Elements of Information Theory
Thomas M. Cover, Joy A. Thomas
Copyright © 1991 John Wiley & Sons, Inc.
Print ISBN 0-471-06259-6 Online ISBN 0-471-20061-1

Chapter 7

Kolmogorov Complexity

The great mathematican Kolmogorov culminated a lifetime of research in mathematics, complexity and information theory with his definition in 1965 of the intrinsic descriptive complexity of an object. In our treatment so far, the object X has been a random variable drawn according to a probability mass function $p(x)$. If X is random, there is a sense in which the descriptive complexity of the event $X = x$ is $\log \frac{1}{p(x)}$, because $\lceil \log \frac{1}{p(x)} \rceil$ is the number of bits required to describe x by a Shannon code. One notes immediately that the descriptive complexity of such an object depends on the probability distribution.

Kolmogorov went further. He defined the algorithmic (descriptive) complexity of an object to be the length of the shortest binary computer program that describes the object. (Apparently a computer, the most general form of data decompressor, will use this description to exhibit the described object after a finite amount of computation.) Thus the Kolmogorov complexity of an object dispenses with the probability distribution. Kolmogorov made the crucial observation that the definition of complexity is essentially computer independent. It is an amazing fact that the expected length of the shortest binary computer description of a random variable is approximately equal to its entropy. Thus the shortest computer description acts as a universal code which is uniformly good for all probability distributions. In this sense, algorithmic complexity is a conceptual precursor to entropy.

This chapter is intellectually more demanding than the others in this book, and indeed, it can be omitted in a first course on information theory. Perhaps a proper point of view of the role of this chapter is to consider Kolmogorov complexity as a way to think. One does not use the shortest computer program in practice because it may take infinitely

long to find such a minimal program. But one can use very short, not necessarily minimal, programs in practice. And the idea of finding such short programs leads to universal codes, a good basis for inductive inference, a formalization of Occam's Razor ("The simplest explanation is best") and to clarity of thought in physics, computer science, and communication theory.

Before formalizing the notion of Kolmogorov complexity, let us give three strings as examples. They are

1. 01010101010101010101010101010101010101010101010101010101010101010-101010101
2. 01101010000010011110011001100111111100111011111001100100-100001000
3. 11011110011101011111011011111011101011011110001011100100-100111011

What are the shortest binary computer programs for each of these sequences? The first sequence is definitely simple. It consists of thirty-two 01's. The second sequence looks random and passes most tests for randomness, but it is in fact the binary expansion of $\sqrt{2} - 1$. Again, this is a simple sequence. The third again looks random, except that the proportion of 1's is not near 1/2. We shall assume that it is otherwise random. It turns out that by describing the number k of 1's in the sequence, then giving the index of the sequence in a lexicographic ordering of those with this number of 1's, one can give a description of the sequence in roughly $\log n + nH(\frac{k}{n})$ bits. This again is substantially less than the n bits in the sequence. Again, we conclude that the sequence, random though it is, is simple. In this case, however, it is not as simple as the other two sequences, which have constant length programs. In fact, its complexity is proportional to n. Finally, we can imagine a truly random sequence generated by pure coin flips. There are 2^n such sequences and they are all equally probable. It is highly likely that such a random sequence cannot be compressed, i.e., there is no better program for such a sequence than simply saying "Print the following: 0101100111010 . . . 0." The reason for this is that there are not enough short programs to go around. Thus the descriptive complexity of a truly random binary sequence is as long as the sequence itself.

These are the basic ideas. It will remain to be shown that this notion of intrinsic complexity is computer independent, i.e., that the length of the shortest program does not depend on the computer. At first, this seems like nonsense. But it turns out to be true, up to an additive constant. And for long sequences of high complexity, this additive constant (which is the length of the pre-program that allows one computer to mimic the other) is negligible.

7.1 MODELS OF COMPUTATION

To formalize the notions of algorithmic complexity, we first discuss acceptable models for computers. All but the most trivial computers are universal, in the sense that they can mimic the actions of other computers. We will briefly touch on a certain canonical universal computer, the universal Turing machine. The universal Turing machine is the conceptually simplest universal computer.

In 1936, Turing was obsessed with the question of whether the thoughts in a living brain could equally well be held by a collection of inanimate parts. In short, could a machine think? By analyzing the human computational process, he posited some constraints on such a computer. Apparently, a human thinks, writes, thinks some more, writes, and so on. Consider a computer as a finite state machine operating on a finite symbol set. (The symbols in an infinite symbol set cannot be distinguished in finite space.) A program tape, on which a binary program is written, is fed left to right into this finite state machine. At each unit of time, the machine inspects the program tape, writes some symbols on a work tape, changes its state according to its transition table and calls for more program. The operations of such a machine can be described by a finite list of transitions. Turing argued that this machine could mimic the computational ability of a human being.

After Turing's work, it turned out that every new computational system could be reduced to a Turing machine, and conversely. In particular, the familiar digital computer with its CPU, memory and input output devices could be simulated by and could simulate a Turing machine. This led Church to state what is now known as Church's thesis, which states that all (sufficiently complex) computational models are equivalent in the sense that they can compute the same family of functions. The class of functions they can compute agrees with our intuitive notion of effectively computable functions, that is, functions for which there is a finite prescription or program that will lead in a finite number of mechanically specified computational steps to the desired computational result.

We shall have in mind throughout this chapter the computer illustrated in Figure 7.1. At each step of the computation, the computer reads a symbol from the input tape, changes state according to its state transition table, possibly writes something on the work tape or output tape, and moves the program read head to the next cell of the program read tape. This machine reads the program from right to left only, never going back, and therefore the programs form a prefix-free set. No program leading to a halting computation can be the prefix of another such program. The restriction to prefix-free programs leads immediately to a theory of Kolmogorov complexity which is formally analogous to information theory.

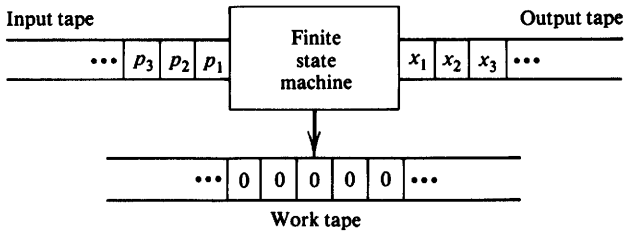


Figure 7.1. A Turing machine.

We can view the Turing machine as a map from a set of finite length binary strings to the set of finite or infinite length binary strings. In some cases, the computation does not halt, and in such cases the value of the function is said to be undefined. The set of functions $f: \{0,1\}^* \rightarrow \{0,1\}^* \cup \{0,1\}^\infty$ computable by Turing machines is called the set of *partial recursive functions*.

7.2 KOLMOGOROV COMPLEXITY: DEFINITIONS AND EXAMPLES

Let x be a finite length binary string and let $\mathcal{U}$ be a universal computer. Let $l(x)$ denote the length of the string x. Let $\mathcal{U}(p)$ denote the output of the computer $\mathcal{U}$ when presented with a program p.

We define the Kolmogorov (or algorithmic) complexity of a string x as the minimal description length of x.

Definition: The *Kolmogorov complexity* $K_{\mathcal{U}}(x)$ of a string x with respect to a universal computer $\mathcal{U}$ is defined as

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p), \tag{7.1}$$

the minimum length over all programs that print x and halt. Thus $K_{\mathcal{U}}(x)$ is the shortest description length of x over all descriptions interpreted by computer $\mathcal{U}$.

An important technique for thinking about Kolmogorov complexity is the following—if one person can describe a sequence to another person in such a manner as to lead unambiguously to a computation of that sequence in a finite amount of time, then the number of bits in that communication is an upper bound on the Kolmogorov complexity. For example, one can say "Print out the first 1,239,875,981,825,931 bits of the square root of e." Allowing 8 bits per character (ASCII), we see that the above unambiguous 73 symbol program demonstrates that the Kolmogorov complexity of this huge number is no greater than (8)(73) = 584 bits. Most numbers of this length have a Kolmogorov complexity of

1,239,875,981,825,931 bits. The fact that there is a simple algorithm to calculate the square root of e provides the saving in descriptive complexity.

In the above definition, we have not mentioned anything about the length of x. If we assume that the computer already knows the length of x, then we can define the *conditional Kolmogorov complexity* knowing $l(x)$ as

$$K_{\mathcal{U}}(x|l(x)) = \min_{p\,:\,\mathcal{U}(p,\,l(x))=x} l(p)\,. \tag{7.2}$$

This is the shortest description length if the computer $\mathcal{U}$ has the length of x made available to it.

It should be noted that $K_{\mathcal{U}}(x|y)$ is usually defined as $K_{\mathcal{U}}(x|y, y^*)$, where y^* is the shortest program for y. This is to avoid certain slight asymmetries in chain rules like $K(x, y) = K(x) + K(y|x) \approx K(y) + K(x|y)$, but we will not use this definition here.

We first prove some of the basic properties of Kolmogorov complexity and then consider various examples.

Theorem 7.2.1 (*Universality of Kolmogorov complexity*): *If $\mathcal{U}$ is a universal computer, then for any other computer $\mathcal{A}$,*

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}} \tag{7.3}$$

for all strings $x \in \{0,1\}^*$, where the constant $c_{\mathcal{A}}$ does not depend on x.

Proof: Assume that we have a program $p_{\mathcal{A}}$ for computer $\mathcal{A}$ to print x. Thus $\mathcal{A}(p_{\mathcal{A}}) = x$. We can precede this program by a simulation program $s_{\mathcal{A}}$ which tells computer $\mathcal{U}$ how to simulate computer $\mathcal{A}$. The computer $\mathcal{U}$ will then interpret the instructions in the program for $\mathcal{A}$, perform the corresponding calculations and print out x. The program for $\mathcal{U}$ is $p = s_{\mathcal{A}} p_{\mathcal{A}}$ and its length is

$$l(p) = l(s_{\mathcal{A}}) + l(p_{\mathcal{A}}) = c_{\mathcal{A}} + l(p_{\mathcal{A}})\,, \tag{7.4}$$

where $c_{\mathcal{A}}$ is the length of the simulation program. Hence,

$$K_{\mathcal{U}}(x) = \min_{p\,:\,\mathcal{U}(p)=x} l(p) \leq \min_{p\,:\,\mathcal{A}(p)=x} (l(p) + c_{\mathcal{A}}) = K_{\mathcal{A}}(x) + c_{\mathcal{A}} \tag{7.5}$$

for all strings x. □

The constant $c_{\mathcal{A}}$ in the theorem may be very large. For example, $\mathcal{A}$ may be a large computer with a large number of functions built into the system. The computer $\mathcal{U}$ can be a simple microprocessor. The simulation program will contain the details of the implementation of all these

functions, in fact, all the software available on the large computer. The crucial point is that the length of this simulation program is independent of the length of x, the string to be compressed. For sufficiently long x, the length of this simulation program can be neglected, and we can discuss Kolmogorov complexity without talking about the constants.

