Kolmogorov Complexity

*Lance Fortnow**

1. Introduction

010101010101010101010101

100111011101011100100110

110100110010110100101100

Consider the three strings shown above. Although all are 24-bit binary strings and therefore equally likely to represent the result of 24 flips of a fair coin, there is a marked difference in the complexity of *describing* each of the three. The first string can be fully described by stating that it is a 24-bit string with a 1 in position n iff n is odd, the third has the property that position i is a 1 iff there are an odd number of 1's in the binary expansion of position i, but the second string appears not to have a similarly simple description, and thus in order to describe the string we are required to recite its contents verbatim.

What is a description? Fix $\Sigma = \{0,1\}$. Let $f : \Sigma^* \mapsto \Sigma^*$. Then (relative to f), a description of a string σ is simply some τ with $f(\tau) = \sigma$.

Care is needed in using this definition. Without further restrictions, paradoxes are possible (consider the description *"the smallest positive integer not describable in fewer than fifty words"*). We restrict f by requiring that it be *computable,* although not necessarily total, but do not initially concern ourselves with the time-complexity of f.

We are now able to define Kolmogorov Complexity C_f:

Definition 1.1. $C_f(x) = \begin{cases} \min\{|p| : f(p) = x\} & \text{if } x \in \operatorname{ran} f \\ \infty & \text{otherwise} \end{cases}$

It would be better to have a fixed notion of description rather than having to define f each time. Or, indeed, to have a notion of complexity that does not vary according to which f we choose. To some extent this problem is unavoidable, but we can achieve a sort of independence if we use a Universal Turing Machine

*This article was prepared from notes of the author taken by Amy Gale in Kaikoura, January 2000.

(UTM). As is well known, there exists a Turing Machine U such that for all partial computable f, there exists a program p such that for all y, $U(p, y) = f(y)$.

We define a partial computable function g by letting $g(0^{|p|}1py) = U(p, y)$. The following basic fact is not difficult to see, and removes the dependence on f. Thus it allows us to talk about *the* Kolmogorov complexity.

Claim 1.1. *For all partial computable f there exists a constant c such that for all x, $C_g(x) \leq C_f(x) + c$. (In fact, $c = 2|p| + 1$, where p is a program for f.)*

Now define $C(x) = C_g(x)$. As we have seen, this is within a constant of other complexities. We can also extend Claim 1.1 to binary functions. Let g be a binary partial computable function. Then we define the *conditional complexity* $C_g(x|y) = \min\{|p| : g(p, y) = x\}$. Again we can drop the g for a universal function. Note that by this definition, $C(x|\epsilon) = C(x)$, where ϵ is the empty string, since for a universal binary g, $g(x, \epsilon)$ is equivalent to a universal unary function.

The idea is that $C(x)$ gives us a way to describe the randomness of x. Before we turn to some applications of Kolmogorov complexity, here are some easy properties of $C(x)$.

1. $C(x) \leq |x| + c$ for all x. (We can see intuitively that there is a program I that just prints out its input, and that $C(x) \leq C_I(x) + c$.)

2. $C(xx) \leq C(x) + c$ for all x. (We can take a program for x and put it in a loop, increasing the program size by only a constant.)

3. For any partial computable h, $C(h(x)) \leq C(x) + c_h$, where c_h is the length of a description of h.

4. $C(x|y) \leq C(x) + c$. (At worst, y is of no benefit in describing x.)

For the present article, we think of $\langle x, y\rangle$ as the concatenation of x and y. One very desirable property we would like is that

$$C(\langle x, y\rangle) \stackrel{?}{\leq} C(x) + C(y) + c.$$

However, there is a problem here: suppose we had programs p and q such that $g(p) = x$ and $g(q) = y$. The difficulty is that we want to encode p and q together in such a way as to make it possible to extract them both intact at a later point. If we simply encode them as the string pq, we will not know where p ends and q begins. If we add additional elements, for example by encoding as $0^{|p|}1pq$, then a lot of space is wasted. Trying $|p|pq$[1] is still problematic since we don't know where $|p|$ ends. One way is to *self-delimit* the strings by encoding $|p|$ by local doubling with an end point marker. That is, if $|p| = 1010$ then $|p|$ is coded by 1100110001, with 01 being the end marker. Of course such a trick could be repeated (by using $|p'|p'pq$, where $p' = |p|$, and encoding $|p'|$ by local doubling with an end point

