least one of the following holds*

1. *A is a tally set,*
2. *A is* E*-complete,*
3. *A is NP-hard under honest Turing reductions, or*
4. *A is* P*-bi-immune.*

Fortnow and Kummer [FK96] give relativized examples of sets A not in P without hard instances even if we only required the instance complexity to be as great as the CD complexity.

4 Size of Sets

By just printing the string we have $\mathrm{C}(x) \leq |x| + O(1)$ for all x. By a simple counting argument, for some x, this is tight, or there wouldn't be enough programs to print all of the strings. By a similar argument we can get a more general result.

Lemma 4.1 *Let A be a subset of Σ^*. For every n such that $A \cap \Sigma^n$ is not empty, there must exist some x of length n in A such that that $\mathrm{C}(x) \geq \log |A|$.*

We can also prove upper bounds on $\mathrm{C}(x)$ for computably enumerable A.

Lemma 4.2 *Let A be computably enumerable. There is a c such that for every x in A,*

$$\mathrm{C}(x|n) \leq \log |A \cap \Sigma^n| + O(1).$$

What if we now look at polynomial-time computations? Lemma 4.1 still applies but does Lemma 4.2 still hold? It depends on the exact variation of Kolmogorov complexity that we use.

A small set A in P could have very hard to find strings, provably so if UE $\neq$ E, so we can't expect Lemma 4.2 to hold if we look at C^t complexity. The situation changes if we use CD^t complexity.

Theorem 4.3 (Sipser [Sip83]) *For every polynomial-time computable set A there exists a polynomial p and constant c such that for every n, for most r in $\Sigma^{p(n)}$ and every $x \in A^{=n}$,*

$$\mathrm{CD}^p(x|r) \leq \log(|A^{=n}|) + c \log(n)$$

To prove Theorem 4.3, Sipser uses the random string r to generate hash functions so one can describe an element A from where the hash functions maps it to.

Can we remove the need for the random string? Buhrman, Fortnow and Laplante [BFL02] show that we can if we allow a weaker bound by using some extra information to encode the hash function.

Theorem 4.4 (Buhrman-Fortnow-Laplante) *For every polynomial-time computable set A there exists a polynomial p and constant c such that for every n, for every $x \in A^{=n}$,*

$$\mathrm{CD}^p(x) \leq 2 \log(|A^{=n}|) + c \log(n).$$

Buhrman, Laplante and Miltersen [BLM00] show that Theorem 4.4 tight relative to an oracle.

Theorem 4.5 (Buhrman-Laplante-Miltersen) *For every polynomial p and sufficiently large n there exists a set of strings $A \subseteq \{0,1\}^n$ containing more than $2^{n/50}$ strings such that there is an x in A with*

$$\mathrm{CD}^{p,A}(x) \geq 2\log(\|A^{=n}\|) - O(1)$$

Using various results on extractors [Ta-96, LRVW03], Buhrman, Fortnow and Laplante [BFL02] improve Theorem 4.4 for most strings in A.

Theorem 4.6 (Buhrman-Fortnow-Laplante) *For any set A in P, for any constants $\alpha, \varepsilon > 0$, there is a polynomial p such that for all n and for all but an ε fraction of the $x \in A^{=n}$,*

$$\mathrm{CD}^p(x) \leq \min\{\log\|A^{=n}\| + \log^{O(1)}(n), (1+\alpha)\log\|A^{=n}\| + O(\log(n))\}.$$

5 The Power of Random Strings

Define $R = \mathrm{C}[\geq n/2]$, the set of strings with Kolmogorov complexity at least half their length. Every r.e. set can be reduced to R. What can we say about R in terms of computational complexity?

Allender, Buhrman, Koucký, van Melkebeek and Ronneburger [ABK$^+$02] show that PSPACE is in P^R and EXP is in NP^R. Allender, Buhrman and Koucký [ABK04] show how to use the techniques of Allender et. al. [ABK$^+$02] to show that even NEXP is in NP^R. In this case one gets an exponential speed-up by having access to the set of random strings.

Can we characterize PSPACE in terms of P^R? Not directly since R is not computable. Perhaps PSPACE is the set of all recursive sets in P^R. Allender, Buhrman and Koucký [ABK04] show that the choice of the universal machine could affect his answer. They conjecture that a recursive set is in PSPACE if and only if is in P^R no matter what universal machine is chosen.

Consider the set $R_{\mathrm{Kt}} = \mathrm{Kt}[\geq n/2]$. Allender et. al. [ABK$^+$02] show that R_{Kt} sits in EXP and every language in EXP reduces to R_{Kt} through *non-uniform* polynomial-time Turing reductions.