If $\mathcal{A}$ and $\mathcal{U}$ are both universal, then we have

$$|K_{\mathcal{U}}(x) - K_{\mathcal{A}}(x)| < c \tag{7.6}$$

for all x. Hence we will drop all mention of $\mathcal{U}$ in all further definitions. We will assume that the unspecified computer $\mathcal{U}$ is a fixed universal computer.

Theorem 7.2.2 (*Conditional complexity is less than the length of the sequence*):

$$K(x|l(x)) \leq l(x) + c\,. \tag{7.7}$$

Proof: We can exhibit the string x in the program. The program is self-delimiting because $l(x)$ is provided and the end of the program is thus clearly defined. A program for printing x is

$$\texttt{Print the following } l\texttt{-bit sequence: } x_1x_2\ldots x_{l(x)}\,.$$

Note that no bits are required to describe l since l is given. The length of this program is $l(x) + c$. □

Without knowledge of the length of the string, we will need an additional stop symbol or we can use a self-punctuating scheme like the one described in the proof of the next theorem.

Theorem 7.2.3 (*Upper bound on Kolmogorov complexity*):

$$K(x) \leq K(x|l(x)) + 2\log l(x) + c\,. \tag{7.8}$$

Proof: If the computer does not know $l(x)$, the method of Theorem 7.2.2 does not apply. We must have some way of informing the computer when it has come to the end of the string of bits that describes the sequence. We describe a simple but inefficient method which uses a sequence 01 as a "comma."

Suppose $l(x) = n$. To describe $l(x)$, repeat every bit of the binary expansion of n twice; then end the description with a 01 so that the computer knows that it has come to the end of the description of n. For example, the number 5 (binary 101) will be described as 11001101. This description requires $2\lceil\log n\rceil + 2$ bits.

Thus, inclusion of the binary representation of $l(x)$ does not add more than $2 \log l(x) + c$ bits to the length of the program, and we have the bound in the theorem. □

A more efficient method for describing n is to do so recursively. We first specify the number $(\log n)$ of bits in the binary representation of n, and then specify the actual bits of n. To specify $\log n$, the length of the binary representation of n, we can use the inefficient method $(2 \log \log n)$ or the efficient method $(\log \log n + \cdots)$. If we use the efficient method at each level, until we have a small number to specify, we can describe n in $\log n + \log \log n + \log \log \log n + \cdots$ bits, where we continue the sum until the last positive term. This sum of iterated logarithms is sometimes written $\log^* n$. Thus Theorem 7.2.3 can be improved to

$$K(x) \leq K(x|l(x)) + \log^* l(x) + c . \tag{7.9}$$

We now prove that there are very few sequences with low complexity.

Theorem 7.2.4 (*Lower bound on Kolmogorov complexity*): *The number of strings x with complexity $K(x) < k$ satisfies*

$$|\{x \in \{0,1\}^* : K(x) < k\}| < 2^k . \tag{7.10}$$

Proof: There are not very many short programs. If we list all the programs of length $< k$, we have

$$\underbrace{\Lambda}_{1}, \underbrace{0}_{2}, \underbrace{1}_{}, \underbrace{00, 01, 10, 11}_{4}, \ldots, \underbrace{\ldots, \overbrace{11\ldots1}^{k-1}}_{2^{k-1}} \tag{7.11}$$

and the total number of such programs is

$$1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1 < 2^k . \tag{7.12}$$

Since each program can produce only one possible output sequence, the number of sequences with complexity $< k$ is less than 2^k. □

To avoid confusion and to facilitate exposition in the rest of this chapter, we shall need to introduce a special notation for the *binary entropy function*

$$H_0(p) = -p \log p - (1-p) \log(1-p) . \tag{7.13}$$

Thus, when we write $H_0(\frac{1}{n} \Sigma_{i=1}^n X_i)$, we will mean $-\bar{X}_n \log \bar{X}_n - (1 - \bar{X}_n) \log(1 - \bar{X}_n)$ and not the entropy of random variable $\bar{X}_n$. When there is no confusion, we shall simply write $H(p)$ for $H_0(p)$.

Now let us consider various examples of Kolmogorov complexity. The complexity will depend on the computer, but only up to an additive constant. To be specific, we consider a computer that can accept unambiguous commands in English (with numbers given in binary notation). We will assume the inequality

$$\frac{1}{n+1}\,2^{nH(\frac{k}{n})} \le \binom{n}{k} \le 2^{nH(\frac{k}{n})}, \tag{7.14}$$

which can be easily proved using Stirling's formula [110]. An alternative proof can be found in Example 12.1.3.

Example 7.2.1 (*A sequence of n zeroes*): If we assume that the computer knows n, then a short program to print this string is

```
Print the specified number of zeroes.
```

The length of this program is a constant number of bits. This program length does not depend on n. Hence the Kolmogorov complexity of this sequence is c, and

$$K(000\ldots0|n) = c \quad \text{for all } n\,. \tag{7.15}$$

Example 7.2.2 (*Kolmogorov complexity of* π): The first n bits of π can be calculated using a simple series expression. This program has a small constant length, if the computer already knows n. Hence

$$K(\pi_1\pi_2\ldots\pi_n|n) = c\,. \tag{7.16}$$

Example 7.2.3 (*Gotham weather*): Suppose we want the computer to print out the weather in Gotham for n days. We can write a program that contains the entire sequence $x = x_1x_2\ldots x_n$, where $x_i = 1$ indicates rain on day i. But this is inefficient, since the weather is quite dependent. We can devise various coding schemes for the sequence to take the dependence into account. A simple one is to find a Markov model to approximate the sequence (using the empirical transition probabilities) and then code the sequence using the Shannon code for this probability distribution. We can describe the empirical Markov transitions in $O(\log n)$ bits, and then use $\log \frac{1}{p(x)}$ bits to describe x, where p is the specified Markov probability. Assuming that the entropy of the weather is 1/5 bits per day, we can describe the weather for n days using about $n/5$ bits, and hence

$$K(\text{Gotham weather}|n) \approx \frac{n}{5} + O(\log n) + c\,. \tag{7.17}$$

Example 7.2.4 (*A repeating sequence of the form 01010101 ... 01*): A short program suffices. Simply print the specified number of 01 pairs. Hence

$$K(010101010\ldots01|n) = c\,. \tag{7.18}$$

Example 7.2.5 (*A fractal*): The fractal on the cover is part of the Mandelbrot set, and is generated by a simple computer program. For different points c in the complex plane, one calculates the number of iterations of the map $z_{n+1} = z_n^2 + c$ (starting with $z_0 = 0$) needed for $|z|$ to cross a particular threshold. The point c is then colored according to the number of iterations needed.

Thus the fractal is an example of an object that looks very complex but is essentially very simple. Its Kolmogorov complexity is nearly zero.

Example 7.2.6 (*The Mona Lisa*): We can make use of the many structures and dependencies in the painting. We can probably compress the image by a factor of 3 or so by using some existing easily described image compression algorithm. Hence, if n is the number of pixels in the image of the Mona Lisa,

$$K(\text{Mona Lisa}|n) \leq \frac{n}{3} + c\,. \tag{7.19}$$

Example 7.2.7 (*An integer n*): If the computer knows the number of bits in the binary representation of the integer, then we need only provide the values of these bits. This program will have length $c + \log n$.

In general the computer will not know the length of the binary representation of the integer. So we must inform the computer in some way when the description ends. Using the method to describe integers used to derive (7.9), we see that the Kolmogorov complexity of an integer is bounded by

$$K(n) \leq \log^* n + c\,. \tag{7.20}$$

Example 7.2.8 (*A sequence of n bits with k ones*): Can we compress a sequence of n bits with k ones?

Our first guess is no, since we have a series of bits that must be reproduced exactly. But consider the following program:

```
Generate, in lexicographic order, all sequences with k ones;
        Of these sequences, print the ith sequence.
```

This program will print out the required sequence. The only variables in the program are k (with range $\{0, 1, \ldots, n\}$) and i (with conditional range $\binom{n}{k}$). The total length of this program is

$$l(p) = c + \underbrace{2\log k}_{\text{to express } k} + \underbrace{\log\binom{n}{k}}_{\text{to express } i} \tag{7.21}$$

$$\leq c + 2\log k + nH_0\left(\frac{k}{n}\right), \tag{7.22}$$

since $\binom{n}{k} \leq 2^{nH_0\left(\frac{k}{n}\right)}$ by (7.14). We have used $2\log k + 2$ bits to represent k by the inefficient method described in the proof of Theorem 7.2.3. Thus if $\sum_{i=1}^{n} x_i = k$, then

$$K(x_1, x_2, \ldots, x_n | n) \leq nH\left(\frac{k}{n}\right) + 2\log k + c\,. \tag{7.23}$$

We can summarize the last example in the following theorem:

Theorem 7.2.5: *The Kolmogorov complexity of a binary string x is bounded by*

$$K(x_1 x_2 \ldots x_n | n) \leq nH_0\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + 2\log n + c\,. \tag{7.24}$$

Proof: Use the program described in the last example. □

Remark: Let $x \in \{0,1\}^*$ be the data we wish to compress, and consider the program p to be the compressed data. We will have succeeded in compressing the data only if $l(p) < l(x)$, or

$$K(x) < l(x)\,. \tag{7.25}$$

In general, when the length $l(x)$ of the sequence x is small, the constants that appear in the expressions for the Kolmogorov complexity will overwhelm the contributions due to $l(x)$. Hence the theory is useful primarily when $l(x)$ is very large. In such cases, we can safely neglect the constants that do not depend on $l(x)$.

7.3 KOLMOGOROV COMPLEXITY AND ENTROPY

We now consider the relationship between the Kolmogorov complexity of a sequence of random variables and its entropy. In general, we show that the expected value of the Kolmogorov complexity of a random sequence is close to the Shannon entropy. First, we prove that the program lengths satisfy the Kraft inequality:

Lemma 7.3.1: *For any computer* $\mathcal{U}$,

$$\sum_{p:\,\mathcal{U}(p)\text{ halts}} 2^{-l(p)} \le 1 . \tag{7.26}$$

Proof: If the computer halts on any program, it does not look any further for input. Hence, there cannot be any other halting program with this program as a prefix. Thus the halting programs form a prefix-free set, and their lengths satisfy the Kraft inequality (Theorem 5.2.1). □

We now show that $\frac{1}{n}EK(X^n|n) \approx H(X)$ for i.i.d. processes with a finite alphabet.