[1] In this article, we identify a natural number such as $|p|$ with its binary representation.

marker) etc. This encoding is more or less best possible, and gives the following tight bound on $C(\langle x, y\rangle)$. (In this article, all logs are "$\log_2$".)

$$\begin{aligned} C(\langle x, y\rangle) &\leq C(x) + C(y) + 2\log C(x) + c \\ C(\langle x, y\rangle) &\leq C(x) + C(y) + \log C(x) + 2\log\log C(x) + c \\ C(\langle x, y\rangle) &\leq C(x) + C(y) + \log C(x) + \log\log C(x) + 2\log\log\log C(x) + c \\ &\vdots \end{aligned}$$

This brings us to randomness. The idea is that a string is random if it cannot be compressed. That is, if it has no short description. Using $C(x)$ we can formalize this idea via the following.

Theorem 1.2. *For all n, there exists some x with $|x| = n$ such that $C(x) \geq n$. Such x are called (Kolmogorov)* **random**.

Proof. Suppose not. Then for all x, $C(x) < n$. Thus for all x there exists some p_x such that $g(p_x) = x$ and $|p_x| < n$. Obviously, if $x \neq y$ then $p_x \neq p_y$.

But there are $2^n - 1$ programs of length less than n, and 2^n strings of length n. By the pigeonhole principle, if all strings of length n have a program shorter than n, then there must be some program that produces two different strings. Clearly this is absurd, so it must be the case that at least one string of length n has a program of length at least n. □

2. Some Applications

Aside from the intrinsic interest in the notion of randomness itself, the concept of incompressibility can be used to give alternative proofs of many classical theorems. Here are some examples.

Theorem 2.1. *There are infinitely many primes.*

Proof. Suppose not. Then there are k primes $p_1, p_2, \ldots, p_k$ for some $k \in \mathbb{N}$.

Thus we can take any number $m \in \mathbb{N}$ and write it as a product of these k primes:

$$m = p_1^{e_1} \cdots p_k^{e_k}.$$

Let m be Kolmogorov random and have length n. We can describe m by $e_1, \ldots, e_k$. We claim that this gives a short description of m. First, $e_i \leq \log m$. Thus $|e_i| \leq \log\log m$. Hence, $|\langle e_1, \ldots, e_k\rangle| \leq 2k\log\log m$. Therefore, as $m \leq 2^{n+1}$, $|\langle e_1, \ldots, e_k\rangle| \leq 2k\log(n+1)$, so $C(m) \leq 2k\log(n+1) + c$. For large enough n, this contradicts $C(m) \geq n$, which follows from the fact that m is random. □

Of course the dubious reader could well say that the proof above is more complex than the original one. However, the following result is quite close to the real "prime number theorem" and is definitely easier.

Let p_m be the mth prime. It is reasonable to ask how big p_m is, and Kolmogorov complexity allows us to put a bound on this value.

Let p_i be the largest prime that divides m. Then we can describe m by specifying p_i and $\frac{m}{p_i}$; in fact all we need is i and $\frac{m}{p_i}$ because we can compute p_i given i. For m random, we have the following.

$$\begin{aligned} C(m) &\leq C(\langle i, \frac{m}{p_i} \rangle) \\ &\leq 2\log|i| + |i| + |\frac{m}{p_i}|, \end{aligned}$$

so

$$\log m \leq 2\log\log i + \log i + \log m - \log p_i.$$

Canceling this gives us

$$\begin{aligned} \log p_i &\leq \log i + 2\log\log i \\ p_i &\leq i(\log i)^2. \end{aligned}$$

The classical theorem is that the i-th prime is below $i \log i$, so the above is pretty close.

Interestingly, most strings are "close" to being random.

Theorem 2.2. *For all k and n,*

$$|\{x \in \Sigma^n : C(x) \geq |x| - k\}| \geq 2^n(1 - 2^{-k}).$$

Proof. The number of programs of size less than 2^{n-k} is $2^{n-k} - 1 < 2^{n-k}$, which leaves over $2^n - 2^{n-k} = 2^n(1 - 2^{-k})$ programs of length $n - k$ or greater. □

Thus, for example, 99.9% of all strings x have $C(x) \geq |x| - 10$. The intuition is that we can easily generate a string that is close to random if we are provided with a simple random process, for example a coin that can be tossed.