The techniques used by these papers build on derandomization tools first developed by Nisan and Wigderson [NW94]. Nisan and Wigderson show how to create pseudorandom generators based on hardness assumptions. Using strings from R allows one to break these generators and thus the hardness assumptions.

Allender [All01] first realized one can also use random strings for derandomization. There exists a polynomial-time algorithm A and a constant k such that for any circuit C that accepts at least half its inputs and any string x such that $\mathrm{CT}(x) \geq |C|^k$, $A(x, |C|)$ will output a list of strings $y_1, \ldots, y_{|x|^k}$ such that $C(y_i)$ will accept for some y_i. The proof uses the fact that x describes a function on $\log|x|$ bits with exponential circuit complexity and using Impagliazzo-Wigderson [IW97] and Umans [Uma03] one can derandomize using such functions.

6 Universal Distributions

Consider the function $\mathbf{m}(x) = 2^{-K(x)}$. Since the set of programs is prefix-free, we have by Kraft's inequality that $\sum_x \mathbf{m}(x) \leq 1$. Since $\mathbf{m}(x) > 0$ for all x, $\mathbf{m}$ is a semimeasure. $\mathbf{m}$ is also enumerable, i.e., there is a recursive function $f(x,k)$ such that f is monotonically increasing and $\lim_k f(x,k) = \mathbf{m}(x)$. Levin [Lev74] showed that $\mathbf{m}$ is universal for this class.

Theorem 6.1 (Levin) *For any $\tau : \Sigma^* \to \Re$ that is an enumerable semimeasure there is a constant c such that for all x,*

$$\mathbf{m}(x) \geq \tau(x)/c.$$

Levin [Lev86] defined a notion of running in polynomial-time on average relative to a distribution.

Definition 6.2 (Levin) *An algorithm runs in time* polynomial on average *with respect to a distribution τ if there exists a constant k such that*

$$\sum_{x \in \Sigma^*} \frac{t^{1/k}(x)}{|x|} \tau(x) < 1$$

where $t(x)$ is the running time the algorithm uses on input x.

Li and Vitányi [LV92] show that under the universal distribution $\mathbf{m}$, polynomial-time on average is the same as polynomial-time in the worst case.

Theorem 6.3 (Li-Vitányi) *If A is an algorithm that runs in time polynomial on average relative to $\mathbf{m}$ then there exists a polynomial p such that the running time of A on input x is bounded by $p(|x|)$ for all x.*

We can consider time-bounded versions of $\mathbf{m}$ by defining $\mathbf{m}^t(x) = 2^{-K^t(x)}$ also a semimeasure. The measure $\mathbf{m}^t$ does not quite have the time-bounded enumerable or universal properties that we would like.

Definition 6.4 *A distribution $\tau(x)$ is* polynomial-time computable *if the cumulative probability function $\tau^*(x) = \sum_{y \leq x} \tau(y)$ is computable in polynomial time. The distribution τ is P-samplable if there is a probabilistic algorithm that on no input will produce a string x with probability $\tau(x)$ and halt in time bounded by a fixed polynomial in its output size.*

Every polynomial-time computable distribution is P-samplable. We would like to say that $\mathbf{m}^t(x)$ is in some sense universal for polynomial-time computable distributions or P-samplable distributions but we can only show one direction for each.

Theorem 6.5

1. *For any polynomial-time computable distribution τ there exists a polynomial p and a constant c such that*
$$\mathbf{m}^p(x) \geq \tau(x)/c.$$
2. *For all polynomials p there exists a P-samplable distribution τ and a constant c such that*
$$\tau(x) \geq \mathbf{m}^p(x)/c.$$

Schuler [Sch99] shows that a tighter characterization is unlikely.

Theorem 6.6 (Schuler) *If pseudorandom generators exist then there is a polynomial p such that for all polynomial-time computable distributions τ and constants c there is an x such that*

$$\tau(x) < \mathbf{m}^p(x)/c.$$

Antunes, Fortnow and Vinodchandran [AFV03] give a version of Theorem 6.3 for $\mathbf{m}^t$.

Theorem 6.7 (Antunes-Fortnow-Vinodchandran) *For all time-constructible functions t, an algorithm runs in time polynomial on average with respect to $\mathbf{m}^t$ if and only if the algorithm runs in time $2^{O(C^t(x) - C(x) + log|x|)}$ for all inputs x.*

Note that as t gets large, Theorem 6.7 approaches Theorem 6.3.