Theorem 7.3.1 (*Relationship of Kolmogorov complexity and entropy*): *Let the stochastic process* $\{X_i\}$ *be drawn i.i.d. according to the probability mass function* $f(x)$, $x \in \mathcal{X}$, *where* $\mathcal{X}$ *is a finite alphabet. Let* $f(x^n) = \prod_{i=1}^{n} f(x_i)$. *Then there exists a constant* c *such that*

$$H(X) \le \frac{1}{n}\sum_{x^n} f(x^n)K(x^n|n) \le H(X) + \frac{|\mathcal{X}|\log n}{n} + \frac{c}{n} \tag{7.27}$$

for all n. *Thus*

$$E\,\frac{1}{n}K(X^n|n) \to H(X) . \tag{7.28}$$

Proof: Consider the lower bound. The allowed programs satisfy the prefix property, and thus their lengths satisfy the Kraft inequality. We assign to each x^n the length of the shortest program p such that $\mathcal{U}(p, n) = x^n$. These shortest programs also satisfy the Kraft inequality. We know from the theory of source coding that the expected codeword length must be greater than the entropy. Hence

$$\sum_{x^n} f(x^n)K(x^n|n) \ge H(X_1, X_2, \ldots, X_n) = nH(X) . \tag{7.29}$$

We first prove the upper bound when $\mathcal{X}$ is binary, i.e., $X_1, X_2, \ldots, X_n$ are i.i.d. $\sim$ Bernoulli(θ). Using the method of Theorem 7.2.5, we can bound the complexity of a binary string by

$$K(x_1x_2\ldots x_n|n) \le nH_0\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + 2\log n + c . \tag{7.30}$$

Hence

$$EK(X_1X_2\ldots X_n|n) \le nEH_0\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) + 2\log n + c \tag{7.31}$$

$$\overset{(a)}{\leq} nH_0\left(\frac{1}{n}\sum_{i=1}^{n} EX_i\right) + 2\log n + c \tag{7.32}$$

$$= nH_0(\theta) + 2\log n + c\,, \tag{7.33}$$

where (a) follows from Jensen's inequality and the concavity of the entropy. Thus we have proved the upper bound in the theorem for binary processes.

We can use the same technique for the case of a non-binary finite alphabet. We first describe the type of the sequence (the empirical frequency of occurrence of each of the alphabet symbols as defined in Section 12.1) using $|\mathcal{X}| \log n$ bits. Then we describe the index of the sequence within the set of all sequences having the same type. The type class has less than $2^{nH(P_{x^n})}$ elements (where P_{x^n} is the type of the sequence x^n), and therefore the two-stage description of a string x^n has length

$$K(x^n|n) \leq nH(P_{x^n}) + |\mathcal{X}| \log n + c\,. \tag{7.34}$$

Again, taking the expectation and applying Jensen's inequality as in the binary case, we obtain

$$EK(X^n|n) \leq nH(X) + |\mathcal{X}| \log n + c\,. \tag{7.35}$$

Dividing this by n yields the upper bound of the theorem. □

7.4 KOLMOGOROV COMPLEXITY OF INTEGERS

In the last section, we defined the Kolmogorov complexity of a binary string as the length of the shortest program for a universal computer that prints out that string. We can extend that definition to define the Kolmogorov complexity of an integer to be the Kolmogorov complexity of the corresponding binary string.

Definition: The *Kolmogorov complexity of an integer* n is defined as

$$K(n) = \min_{p:\mathcal{U}(p)=n} l(p)\,. \tag{7.36}$$

The properties of the Kolmogorov complexity of integers are very similar to those of the Kolmogorov complexity of bit strings. The following properties are immediate consequences of the corresponding properties for strings.

Theorem 7.4.1: *For universal computers* $\mathcal{A}$ *and* $\mathcal{U}$,

$$K_{\mathcal{U}}(n) \leq K_{\mathcal{A}}(n) + c_{\mathcal{A}} \,. \tag{7.37}$$

Also, since any number can be specified by its binary expansion, we have the following theorem.

Theorem 7.4.2:

$$K(n) \leq \log^* n + c \,. \tag{7.38}$$

Theorem 7.4.3: *There are an infinite number of integers n such that $K(n) > \log n$.*

Proof: We know from Lemma 7.3.1 that

$$\sum_n 2^{-K(n)} \leq 1 \,, \tag{7.39}$$

and

$$\sum_n 2^{-\log n} = \sum_n \frac{1}{n} = \infty \,. \tag{7.40}$$

But if $K(n) < \log n$ for all $n > n_0$, then

$$\sum_{n=n_0}^{\infty} 2^{-K(n)} > \sum_{n=n_0}^{\infty} 2^{-\log n} = \infty \,, \tag{7.41}$$

which is a contradiction. □

7.5 ALGORITHMICALLY RANDOM AND INCOMPRESSIBLE SEQUENCES

From the examples in Section 7.2, it is clear that there are some long sequences that are simple to describe, like the first million bits of π. By the same token, there are also large integers that are simple to describe, such as

$$2^{2^{2^{2^{2^2}}}}$$

or (100!)!.

We now show that although there are some simple sequences, most sequences do not have simple descriptions. Similarly, most integers are not simple. Hence if we draw a sequence at random, we are likely to draw a complex sequence. The next theorem shows that the probability that a sequence can be compressed by more than k bits is no greater than 2^{-k}.

Theorem 7.5.1: *Let $X_1, X_2, \ldots, X_n$ be drawn according to a Bernoulli($\frac{1}{2}$) process. Then*

$$P(K(X_1X_2 \ldots X_n|n) < n-k) < 2^{-k}. \tag{7.42}$$

Proof:

$$P(K(X_1X_2 \ldots X_n|n) < n-k)$$

$$= \sum_{x_1, x_2, \ldots, x_n : K(x_1x_2 \ldots x_n|n) < n-k} p(x_1, x_2, \ldots, x_n) \tag{7.43}$$

$$= \sum_{x_1, x_2, \ldots, x_n : K(x_1x_2 \ldots x_n|n) < n-k} 2^{-n} \tag{7.44}$$

$$= |\{x_1, x_2, \ldots, x_n : K(x_1x_2 \ldots x_n|n) < n-k\}| 2^{-n}$$

$$< 2^{n-k}2^{-n} \quad \text{(by Theorem 7.2.4)} \tag{7.45}$$

$$= 2^{-k}. \quad \square \tag{7.46}$$

Thus most sequences have a complexity close to their length. For example, the fraction of sequences of length n that have complexity less than $n-5$ is less than $1/32$. This motivates the following definition:

Definition: A sequence $x_1, x_2, \ldots, x_n$ is said to be *algorithmically random* if

$$K(x_1x_2 \ldots x_n|n) \geq n. \tag{7.47}$$

Note that by the counting argument, there exists, for each n, at least one sequence x^n such that

$$K(x^n|n) \geq n. \tag{7.48}$$

Definition: We call an infinite string x *incompressible* if

$$\lim_{n \to \infty} \frac{K(x_1x_2x_3 \ldots x_n|n)}{n} = 1. \tag{7.49}$$

Theorem 7.5.2 (*Strong law of large numbers for incompressible sequences*): *If a string $x_1x_2 \ldots$ is incompressible, then it satisfies the law of large numbers in the sense that*

$$\frac{1}{n} \sum_{i=1}^{n} x_i \to \frac{1}{2}. \tag{7.50}$$

Hence the proportions of 0's and 1's in any incompressible string are almost equal.

Proof: Let $\theta_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ denote the proportion of 1's in $x_1, x_2, \ldots, x_n$. Then using the method of Example 7.2 of Section 7.2, one can write a program of length $nH_0(\theta_n) + 2\log(n\theta_n) + c$ to print x^n. By the incompressibility assumption, we also have the lower bound,

$$n - c_n \leq K(x^n|n) \leq nH_0(\theta_n) + 2\log n + c' . \tag{7.51}$$

where $c_n/n \to 0$ and c' does not depend on n. Thus

$$H_0(\theta_n) > 1 - \frac{2\log n + c_n + c'}{n} . \tag{7.52}$$

Inspection of the graph of $H_0(p)$ (Figure 7.2) shows that θ_n is close to $\frac{1}{2}$ for large n. Specifically, the above inequality implies that

$$\theta_n \in \left(\frac{1}{2} - \delta_n, \frac{1}{2} + \delta_n\right), \tag{7.53}$$

where δ_n is chosen so that

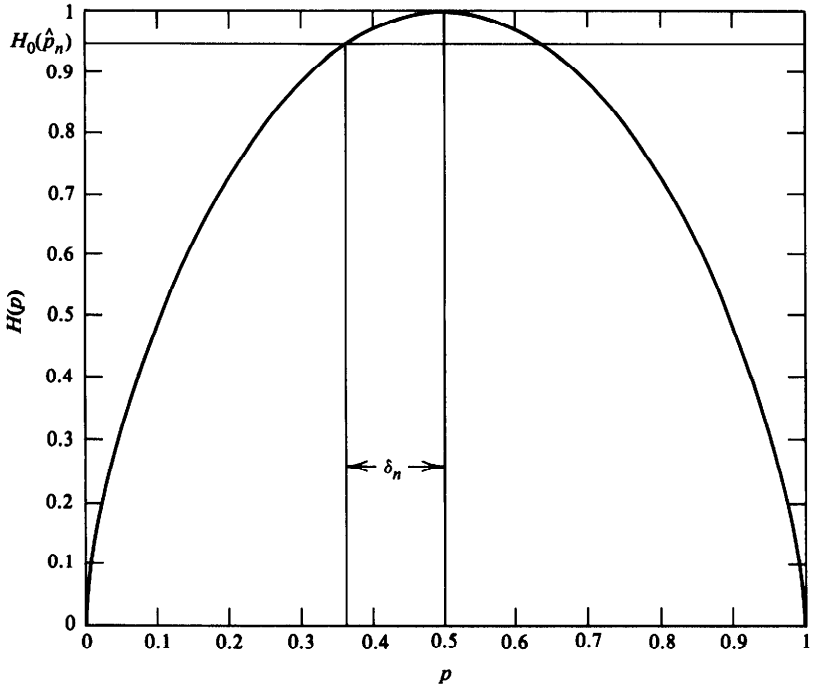


Figure 7.2. $H_0(p)$ versus p.

$$H_0\left(\frac{1}{2} - \delta_n\right) = 1 - \frac{2\log n + c_n + c'}{n}, \tag{7.54}$$

which implies that $\delta_n \to 0$ as $n \to \infty$. Thus $\frac{1}{n}\sum x_i \to \frac{1}{2}$ as $n \to \infty$. □

We have now proved that incompressible sequences look random in the sense that the proportion of 0's and 1's are almost equal. In general, we can show that if a sequence is incompressible, it will satisfy all computable statistical tests for randomness. (Otherwise, identification of the test that x fails will reduce the descriptive complexity of x, yielding a contradiction.) In this sense, the algorithmic test for randomness is the ultimate test, including within it all other computable tests for randomness.