Randomness has interesting interactions with classical computability theory. Consider the non-random, co-computably enumerable set of strings

$$A = \{x : C(x) \geq \frac{|x|}{2}\}.$$

Here is an easy proof that A is a so-called "immune" set.

Theorem 2.3. *If B is a computably enumerable subset of A then B is finite.*

Proof. Suppose that B is computably enumerable and infinite, and $B \subseteq A$. Consider the function h, where $h(n)$ is the first element to be enumerated into B whose length is n or greater. Then h is total (because otherwise B could not be infinite)

and computable (by dovetailing). We have $h(n) \in B \subseteq A$ and by the definition of A,

$$C(h(n)) \geq \frac{|h(n)|}{2} \geq \frac{n}{2}.$$

But

$$C(h(n)) \leq C(n) + c \leq \log n + c,$$

which gives us a contradiction, because for any c, $\frac{n}{2} > \log n + c$ given a large enough n. □

This theorem has a nice application in logic. We consider what can be proved in a given proof system, for example Peano arithmetic. If we assume the system is complete and sound, then we can either prove or disprove all theorems of the form "x is random". We can define the set

$$B = \{x : \text{ there is a proof that } x \text{ is random}\}.$$

Then $B \subseteq A$, and B is computably enumerable (since we can determine the membership of x in B by searching the proof space). By the above theorem, B must therefore be finite, which we know to be a contradiction. This provides an easy view on Gödel's Incompleteness Theorem. *Almost every random number cannot be proven so!*

3. Runs in Random Strings

Suppose x is random, with $C(x) = |x| = n$. What is the longest run of zeroes in x? The first intuition would be that there are only very short runs of zeroes in a random string. We can show that the longest run can have length of order $\log n$ — if there were many *more* consecutive zeroes, we could compress them. Suppose x is a random string of length n and $x = u0^{2\log n}v$ for some u and v. Then we can fully describe x by u and v, noting we can compute n given $|u| + |v|$. We have

$$\begin{aligned} C(x) &\leq |u| + |v| + \log|u| + 2\log\log|u| + c \\ &\leq n - 2\log n + \log n + 2\log\log n + c. \end{aligned}$$

Hence,

$$C(x) \leq n - \log n + 2\log\log n + c,$$

a contradiction to x's randomness for long enough x. This is called a "cut and paste" argument and is quite typical of arguments using Kolmogorov complexity.

More surprisingly, one can show that x must have relatively long runs of zeros. To show this we need to develop some new machinery.

Recall that

$$C(x|y) = \min\{|p| : g(p, y) = x\},$$

where g is a universal binary partial computable function.

There are two main things to remember:

1. $(\forall n)(\exists x \in \Sigma^n)(C(x) \geq n)$, and
2. $(\forall x)(C(x) \leq |x| + c)$.

That is, for all n there is at least one random string of length n, but no string of length n has complexity greater than n by more than a constant.

We now show a close relationship between the size of a set and the maximum Kolmogorov complexity of a string in that set.

Theorem 3.1.

- *Let A be finite.* $(\forall y \in \Sigma^*)(\exists x \in A)(C(x|y) \geq \log |A|)$.
- *Let $B \subseteq \Sigma^* \times \Sigma^*$ be an infinite computably enumerable set such that the sets of the form $B_y = \{x : \langle x, y\rangle \in B\}$ are finite for all y. Then*

$$(\forall x, y : x \in B_y)(C(x|y) \leq \log |B_y| + c),$$

where c is independent of both x and y.

Proof. The first item follows from a simple counting argument similar to the proofs of Theorems 1.2 and 2.2. For the second item consider the generator program for B. It will enumerate elements of B_y in some order $x_1, \ldots, x_{|B_y|}$. We can describe x by the program for B, y and the i such that $x = x_i$. □

Now suppose a random string x with $n = |x|$ has no runs of $\frac{1}{2}\log n = \log \sqrt{n}$ zeros. Break x into $2n/\log n$ segments of length $\log \sqrt{n}$. Each segment must be one of only $\sqrt{n} - 1$ possibilities, since $0^{\log \sqrt{n}}$ cannot be in any segment. The total number of possibilities for x is at most

$$(\sqrt{n} - 1)^{2n/\log n} = \sqrt{n}^{2n/\log n}(1 - 1/\sqrt{n})^{2n/\log n} \approx 2^n e^{-\frac{2\sqrt{n}}{\log n}}.$$

We can enumerate these strings easily, so by Theorem 3.1, $C(x) \leq n - \Omega(\sqrt{n}/\log n)$, contradicting the fact that x is random.