7 Universal Search

Levin [Lev73a] has a well-known result producing a factoring algorithm within a constant factor of optimal. Let us consider this work in the context of Kolmogorov complexity.

Lemma 7.1 (Levin) *There exists an algorithm M that enumerates every string, where string x will be outputted in time $2^{\mathrm{Kt}(x)}$ steps.*

The algorithm simulates each program p for a $2^{-|p|}$ fraction of the time. If p produces x in t steps the algorithm will take $t2^{|p|}$ steps which is $2^{\mathrm{Kt}(x)}$ if p is the shortest program that generates x.

Levin calls the value $O(2^{\mathrm{Kt}}(x))$ the *age* of string x. If we give the algorithm an input string y, then Lemma 7.1 holds with the stronger time bound $O(2^{\mathrm{Kt}(x|y)})$.

Lemma 7.2 *For any total recursive algorithm A, $\mathrm{Kt}(A(x)|x) \leq \log t(x) + c_A$ where $t(x)$ is the running time of algorithm $A(x)$ and c_A is a constant depending only on A.*

Combining Lemmas 7.1 and 7.2 we get the following optimality result.

Theorem 7.3 *For any total recursive algorithm A, $M(x)$ will enumerate $A(x)$ in time $c_A t(x)$ where $t(x)$ is the running time of $A(x)$ and c_A is a constant depending only on A.*

The algorithm $M(x)$ will enumerate many values including $A(x)$ and we may not know which value is correct. We can eliminate this problem if we have an an algorithm $B(x, y)$ that accepts exactly when $y = A(x)$.

Theorem 7.4 *Given M and B as described above, there exists an algorithm D that on x produces the single y such that $B(x, y)$ accepts. Let A be any total recursive algorithm such that $B(x, A(x))$ accepts. $D(x)$ will produce $A(x)$ in time $c_A r(x)t(x)$ where $r(x)$ is the running time of B on $(x, A(x))$, $t(x)$ is the running time of A on input x and c_A is a constant depending only on A.*

Note that the algorithm D is independent of the description of A.

Consider the example where $B(x, y)$ accepts if y is a list of the prime factors of x. We can compute B in polynomial time [AKS02]. Thus the algorithm D will run within a polynomial factor of any algorithm that factors A. In particular if factoring is computable in polynomial time then D will factor in polynomial time even if we don't know the original factoring algorithm.

Of course we must note that c_A will, in most cases, be so large as to make Theorem 7.4 no more than a theoretical oddity.

8 Kolmogorov Characterization of Complexity Classes

We can often get characterizations of complexity classes using Kolmogorov complexity.

Definition 8.1 *A language L is P-printable if there exists a polynomial time computable function f such that $f(1^n)$ enumerates exactly the strings in L of length n.*

Every P-printable set is sparse, i.e., there is at most a polynomial number of strings at every length.

Lemma 8.2 *For every k, $\mathrm{C}[k \log n, n^k]$ is P-printable.*

We just simulate all of the short programs.

Lemma 8.3 *For every P-printable set L, $L \subseteq \mathrm{C}[k \log n, n^k]$ for some k.*

We can describe any string x in L enumerated by f by a description of f and the index of x in the list of strings enumerated by f.

Since the intersection of a set in P and a P-printable set is P-printable we get the following characterizations of P-printable sets.

Theorem 8.4 *For any set L in* P *the following are equivalent.*

1. *L is P-printable,*
2. *for some k, L is a subset of $\mathrm{C}[k \log n, n^k]$, and*
3. *for some k, $\mathrm{Ct}(x) \leq k \log n$ for all x in L.*

Now let us turn to the class P/poly, nonuniform polynomial-time.

Definition 8.5 *A language L is in P/poly if there exists a sparse set S such that L is computable in polynomial-time with access to an oracle for S.*

We will give some characterizations of P/poly using Kolmogorov complexity but first we need to define the characteristic sequence of a set A.

Definition 8.6 *Consider the infinite list of strings $\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$ that enumerates Σ^*. A characteristic sequence of a set A, σ_A is an infinite binary sequence whose ith bit is 1 if the ith string of Σ^* is in A. The finite sequence σ_A^n is the characteristic sequence of A through all of the strings of length up to n.*

Note $|\sigma_A^n| = 2^{n+1} - 1$.

We first characterize P/poly by CT complexity.

Theorem 8.7 *A language A is in P/poly if and only if there is a constant c such that*

$$\mathrm{CT}(\sigma_A^n) \leq n^c$$

for all n.

Interestingly enough, strings with very high Kolmogorov complexity can also characterize P/poly.