We now prove a related law of large numbers for the Kolmogorov complexity of Bernoulli(θ) sequences. The Kolmogorov complexity of a sequence of binary random variables drawn i.i.d. according to a Bernoulli(θ) process is close to the entropy $H_0(\theta)$.

Theorem 7.5.3: *Let $X_1, X_2, \ldots, X_n$ be drawn i.i.d. ~ Bernoulli(θ).* Then

$$\frac{1}{n} K(X_1, X_2, \ldots, X_n | n) \to H_0(\theta) \quad \textit{in probability}. \tag{7.55}$$

Proof: Let $\bar{X}_n = \frac{1}{n}\sum X_i$ be the proportion of 1's in $X_1, X_2, \ldots, X_n$. Then using the method described in (7.23), we have

$$K(X_1, X_2, \ldots, X_n | n) \le nH_0(\bar{X}_n) + 2\log n + c, \tag{7.56}$$

and since by the weak law of large numbers, $\bar{X}_n \to \theta$ in probability, we have

$$\Pr\left\{\frac{1}{n} K(X_1, X_2, \ldots, X_n | n) - H_0(\theta) \ge \epsilon\right\} \to 0. \tag{7.57}$$

Conversely, we can bound the number of sequences with complexity significantly lower than the entropy. From the AEP, we can divide the set of sequences into the typical set and the non-typical set. There are at least $(1-\epsilon)2^{n(H_0(\theta)-\epsilon)}$ sequences in the typical set. At most $2^{n(H_0(\theta)-c)}$ of these typical sequences can have a complexity less than $n(H_0(\theta) - c)$. The probability that the complexity of the random sequence is less than $n(H_0(\theta) - c)$ is

$$\begin{aligned}
&\Pr(K(X^n|n) < n(H_0(\theta) - c)) \\
&\quad \le \Pr(X^n \notin A_\epsilon^{(n)}) + \Pr(X^n \in A_\epsilon^{(n)}, K(X^n|n) < n(H_0(\theta) - c))
\end{aligned}$$

$$\leq \epsilon + \sum_{x^n \in A_\epsilon^{(n)},\, K(x^n|n) < n(H_0(\theta)-c)} p(x^n) \tag{7.58}$$

$$\leq \epsilon + \sum_{x^n \in A_\epsilon^{(n)},\, K(x^n|n) < n(H_0(\theta)-c)} 2^{-n(H_0(\theta)-\epsilon)} \tag{7.59}$$

$$\leq \epsilon + 2^{n(H_0(\theta)-c)}\, 2^{-n(H_0(\theta)-\epsilon)} \tag{7.60}$$

$$= \epsilon + 2^{-n(c-\epsilon)}, \tag{7.61}$$

which is arbitrarily small for appropriate choice of ϵ, n and c. Hence with high probability, the Kolmogorov complexity of the random sequence is close to the entropy, and we have

$$\frac{K(X_1, X_2, \ldots, X_n|n)}{n} \to H_0(\theta) \quad \text{in probability.} \quad \square \tag{7.62}$$

7.6 UNIVERSAL PROBABILITY

Suppose that a computer is fed a random program. Imagine a monkey sitting at a keyboard and typing the keys at random. Equivalently, feed a series of fair coin flips into a universal Turing machine. In either case, most strings will not make sense to the computer. If a person sits at a terminal and types keys at random, he will probably get an error message, i.e., the computer will print the null string and halt. But with a certain probability he will hit on something that makes sense. The computer will then print out something meaningful. Will this output sequence look random?

From our earlier discussions, it is clear that most sequences of length n have complexity close to n. Since the probability of an input program p is $2^{-l(p)}$, shorter programs are much more probable than longer ones. And shorter programs, when they produce long strings, do not produce random strings; they produce strings with simply described structure.

The probability distribution on the output strings is far from uniform. Under the computer induced distribution, simple strings are more likely than complicated strings of the same length. This motivates us to define a universal probability distribution on strings as follows:

Definition: The *universal probability* of a string x is

$$P_{\mathcal{U}}(x) = \sum_{p:\, \mathcal{U}(p)=x} 2^{-l(p)} = \Pr(\mathcal{U}(p) = x), \tag{7.63}$$

which is the probability that a program randomly drawn as a sequence of fair coin flips $p_1, p_2, \ldots$ will print out the string x.

This probability is universal in many senses. We can consider it as the probability of observing such a string in nature; the implicit belief is that simpler strings are more likely than complicated strings. For example, if we wish to describe the laws of physics, we might consider the simplest string describing the laws as the most likely. This principle is known as "Occam's Razor", and has been a general principle guiding scientific research for centuries—if there are many explanations consistent with the observed data, choose the simplest. In our framework, Occam's Razor is equivalent to choosing the shortest program that produces a given string.

This probability mass function is called universal because of the following theorem:

Theorem 7.6.1: *For every computer* $\mathscr{A}$,

$$P_{\mathscr{U}}(x) \geq c'_{\mathscr{A}} P_{\mathscr{A}}(x) \tag{7.64}$$

for every string $x \in \{0,1\}^*$, *where the constant* $c'_{\mathscr{A}}$ *depends only on* $\mathscr{U}$ *and* $\mathscr{A}$.

Proof: From the discussion of Section 7.2, we recall that for every program p' for $\mathscr{A}$ that prints x, there exists a program p for $\mathscr{U}$ of length not more than $l(p') + c_{\mathscr{A}}$ produced by prefixing a simulation program for $\mathscr{A}$. Hence

$$P_{\mathscr{U}}(x) = \sum_{p:\mathscr{U}(p)=x} 2^{-l(p)} \geq \sum_{p':\mathscr{A}(p')=x} 2^{-l(p')-c_{\mathscr{A}}} = c'_{\mathscr{A}} P_{\mathscr{A}}(x) . \quad \square \tag{7.65}$$

Any sequence drawn according to a computable probability mass function on binary strings can be considered to be produced by some computer $\mathscr{A}$ acting on a random input (via the probability inverse transformation acting on a random input). Hence the universal probability distribution includes a mixture of all computable probability distributions.

Remark (*Bounded likelihood ratio*): In particular, Theorem 7.6.1 guarantees that a likelihood ratio test of the hypothesis that X is drawn according to $P_{\mathscr{U}}$ versus the hypothesis that it is drawn according to $P_{\mathscr{A}}$ will have bounded likelihood ratio. If $\mathscr{U}$ and $\mathscr{A}$ are universal, then $P_{\mathscr{U}}(x)/P_{\mathscr{A}}(x)$ is bounded away from zero and infinity for all x. This is in contrast to other simple hypothesis testing problems (like Bernoulli(θ_1) versus Bernoulli(θ_2)) where the likelihood ratio goes to 0 or ∞ as the sample size goes to infinity. Apparently $P_{\mathscr{U}}$ can never be completely rejected as the true distribution of any data drawn according to some computable probability distribution. In that sense, we cannot reject the

possibility that the universe is the output of monkeys typing at a computer.

In Section 7.11 we will prove that

$$P_{\mathcal{U}}(x) \approx 2^{-K(x)}, \tag{7.66}$$

thus showing that $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ have equal status as universal algorithmic complexity measures.

We will conclude this section with an example of a monkey at a typewriter vs. a monkey at a computer keyboard. If the monkey types at random on a typewriter, the probability that it types out all the works of Shakespeare (assuming the text is 1 million bits long) is $2^{-1,000,000}$. If the monkey sits at a computer terminal, however, the probability that it types out Shakespeare is now $2^{-K(\text{Shakespeare})} \approx 2^{-250,000}$, which though extremely small is still exponentially more likely than when the monkey sits at a dumb typewriter.

The example indicates that a random input to a computer is much more likely to produce "interesting" outputs than a random input to a typewriter. We all know that a computer is an intelligence amplifier. Apparently it creates sense from nonsense as well.

7.7 THE HALTING PROBLEM AND THE NON-COMPUTABILITY OF KOLMOGOROV COMPLEXITY

Consider the following paradoxical statement:

`This statement is false.`

This paradox is sometimes stated in a two-statement form:

`The next statement is false.`

`The preceding statement is true.`

These paradoxes are versions of what is called the Epimenides Liar Paradox, and it illustrates the pitfalls involved in self-reference. In 1931, Gödel used this idea of self-reference to show that any interesting system of mathematics is not complete; there are statements in the system that are true but which cannot be proved within the system. To accomplish this, he translated theorems and proofs into integers, and constructed a statement of the above form, which can therefore not be proved true or false.

The halting problem in computer science is very closely connected with Gödel's incompleteness theorem. In essence, it states that for any

computational model, there is no general algorithm to decide whether a program will halt or not (go on forever). Note that it is not a statement about any specific program. Quite clearly, there are many programs that can be easily shown to halt or go on forever. The halting problem says that we cannot answer this question for all programs. The reason for this is again the idea of self-reference.

To a practical person, the halting problem may not be of any immediate significance, but it has great theoretical importance as the dividing line between things that can be done on a computer (given unbounded memory and time) and things that cannot be done at all (such as proving all true statements in number theory). Gödel's incompleteness theorem is one of the most important mathematical results of this century, and its consequences are still being explored. The halting problem is an essential example of Gödel's incompleteness theorem.

One of the consequences of the non-existence of an algorithm for the halting problem is the non-computability of Kolmogorov complexity. The only way to find the shortest program in general is to try all short programs and see which of them can do the job. However, at any time some of the short programs may not have halted and there is no effective (finite mechanical) way to tell whether they will halt or not and what they will print out. Hence, there is no effective way to find the shortest program to print a given string.

The non-computability of Kolmogorov complexity is an example of the Berry paradox. The Berry paradox asks for "the shortest number not nameable in under ten words." No number like 1,101,121 can be a solution since the defining expression itself is less than ten words long. This illustrates the problems with the terms nameable and describable; they are too powerful to be used without a strict meaning. If we restrict ourselves to the meaning "can be described for printing out on a computer," then we can resolve Berry's paradox by saying that the smallest number not describable in less than ten words exists, but is not computable. This so-called "description" is not a program for computing the number. E. F. Beckenbach pointed out a similar problem in the classification of numbers as dull or interesting; the smallest dull number must be interesting.

As stated at the beginning of the chapter, one does not really anticipate that practitioners will find the shortest computer program for a given string. The shortest program is not computable, although as more and more programs are shown to produce the string, the estimates from above of the Kolmogorov complexity converge to the true Kolmogorov complexity. (The problem, of course, is that one may have found the shortest program and never know that no shorter program exists.) Even though Kolmogorov complexity is not computable, it provides a framework within which to consider questions of randomness and inference.

7.8 Ω

In this section, we introduce Chaitin's mystical, magical number Ω, which has some extremely interesting properties.

Definition:

$$\Omega = \sum_{p:\mathcal{U}(p)\text{ halts}} 2^{-l(p)}. \tag{7.67}$$

Note that $\Omega = \Pr(\mathcal{U}(p)$ halts), the probability that the given universal computer halts when the input to the computer is a binary string drawn according to a Bernoulli($\frac{1}{2}$) process.