Theorem 3.1 also allows us to consider Kolmogorov complexity over objects besides strings. For example, let B_n be the set of permutations on $\{1, \ldots, n\}$ encoded into strings in some natural way. Then

$$(\exists x \in B_n)(C(x|n) \geq \log |B_n| = n \log n)$$

and

$$(\forall x \in B_n)(C(x|n) \leq \log |B_n| + c).$$

4. Symmetry of Information

One important theorem is the following. We know that

$$C(\langle x, y\rangle) \leq C(y|x) + C(x) + O(\log n),$$

where $n = \max\{|x|, |y|\}$. Surprisingly, this inequality is essentially tight.

Theorem 4.1.

$$C(y|x) + C(x) \leq C(\langle x, y\rangle) + O(\log n),$$

where $n = \max\{|x|, |y|\}$.

Proof. Define the sets

$$A = \{\langle u, v\rangle : C(\langle u, v\rangle) \leq C(\langle x, y\rangle)\}$$

and

$$A_u = \{v : \langle u, v\rangle \in A\}.$$

A is finite and recursively enumerable given $\langle x, y\rangle$, and likewise A_u for all u. We take $e \in \mathbb{N}$ such that $2^{e+1} > |A_x| \geq 2^e$. Then

$$C(y|x) \quad \leq \quad \log |A_x| + O(1) = e + O(1).$$

Now consider the set $B = \{u : |A_u| \geq 2^e\}$. It is clear that $x \in B$. Now, what is $|B|$?

$$|B| \leq \frac{|A|}{2^e} \leq \frac{2^{C(\langle x,y\rangle)}}{2^e}.$$

This is independent of the pairing function used, provided the function is injective. Note that $|\bigcup_u A_u| \leq |A|$.

We now have

$$\begin{aligned} C(x) &\leq |e| + \log \frac{2^{C(\langle x,y\rangle)}}{2^e} + 2\log|e| \\ &\leq C(\langle x, y\rangle) - e + O(\log n), \end{aligned}$$

and thus

$$C(x) + C(y|x) \leq C(\langle x, y\rangle) + O(\log n)$$

as required. □

Theorem 4.1 is often referred to as "Symmetry of Information". We define the information content of y in x as the difference in the sizes of the programs needed to describe y given x as opposed to not given x, i.e., $I(x : y) = C(y) - C(y|x)$. The following Corollary of Theorem 4.1 shows that the amount of information of y in x is roughly the same as the amount of information of x in y.

Corollary 4.2. $I(x : y) = I(y : x) \pm O(\log n)$, *where* $n = \max\{|y|, |x|\}$.

Proof.

$$\begin{aligned} C(\langle x, y\rangle) &= C(y) + C(x|y) \pm O(\log n) \\ &= C(x) + C(y|x) \pm O(\log n), \end{aligned}$$

and hence

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

□

5. Prefix-Free Complexity

A problem with Kolmogorov complexity is that we are not always able to determine where one string stops and another begins. A solution to this is to use prefix-free languages.

Definition 5.1. We say a language $A \subseteq \Sigma^*$ is **prefix-free** if, for all x, y in A, if $x \neq y$ then x is not a prefix of y.

We say a function f is prefix-free if $\operatorname{dom} f$ is prefix-free.

Now we consider Kolmogorov complexity with respect to prefix-free codes. If this is to be analogous with our original characterization of Kolmogorov complexity, the first question we must ask is whether there is a universal prefix-free function, that is, one that provides descriptions that are within a constant of those provided by any prefix-free function, as an analog of the universal Turing Machine.

Definition 5.2. A **prefix-free machine** is a Turing machine with an input tape, some number of work tapes and an output tape. The input head can only read from left to right. At each stage there are three possible actions for the machine:

1. read a bit from the input and move the head to the right,
2. halt and output, or
3. go into an infinite loop.