Definition 8.8 *A set A is* random *if $\mathrm{C}(\sigma_A^n) \geq |\sigma_A^n|$ for infinitely many n.*

By Theorem 8.7, A is not in P/poly. But everything useful we can derive from A is in P/poly.

If B is recursive and B is in P^A for some random A then B is in BPP and thus in P/poly. Antunes, Fortnow and van Melkebeek [AFvM01] generalize this idea.

Theorem 8.9 (Antunes-Fortnow-van Melkebeek) *The following are equivalent for all recursive languages L.*

1. *L is in P/poly, and*
2. *There exists a set A and a constant k such that L is in P^A and*

$$\mathrm{CT}(\sigma_A^n) \leq \mathrm{C}(\sigma_A^n) + n^k$$

for all n.

9 Conclusions

In this survey we have shown a number of interesting connections between Kolmogorov Complexity and Computational Complexity. We have only given a flavor of these connections. We have not covered areas like connections to quantum computing, computational depth, minimally sufficient statistics, connections to biology and other sciences and many others.

For those who enjoy Kolmogorov Complexity, think Kolmogorovly! Often a research paper in almost any area becomes more interesting when viewed through a Kolmogorov lens.

May all your programs be short and quick.

Acknowledgments

We thank Eric Allender and Sophie Laplante for helpful discussions on some of the results mentioned in this survey.

References

[Aar] S. Aaronson. The complexity zoo. http://www.cs.berkeley.edu/~aaronson/zoo.html.

[ABK+02] E. Allender, H. Burhman, M. Koucký, D. van Melkbeek, and D. Ronneburger. Power from random strings. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 669–678. IEEE, New York, 2002.

[ABK04] E. Allender, H. Buhrman, and M. Koucký. What can be efficiently reduced to the K-random strings? In *Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science. Springer, Berlin, 2004. To appear.

[AFV03] L. Antunes, L. Fortnow, and 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.

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

[AKS02] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Unpublished manuscript, Indian Institute of Technology Kanpur, 2002.

[All01] E. Allender. When worlds collide: Derandomization, lower bounds, and kolmogorov complexity. In *Proceedings of the 21st Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *Lecture Notes in Computer Science*, pages 1–15. Springer, Berlin, Germany, 2001.

[BFL02] H. Buhrman, L. Fortnow, and S. Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM Journal on Computing*, 31(3):887–905, 2002.

[BLM00] H. Buhrman, S. Laplante, and P. Miltersen. New bounds for the language compression problem. In *Proceedings of the 15th IEEE Conference on Computational Complexity*, pages 126–130. IEEE Computer Society, Los Alamitos, 2000.

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

[HS01] S. Homer and A. Selman. *Computability and Complexity Theory.* Springer, New York, 2001.

[IW97] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on the Theory of Computing*, pages 220–229. ACM, New York, 1997.

[JKS03] T. Jayram, R. Kumar, and D. Sivakumar. Two applications of information complexity. In *Proceedings of the 35th ACM Symposium on the Theory of Computing*, pages 673–682. ACM, New York, 2003.

[Lev73a] L. Levin. Universal search problems. *Problems of Information Transmission*, 9:265–266, 1973.

[Lev73b] L. Levin. Universal'nyĭe perebornyĭe zadachi (Universal search problems: in Russian). *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. Corrected English translation in [Tra84].

[Lev74] L. Levin. Laws of information conservation (non-growth) and aspects of the foundations of probability theory. *Problems of Information Transmission*, 10:206–210, 1974.

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

[LRVW03] C. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors. In *Proceedings of the 35th ACM Symposium on the Theory of Computing*, pages 602–611. ACM, New York, 2003.

[LV92] M. Li and P. Vitányi. 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.* Graduate Texts in Computer Science. Springer, New York, second edition, 1997.

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

[OKSW94] P. Orponen, K. Ko, U. Schöning, and O. Watanabe. Instance complexity. *Journal of the ACM*, 41(1):96–121, 1994.

[Raz95] A. Razborov. Bounded Arithmetic and lower bounds in Boolean complexity. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, volume 13 of *Progress in Computer Science and Applied Logic*, pages 344–386. Birkhäuser, Boston, 1995.

[Sch99] R. Schuler. Universal distributions and time-bounded kolmogorov complexity. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 434–443. Springer, 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. ACM, New York, 1983.

[Ta-96] A. Ta-Shma. On extracting randomness from weak random sources (extended abstract). In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 276–285. ACM, New York, 1996.

[Tra84] R. Trakhtenbrot. A survey of Russian approaches to *Perebor* (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.