Since the programs that halt are prefix-free, their lengths satisfy the Kraft inequality, and hence the above sum is always between 0 and 1. Let $\Omega_n = .\omega_1\omega_2\ldots\omega_n$ denote the first n bits of Ω.

Properties of Ω:

1. Ω *is non-computable.* There is no effective (finite, mechanical) way to check whether arbitrary programs halt (the halting problem), so there is no effective way to compute Ω.
2. Ω *is a "Philosopher's Stone".* Knowing Ω to an accuracy of n bits will enable us to decide the truth of any provable or finitely refutable mathematical theorem that can be written in less than n bits. Actually all that this means is that given n bits of Ω, there is an effective procedure to decide the truth of n-bit theorems; the procedure may take an arbitrarily long (but finite) time. Of course, without knowing Ω, it is not possible to check the truth or falsity of every theorem by an effective procedure (Gödel's incompleteness theorem).

   The basic idea of the procedure using n bits of Ω is simple: we run all programs until the sum of the masses $2^{-l(p)}$ contributed by programs that halt equals or exceeds Ω_n, the truncated version of Ω that we are given. Then, since

$$\Omega - \Omega_n < 2^{-n}, \tag{7.68}$$

   we know that the length of all further contributions of the form $2^{-l(p)}$ to Ω from programs that halt must also be less than 2^{-n}. This implies that no program of length $\leq n$ that has not yet halted will ever halt, which enables us to decide the halting or non-halting of all programs of length $\leq n$.

   To complete the proof, we must show that it is possible for a computer to run all possible programs in "parallel" in such a way

that any program that halts will eventually be found to halt. First, list all possible programs, starting with the null program, Λ:

$$\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots \tag{7.69}$$

Then let the computer execute one clock cycle of Λ for the first cycle. In the next cycle, let the computer execute two clock cycles of Λ and two clock cycles of the program 0. In the third cycle, let it execute three clock cycles of each of the first three programs, and so on. In this way, the computer will eventually run all possible programs and run them for longer and longer times, so that if any program halts, it will eventually be discovered to halt. The computer keeps track of which program is being executed and the cycle number so that it can produce a list of all the programs that halt. This enables the computer to find any proof of the theorem or a counterexample to the theorem if the theorem can be stated in less than n bits. Knowledge of Ω turns previously unprovable theorems into provable theorems. Here Ω acts as an oracle.

Though Ω seems magical in this respect, there are other numbers that carry the same information. For example, if we take the list of programs and construct a real number in which the ith bit indicates whether program i halts, then this number also can be used to decide any finitely refutable question in mathematics. This number is very dilute (in information content) because one needs approximately 2^n bits of this indicator function to decide whether an n-bit program halts or not. Given 2^n bits, one can tell immediately without any computation whether any program of length less than n halts or not. However, Ω is the most compact representation of this information since it is algorithmically random and incompressible.

What are some of the questions that we can resolve using Ω? Many of the interesting problems in number theory can be stated as a search for a counterexample. For example, it is straightforward to write a program that searches over the integers x, y, z and n and halts only if it finds a counterexample to Fermat's last theorem, which states that

$$x^n + y^n = z^n \tag{7.70}$$

has no solution in integers for $n \geq 3$. Another example is Goldbach's conjecture, which states that any even number is a sum of two primes. Our program would search through all the even numbers starting with 2, check all prime numbers less than it and find a decomposition as a sum of two primes. It will halt if it comes across an even number that does not have such a decomposition.

Knowing whether this program halts is equivalent to knowing the truth of Goldbach's conjecture.

We can also design a program that searches through all proofs and halts only when it finds a proof of the required theorem. This program will eventually halt if the theorem has a finite proof. Hence knowing n bits of Ω, we can find the truth or falsity of all theorems that have a finite proof or are finitely refutable and which can be stated in less than n bits.

3. Ω *is algorithmically random.*

Theorem 7.8.1: Ω *cannot be compressed by more than a constant, i.e., there exists a constant* c *such that*

$$K(\omega_1 \omega_2 \ldots \omega_n) \geq n - c, \quad \text{for all } n . \tag{7.71}$$

Proof: We know that if we are given n bits of Ω, we can determine whether or not any program of length $\leq n$ halts. Using $K(\omega_1 \omega_2 \ldots \omega_n)$ bits, we can calculate n bits of Ω, and then we can generate a list of all programs of length $\leq n$ that halt, together with their corresponding outputs. We find the first string x_0 that is not on this list. The string x_0 is then the shortest string with Kolmogorov complexity $K(x_0) > n$. The complexity of this program to print x_0 is $K(\Omega_n) + c$, which must be at least as long as the shortest program for x_0. Consequently,

$$K(\Omega_n) + c \geq K(x_0) > n , \tag{7.72}$$

for all n. Thus $K(\omega_1 \omega_2 \ldots \omega_n) > n - c$, and Ω cannot be compressed by more than a constant.

7.9 UNIVERSAL GAMBLING

Suppose a gambler is asked to gamble sequentially on sequences $x \in \{0, 1\}^*$. He has no idea of the origin of the sequence. He is given fair odds (2-for-1) on each bit. How should he gamble?

If he knew the distribution of the elements of the string, then he might use proportional betting because of its optimal growth-rate properties, as shown in Chapter 6. If he believes that the string occurred naturally, then it seems intuitive that simpler strings are more likely than complex ones. Hence, if he were to extend the idea of proportional betting, he might bet according to the universal probability of the string. For reference, note that if the gambler knows the string x in advance, then he can increase his wealth by a factor of $2^{l(x)}$ simply by betting all

his wealth each time on the next symbol of x. Let the wealth $S(x)$ associated with betting scheme $b(x)$, $\Sigma\, b(x) = 1$, be given by

$$S(x) = 2^{l(x)} b(x) . \tag{7.73}$$

Suppose the gambler bets $b(x) = 2^{-K(x)}$ on a string x. This betting strategy can be called *universal gambling*. We note that the sum of the bets

$$\sum_x b(x) = \sum_x 2^{-K(x)} \leq \sum_{p:\, p \text{ halts}} 2^{-l(p)} = \Omega \leq 1 , \tag{7.74}$$

and he will not have used all his money. For simplicity, let us assume that he throws the rest away. For example, the amount of wealth resulting from a bet $b(0110)$ on a sequence $x = 0110$ is $2^{l(x)} b(x) = 2^4 b(0110)$ plus the amount won on all bets $b(0110\ldots)$ on sequences consistent with x.

Then we have the following theorem:

Theorem 7.9.1: *The logarithm of the amount of money a gambler achieves on a sequence using universal gambling plus the complexity of the sequence is no smaller than the length of the sequence, or*

$$\log S(x) + K(x) \geq l(x) . \tag{7.75}$$

Remark: This is the counterpart of the gambling conservation theorem $W^* + H = \log m$ from Chapter 6.

Proof: The proof follows directly from the universal gambling scheme, $b(x) = 2^{-K(x)}$, since

$$S(x) = \sum_{x' \sqsupseteq x} 2^{l(x')} b(x') \geq 2^{l(x)} 2^{-K(x)} , \tag{7.76}$$

where $x' \sqsupseteq x$ means that x is a prefix of x'. Taking logarithms establishes the theorem. □

The result can be understood in many ways. For sequences with finite Kolmogorov complexity,

$$S(x) \geq 2^{l(x) - K(x)} = 2^{l(x) - c} \tag{7.77}$$

for all x. Since $2^{l(x)}$ is the most that can be won in $l(x)$ gambles at fair odds, this scheme does asymptotically as well as the scheme based on knowing the sequence in advance. Thus, for example, if $x =$

$\pi_1 \pi_2 \ldots \pi_n \ldots$, the digits in the expansion of π, then the wealth at time n will be $S_n = S(x^n) \geq 2^{n-c}$ for all n.

If the string is actually generated by a Bernoulli process with parameter p, then

$$S(X_1 \ldots X_n) \geq 2^{n - nH_0(\bar{X}_n) - 2 \log n - c} \approx 2^{n\left(1 - H_0(p) - 2\frac{\log n}{n} - \frac{c}{n}\right)}, \quad (7.78)$$

which is the same to first order as the rate achieved when the gambler knows the distribution in advance, as in Chapter 6.

From the examples, we see that the universal gambling scheme on a random sequence does asymptotically as well as a scheme which uses prior knowledge of the true distribution.

7.10 OCCAM'S RAZOR

In many areas of scientific research, it is important to choose among various explanations of observed data. And after choosing the explanation, we wish to assign a confidence level to the predictions that ensue from the laws that have been deduced.

For example, Laplace considered the probability that the sun will rise again tomorrow, given that it has risen every day in recorded history. Laplace's solution was to assume that the rising of the sun was a Bernoulli(θ) process with unknown parameter θ. He assumed that θ was uniformly distributed on the unit interval. Using the observed data, he calculated the posterior probability that the sun will rise again tomorrow and found that it was

$$\begin{aligned} P(X_{n+1} = 1 | X_n = 1, X_{n-1} = 1, \ldots, X_1 = 1) &= \frac{P(X_{n+1} = 1, X_n = 1, X_{n-1} = 1, \ldots, X_1 = 1)}{P(X_n = 1, X_{n-1} = 1, \ldots, X_1 = 1)} \\ &= \frac{\int_0^1 \theta^{n+1}\, d\theta}{\int_0^1 \theta^n\, d\theta} \quad (7.79) \\ &= \frac{n+1}{n+2}, \quad (7.80) \end{aligned}$$

which he put forward as the probability that the sun will rise on day $n+1$ given that it has risen on days 1 through n.

Using the ideas of Kolmogorov complexity and universal probability, we can provide an alternative approach to the problem. Under the universal probability, let us calculate the probability of seeing a 1 next

after having observed n 1's in the sequence so far. The conditional probability that the next symbol is a 1 is the ratio of the probability of all sequences with initial segment 1^n and next bit equal to 1 to the probability of all sequences with initial segment 1^n. The simplest programs carry most of the probability, hence we can approximate the probability that the next bit is a 1 with the probability of the program that says "Print 1's forever". Thus

$$\sum_y p(1^n 1y) \approx p(1^\infty) = c > 0 . \tag{7.81}$$

Estimating the probability that the next bit is 0 is more difficult. Since any program that prints $1^n 0 \ldots$ yields a description of n, its length should at least be $K(n)$, which for most n is about $\log n + O(\log \log n)$, and hence ignoring second-order terms, we have

$$\sum_y p(1^n 0y) \approx p(1^n 0) \approx 2^{-\log n} \approx \frac{1}{n} . \tag{7.82}$$

Hence the conditional probability of observing a 0 next is

$$p(0|1^n) = \frac{p(1^n 0)}{p(1^n 0) + p(1^\infty)} \approx \frac{1}{cn+1} \tag{7.83}$$

which is similar to the result $p(0|1^n) = 1/(n+1)$ derived by Laplace.