Prefix-free machines were considered by several authors, notably Chaitin and Martin-Löf. (See Li and Vitányi [3] for the history here.) We say that a machine M accepts a function f if for all x, y, if $f(x) = y$ then the machine M reads exactly all bits of x, then outputs y and halts, while if $f(x)$ is undefined then M does not halt on input x.

Theorem 5.1. *Every prefix-free partial computable function can be accepted by a prefix-free machine, and there is a universal prefix-free machine.*

The proof of this theorem is not completely obvious, but not too difficult. The proof sketch runs as follows. Let f be partial computable, and $\operatorname{dom} f$ prefix-free. The corresponding prefix-free machine acts as follows. Read a bit of the input z. Before reading any more, simulate f simultaneously on all y such that z is a prefix of y, until $f(y)$ halts, if ever. If $y = z$, output $f(y)$, and otherwise, if $y \neq z$, read the next bit of input.

This argument produces a prefix-free version of each partial computable function, and gives us our universal machine, which on input $0^{|p|}1px$ uses p as the program for some f and simulates the above prefix-free machine for f on input x, and a corresponding universal function h.

Clearly $\operatorname{dom} h$ is prefix-free, and there is a constant c such that

$$C_h(x) \leq C_f(x) + c$$

for any prefix-free partial computable function f. The prefix-free complexity is then defined as before:

$$K(x) = C_h(x).$$

Notice that we get the following:

Theorem 5.2. $K(\langle x, y\rangle) \leq K(x) + K(y) + c.$

We don't need the $\log n$ factor any more. This is because if $h(p) = x$ and $h(q) = y$ then, by prefix-freeness, p is the only initial segment of the string pq that will give a halting computation.

Using the same counting argument as before, we see that

Theorem 5.3. $(\forall n)(\exists x \in \Sigma^n)(K(x) \geq n).$

So we have gained something. On the principle that there are no free lunches (except at this conference), we also lose something. Recall that we had

$$C(x) \leq |x| + c.$$

Now we no longer have this because we need a prefix-free way of describing x. We get the following instead.

$$K(x) \leq 2\log|x| + |x| + c,$$

or refining as before,

$$K(x) \leq 2\log\log|x| + \log|x| + |x| + c, \text{ etc.}$$

We remark that a counting argument demonstrates that

$$(\forall c)(\exists x)(K(x) \geq |x| + \log|x| + c).$$

6. Kraft's Inequality and the Universal Measure

The following theorem is the basis for assigning complexity to infinite sets.

Theorem 6.1 (Kraft's Inequality). *If A is prefix-free then*

$$\sum_{x \in A} 2^{-|x|} \leq 1.$$

Proof. Let R_x denote the interval of real numbers whose dyadic expansion begins with $0.x\ldots$. Then $|R_x| = 2^{-|x|}$, where $|R_x|$ denotes the length of the interval. Note that if $x \neq y$ with $x, y \in A$, then $R_x \cap R_y = \emptyset$. The result follows. □

This allows us to assign a measure: set $\mu(x) = 2^{-K(x)}$. Note that $\mu : \Sigma^* \mapsto [0,1]$ and $\sum_x \mu(x) < 1$. This measure assigns a weight to each string according to its Kolmogorov complexity. The shorter the string's description, the heavier the weighting.

The measure $\mu(x)$ is semicomputable, i.e. there exists a computable function $f(x,k)$, nondecreasing in k, such that

$$\lim_{k \to \infty} f(x,k) = \mu(x).$$

The measure $\mu(x)$ is universal for the semicomputable measures.

Fact 6.2. *Let $\tau(x)$ be any semicomputable function such that $\sum_{x \in \Sigma^*} \tau(x) \leq 1$. There exists a constant c such that for all x, $\tau(x) \leq c\mu(x)$.*

Take any algorithm and look at the average case under μ. This equals the worst case. Specifically, let $T(x)$ be the running time of some algorithm. Let $T_w(n) = \max_{x \in \Sigma^n} T(x)$, and $T_{ave}(n) = \frac{\Sigma_{x \in \Sigma^n} \mu(x) T(x)}{\Sigma_{x \in \Sigma^n} \mu(x)}$. Then we have the following.

Theorem 6.3. $T_w(n) = O(T_{ave}(n))$.

Proof. Let $\mu(n) = \sum_{x \in \Sigma^n} \mu(x)$. Let $\mu'(x)$ be the distribution that puts $\mu(n)$ weight at the lexicographically first string x of length n that maximizes $T(x)$. Theorem 6.3 follows from the universality of $\mu(x)$. □

7. Time-Bounded Kolmogorov Complexity

One of the problems with Kolmogorov complexity is that the shortest description of a string can run very slowly. In fact, it must often do so, lest the set of random strings have an infinite computably enumerable subset. One very useful variant of classical complexity is the time-bounded version. For a function t, we have:

$$C_f^t(x|y) = \min\{|p| : f(p,y) = x \text{ using time at most } t(|x| + |y|)\}.$$

Here and below, we use the convention that this quantity is ∞ if there is no such p.

There is a universality theorem in this setting. A function f is time-constructible if there exists a Turing machine that on input n outputs $t(n)$ using at most $t(n)$ time.

Fact 7.1. *There exists a computable g such that for all computable f and time-constructible t there is a constant c such that*

$$C_g^{t \log t}(x|y) \leq C_f^t(x|y) + c.$$

We define $C^t(x|y) = C_g^t(x|y)$ and $C^t(x) = C^t(x|\epsilon)$.

We can also look at the programs that *distinguish* x rather than *generate* x.

$$CD_f^t(x|y) = \min\{|p| : f(p,y,x) = 1 \ \wedge \ f(p,y,z) = 0 \text{ if } z \neq x \ \wedge \\ \forall z(f(p,y,z) \text{ uses at most time } t(|y|+|z|))\}.$$

We define $CD^t(x|y)$ and $CD^t(x)$ as above.

Without the time bound, $C(x|y) \leq CD(x|y) + O(1)$ for all x and y: Given the CD program p for x and y search for the first z such that $f(p,y,z) = 1$ and by definition $z = x$. However, for polynomial t we may not have enough time to perform this search.

So what is the relationship between C and CD for polynomial-time bounds?

$$(\forall \text{ poly } p)(\exists \text{ poly } q)(CD^q(x|y) \leq C^p(x|y) + c).$$

That is, a program that can generate x can easily give a program that distinguishes x. The converse is *much* harder.

Theorem 7.2. *The statement*

$$(\forall \textit{ poly } p)(\exists \textit{ poly } q)(C^q(x|y) \leq CD^p(x|y) + c \log |x|)$$

is equivalent to
"There is a polynomial time computable function f such that for all formulas φ with exactly one satisfying assignment, $f(\varphi)$ outputs that assignment."

The existence of such a function f is thought to be very unlikely; indeed, it is thought to be only slightly weaker than $P = NP$.

8. Sizes of Sets

It follows from Theorem 3.1 that if A is computably enumerable then $(\forall x \in A \cap \Sigma^n)(C(x|n) \leq \log |A \cap \Sigma^n| + O(1))$. Again it might take a long time to enumerate elements of A. We can obtain a similar result for a time-bounded version with CD.

Theorem 8.1. *For $A \in P$, there are a polynomial p and a constant c such that*

$$(\forall x \in A \cap \Sigma^n)(CD^p(x) \leq 2 \log |A \cap \Sigma^n| + c \log n).$$

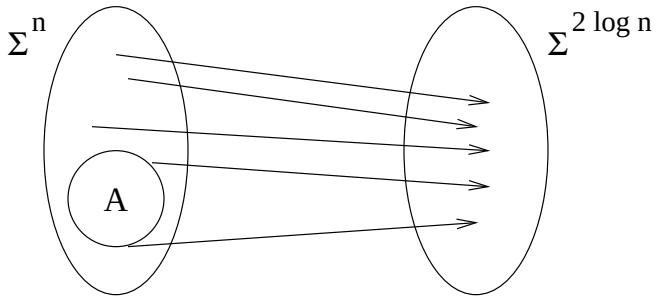

Figure 1: Hashing

We don't know how sharp this result is. There are relativized worlds B where there is an $A \in P^B$ such that for all polynomials q, there is an $x \in \Sigma^n$ with

$$CD^{q,B}(x) \geq 2 \log |A \cap \Sigma^n|,$$

where $CD^{q,B}(x)$ is defined as $CD^q(x)$, but relativized to B. This result means that the bound is tight as far as "known" techniques are concerned.