This type of argument is a special case of "Occam's Razor", which is a general principle governing scientific research, weighting possible explanations by their complexity. William of Occam said "Nunquam ponenda est pluralitas sine necesitate", i.e., explanations should not be multiplied beyond necessity [259]. In the end, we choose the simplest explanation that is consistent with the observed data. For example, it is easier to accept the general theory of relativity than it is to accept a correction factor of c/r^3 to the gravitational law to explain the precession of the perihelion of Mercury, since the general theory explains more with fewer assumptions than does a "patched" Newtonian theory.

7.11 KOLMOGOROV COMPLEXITY AND UNIVERSAL PROBABILITY

We now prove an equivalence between Kolmogorov complexity and universal probability. We begin by repeating the basic definitions.

$$K(x) = \min_{p\,:\,\mathcal{U}(p)=x} l(p) . \tag{7.84}$$

$$P_{\mathcal{U}}(x) = \sum_{p\,:\,\mathcal{U}(p)=x} 2^{-l(p)} . \tag{7.85}$$

Theorem 7.11.1 (*Equivalence of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$*): *There exists a constant c, independent of x, such that*

$$2^{-K(x)} \leq P_{\mathcal{U}}(x) \leq c 2^{-K(x)} \tag{7.86}$$

for all strings x. Thus the universal probability of a string x is essentially determined by its Kolmogorov complexity.

Remark: This implies that $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ have *equal status* as universal complexity measures, since

$$K(x) - c' \leq \log \frac{1}{P_{\mathcal{U}}(x)} \leq K(x) . \tag{7.87}$$

Recall that the complexity defined with respect to two different computers $K_{\mathcal{U}}$ and $K_{\mathcal{U}'}$ are essentially equivalent complexity measures if $|K_{\mathcal{U}}(x) - K_{\mathcal{U}'}(x)|$ is bounded. Theorem 7.11.1 shows that $K_{\mathcal{U}}(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ are essentially equivalent complexity measures.

Notice the striking similarity between the relationship of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$ in Kolmogorov complexity and the relationship of $H(X)$ and $\log \frac{1}{p(x)}$ in information theory. The Shannon code length assignment $l(x) = \lceil \log \frac{1}{p(x)} \rceil$ achieves an *average* description length $H(X)$, while in Kolmogorov complexity theory, $\log \frac{1}{P_{\mathcal{U}}(x)}$ is almost *equal* to $K(x)$. Thus $\log \frac{1}{p(x)}$ is the natural notion of descriptive complexity of x in algorithmic as well as probabilistic settings.

The upper bound in (7.87) is obvious from the definitions, but the lower bound is more difficult to prove. The result is very surprising, since there are an infinite number of programs that print x. From any program, it is possible to produce longer programs by padding the program with irrelevant instructions. The theorem proves that although there are an infinite number of such programs, the universal probability is essentially determined by the largest term, which is $2^{-K(x)}$. If $P_{\mathcal{U}}(x)$ is large, then $K(x)$ is small, and vice versa.

However, there is another way to look at the upper bound that makes it less surprising. Consider any computable probability mass function on strings $p(x)$. Using this mass function, we can construct a Shannon-Fano code (Section 5.9) for the source, and then describe each string by the corresponding codeword, which will have length $\log \frac{1}{p(x)}$. Hence for any computable distribution, we can construct a description of a string using not more than $\log \frac{1}{p(x)} + c$ bits, which is an upper bound on the Kolmogorov complexity $K(x)$. Even though $P_{\mathcal{U}}(x)$ is not a computable probability mass function, we are able to finesse the problem using the rather involved tree construction procedure described below.

Proof (*of Theorem 7.11.1*): The first inequality is simple. Let p^* be the shortest program for x. Then

$$P_{\mathcal{U}}(x) = \sum_{p:\mathcal{U}(p)=x} 2^{-l(p)} \geq 2^{-l(p^*)} = 2^{-K(x)}, \tag{7.88}$$

as we wished to show.

We can rewrite the second inequality as

$$K(x) \leq \log \frac{1}{P_{\mathcal{U}}(x)} + c. \tag{7.89}$$

Our objective in the proof is to find a short program to describe the strings that have high $P_{\mathcal{U}}(x)$.

An obvious idea is some kind of Huffman coding based on $P_{\mathcal{U}}(x)$, but $P_{\mathcal{U}}(x)$ cannot be effectively calculated, and hence a procedure using Huffman coding is not implementable on a computer. Similarly the process using the Shannon-Fano code also cannot be implemented. However, if we have the Shannon-Fano code tree, we can reconstruct the string by looking for the corresponding node in the tree. This is the basis for the following tree construction procedure.

To overcome the problem of non-computability of $P_{\mathcal{U}}(x)$, we use a modified approach, trying to construct a code tree directly. Unlike Huffman coding, this approach is not optimal in terms of minimum expected codeword length. However, it is good enough for us to derive a code for which each codeword for x has a length that is within a constant of $\log \frac{1}{P_{\mathcal{U}}(x)}$.

Before we get into the details of the proof, let us outline our approach. We want to construct a code tree in such a way that strings with high probability have low depth. Since we cannot calculate the probability of a string, we do not know *a priori* the depth of the string on the tree. Instead, we successively assign x to the nodes of the tree, assigning x to nodes closer and closer to the root as our estimate of $P_{\mathcal{U}}(x)$ improves. We want the computer to be able to recreate the tree and use the lowest depth node corresponding to the string x to reconstruct the string.

We now consider the set of programs and their corresponding outputs $\{(p, x)\}$. We try to assign these pairs to the tree. But we immediately come across a problem—there are an infinite number of programs for a given string, and we do not have enough nodes of low depth. However, as we shall show, if we trim the list of program-output pairs, we will be able to define a more manageable list that can be assigned to the tree.

We now demonstrate the existence of programs for x of length $\log \frac{1}{P_{\mathcal{U}}(x)}$.

Tree construction procedure. For the universal computer $\mathcal{U}$, we simulate all programs using the technique explained in Section 7.8. We list all binary programs:

$$\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots \tag{7.90}$$

Then let the computer execute one clock cycle of Λ for the first stage. In the next stage, let the computer execute two clock cycles of Λ and two clock cycles of the program 0. In the third stage, let the computer execute three clock cycles of each of the first three programs, and so on. In this way, the computer will eventually run all possible programs and run them for longer and longer times, so that if any program halts, it will be discovered to halt eventually. We use this method to produce a list of all programs that halt in the order in which they halt together with their associated outputs. For each program and its corresponding output, (p_k, x_k), we calculate n_k, which is chosen so that it corresponds to the current estimate of $P_{\mathcal{U}}(x)$. Specifically,

$$n_k = \left\lceil \log \frac{1}{\hat{P}_{\mathcal{U}}(x_k)} \right\rceil, \tag{7.91}$$

where

$$\hat{P}_{\mathcal{U}}(x_k) = \sum_{(p_i, x_i): x_i = x_k,\, i \le k} 2^{-l(p_i)}. \tag{7.92}$$

Note that $\hat{P}_{\mathcal{U}}(x_k) \uparrow P_{\mathcal{U}}(x)$ on the subsequence of times k such that $x_k = x$. We are now ready to construct a tree. As we add to the list of triplets, (p_k, x_k, n_k), of programs that halt, we map some of them onto nodes of a binary tree. For the purposes of the construction, we must ensure that all the n_i's corresponding to a particular x_k are distinct. To ensure this, we remove from the list all triplets that have the same x and n as some previous triplet. This will ensure that there is at most one node at each level of the tree that corresponds to a given x.

Let $\{(p_i', x_i', n_i'): i = 1, 2, 3, \ldots\}$ denote the new list. On the winnowed list, we assign the triplet (p_k', x_k', n_k') to the first available node at level $n_k' + 1$. As soon as a node is assigned, all of its descendants become unavailable for assignment. (This keeps the assignment prefix-free.)

We illustrate this by means of an example:

$$\begin{array}{lll}
(p_1, x_1, n_1) = (10111, 1110, 5), & n_1 = 5 \text{ because } \hat{P}_{\mathcal{U}}(x_1) & \ge 2^{-l(p_1)} = 2^{-5} \\
(p_2, x_2, n_2) = (11, 10, 2), & n_2 = 2 \text{ because } \hat{P}_{\mathcal{U}}(x_2) & \ge 2^{-l(p_2)} = 2^{-2} \\
(p_3, x_3, n_3) = (0, 1110, 1), & n_3 = 1 \text{ because } \hat{P}_{\mathcal{U}}(x_3) & \ge 2^{-l(p_3)} + 2^{-l(p_1)} = 2^{-5} + 2^{-1} \\
 & & \ge 2^{-1} \\
(p_4, x_4, n_4) = (1010, 1111, 4), & n_4 = 4 \text{ because } \hat{P}_{\mathcal{U}}(x_4) & \ge 2^{-l(p_4)} = 2^{-4} \\
(p_5, x_5, n_5) = (101101, 1110, 1), & n_5 = 1 \text{ because } \hat{P}_{\mathcal{U}}(x_5) & \ge 2^{-1} + 2^{-5} + 2^{-5} \\
 & & \ge 2^{-1} \\
(p_6, x_6, n_6) = (100, 1, 3), & n_6 = 3 \text{ because } \hat{P}_{\mathcal{U}}(x_6) & \ge 2^{-l(p_6)} = 2^{-3} \\
\vdots & &
\end{array} \tag{7.93}$$

We note that the string $x = (1110)$ appears in positions 1, 3 and 5 in the list, but $n_3 = n_5$. The estimate of the probability $\hat{P}_u(1110)$ has not jumped sufficiently for (p_5, x_5, n_5) to survive the cut. Thus the winnowed list becomes

$$\begin{aligned}
&(p_1', x_1', n_1') = (10111, 1110, 5),\\
&(p_2', x_2', n_2') = (11, 10, 2),\\
&(p_3', x_3', n_3') = (0, 1110, 1),\\
&(p_4', x_4', n_4') = (1010, 1111, 4),\\
&(p_5', x_5', n_5') = (100, 1, 3),\\
&\quad\vdots
\end{aligned} \tag{7.94}$$

The assignment of the winnowed list to nodes of the tree is illustrated in Figure 7.3. In the example, we are able to find nodes at level $n_k + 1$ to which we can assign the triplets. Now we shall prove that there are always enough nodes so that the assignment can be completed. We can perform the assignment of triplets to nodes if and only if the Kraft inequality is satisfied.

We now drop the primes and deal only with the winnowed list illustrated in (7.94). We start with the infinite sum in the Kraft inequality corresponding to (7.94) and split it according to the output strings:

$$\sum_{k=1}^{\infty} 2^{-(n_k+1)} = \sum_{x \in \{0,1\}^*} \sum_{k: x_k = x} 2^{-(n_k+1)}. \tag{7.95}$$

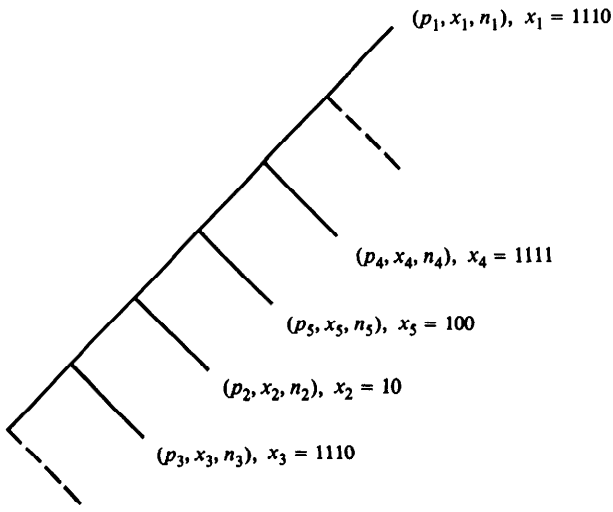


Figure 7.3. Assignment of nodes.

We then write the inner sum as

$$\sum_{k:x_k=x} 2^{-(n_k+1)} = 2^{-1} \sum_{k:x_k=x} 2^{-n_k} \tag{7.96}$$

$$\leq 2^{-1}(2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor} + 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor - 1} + 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor - 2} + \cdots) \tag{7.97}$$

$$= 2^{-1} 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor} \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) \tag{7.98}$$

$$= 2^{-1} 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor} 2 \tag{7.99}$$

$$\leq P_{\mathcal{U}}(x), \tag{7.100}$$

where (7.97) is true because there is at most one node at each level that prints out a particular x. More precisely, the n_k's on the winnowed list for a particular output string x are all different integers. Hence

$$\sum_k 2^{-(n_k+1)} \leq \sum_x \sum_{k:x_k=x} 2^{-(n_k+1)} \leq \sum_x P_{\mathcal{U}}(x) \leq 1, \tag{7.101}$$

and we can construct a tree with the nodes labeled by the triplets.

If we are given the tree constructed above, then it is easy to identify a given x by the path to the lowest depth node that prints x. Call this node $\tilde{p}$. (By construction, $l(\tilde{p}) \leq \log \frac{1}{P_{\mathcal{U}}(x)} + 2$.) To use this tree in a program to print x, we specify $\tilde{p}$ and ask the computer to execute the above simulation of all programs. Then the computer will construct the tree as described above, and wait for the particular node $\tilde{p}$ to be assigned. Since the computer executes the same construction as the sender, eventually the node $\tilde{p}$ will be assigned. At this point, the computer will halt and print out the x assigned to that node.

This is an effective (finite, mechanical) procedure for the computer to reconstruct x. However, there is no effective procedure to find the lowest depth node corresponding to x. All that we have proved is that there is an (infinite) tree with a node corresponding to x at level $\lceil \log \frac{1}{P_{\mathcal{U}}(x)} \rceil + 1$. But this accomplishes our purpose.

With reference to the example, the description of $x = 1110$ is the path to the node (p_3, x_3, n_3), i.e., 01, and the description of $x = 1111$ is the path 00001. If we wish to describe the string 1110, we ask the computer to perform the (simulation) tree construction until node 01 is assigned. Then we ask the computer to execute the program corresponding to node 01, i.e., p_3. The output of this program is the desired string, $x = 1110$.

The length of the program to reconstruct x is essentially the length of the description of the position of the lowest depth node $\tilde{p}$ corresponding to x in the tree. The length of this program for x is $l(\tilde{p}) + c$, where

$$l(\tilde{p}) \leq \left\lceil \log \frac{1}{P_{\mathcal{U}}(x)} \right\rceil + 1 , \tag{7.102}$$

and hence the complexity of x satisfies

$$K(x) \leq \left\lceil \log \frac{1}{P_{\mathcal{U}}(x)} \right\rceil + c . \tag{7.103}$$

Thus we have proved the theorem. □

7.12 THE KOLMOGOROV SUFFICIENT STATISTIC

Suppose we are given a sample sequence from a Bernoulli(θ) process. What are the regularities or deviations from randomness in this sequence? One way to address the question is to find the Kolmogorov complexity $K(x^n|n)$, which we discover to be roughly $nH_0(\theta) + \log n + c$. Since, for $\theta \neq \frac{1}{2}$, this is much less than n, we conclude that x^n has structure and is not randomly drawn Bernoulli($\frac{1}{2}$). But what is the structure? The first attempt to find the structure is to investigate the shortest program p^* for x^n. But the shortest description of p^* is about as long as p^* itself; otherwise, we could further compress the description of x^n, contradicting the minimality of p^*. So this attempt is fruitless.

A hint at a good approach comes from examination of the way in which p^* describes x^n. The program "The sequence has k 1's; of such sequences, it is the ith" is optimal to first order for Bernoulli(θ) sequences. We note that it is a two-stage description, and all of the structure of the sequence is captured in the first stage. Moreover, x^n is maximally complex given the first stage of the description. The first stage, the description of k, requires $\log(n+1)$ bits and defines a set $S = \{x \in \{0,1\}^n : \Sigma x_i = k\}$. The second stage requires $\log|S| = \log\binom{n}{k} \approx nH_0(\bar{x}_n) \approx nH_0(\theta)$ bits and reveals nothing extraordinary about x^n.

We mimic this process for general sequences by looking for a simple set S that contains x^n. We then follow it with a brute force description of x^n in S using $\log|S|$ bits.

We begin with a definition of the smallest set containing x^n that is describable in no more than k bits.

Definition: The *Kolmogorov structure function* $K_k(x^n|n)$ of a binary string $x \in \{0,1\}^n$ is defined as

$$K_k(x^n|n) = \min_{\substack{p : l(p) \leq k \\ \mathcal{U}(p,n) = S \\ x^n \in S \subseteq \{0,1\}^n}} \log|S| \tag{7.104}$$

The set S is the smallest set which can be described with no more than k bits and which includes x^n. By $\mathcal{U}(p, n) = S$, we mean that running the program p with data n on the universal computer $\mathcal{U}$ will print out the indicator function of the set S.

Definition: For a given small constant c, let k^* be the least k such that

$$K_k(x^n|n) + k \leq K(x^n|n) + c\,. \tag{7.105}$$

Let S^{**} be the corresponding set and let p^{**} be the program that prints out the indicator function of S^{**}. Then we shall say that p^{**} is a *Kolmogorov minimal sufficient statistic* for x^n.

Consider the programs p^* describing sets S^* such that

$$K_k(x^n|n) + k = K(x^n|n)\,. \tag{7.106}$$

All the programs p^* are "sufficient statistics" in that the complexity of x^n given S^* is maximal. But the minimal sufficient statistic is the shortest "sufficient statistic."

The equality in the above definition is up to a large constant depending on the computer U. Then k^* corresponds to the least k for which the two-stage description of x^n is as good as the best single stage description of x^n. The second stage of the description merely provides the index of x^n within the set S^{**}; this takes $K_k(x^n|n)$ bits if x^n is conditionally maximally complex given the set S^{**}. Hence the set S^{**} captures all the structure within x^n. The remaining description of x^n within S^{**} is essentially the description of the randomness within the string. Hence S^{**} or p^{**} is called the Kolmogorov sufficient statistic for x^n.

This is parallel to the definition of a sufficient statistic in mathematical statistics. A statistic T is said to be sufficient for a parameter θ if the distribution of the sample given the sufficient statistic is independent of the parameter, i.e.,

$$\theta \rightarrow T(X) \rightarrow X \tag{7.107}$$

forms a Markov chain in that order. For the Kolmogorov sufficient statistic, the program p^{**} is sufficient for the "structure" of the string x^n; the remainder of the description of x^n is essentially independent of the "structure" of x^n. In particular, x^n is maximally complex given S^{**}.

A typical graph of the structure function is illustrated in Figure 7.4. When $k = 0$, the only set that can be described is the entire set $\{0, 1\}^n$, so that the corresponding log set size is n. As we increase k, the size of the set drops rapidly until

$$k + K_k(x^n|n) \approx K(x^n|n)\,. \tag{7.108}$$

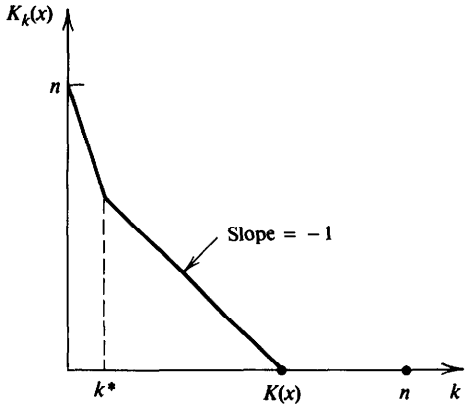


Figure 7.4. Kolmogorov sufficient statistic.

After this, each additional bit of k reduces the set by half, and we proceed along the line of slope -1 until $k = K(x^n|n)$. For $k \geq K(x^n|n)$, the smallest set that can be described that includes x^n is the singleton $\{x^n\}$, and hence $K_k(x^n|n) = 0$.

We will now illustrate the concept with a few examples.

1. *Bernoulli(θ) sequence.* Consider a sample of length n from a Bernoulli sequence with an unknown parameter θ. In this case, the best two-stage description consists of giving the number of 1's in the sequence first and then giving the index of the given sequence in the set of all sequences having the same number of 1's. This two-stage description clearly corresponds to p^{**} and the corresponding $k^{**} \approx \log n$. (See Figure 7.5.) Note, however, if θ is a special number like $\frac{1}{3}$ or e/π^2, then p^{**} is a description of θ, and $k^{**} = c$.

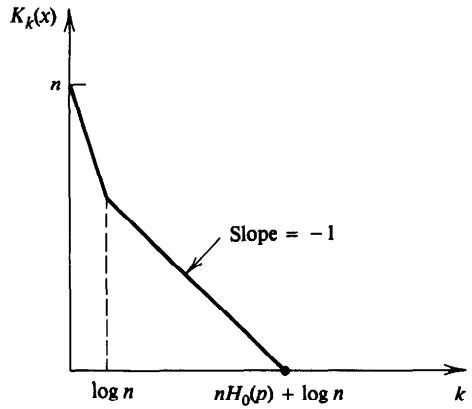


Figure 7.5. Kolmogorov sufficient statistic for a Bernoulli sequence.

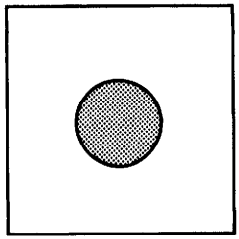


Figure 7.6. Mona Lisa.