If P = NP then we have for every A in P there exist a polynomial p and a constant c such that

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

Since if P = NP, C and CD complexity are basically equal, we can replace CD by C in Equation (1).

If we only concern ourselves with *most* of the strings of A, we can substantially improve on Theorem 8.1.

Theorem 8.2. *For $A \in P$ and $\delta < 1$, there are a polynomial p and a constant c such that for a δ fraction of the $x \in \Sigma^n$,*

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

The closer we get to all of the strings, the worse the approximation is. For "nearly all" strings we can get close to a log factor, but to get all strings we need $2 \log$.

The proof of theorem 8.1 is quite interesting in technique, and uses *hashing*. Consider Figure 1.

Here we have a hash function h taking Σ^n to $\Sigma^{2 \log n}$. Suppose that h can be taken to be injective when restricted to A. Then we can describe each element of A by where it maps to. Note that this works only for distinguishing, not for generating. That is, if we know h and the image of x, we can distinguish x. The method is intrinsically *nonuniform*. The necessary lemma is that if we define

$$h_p(x) = x \bmod p$$

then if $\{x_1, \ldots, x_d\} \subseteq \{1, \ldots, 2^n\}$, there is a $p \leq 4dn^2$ with $x_i \neq x_j \bmod p$ for $i \neq j$. Given this fact, we can describe a h via p.

Theorem 8.3 (Sipser [4]). *For all $A \in P$, there are a polynomial q and a constant c such that, for all n and most strings r of length $q(n)$,*

$$(\forall x \in A \cap \Sigma^n)(CD^q(x|r) \leq \log |A \cap \Sigma^n| + c \log n).$$

A famous theorem in computational complexity is the Valiant-Vazirani Theorem. Take a formula φ. Then there is a randomized polynomial time reduction $\varphi \mapsto \psi$ such that if φ is not satisfiable then ψ is not satisfiable and if φ is satisfiable then ψ is satisfiable with probability at least $1/|\varphi|^k$ for some k.

This theorem follows from Theorem 8.3. Think of A as the space of satisfying assignments of some formula φ. Pick both r and the *CD* program p at random. With probability at least $1/|\varphi|^k$ for some k, p will be a *CD* program for some satisfying assignment of φ. Let ψ be the formula that encodes whether p and r accept some satisfying assignment of φ. If p is really a *CD* program than ψ will have exactly one solution as promised.

9. P-printable Sets

A set A is called *P-printable* if there is a polynomial time computable $f : \mathbb{N} \to 2^{\Sigma^*}$ such that $f(n) = A \cap \Sigma^n$. Note that A is P-printable implies that A is sparse in the sense that it has only polynomially many elements of length less than or equal to n. It is not clear whether all sparse sets in P are P-printable, but this is thought not to hold.

Consider $B_k = \{x : C^{n^k}(x) \leq k \log |x|\}$. For all k the set B_k is P-printable. (To print $B_k \cap \Sigma^n$, run all the n^k-time programs of length $k \log n$.) The following is a characterization of P-printability.

Theorem 9.1. *For all A in P, A is P-printable iff $A \subseteq B_k$ for some k.*

Proof. The "if" direction is clear. Now suppose that A is P-printable. Then the runtime for the function $f(n) = A \cap \Sigma^n$ is bounded by n^k for some k. Let $x \in A \cap \Sigma^n = \{x_1, \ldots, x_j\}$. Here $j \leq n^k$, so we can encode the bit vector of $x = x_i$ by $k \log n$ bits in polynomial time. □

10. References

Li and Vitányi [3] have an excellent book on Kolmogorov complexity. This book studies in depth most of the topics discussed in these notes. I strongly recommend that anyone with an interest in this area take a look at this book.

The material from Section 7 comes from recent research papers by Buhrman, Fortnow and Laplante [1] and Buhrman, Laplante and Miltersen [2].

References

[1] H. Buhrman, L. Fortnow, and S. Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM Journal on Computing*, 2001. To appear.

[2] 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.

[3] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer, New York, second edition, 1997.

[4] 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.