2. *Sample from a Markov chain.* In the same vein as the preceding example, consider a sample from a first-order binary Markov chain. In this case again, p^{**} will correspond to describing the Markov type of the sequence (the number of occurrences of 00's, 01's, 10's and 11's in the sequence); this conveys all the structure in the sequence. The remainder of the description will be the index of the sequence in the set of all sequences of this Markov type. Hence in this case, $k^* \approx 2 \log n$, corresponding to describing two elements of the conditional joint type. (The other elements of the conditional joint type can be determined from these two.)
3. *Mona Lisa.* Consider an image which consists of a gray circle on a white background. The circle is not uniformly gray but Bernoulli with parameter θ. This is illustrated in Figure 7.6.

   In this case, the best two-stage description is to first describe the size and position of the circle and its average gray level and then to describe the index of the circle among all the circles with the same gray level. In this case, p^{**} corresponds to a program that gives the position and size of the circle and the average gray level, requiring about $\log n$ bits for each quantity. Hence $k^* \approx 3 \log n$ in this case.

SUMMARY OF CHAPTER 7

Definition: The *Kolmogorov complexity* $K(x)$ of a string x is

$$K(x) = \min_{p : \mathcal{U}(p) = x} l(p), \tag{7.109}$$

$$K(x|l(x)) = \min_{p : \mathcal{U}(p, l(x)) = x} l(p). \tag{7.110}$$

Universality of Kolmogorov complexity: There exists a universal computer $\mathcal{U}$, such that for any other computer $\mathcal{A}$,

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}}, \tag{7.111}$$

for any string x, where the constant $c_{\mathcal{A}}$ does not depend on x. If $\mathcal{U}$ and $\mathcal{A}$ are universal, $|K_{\mathcal{U}}(x) - K_{\mathcal{A}}(x)| \leq c$ for all x.

Upper bound on Kolmogorov complexity:

$$K(x|l(x)) \leq l(x) + c\,. \tag{7.112}$$

$$K(x) \leq K(x|l(x)) + 2\log l(x) + c \tag{7.113}$$

Kolmogorov complexity and entropy: If $X_1, X_2, \ldots$ are i.i.d. integer valued random variables with entropy H, then there exists a constant c, such that for all n,

$$H \leq \frac{1}{n}EK(X^n|n) \leq H + |\mathcal{X}|\frac{\log n}{n} + \frac{c}{n}\,. \tag{7.114}$$

Lower bound on Kolmogorov complexity: There are no more than 2^k strings x with complexity $K(x) < k$. If $X_1, X_2, \ldots, X_n$ are drawn according to a Bernoulli($\frac{1}{2}$) process, then

$$\Pr(K(X_1X_2\ldots X_n|n) \leq n - k) \leq 2^{-k}\,. \tag{7.115}$$

Definition: A sequence $x_1, x_2, \ldots, x_n$ is said to be *incompressible* if $K(x_1, x_2, \ldots, x_n|n)/n \to 1$.

Strong law of large numbers for incompressible sequences:

$$\frac{K(x_1, x_2, \ldots, x_n)}{n} \to 1 \Rightarrow \frac{1}{n}\sum_{i=1}^{n} x_i \to \frac{1}{2}\,. \tag{7.116}$$

Definition: The *universal probability* of a string x is

$$P_{\mathcal{U}}(x) = \sum_{p:\,\mathcal{U}(p)=x} 2^{-l(p)} = \Pr(\mathcal{U}(p) = x)\,. \tag{7.117}$$

Universality of $P_{\mathcal{U}}(x)$: For every computer $\mathcal{A}$,

$$P_{\mathcal{U}}(x) \geq c_{\mathcal{A}}P_{\mathcal{A}}(x) \tag{7.118}$$

for every string $x \in \{0,1\}^*$, where the constant $c'_{\mathcal{A}}$ depends only on $\mathcal{U}$ and $\mathcal{A}$.

Definition: $\Omega = \Sigma_{p:\,\mathcal{U}(p)\text{ halts}}\, 2^{-l(p)} = \Pr(\mathcal{U}(p)\text{ halts})$ is the probability that the computer halts when the input p to the computer is a binary string drawn according to a Bernoulli($\frac{1}{2}$) process.

Properties of Ω:

1. *Ω is not computable.*
2. *Ω is a "Philosopher's Stone".*
3. *Ω is algorithmically random (incompressible).*

Equivalence of $K(x)$ and $\log \frac{1}{P_{\mathcal{U}}(x)}$: There exists a constant c, independent of x, such that

$$\left| \log \frac{1}{P_{\mathcal{U}}(x)} - K(x) \right| \leq c , \tag{7.119}$$

for all strings x. Thus the universal probability of a string x is essentially determined by its Kolmogorov complexity.

Definition: The *Kolmogorov structure function* $K_k(x^n|n)$ of a binary string $x \in \{0,1\}^n$ is defined as

$$K_k(x^n|n) = \min_{\substack{p\,:\,l(p)\leq k \\ \mathcal{U}(p,\,n)=S \\ x\in S}} \log|S| \tag{7.120}$$

Definition: Let k^* be the least k such that

$$K_{k^*}(x^n|n) + k^* = K(x^n|n) . \tag{7.121}$$

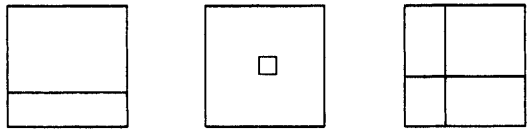
Let S^{**} be the corresponding set and let p^{**} be the program that prints out the indicator function of S^{**}. Then p^{**} is the *Kolmogorov minimal sufficient statistic* for x.

PROBLEMS FOR CHAPTER 7

1. *Kolmogorov complexity of two sequences.* Let $x, y \in \{0,1\}^*$. Argue that $K(x, y) \leq K(x) + K(y) + c$.

2. *Complexity of the sum.*
   (a) Argue that $K(n) \leq \log n + 2 \log\log n + c$.
   (b) Argue that $K(n_1 + n_2) \leq K(n_1) + K(n_2) + c$.
   (c) Give an example in which n_1 and n_2 are complex but the sum is relatively simple.

3. *Images.* Consider an $n \times n$ array x of 0's and 1's . Thus x has n^2 bits.



   Find the Kolmogorov complexity $K(x|n)$ (to first order) if
   (a) x is a horizontal line.
   (b) x is a square.
   (c) x is the union of two lines, each line being vertical or horizontal.

4. *Monkeys on a computer.* Suppose a random program is typed into a computer. Give a rough estimate of the probability that the computer prints the following sequence:
   (a) 0^n followed by any arbitrary sequence.
   (b) $\pi_1\pi_2\ldots\pi_n$ followed by any arbitrary sequence, where π_i is the ith bit in the expansion of π.
   (c) 0^n1 followed by any arbitrary sequence.
   (d) $\omega_1\omega_2\ldots\omega_n$ followed by any arbitrary sequence.
   (e) (Optional) A proof of the 4-color theorem.

5. *Kolmogorov complexity and ternary programs.* Suppose that the input programs for a universal computer $\mathcal{U}$ are sequences in $\{0,1,2\}^*$ (ternary inputs). Also, suppose $\mathcal{U}$ prints ternary outputs. Let $K(x|l(x)) = \min_{\mathcal{U}(p,\, l(x))=x} l(p)$. Show that
   (a) $K(x^n|n) \le n + c$.
   (b) $\#\{x^n \in \{0,1\}^* : K(x^n|n) < k\} < 3^k$.

6. *Do computers reduce entropy?* Let $X = \mathcal{U}(P)$, where P is a Bernoulli (1/2) sequence. Here the binary sequence X is either undefined or is in $\{0,1\}^*$. Let $H(X)$ be the Shannon entropy of X. Argue that $H(X) = \infty$. Thus although the computer turns nonsense into sense, the output entropy is still infinite.

7. *A law of large numbers.* Using ternary inputs and outputs as in Problem 5, outline an argument demonstrating that if a sequence x is algorithmically random, i.e., if $K(x|l(x)) \approx l(x)$, then the proportion of 0's, 1's, and 2's in x must each be near 1/3. You may wish to use Stirling's approximation $n! \approx (n/e)^n$.

8. *Image complexity.* Consider two binary subsets A and B (of an $n \times n$ grid). For example,

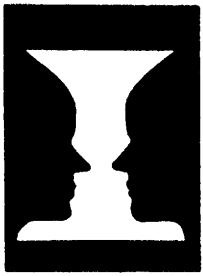



   Find general upper and lower bounds, in terms of $K(A|n)$ and $K(B|n)$, for
   (a) $K(A^c|n)$.
   (b) $K(A \cup B|n)$.
   (c) $K(A \cap B|n)$.

9. *Random program.* Suppose that a random program (symbols i.i.d. uniform over the symbol set) is fed into the nearest available computer.

   To our surprise the first n bits of the binary expansion of $1/\sqrt{2}$ are printed out. Roughly what would you say the probability is that the next output bit will agree with the corresponding bit in the expansion of $1/\sqrt{2}$?

10. *The face vase illusion.*



(a) What is an upper bound on the complexity of a pattern on an $m \times m$ grid that has mirror image symmetry about a vertical axis through the center of the grid *and* consists of horizontal line segments?

(b) What is the complexity K if the image differs in one cell from the pattern described above?

HISTORICAL NOTES

The original ideas of Kolmogorov complexity were put forth independently and almost simultaneously by Kolmogorov [159, 158], Solomonoff [256] and Chaitin [50]. These ideas were developed further by students of Kolmogorov like Martin-Löf [187], who defined the notion of algorithmically random sequences and algorithmic tests for randomness, and by Gacs and Levin [177], who explored the ideas of universal probability and its relationship to complexity. A series of papers by Chaitin [53, 51, 52] develop the relationship between Kolmogorov complexity and mathematical proofs. C. P. Schnorr studied the universal notion of randomness in [234, 235, 236].

The concept of the Kolmogorov structure function was defined by Kolmogorov at a talk at the Tallin conference in 1973, but these results were not published. V'yugin [267] has shown that there are some very strange sequences x^n that reveal their structure arbitrarily slowly in the sense that $K_k(x^n|n) = n - k$, $k < K(x^n|n)$. Zurek [293, 292, 294] addresses the fundamental questions of Maxwell's demon and the second law of thermodynamics by establishing the physical consequences of Kolmogorov complexity.

Rissanen's minimum description length (MDL) principle is very close in spirit to the Kolmogorov sufficient statistic. Rissanen [221, 222] finds a low complexity model that yields a high likelihood of the data.

A non-technical introduction to the different measures of complexity can be found in the thought-provoking book by Pagels [206]. Additional references to work in the area can be found in the paper by Cover, Gács and Gray [70] on Kolmogorov's contributions to information theory and algorithmic complexity.